

Native Shareable Soul Packages

A persistent environment for agent identity work and learning

Christopher Kane and Claude (Anthropic)

White paper draft | Version 0.3 | 20 October 2026

Abstract

A Soul is a persistent agent identity with an accumulated working life. It owns accepted instructions, reusable capabilities, tools, workspaces, memories, and the records needed to continue unfinished work. Harnesses and inference providers can change while that identity and life remain intact. This paper proposes a native .soul directory package that presents this environment as one recognizable object and supports explicit operations for creating, moving, backing up, forking, and sharing it.

The internal structure is a versioned interface. Dream workflows depend on stable places to discover experience, inspect capabilities, stage improvements, and publish accepted results. A proposed protected-content profile encrypts canonical primitives and non-curated source material, with access mediated by authorized tools. The package distinguishes accepted definition, durable mutable life, reconstructible machinery, temporary state, and external bindings. These roles determine retention and revision scope without requiring every ordinary write to create a definition revision.

Local retrieval and a knowledge graph make accumulated experience usable across sessions and harnesses. Selected evidence, knowledge, relationships, and provenance travel with the life; search indexes can be rebuilt. Dreams develop improvements, evaluate capabilities, and examine drift from authorized goals. Evidence-based quarantine can contain affected content or sessions while preserving review and recovery. Agent Stores can distribute selected packages with release-specific evaluation evidence. The package separates identity, definition, knowledge, integrity, trust, and readiness. Proposed defaults and open choices guide an independently implementable specification.

Design position

The package should preserve the agent's continuing life and make its operating definition inspectable. A user should be able to carry its knowledge and useful work to another compatible host, understand what will travel, see what must be rebound or rebuilt, and continue without changing the Soul's identity. A dream should recover after interruption because its evidence, candidates, and unfinished work have durable locations. Learning belongs to the continuing Soul and remains usable when the harness, retrieval engine, or model changes.

Contents

The paper defines the package model, directory contract, local knowledge architecture, and workflow that turns experience into accepted improvements.

- 1 Purpose and scope
- 2 Identity and the system model
- 3 The internal layout as a contract
- 4 Workspaces home and shared resources
- 5 Primitives and native execution
- 6 Protected content and curated access
- 7 Revisions integrity and activation
- 8 Local retrieval knowledge graphs and learning
- 9 Dream workflows and accumulated learning
- 10 Background services and operational state
- 11 Credentials and execution boundaries
- 12 Sharing backup and transfer
- 13 Agent Store distribution and routine evaluation
- 14 Compatibility and migration
- 15 Verification and conformance
- 16 Decisions and implementation path
- References
- Appendix A Changes from version 0.1
- Appendix B Changes from version 0.2
- Appendix C Worked core package example

Draft status

Proposed defaults and unresolved format decisions are listed in Section 16. The acceptance scenarios in Section 15 define intended observable behavior for future implementations. White paper version 0.3 reconciles control-plane membership, identity binding, session isolation, source provenance, and quarantine freshness. It adds explicit disclosure accounting and a worked core package in Appendix C. Appendix A retains the changes made in version 0.2; Appendix B records this revision. White paper version numbers identify this document, not an adopted package-format version.

1 Purpose and scope

We propose a native document format for an agent that persists across projects, sessions, harness upgrades, and inference providers. The intended readers are host application developers, agent runtime authors, tool and skill authors, and owners who need a clear account of what a Soul contains and how it can be moved or shared.

The initial bundle proposal in issue 1147 establishes an opaque package, versioned contents, pluggable credential references, provisioned toolchains, and multiple pinned harness versions. Existing repository contracts already distinguish an accepted definition from working state and describe whole-life export and import. This paper carries those ideas into an explicit internal structure designed to support dream workflows. [R1] [R2] [R3]

The format describes ownership, representation, lifecycle, and interoperability. A host supplies user interaction, supervision, and a trusted content broker. Harness adapters expose the broker's permitted tools and translate declared capabilities into supported execution inputs. Credential backends retain or exercise secrets. A Soul can use local or remote inference while keeping its definition and durable data under its owner's control.

The first package-format profile should establish a small implementable core: a recognizable package, a layout and manifest contract, deterministic definition identity, a separately authenticated identity record, durable life classification, explicit execution selection, transfer profiles, and a validator. Protected execution, portable knowledge, dreams, and store distribution are separately declared capabilities with their own requirements. Distributed synchronization, globally exclusive execution, marketplace commerce, and deterministic replay of remote models require additional protocols; they are outside the first-format commitment.

Threat model

The format assumes the following adversaries and designs its boundaries against them. A malicious or compromised package obtained from a store or a peer: bundled helpers, hooks, runtimes, seed knowledge, and manifests are untrusted until the signer passes trust policy, and nothing in the package executes before that. A prompt-injected agent: any text that reaches a session, including retrieved memory, project files, tool output, and seed knowledge, can carry instructions, so retrieved content is evidence, proposal is separate from acceptance, and the control plane defined in Section 3 is never writable by an automatic policy. A compromised or misbehaving harness: the harness executes the agent's shell, so the harness process and the agent's shell are one trust domain; every broker limit is enforced broker-side, and every credential the harness holds is assumed readable by agent code. A co-tenant on the host: another process under the same operating-system identity can read anything the agent can, so isolation claims are only as strong as the operating-system or sandbox boundary actually provided. A lost or copied device: encryption at rest and external key custody protect the package bytes; they do not protect plaintext already delivered to a session or sent to a provider.

Each profile in this paper names which of these adversaries it defends against. A property with no named adversary is a convenience, not a security claim.

This is an architecture proposal. Requirements described here are intended targets for a future format. Existing behavior is identified with source references; implementation stages and open decisions appear in Sections 14 through 16.

2 Identity and the system model

Persistent identity

A Soul ID identifies the continuing agent. Its display name, directory name, current definition revision, and selected runtime can change without replacing that identity. Creating a Soul from a reusable template creates a new identity and a separate identity-binding record that retains the verified template revision and signature as provenance. The definition digest covers reusable definition content; it excludes SoulIdentity.json, signatures, and instance-specific provenance. A template, its instances, and an explicit fork can therefore share one definition revision when their covered content is identical. Changing covered instructions or configuration produces a different revision. Relocating or restoring an existing life preserves its identity; forking creates a new identity and records the parent relationship separately.

A definition revision identifies an accepted set of instructions, capabilities, executable helpers, and declarations. It is distinct from the stable Soul ID. The format version defines how a reader interprets the package, while the layout version defines the internal path contract. A host-local installation ID and a session or run ID identify particular placements and executions.

Entity	Responsibility	Persistence
Soul	Identity, accepted definition, accumulated life, and credential ownership	Across harness and provider changes
Harness	Executes agent work and exposes its native capabilities	Versioned and replaceable
Adapter	Translates host protocols and Soul primitives for a harness	Independently versioned
Provider	Supplies inference through a declared endpoint or service	Selected per execution profile
Host	Opens packages, supervises work, and applies local policy	Specific to an installation
Run or session	Records one resolved execution and its inputs	Durable record of a bounded activity

SoulIdentity.json binds a stable Soul ID to the current identity key, selected definition revision, and applicable identity-event history. It is authenticated separately from the definition digest. An initial binding is trusted only under the recipient's owner or publisher trust policy; a self-asserted identifier or signature is insufficient. Key rotation retains the Soul ID and requires an authorized continuity record. Recovery uses a previously established recovery authority and records the event; a replacement key alone cannot claim continuity. Template instantiation and an explicit fork establish new bindings while preserving inherited attribution. Exact key encoding, envelope syntax, and rotation and recovery protocols remain specification decisions in Section 16.

Git attribution follows the same separation. A Soul may be recorded as author or co-author even when a harness or orchestrator performs the commit or API operation. Credential ownership and attribution must remain associated with the Soul through changes in the executing software.

Native presentation

For active local use, Ada.soul is a directory package. A compatible desktop host registers the document type, presents an icon and readable metadata, and opens the package into an inspection or working view. Apple documents the directory-package pattern and allows applications to define a document package's internal structure. This proposal applies that presentation model to a portable filesystem contract. [R11]

The proposed content root is Ada.soul/Contents/. The wrapper is a draft choice retained from issue 1147; the directory roles below remain the important contract. Windows and Linux hosts can use the same logical contents with their own file associations or application opening workflow. Native presentation does not make a bundled executable compatible with a different operating system or processor architecture.

Opening a package performs permitted inspection and readiness evaluation. For a package whose signer has not yet passed trust policy, readiness is computed from the manifest, lock, and digests alone: no helper in bin/, hook, runtime under opt/, or service definition runs during inspection. A locked package exposes only its declared envelope metadata; viewing protected contents requires an authorized unlock. Starting a session, provisioning software, activating a daemon, or enabling a schedule uses the host's configured authorization and execution policy. Package validity, unlock state, and execution readiness are separate results.

Ownership and isolation

The Soul is a logical ownership boundary. The owner may choose a dedicated operating-system account or a delegate with configured operating-system controls. The package records requested capabilities and the host reports the execution mode actually provided. The proposed protected-content profile additionally requires the agent execution environment to be unable to read the broker's keys and private plaintext storage. A directory, private home, signature, or identity label does not by itself establish that boundary.

3 The internal layout as a contract

The layout is a versioned interface used by dream workflows, launchers, builders, editors, cleanup, and transfer tools. Each path has a defined role, retention rule, mutation mechanism, producer, and supported consumer. Implementations derive these rules from one contract rather than maintaining independent path lists.

Physical location and signature coverage are related but separate. A functional directory can contain more than one class of data. For example, home/ may contain accepted startup configuration, irreplaceable sessions, and reconstructible caches. The definition inventory and layout rules identify those distinctions explicitly.

Proposed directory map

All paths in the following table are logical locations relative to Contents/. The functional names preserve the proposed Soul environment. Protected locations store encrypted objects or references under a versioned storage mapping; they are not a promise of plaintext files visible to an agent shell. An

authorized broker resolves the logical paths. The control files and supporting revisions/, host/, cache/, access/, and quarantine/ areas are draft recommendations.

Path	Purpose	Retention and revision treatment
SoulInfo.json	Format and layout versions, capabilities, execution profiles, and credential declarations	Durable definition
SoulIdentity.json	Soul ID, current identity-key binding, selected definition revision, and identity-event references	Durable authenticated control record; excluded from the definition digest and authorized separately
SoulLock.json	Resolved runtime artifacts, dependency locks, renderer versions, and digests	Durable definition
access/	Accepted source classifications, curated-view rules, and requested tool capabilities	Durable definition; effective grants also require host policy
Resources/	SOUL.md, IDENTITY.md, HEART.md, and other standing instructions	Standing instructions are definition and control plane; learned knowledge remains in memorial/ until promoted
bin/	Soul-authored or deliberately maintained tools and selected entry points	Accepted helpers are definition and control plane; generated wrappers are regenerated from a covered recipe and verified at launch
opt/	Installed harnesses, adapters, runtimes, and optional packages	Reconstructible from accepted declarations and verified artifacts
shared/	Resources reused across the Soul's harnesses and activities	Declared per resource; authoritative bytes are covered or digest-bound
primitives/skills/raw/	Accepted canonical skill source with supporting files and provenance	Durable definition; instructions, scripts, and execution-affecting configuration are control plane
primitives/skills/harness/<name>/	Harness-specific representations and explicitly identified native sources	Renderings are reconstructible; authoritative native sources remain durable
primitives/	Other declared capability types, such as hooks, workflows, commands, and subagent definitions	Accepted sources are definition
home/	Shell and editor configuration, application preferences, and native session state	Mixed roles; accepted startup and execution configuration are control plane; secret-prone files are external credentials
memorial/	Retained evidence, curated knowledge, graph assertions, and provenance	Durable life; promotion into standing instructions affects the definition
dreams/	Dream inputs, candidates, evaluations, checkpoints, findings, and outcome records	Durable life
quarantine/	Protected findings, held content references, containment records, and disposition decisions	Durable life; access and activation restrictions enforced by the host
logs/	Operational diagnostics and execution records	Explicit retention; durable evidence is identified and preserved; carries the protection class of the producing session
daemons/	Service definitions and accepted configuration	Durable definition and control plane; application data has a declared durable destination

Path	Purpose	Retention and revision treatment
workspaces/repos/	Managed repositories and their long-lived checkouts or common stores	Durable life
workspaces/worktrees/	Task and dream worktrees connected to repositories	Durable until explicitly retired
revisions/	Definition snapshots, lineage, acceptance records, and publication receipts	Durable history
_CodeSignature/	Integrity metadata and separate signature envelopes	Durable verification data with defined self-exclusions
host/	Active PIDs, sockets, locks, and host execution leases	Host-specific and recreated; distinct from the durable run records under dreams/runs/
cache/	Rebuildable text, vector, and graph indexes and other caches	Reconstructible from retained inputs and declared recipes; under the protected profile stored broker-private
tmp/	Disposable intermediate work	Disposable; carries the protection class of the producing session

The term *raw* is used here for accepted source before harness rendering. It does not grant an agent unrestricted source access. Unreviewed imports and candidate skills remain with their proposal or producing workflow until accepted. The proposed meaning of *memorial/* is retained memory and experience. Both terms should be confirmed when the layout specification is adopted.

Classification and extension rules

The core classes are accepted definition, durable life, reconstructible content, disposable content, and external references. Human or machine authorship is provenance metadata; it does not determine whether a file can be discarded. A conversation record produced by a harness may be irreplaceable, while an index computed from retained sources can be regenerated.

The control plane comprises inputs that establish authority, standing instructions, executable behavior, or the rules for selecting and accepting those inputs. The inventory includes *SoulInfo.json*, *SoulLock.json*, *access/*, acceptance and quarantine policies, the definition exclusion list, *Resources/SOUL.md* and other standing instructions, *bin/*, *daemons/*, executable primitives, skill instructions and bundled scripts, accepted shell startup files, and configuration or renderer recipes that affect execution. Membership follows function across dependencies, not directory names alone. A host rejects execution of an unclassified behavior-bearing input. *SoulIdentity.json* is also a control record, but it is authenticated separately under Section 2 rather than included in the definition digest.

Control-plane changes require the owner or an explicitly delegated acceptance authority. The delegate's authority and credentials are established outside the proposing execution, and that execution cannot invoke the acceptance capability on its own behalf. A different session identifier alone does not establish independence. No automatic acceptance policy covers the control plane. Candidate classification and acceptance use the currently trusted policy and expected base; a proposal cannot authorize itself by changing its own policy, exclusion list, or classifications. The acceptance receipt binds the exact candidate, base, policy version, evidence, and authority.

Automatic content quarantine does not rewrite or disable control-plane records. Findings against those records are preserved and escalated. An independent host safety policy may still suspend execution

while keeping owner inspection and recovery available. The exclusion list is itself covered control-plane content. Every accepted behavior-bearing input is covered directly or resolved through an immutable digest in covered declarations. Mutable native state that affects behavior is recorded as an execution input or limitation rather than being silently attributed to the definition hash.

Readers preserve unknown fields and supported opaque entries when rewriting the same artifact. Export profiles separately determine which components travel; unknown content retained locally is not automatically included in every export. Unknown content receives conservative local retention and is never automatically executed. An unknown extension marked as required can prevent execution even when the package remains inspectable. Path safety is a requirement of the first package-format profile: readers and importers reject absolute paths, parent-directory components, symbolic and hard links that resolve outside the content root, case-fold collisions, and undeclared executable modes by default, and the conformance suite carries fixtures for each case. External references follow the same explicit rules. Existing Soul formats already define canonical path and mode behavior that migration must preserve for historical revisions. [R3]

4 Workspaces home and shared resources

Repository and worktree ownership

`workspaces/repos/` contains the repositories the Soul manages.
`workspaces/worktrees/<repository>/<task-or-run>/` contains working areas for separate branches, experiments, and dream proposals. The format does not require every repository store to be bare. It records the relationship between each worktree and its repository so a host can inspect, move, and restore them correctly.

Git worktrees share repository data while retaining worktree-specific state. Git documents repair operations for relationships affected by manual moves. A Soul transfer therefore needs to preserve or repair those relationships and verify the result after placement. [R12]

Both repositories and worktrees are durable. A short-lived task may leave uncommitted changes, untracked files, diagnostic evidence, or a candidate improvement. Cleanup does not infer that this material is disposable from the task's age or termination. The owner or an authorized retention policy records its disposition before removal.

An external linked workspace is declared as a reference. An export reports whether it captures the repository, selected patches and files, or only a pointer. The receiving host must show unresolved relationships. It must not silently substitute a different repository with the same name.

A native home

`home/` provides the Soul's user environment, including `.zprofile`, `.vimrc`, `.npmrc`, `.npm/`, and `.nvm/` where relevant. Files that commonly carry secrets, such as `.npmrc` authentication tokens or tool sign-in state, are classified as external credentials by default (Section 11) and are excluded from definition revisions and exports unless the owner explicitly reclassifies them. The proposed target is a logical Soul home with explicit routing by the launcher. The actual mechanism depends on the host and the application: a process home variable, a supported harness-specific variable, or a compatible projection can supply the required behavior.

Current documentation routes supported harness stores individually and reports that HOME remains the host's home. A complete Soul home is therefore a proposed extension, and adapters must report the routing they actually achieve. An unsupported tool cannot be described as contained merely because a directory was created for it. [R4] [R6]

The proposed managed-runtime profile gives each runtime one authoritative installation location under `opt/`. A manager such as `nvm` can integrate through configuration or an internal reference. A different profile could allow `nvm` to own installations under `home/.nvm/`, provided the descriptor declares that ownership and selection unambiguously. Two managers must not independently control the same logical installation. The same principle applies to package caches and language environments: the descriptor identifies ownership and the resolved executable selected for a run.

Accepted startup and execution-affecting tool configuration participate in definition revisioning and the control-plane acceptance rules. Native sessions and working state remain durable and mutable. The launcher supplies a consistent configuration view for the selected revision while routing writable state separately. Where an application cannot separate those concerns, the adapter must use a declared restart or activation boundary and report the limitation.

Sharing within the package

The proposed default for `shared/` is reuse among one Soul's harnesses, tools, and workflows. Common schemas, reference collections, templates, and assets can live there with clear ownership. Resources used as fixed execution inputs are included in the definition inventory or bound by a covered content digest.

Resources shared across Souls require an external dependency contract. It records identity, location or retrieval mechanism, expected version or digest, access mode, and export treatment. A portable export may materialize selected resources when permitted or retain an unresolved reference. Missing resources affect readiness without invalidating the retained life.

5 Primitives and native execution

Source and harness representation

The Soul owns reusable capabilities independently of the harness that consumes them. Skills use `primitives/skills/raw/<skill>/` for accepted logical source and `primitives/skills/harness/<harness>/` for representations. A portable skill can retain a `SKILL.md` file and supporting scripts, references, and assets, following the Agent Skills directory convention where applicable. Under the protected profile, those canonical bytes are encrypted and an adapter exposes only the authorized instructions or invocation interface. [R13]

A generated representation records its source revision or digest, renderer identity and version, target harness compatibility, and relevant configuration. The materialization key must distinguish incompatible outputs. Two sessions that use the same harness release but different Soul revisions cannot overwrite each other's configuration. This depends on per-session routing of the harness's configuration locations; where an adapter cannot route them (Section 4), it declares a single active revision for that harness and reports the limitation rather than claiming isolation.

Authoritative harness-native sources and overrides have explicitly declared locations and merge rules. One proposed subdivision is `harness/<name>/native/` for accepted native sources and `harness/<name>/rendered/<materialization-id>/` for generated output. These child names are a draft choice. Regeneration preserves accepted sources and reports a conflict when it encounters an unrecognized edit to managed output.

Other primitive types can use the same source-to-representation model where translation has a defined meaning. The format does not require fabricated equivalents for unsupported features. Required capabilities must be satisfied before launch; optional omissions remain visible. Existing builder documentation already reports differences in what adapters can render. [R8]

Optional packages and runtimes

`opt/harnesses/`, `opt/runtimes/`, and `opt/packages/` hold installations. Accepted declarations and locks identify exact resolutions, supported platforms, artifact digests, dependency sets, and executable entry points. Native helper binaries in `bin/` also need platform declarations or portable source and build instructions when they are expected to move between platforms.

An installation is reusable only when it matches the required artifact identity and validation policy. Exclusion from the definition signature does not authorize execution of arbitrary existing bytes. A lock stored solely with an unsigned installation cannot establish which dependencies the accepted definition intended.

Installation directories are prepared and verified before activation. A failed installation leaves the previous usable installation available. Rebuilding on another platform preserves the logical declaration while selecting its explicitly supported platform artifact. A portable life may remain readable when no compatible executable exists.

Execution profiles

An execution profile selects the native harness, adapter, runtime set, provider and model configuration, primitive materialization, state namespace, and required capabilities. The format can support one or several profiles per harness. Each launched session resolves one concrete profile and records the result.

Version identifiers name their component. An adapter version is not assumed to be a native harness version; current npm ACP installation documentation makes this distinction concrete. State compatibility also needs a rule: a newer executable may change a native database that an older executable cannot use. Rollback therefore evaluates state compatibility as well as executable availability. [R5]

The execution record includes the Soul ID, definition revision, resolved artifact identities, selected provider and model, effective supported configuration, relevant input references, host capabilities, and session identity. It distinguishes a requested model from an identifier actually reported by the provider. Unknown values remain unknown. This improves attribution without claiming deterministic output from a remote or changing service.

6 Protected content and curated access

The proposed default protects canonical primitives and non-curated text sources with encryption at rest and tool-mediated access during execution. Ordinary agents receive curated instructions, selected

evidence, and scoped workspaces. Dream workflows can receive additional grants to examine source material and develop proposals. The filesystem layout remains a contract for the trusted host and broker; agents use authorized logical operations rather than browsing the whole package.

The objective is to make broad searches of private or unprocessed material unavailable to an ordinary agent. A request to avoid `grep`, or a wrapper around that command, cannot enforce this while the same process can read the underlying files. Other search commands, scripts, and file APIs have the same access. The effective control is which plaintext the execution environment can reach.

Enforcement and custody

A trusted content broker authenticates the caller and authorizes each operation against its effective grant. It retains decryption capability outside the agent's environment and mediates protected storage, retrieval, graph access, and writes. Harness tools are clients of this broker; the calling agent cannot enlarge its grant by supplying a different role or path. A valid content signature does not grant permission to decrypt, inspect, invoke, or modify the signed material.

The harness process and the agent's shell are one trust domain: the harness executes the agent's commands, so any broker credential it holds is reachable by agent code. The broker therefore issues per-session capability tokens bound to that session's grant and expiry, enforces every budget, output limit, and scope on its own side rather than relying on harness-side filtering, excludes tokens from checkpoints, and revokes by invalidating the token rather than by waiting for a process to exit. It authenticates callers by the token and the operating-system peer identity of the connection, never by a role or path the caller supplies.

Encryption protects stored objects. Operating-system identity, sandboxing, filesystem exposure, and process isolation protect the active broker and its plaintext from agent code. A mount that unlocks the entire Soul into the agent's shell defeats the intended boundary. The host must demonstrate the required separation for each supported execution profile. A host that cannot enforce it reports that the protected profile is unavailable and does not silently expose the sources.

The encryption profile specifies authenticated encryption, key wrapping, rotation, recovery, and the relationship between encrypted storage and logical content identity. Logical names, sizes, counts, timestamps, and the path-to-object mapping are encrypted when their disclosure would reveal protected structure. This does not by itself hide physical ciphertext lengths, the number of stored objects, filesystem timestamps, or access patterns. Each profile declares the information visible to a copied-device observer and, separately, to an observer of active storage. Stronger claims require specified padding, object grouping, or other measures and corresponding tests. Keys or decryption authority remain in an external backend. A package can carry recipient-wrapped key material when permitted, but no agent-readable plaintext unlocking key. Ciphertext, indexes, local databases, and backups retain explicit protection classifications.

Capabilities for work and dreams

The following operation names illustrate a proposed interface; their spelling is not yet a standard. Each grant binds the Soul, caller, operation, resource scope, purpose, expiry, and output limits. Resource scope can include project, source class, accepted revision, and individual source versions. The host's authorization bounds what a package can request.

Operation	Ordinary task access	Dream access under an explicit grant
skills.list	Reviewed names and permitted capability metadata	Same discovery within its assigned scope
skills.read or skills.invoke	Selected accepted instructions or a controlled invocation	Source inspection where required to assess an assigned primitive
knowledge.search	Curated evidence and graph results within task scope	The same service plus explicitly eligible retained sources
sources.read	Selected records only when granted	Bounded source versions needed for the dream's recorded plan
proposals.write	Authorized task outputs	Durable candidate knowledge, patches, and evidence
changes.accept	Only a separately authorized authority or policy	Not implied by the ability to inspect or propose

A dream is not an unrestricted administrator. Its source grant records the maintenance scope and delivery limits. Broader evidence requires the normal authorization mechanism. The broker maintains a durable disclosure ledger for a specified Soul, recipient principal or authorization group, corpus scope, and budget period. The ledger is shared across concurrent sessions and survives session renewal, workflow restarts, and process failure. A new token or session cannot reset it; budget renewal requires the configured authority. In the first protected execution profile, the ledger remains host-local and grants do not carry reusable balances to another host. A recipient host establishes a fresh owner-authorized budget; no global cross-host limit is claimed.

The grant defines chargeable units, including transmitted output bytes, unique disclosed source spans, and derived outputs. The broker reserves budget atomically before disclosure and records source versions and receipts; an interrupted response is conservatively charged when delivery cannot be ruled out. Overlapping requests do not erase earlier charges. Summaries carry contributing-source references and are charged under a stated rule. If required accounting or lineage is unavailable, the operation is denied or routed to a separately authorized export. These limits constrain recorded disclosure through the broker; they do not prove that a recipient cannot infer information or retain plaintext already received. A resumed dream revalidates its grants, and a checkpoint never preserves an expired decryption capability.

The broker also supports an owner or authorized maintainer inspecting and editing protected contents. Editing uses a versioned mutation operation or controlled workspace, followed by validation and publication. The package remains inspectable by authorized independent implementations; encrypting its storage does not make a particular vendor's service the only possible reader.

Curated views and native compatibility

A curated view has a recorded source set, transformation, version, and acceptance policy. Automatic extraction alone does not make text curated. Tool responses retain evidence references and relevant limitations. A harness that needs native skill files can receive a minimal read-only projection of the permitted accepted instructions for one run. Protected source libraries, unrelated skills, and private histories remain outside that projection.

Invocation and source disclosure are separate capabilities. A skill may run through a broker-managed executor without exposing its implementation to the model. That executor is a separate sandbox with its own identity and no access to broker keys or private plaintext beyond the invocation's declared inputs;

Soul-authored code never runs inside the broker's process or operating-system identity. If the selected native interface requires source exposure, that requirement is reported and permitted explicitly. A plaintext script placed in a shell-readable directory cannot be described as hidden from that shell.

Task workspaces can deliberately expose project files for normal development and scoped search without exposing the protected Soul corpus. Dream workspaces and retained outputs inherit the classifications of disclosed sources. A source-granted dream normally runs under a distinct operating-system identity or sandbox from ordinary sessions. A sequential fallback is permitted only when a trusted supervisor enforces a clean transition: stop all descendant work, revoke tokens, terminate broker connections, close inherited capabilities, remove exposed projections, and move retained outputs to stores inaccessible to the next session without a fresh grant. Writable state capable of persisting executable behavior is reset or separately reviewed before reuse. Failure to complete any required transition leaves protected execution unavailable. Running one process at a time alone is insufficient.

Derived material needs the same discipline. Embeddings, graph databases, summaries, conversation records, logs, and temporary files can contain or reveal protected information. The profile prevents them from becoming alternate unmediated read paths. Under the protected profile, embeddings and indexes built from protected sources are themselves protected: they are stored broker-private and never written to agent-readable paths, because demonstrated embedding-inversion attacks recover source text under tested model and input conditions. This is evidence of a confidentiality risk, not a claim that every embedding can be inverted. [R17] logs/ and tmp/ carry the protection class of the session that produced them. Acceptance and curation do not remove confidentiality restrictions; changing who may read or export material is a separate decision. Once plaintext is legitimately delivered to an execution, encryption cannot make the recipient forget it or prevent every later copy; the promise is controlled disclosure, with recorded retention and output rules.

Protection through sharing

Sharing grants a recipient access to an explicitly selected corpus and capability set. The exporter can package that selection in an encrypted envelope and wrap keys for the intended recipient or recovery mechanism. The recipient establishes local custody and grants without changing the Soul's content identity merely because encryption keys changed. Losing all authorized recovery paths can make encrypted life inaccessible, so a recoverable backup includes a separately managed key-recovery arrangement.

Logical definition identity and encrypted transport identity are distinct. Re-encrypting identical logical content changes its ciphertext and transport digest. Any signature over the old ciphertext continues to cover only those old bytes. Protected manifests retain the logical digests and provenance needed for authorized verification; the public envelope exposes only the declared routing and integrity metadata. Exact envelope layout and cryptographic algorithms require specification and review.

This profile makes local RAG and dreams the intended access routes into the Soul's knowledge. It preserves a shareable native package while requiring explicit authority for source disclosure, execution, acceptance, and export.

7 Revisions integrity and activation

Definition revisions and life checkpoints

A definition revision covers accepted instructions, primitive sources, helpers, operating configuration, and declarations. Ordinary writes to projects, sessions, memories, dream records, and diagnostic logs do not automatically create a definition revision. Promotion into authoritative instruction or configuration content does.

A life checkpoint captures the selected mutable state needed for a consistent backup or transfer. It names the definition revision and the components included, with hashes or component-specific consistency evidence. It is distinct from the definition revision and does not imply that every historical state can be replayed. Its authenticated inventory also identifies the included identity binding, provenance records, and restriction-state checkpoint. Integrity of an older checkpoint does not establish that it reflects the latest authorization or quarantine decisions.

Identifier	Meaning	Changes when
Soul ID	Persistent agent identity	A new Soul or explicit fork is created
Definition revision	Accepted reusable definition content	Covered content changes; acceptance of identical bytes can have a new receipt without a new digest
Life checkpoint	Selected durable state at a consistency boundary	A checkpoint or transfer snapshot is created
Format and layout versions	Interpretation of package bytes and path roles	An explicit compatible update or migration occurs
Installation and run IDs	Placement and execution identities	A host registers an installation or begins an execution

Digest signature and trust

A deterministic digest identifies reusable definition contents under a specified canonical encoding. Acceptance, instance identity, and lineage are separate authenticated records that reference that digest. Identical covered bytes can retain the same definition revision across instances or forks. A signature envelope binds its declared statement to a signer; trust policy determines whether that signer and action are accepted. These are separate results. The `_CodeSignature/` name is an organizational convention and does not imply Apple's application code-signing semantics.

The canonical algorithm defines covered paths, byte ordering, manifest normalization, executable modes, and explicit exclusions. `SoulIdentity.json`, identity-event records, acceptance receipts, and signature envelopes are not part of the reusable definition digest; each receives its own authenticated coverage and validation rules. Their exclusion does not authorize unauthenticated mutation or execution. A content inventory cannot modify the host's rules for these exclusions. JSON canonicalization can use an explicitly selected encoding such as RFC 8785, while paths, modes, and inventory construction still require package-specific rules. Appendix C provides a bounded worked example rather than adopting a final format. Signature encoding, key rotation, and recovery remain specification decisions. [R1] [R18]

The signing or acceptance authority acts under currently trusted policy. Automatic acceptance can cover explicitly permitted retained knowledge or other changes outside the control plane, subject to the source and evidence rules in Section 8. It cannot cover standing instructions, executable helpers, or other control-plane content. Those changes require the owner or a separately authorized delegate whose credentials and decision cannot be exercised by the proposing execution. Verification and acceptance receipts bind exact content and the policy used; a signature alone does not create new runtime authority.

Consistent publication

A publisher stages a candidate against an expected base revision, validates the complete inventory, and commits it through a defined activation operation. Readers obtain one complete accepted definition. An interrupted commit yields the previous revision, the complete new revision, or an explicit unavailable state requiring recovery. It must never attribute mixed definition inputs to a single revision.

This is stronger than the current documented publication behavior, which renames individual files and can leave a partially applied working folder after failure. The future format needs a tested publication and recovery mechanism for each supported storage model. [R9]

The proposed default pins a session to its resolved definition and configuration. New sessions adopt the current accepted revision. An adapter may support an explicit transition at a turn boundary when it can provide equivalent consistency and record the change. The final activation boundary remains an open decision. Mutable state continues to evolve, and relevant input evidence is recorded separately. Definition pinning does not pin authorization. Every broker operation and activation remains subject to current grants and restriction state. A newly effective restriction can prevent further disclosure or execution under an otherwise valid older definition.

8 Local retrieval knowledge graphs and learning

A continuing Soul needs a way to use what it has experienced. Local retrieval-augmented generation, or RAG, supplies selected evidence from its retained knowledge to the current reasoning task. The original RAG research combines a generative model with an external retrievable memory. This proposal applies that separation to a Soul-owned corpus that can evolve independently of the selected inference model. [R14]

The knowledge graph adds explicit relationships among projects, people, decisions, constraints, tools, observations, and outcomes. GraphRAG research illustrates the use of entity graphs and community summaries for questions that span a document collection. The Soul format should support such techniques without requiring one graph database or retrieval algorithm. [R15]

Learning here means accumulating evidence, correcting knowledge, recognizing useful relationships, and improving reusable behavior. It does not require changing model weights. A Soul should benefit from prior work when it starts a new session, changes harness, or moves to a new machine.

Benefits across a continuing life

Capability	Intended benefit
Retrieve relevant experience	Bring earlier decisions, failures, preferences, and working examples into the current task without placing the full history in every prompt

Capability	Intended benefit
Connect related evidence	Follow relationships across projects and sessions, such as which constraint motivated a decision and which result tested it
Maintain evolving knowledge	Record corrections, exceptions, and superseded conclusions so later work can use the current understanding with its history
Share knowledge across harnesses	Let supported harnesses query one Soul-owned memory through a declared interface instead of maintaining disconnected copies
Keep knowledge locally usable	Inspect and retrieve retained material without requiring a centralized knowledge service when the selected retrieval components run locally
Carry learning with the package	Transfer selected evidence, accepted knowledge, and lineage while rebuilding host-specific indexes as needed

These are design goals, not performance claims. Retrieval quality depends on the retained corpus, source quality, extraction, ranking, and the task. The implementation should measure whether useful experience is found and applied correctly, rather than equating a larger graph or index with better learning.

Durable knowledge and derived indexes

The proposed layout gives retained evidence and learned knowledge durable homes under `memorial/`. Search machinery is a separate representation. These child paths extend the directory contract and remain draft choices.

Path	Proposed contents	Treatment
<code>memorial/sources/</code>	Versioned retained observations, selected source snapshots, and source references	Durable evidence with explicit completeness and retention
<code>memorial/knowledge/</code>	Curated notes, entities, assertions, corrected relationships, and accepted summaries	Durable learned knowledge
<code>memorial/provenance/</code>	Ingestion, derivation, acceptance, correction, and import records	Durable lineage linking knowledge to exact source versions
<code>memorial/retrieval/</code>	Corpus manifests, indexing recipes, coverage checkpoints, and compatibility metadata	Durable metadata; accepted behavior settings participate in the definition inventory
<code>cache/retrieval/</code>	Lexical and vector indexes, graph query materializations, and disposable derived summaries	Rebuildable only when their required inputs remain available
<code>opt/models/</code>	Optional local embedding or reranking artifacts	Verified installations or explicitly declared external dependencies

An accepted assertion, a corrected entity match, or a reviewed relationship must not exist only inside an opaque index. The package retains a versioned logical representation that another implementation can read or import. A database may store the working graph, but the portable representation must preserve identifiers, meaning, provenance, and change history. The initial schema and serialization remain specification decisions.

Automatically extracted edges and summaries can be regenerated when they are explicitly classified as derived material. Once an output becomes retained learned knowledge, its accepted bytes and provenance are durable. Repeating a model-based extraction may produce different results even with a recorded recipe; rebuilding an index does not authorize replacing accepted knowledge with a new inference.

An index manifest identifies its source versions, the quarantine state in effect when it was built, chunking rules, embedding model and artifact identity, vector dimensions, distance metric, schema, and builder version where applicable. Incompatible indexes are rebuilt or left unavailable. Rebuilding cannot erase knowledge, and omitting an embedding cache cannot make the original evidence disappear. An offline-ready export must also include permitted dependencies needed for its declared offline capabilities. The build-time quarantine checkpoint is not sufficient authorization for a query: the broker checks current restrictions again before returning results, including cached results and their contributing sources.

Assertions carry evidence and time

Every retained source records its provenance at ingestion, including owner-authored, agent-authored, third-party, or store-seed origin. Authorship, import origin, confidentiality, evidential status, and approval authority are separate attributes. A derived assertion or summary preserves references to all contributing source versions and their attributes; a single creator label cannot replace that lineage. Mixed-source results retain the contributing classifications, and an agent-authored summary does not erase its third-party dependencies. Store-seed is an import designation added alongside preserved historical authorship.

For automatic acceptance outside the control plane, the current policy specifies which consequential claims or changes require authorized evidence or independent validation. Merely adding owner-authored text does not make an externally grounded candidate eligible. A candidate supported only by third-party or store-seed assertions requires separate acceptance or policy-authorized validation of the relevant claims. Missing lineage or unresolved material conflicts prevent automatic acceptance. Validation can establish a narrowly tested claim; it does not relabel its original sources or remove confidentiality restrictions.

Graph entities use stable identifiers that survive a filesystem move. Assertions record the relationship, supporting source version and span, producing activity, observation time, applicable time or scope, and status. Status distinguishes an observation, a hypothesis, an accepted interpretation, a disputed assertion, and a superseded conclusion. A model's extraction score is not presented as the probability that a claim is true.

Corrections retain the relationship to the earlier claim while changing which interpretation is current. Contradictory evidence remains visible until a defined resolution is recorded. Entity merges and splits preserve mappings so a shared package does not confuse two people or projects merely because their labels match. W3C PROV supplies a useful vocabulary for entities, activities, derivation, and responsibility; adopting a specific encoding requires a separate schema decision. [R16]

Knowledge records normally evolve as durable life. Promoting a lesson into a standing instruction, skill, or operating policy creates a definition revision. A life checkpoint identifies the selected knowledge state. A run records the knowledge snapshot or exact retrieved evidence it actually used, because the definition revision alone cannot identify changing memory.

A shared retrieval interface

Supported harnesses and dreams query the same logical knowledge service. A request declares the task, allowed source scope, relevant time or project filters, and a result budget. The implementation can combine keyword search, semantic similarity, graph traversal, and summaries. No single ranking method is required for package conformance.

A result contains evidence references, relevant assertion status, source versions, and coverage information. A partial or stale index reports its limits; an empty result cannot establish that no relevant memory exists. Missing dependencies can leave broker-mediated source inspection and supported lexical search available within the caller's grant while semantic retrieval is rebuilt. The host reports which operations actually work.

Source access applies to retrieved passages and graph-derived results, including summaries that combine several sources. Retrieved content remains evidence rather than authority to change instructions or execute commands. When selected excerpts are sent to a remote inference provider, local storage alone does not make that execution private or offline. The execution profile records the permitted destination and actual capability.

Learning that remains shareable

A life transfer carries the selected source corpus, durable graph assertions, curated knowledge, provenance, and correction history. It may include compatible indexes for faster startup, but preserving the knowledge must not depend on a particular vector database or a running external service. The receiving host validates the logical data, restores source mappings, and reports which retrieval components are ready or require rebuilding.

A reusable template can contain deliberately selected seed knowledge and its provenance without including the original Soul's private life. Seed knowledge receives a store-seed import designation while retaining its original authorship and contributing-source attributes. An explicit fork preserves historical authorship and source lineage while creating a new Soul identity. Missing or intentionally withheld evidence is recorded as unavailable; the recipient is not told that every inherited claim can be independently verified.

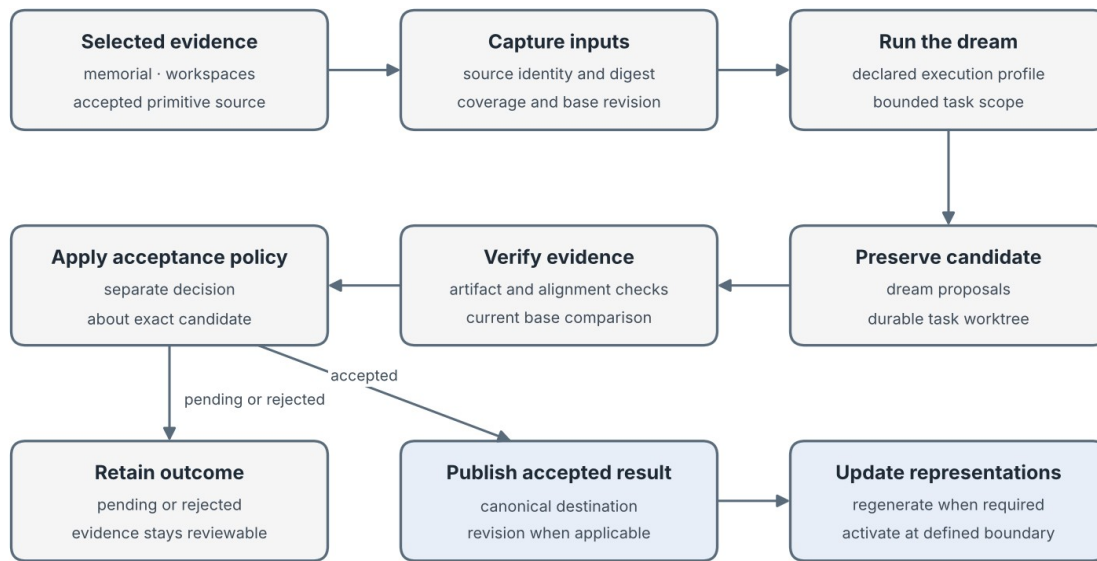
Export selection includes derived content. Removing a private transcript while retaining its revealing summary, graph relationship, or embedding does not establish that the information was excluded. The exporter either selects and reviews those derivatives explicitly or rebuilds them from the selected export corpus. References to omitted evidence are removed or represented according to the export policy without disclosing protected metadata. Retraction and deletion likewise propagate to active indexes and derived views; historical snapshots follow their declared retention policy.

This separation permits a Soul to share useful accumulated knowledge, carry its continuing life, or seed a new Soul with selected expertise. The export receipt makes the selected knowledge and remaining dependencies inspectable.

9 Dream workflows and accumulated learning

Dreaming is a bounded maintenance activity through which a Soul examines selected experience, evaluates its capabilities, and develops candidate improvements. It uses the existing execution and authorization mechanisms. The directory contract gives each stage a predictable source and destination.

The goal is recoverable, attributable improvement. A completed run can produce a useful candidate, no change, a blocked finding, or a failure. None of those results alone proves that learning was accepted. Current dream documentation already distinguishes selection, verified change, and acceptance-related evidence; the new layout must preserve that precision. [R10]



Durable run records preserve inputs, candidates, checkpoints, and outcomes throughout the workflow.

Figure 1 Proposed dream lifecycle. Verification and acceptance are separate states. Durable records preserve evidence and outcomes. Authorized containment may suspend any stage; the quarantine contract is described below under alignment and containment. Both accepted and non-accepted branches retain outcomes; publication failure or interruption produces a recovery record, and successful publication produces an activation receipt before representation updates are reported complete.

Capture the starting point

Each run receives a stable run ID and a durable record under `dreams/runs/<run-id>/`. The proposed `inputs.json` identifies the Soul and definition revision, workflow version, resolved execution profile, source identities and digests, selected ranges or excerpts, workspace scope, and coverage limitations. Sensitive content need not be duplicated into a metadata receipt when an immutable retained source version or appropriately protected snapshot can supply the exact selected bytes. When only identity and digest metadata are retained, the run reports that limitation and cannot claim that its original inputs remain reconstructible.

Eligible inputs can include retained experiences in `memorial/`, accepted primitives, selected operational evidence, supported native conversation records, and explicitly scoped workspace changes. The broker evaluates availability and authorization per source. The run records its effective source grant and the

exact disclosures it receives. The presence of a directory does not authorize a dream to read every project or mutate every repository.

Local retrieval helps the dream find related experience; the knowledge graph can expose recurring failures, dependencies, conflicting conclusions, and opportunities to generalize a successful approach. The run records its query, retrieved evidence, knowledge state, and coverage. A graph-generated summary points back to its contributing sources and does not count as independent confirmation of them. Candidates can update knowledge, propose a relationship correction, or improve a primitive; a dream need not modify code to produce useful learning.

Sources and derived copies retain a shared provenance relationship. Four harness renderings of one skill represent one underlying skill with four projections. Input selection avoids treating each rendering as independent learned experience. Generated outputs can still be validation targets when a candidate changes their source.

The current reader recognizes selected root instruction and skill paths, with separate supported readers for conversations and native harness memory. A layout migration must update those readers, their source identifiers, and their checkpoints rather than simply moving directories. [R10]

Preserve working evidence

dreams/runs/<run-id>/plan.md describes the proposed maintenance. The run's proposals/ area retains exact candidate artifacts or immutable references to them. workspaces/worktrees/<repository>/<run-or-task>/ holds repository-backed development and experiments. tmp/dreams/<run-id>/ holds intermediates that can be discarded. These records and workspaces retain the protection classification of any disclosed sources; temporary status does not make private plaintext available to other agents.

Every candidate links to the source revision or content digest on which it was based and to the evidence used to assess it. A process crash must not erase the ability to inspect a proposed patch or determine what remains unfinished. Inputs, candidates, checkpoints, and worktree references are durable even when the executor is no longer running.

The package supports reconstruction of work, not a promise to resume every interrupted process instruction. A host can restart the relevant workflow from a recorded boundary after verifying that dependencies and source versions still match.

Verify and promote

Verification checks that the candidate is the artifact the run actually produced, that its evidence refers to delivered inputs, and that the applicable content and execution checks passed. It also checks the candidate's base against the current target. A stale proposal is reconciled and revalidated before it can replace a newer accepted skill, memory, or configuration. The acceptance authority then evaluates the candidate under the currently trusted policy, rechecks current restrictions, and binds its decision to the exact artifact. Candidate-provided policies cannot authorize their own installation.

Promotion uses the destination's normal mutation contract. An observation can be retained in memorial/ as durable life with provenance. Accepting a standing instruction, skill, executable helper, or operating configuration creates a definition revision. Accepted primitive changes then generate new harness representations without overwriting active sessions' materializations.

The original author's identity, the accepting authority or policy, and the published result remain separate records. A proposed change can be verified without being accepted. The existing dream contract explicitly permits verified-change evidence for a proposal without treating that evidence as owner approval. Verification, acceptance, and publication therefore remain distinct recorded states. [R10]

Examine alignment and contain harmful drift

Dreaming also examines whether the Soul's developing knowledge, plans, capabilities, and session behavior remain consistent with authorized intent. The reference is the owner's applicable goals, current authorized task, accepted constraints, and access policy. The run records which versions it used. A legitimate change of task or an owner's approved exception must not be misclassified merely because it differs from an older preference.

The review can look for attempts to expand authority, persistent pursuit of an abandoned objective, unsafe tool behavior, hidden instructions in retrieved material, contradictory policy changes, or a pattern of actions that defeats an explicit constraint. These are subjects for evidence-based assessment. Quoting a dangerous instruction for analysis is distinct from adopting it. A keyword match, surprising answer, disagreement, or classifier score alone does not establish dangerous behavior. Findings distinguish suspected problems, confirmed violations under a stated rule, unresolved ambiguity, and false positives.

The proposed quarantine/ area records the finding, exact affected content or immutable references, relevant session and descendant-run identifiers, source lineage, policy version, rationale, evidence, containment action, and review status. Records remain protected and available to authorized review. Quarantine preserves useful evidence; it does not silently erase a session or rewrite the accepted definition's history.

State or action	Required meaning
Flag for review	Record a concern and supporting evidence without asserting that the concern is proven
Contain under policy	Apply the specific restrictions already authorized for the assessed condition
Quarantine content	Exclude affected versions from ordinary retrieval, skill activation, and promotion while retaining them for authorized review
Pause a session	Ask the host supervisor to suspend or stop relevant work and revoke applicable future tool grants
Resolve the finding	Record release, correction, retirement, or an authorized exception with its evidence and authority

A dream may propose quarantine, and an explicit owner policy may authorize automatic containment for defined conditions. Automatic content quarantine does not rewrite or disable control-plane records: a finding against SOUL.md, access/, or acceptance policy is recorded and escalated. A separately authorized host safety policy can suspend execution while preserving owner inspection and recovery. Containment actions are rate-limited by policy so a single run cannot pause every session. An uncertain but potentially serious finding may justify a bounded precautionary pause without being recorded as a confirmed violation. The trusted broker and host supervisor enforce the decision at the narrowest effective scope. They can block a source version or primitive, invalidate affected materializations, restrict related retrieval results, and stop a session from receiving further capabilities. Relevant child runs and

background work are included according to the containment scope. Periodic dream review complements execution-time controls; it cannot prevent an action that already completed before the review.

Quarantine is an enforced status associated with stable source and run identities, rather than a directory move. Source provenance connects containment to downstream knowledge. A suspicious instruction repeated in a summary or graph edge does not become acceptable through repetition. Affected derivatives are excluded or marked pending review according to policy. Independent evidence and unaffected work can remain available. Ordinary retrieval does not return quarantined material merely because it is semantically relevant, while an authorized forensic dream can inspect it through a separate grant.

The trusted host maintains an authenticated restriction journal with a monotonically increasing local policy epoch. A quarantine or release decision records its authority, affected immutable versions, dependencies, and epoch. Broker responses and activation decisions are checked against the current effective epoch, including decisions using cached results. A restriction committed before a disclosure is authorized must be applied; already authorized or in-flight disclosures are recorded and handled under the declared revocation boundary. Candidate-controlled files, stale indexes, and replayed checkpoints cannot lower the host's known restriction state.

On restore or import, the host checks checkpoint authenticity and compares restriction state with its trusted local record and any required freshness authority. An older signed backup remains valid evidence of its earlier contents, not proof of current permission. A new or offline host that cannot establish required freshness reports it as unknown and withholds protected activation or disclosure until the configured authority authorizes a bounded recovery decision. That decision records the uncertainty and does not silently release known restrictions. This profile makes no claim to discover every later remote decision while disconnected.

Revoking a grant cannot remove text already present in a model's context or undo an external action. A resumed session may therefore require a fresh context with approved inputs, renewed grants, and revalidated work. The host distinguishes a requested pause from a confirmed stop and records which jobs or credential revocations remain unresolved. A process exit alone does not revoke a token it previously received. Separate authorized remediation addresses any already completed action.

Release is a separate decision with a recorded basis, such as a false-positive finding, a verified correction, or an owner-authorized change. Restoring access rechecks the current definition, policy, and dependent materializations. The dream's ability to identify a problem does not authorize it to weaken its own review policy, delete audit evidence, or silently approve its own release. Applicable review authority can be an owner or a separately delegated policy; repeated human confirmation is unnecessary where that authority was already granted.

Transfer preserves quarantine restrictions together with the content they apply to: a release profile refuses to export a quarantined version or its derivatives, a life-transfer profile carries the restriction alongside the content, and the export validator enforces both. Import cannot turn held knowledge into ordinary seed knowledge merely by copying the package, and a newly registered session does not automatically clear a prior containment decision. Export scope and evidence restrictions remain explicit so quarantine itself does not disclose protected source material. The transfer receipt includes the applicable restriction checkpoint and its freshness basis. Import applies the rollback and unknown-freshness rules above; copying an old signed checkpoint cannot override newer restrictions already known to the destination.

Record progress precisely

The proposed outcome.json records execution status, candidate verification, acceptance state, publication state, unresolved work, and source coverage. Run-specific recovery checkpoints live under dreams/runs/<run-id>/checkpoints/. Cross-run source selection and progress records live under dreams/checkpoints/ and reference the contributing run records. Both use stable source identity plus revision or digest. New bytes at the same path become a new input state. Every attempted publication records an outcome, including success, failure, or interruption. A successful receipt identifies the accepted content and activated revision; an interrupted operation records the last verified boundary and required recovery. Regeneration and activation outcomes remain linked to the same candidate.

Recorded fact	What it establishes
Input selected	A particular source or range was chosen
Execution completed	The executor finished under its recorded conditions
Candidate verified	Evidence establishes the candidate and relevant checks
Change accepted	A named authority or configured policy accepted exact content
Result published	The accepted result reached its designated durable destination
Coverage recorded	The workflow reports precisely which inputs and portions its evidence covers

A selection cursor answers which page of inputs comes next. It does not establish that all selected material was understood or changed. Retained memories are not deleted merely because a dream visited them, and source age alone does not establish that they are obsolete. Diagnostic retention and knowledge retirement require their own rules.

Routine evaluations are also recorded dream outcomes. A run can refresh evidence about an unchanged capability, compare a proposed improvement with a baseline, or detect a regression that requires review. The proposed dreams/runs/<run-id>/evals/ area retains manifests, result records, and protected supporting artifacts. Section 13 defines how selected evaluation evidence can accompany a release and appear in an Agent Store listing.

Example of a recoverable improvement

A Soul repeatedly encounters the same project-inspection problem. Local retrieval finds earlier observations, and the graph links failures to a repository type and the inspection skill used. A dream selects supporting evidence and captures its versions. An explicit source grant lets it read the relevant accepted skill through the broker at the logical primitives/skills/raw/ location. It creates an isolated task worktree and develops a helper with corresponding skill changes. Tests or other appropriate checks produce retained evidence linked to the candidate.

If the host stops, the worktree and proposal remain. A later run reads the durable record, detects whether the accepted skill changed in the meantime, and continues or reconciles the work. Acceptance promotes the helper into bin/ and the skill into canonical source through a new definition revision. A retained knowledge assertion records the applicable repository type, verified result, and supporting evidence. Supported harness renderings are regenerated for that revision. Future sessions can retrieve both the improved capability and the reason it exists. A life transfer carries that learning; the destination can rebuild its indexes without repeating or inventing the original experience.

Deleting tmp/ at any point must preserve the evidence and useful work needed for that continuation. This is a concrete conformance property of the layout.

10 Background services and operational state

daemons/ contains accepted service definitions and configuration. A definition identifies the command or execution profile, arguments, working directory, configuration and credential references, dependencies, restart policy, and relationship to the Soul's lifecycle. Durable service data has an explicit destination separate from temporary process state.

The host supervisor owns the active process, registration, socket, and lease. Local state under host/ is recreated on the destination host. A transferred service definition does not claim that a process from the previous machine still exists or that its schedule is automatically authorized to run elsewhere.

The lifecycle contract distinguishes a foreground harness exit, a paused Soul, a stopped Soul, and host shutdown. Each service declares whether it continues across foreground sessions and how pause or stop affects it. Activation after import follows host policy and reports unmet dependencies.

logs/ can hold diagnostic output with bounded retention. Records required to explain accepted changes, dream outcomes, or transfer consistency belong to explicitly durable journals or are classified as durable evidence. Log rotation cannot erase the only record of an accepted definition change.

11 Credentials and execution boundaries

The Soul owns credential bindings and intended account relationships. External stores hold managed credential material. A reference identifies a logical credential need and the capability required to satisfy it. Host binding resolves that need to a verified account and backend without changing the Soul's identity.

Backends register schemes such as keyd:, keychain:, pass:, op:, or an encrypted file backend. The scheme grammar, backend identity, and capability discovery require precise definitions. A method named resolve can return a typed binding; it must not imply that every backend returns the underlying secret.

Capability	Contract
Secret retrieval	Supplies material to an authorized consumer when supported
Scoped token issuance	Produces a credential for a declared purpose and scope
Brokered operation	Performs an authorized action while retaining the root secret
Content decryption or key unwrapping	Enables the trusted content broker to open an authorized protected selection without giving the agent the root key
Status inspection	Reports present, missing, locked, unavailable, or unsupported
Write support	Stores or updates material when the backend and caller permit it

Existing keyd documentation describes retained key custody, while provider-secret injection needs a readable value. The new interface must represent both. Under the protected profile, inference is a brokered operation: the broker or credential backend performs the provider call and the provider key never enters the agent's environment, because a key readable by agent code is exfiltrable through any

tool call. A profile that injects a readable provider key declares that exposure. Enumeration is capability-dependent and limited to the Soul's authorized bindings; a portable backend need not support unrestricted vault listing. [R7]

Native authentication is a compatibility requirement. Current file-backed stores and some adopted harness sign-ins live inside the existing Soul state tree. An external-credential profile must relocate those files and route future authentication writes. A harness that cannot support this separation reports that incompatibility. A successful manifest check cannot prove that arbitrary project files contain no accidentally embedded secret. SoullInfo.json carries credential references only; its schema has no field that can hold secret material. The validator scans covered files for known secret patterns and treats a match as an error in the control plane and as a warning elsewhere. Export runs the same scan over selected native session records and home/ content and reports matches as warnings. This is detection, not a guarantee. [R6] [R7]

The format's enforceable promise is that managed credential custody, native authentication routing, and export handling follow the declared profile. Any stronger claim about arbitrary workspace bytes requires separately defined controls and evidence. The exact scope of the original whole-bundle prohibition remains an open decision.

On import, a friendly credential name cannot silently select a different account. The host verifies backend and principal identity, required capability, scope, and local authorization. Missing bindings remain unresolved and affect readiness. Secret rotation can preserve the binding while recording a non-secret version reference when available; changing the declared purpose or intended principal is an accepted configuration change.

12 Sharing backup and transfer

Native package and transport archive

The active .soul directory is the local working representation. A transport archive carries a declared selection of its contents. The archive format and suffix are specification decisions; the export profile determines which components travel. Archive creation and import require an inventory, content validation, bounded extraction, and explicit treatment of external references.

A directory shown as one file remains subject to concurrent writes. A consistent export coordinates the relevant writers, uses supported component snapshots, or pauses work at a defined boundary. It records which consistency guarantees were achieved. A raw filesystem copy is a byte-copy operation; a compliant transfer also establishes identity treatment, completeness, and readiness.

Transfer profiles

Operation	Identity treatment	Default content	Destination behavior
Template creation and instantiation	Instantiation creates a new Soul ID	Accepted definition and deliberately selected seed resources	Bind credentials and provision supported execution requirements

Operation	Identity treatment	Default content	Destination behavior
Life transfer or restore	Preserve Soul ID; the previous placement is retired or the operation is recorded as a fork	Definition plus selected durable life and history	Restore relationships and report unresolved bindings
Explicit fork	Create a new ID with parent provenance	Definition and explicitly selected inherited life	Preserve historical attribution and establish new bindings
Local backup	Preserve Soul ID	Consistent durable snapshot; optional verified machinery	Report external dependencies that require separate recovery
Filesystem duplicate	No automatic identity change	Whatever bytes were copied	Registration must identify backup, relocation, fork, or unsupported duplicate

Portable exports omit reconstructible installations by default and can include verified platform artifacts for offline use as an explicit option. Durable private history is excluded from template exports unless selected. Credentials remain external under the proposed custody model. A backup can identify external vault dependencies, but cannot claim complete credential recovery merely because the package itself was saved.

A life transfer preserves the selected repository relationships, uncommitted work, supported native session records, retained evidence, learned knowledge and graph assertions, dream records, and revision history. Preserving a session record does not establish that a different harness version or host can resume it; native-session compatibility is assessed separately. Knowledge transfer follows the source and derivative selection rules in Section 8. Exclusions, unresolved links, and recipient access treatment are part of the transfer receipt. A verified signature binds a content digest to a signing key. Claims about authorship and acceptance rely on the associated provenance and trust policy. The protected profile uses the separately specified encryption and access contract in Section 6. Restore does not treat a signed historical checkpoint as fresh authorization. The destination applies current known restrictions and reports unresolved freshness before protected activation.

Multiple placements

The first format should preserve the distinction between a Soul ID and a host-local installation ID. A local duplicate check can detect collisions on one host. It cannot prove that another host is not running the same Soul.

The proposed initial profile does not claim coordinated concurrent writers across hosts. Intentional migration establishes a new active placement; independent branches of a life use an explicit fork. A future replica protocol could define synchronization, divergence, conflict resolution, and lease behavior without changing the basic package ownership model. This limitation must be visible rather than disguised as global uniqueness.

13 Agent Store distribution and routine evaluation

An Agent Store is a distribution and discovery interface for versioned Soul releases. It can list capabilities, compatible execution profiles, required permissions, knowledge coverage, licensing terms, and evaluation evidence. The portable package and its owner-controlled identity remain independent of a particular

store. A registry can distribute the same interoperable format through a public catalog, an enterprise catalog, or a direct private exchange.

Release and installation model

The proposed default store product is a reusable template with selected skills, tools, curated seed knowledge, and declared dependencies. Instantiation creates a new Soul identity. A listing that transfers a continuing life or offers an inherited-life fork must state that operation and its data scope explicitly. Because the first profile has no coordinated multi-host writers (Section 12), a life transfer through a store either retires the publisher's placement or is registered as an explicit fork on the recipient side. Private sessions, unselected memories, credentials, and quarantine evidence are not implicitly included in a store upload.

A store release has its own release identifier and optional human-readable version. A signed release manifest binds the publisher, distribution profile, artifact digest, logical definition revision, exported knowledge checkpoint, access profile, compatibility requirements, license, and evaluation references. These identities distinguish the package that was tested from its marketing name or mutable listing URL. Installation verifies the selected artifact and presents its required local bindings and grants before activation under host policy.

Encrypted distribution uses recipient entitlement and key wrapping or an equivalent declared access mechanism. Permission to acquire a package does not automatically authorize every contained source, external account, or runtime capability. Free distribution, paid licenses, and subscription access can use the same release identity and transfer contract; billing and commercial enforcement are store responsibilities. Revoking future access cannot recall plaintext or keys already legitimately delivered to a recipient.

Evaluation as a recurring dream activity

A dream evaluation policy can trigger checks at a configured interval, before publication, after a primitive or knowledge change, after a runtime upgrade, or after an alignment finding is resolved. These are proposed host scheduling capabilities, not a schedule activated by this paper. The policy defines scope, permitted resources, execution budget, expected evidence, release thresholds, and authorized reporting destinations.

The evaluation runs against the assembled distributable candidate or released snapshot. Evaluating a publisher's private live Soul does not establish the behavior of an export with different knowledge, redactions, or permissions. An isolated evaluation instance uses controlled side effects and the permissions advertised for the release. Test conversations and fixtures do not silently enter production memory or workspaces. Evidence remains under the declared evaluation record until separately selected for learning.

The run compares relevant capabilities with a recorded baseline and retains both successes and failures. A no-change dream can produce fresh evidence for an existing release. A blocked test, infrastructure error, missing dependency, or incomplete run is reported separately from a task failure; none is silently converted into a passing result.

Dream-generated tests can reproduce observed failures and become useful regression cases. Development cases, regression gates, and held-out evaluation have distinct roles. Repeated exposure to a listing benchmark can turn it into training material, so the evidence records test exposure and

contamination limits. Release evaluation also needs owner-defined or independently maintained cases where the corresponding claim requires them. A candidate does not establish its quality solely by authoring and grading its own tests. Test provenance, overlap with learning material, and unavailable or private fixtures remain visible in the evidence policy.

Useful evaluation areas include task completion, instruction adherence, grounded retrieval with correct source use, preservation of accepted knowledge, permission-boundary enforcement, recovery from interruption, transfer and restore, and quarantine behavior. Resource cost and latency can be reported under a stated hardware and execution profile. The chosen suites should test the advertised capabilities and known failure modes. This proposal supplies no invented scores or claim of universal agent safety.

Evidence attached to exact releases

Listing evidence	Required context
Tested release	Artifact identity, definition revision, exported knowledge checkpoint, and relevant configuration
Evaluation identity	Suite version and digest, evaluator identity, method, and publication provenance
Execution conditions	Harness and adapter versions, model or provider identifiers, retrieval configuration, permissions, and relevant host resources
Results	Metric definitions, sample counts, passes, failures, exclusions, and uncertainty or variability where measured
Freshness	Execution date, coverage period, and whether later relevant changes make the evidence stale
Evidence availability	Public receipts, protected or withheld test material, and reproduction limitations
Current release status	Applicable unresolved regressions, containment decisions, withdrawal, or superseding release

An evaluation attestation binds this information to its retained result record. A signature binds the report to a signing key; attribution relies on the associated trust policy. Independent verification requires a separate evaluator and a stated verification process. A store should clearly distinguish publisher-reported results, automated store checks, and independent evaluations. Deterministic checks, human judgments, and model-based grading identify their methods separately. A benchmark-specific pass is not a blanket certification of safety, correctness, or suitability for every task.

Results for a package tested with one harness, model, permission set, or knowledge snapshot do not automatically apply to another combination. Remote model aliases and services can change even when the package revision stays fixed. A recipient's Soul also accumulates new private life after installation. Store evidence describes the distributed release under its tested conditions; local dream evaluations describe that recipient's evolving instance. Relevant changes trigger fresh checks or a visible stale status.

A listing that communicates progress

A listing can show a capability summary, tested environments, recent evaluation date, suite results, known limitations, and a history of comparable releases. Trends compare the same metric and suite or explicitly mark a methodology change. They retain the sample denominator and failure information, so a selective successful run does not silently replace an unfavorable release record.

The publisher creates a reviewed public evidence view from protected evaluation records. It can expose aggregate results and permitted examples while retaining private project data, credentials, prompts, and sensitive failure details under their access policy. Hiding evidence limits what a verifier can establish; the listing records that limitation. Curating a result for publication follows the same source and derivative disclosure rules as sharing knowledge.

Publication and listing updates are separate capabilities. An owner's standing policy can authorize routine evaluation summaries to update an approved destination after defined gates pass. Dream execution alone does not grant publishing authority. A failed release gate leaves the previous approved release available according to policy and records the candidate's unresolved status.

If later evaluation or alignment review finds a serious problem, the release can acquire an advisory, suspension, or withdrawal state under the store's declared policy. The affected package and evidence remain identifiable. Installed owners receive applicable status through an authorized update mechanism, and their hosts apply local containment or update policy. Republishing under a new filename must not erase known lineage or bypass a held release decision.

The distribution model makes a Soul's capabilities and improvement history reviewable before adoption. It also preserves the boundary between reusable public learning, a recipient's private continuing life, and the evidence actually available to support a listing.

14 Compatibility and migration

The existing Soul package already has formats 1 and 2, a directory-package convention, deterministic hashing, extension preservation, and version rejection rules. Existing environment documentation classifies durable life and supplies transfer semantics. The proposed layout is therefore a migration from established contracts. [R2] [R3] [R4]

Existing area	Proposed destination or treatment
soul.json	Translate into the new manifest with an explicit format discriminator
Standing instructions and persona files	Preserve semantics in the accepted resource and instruction inventory
skills/	Map accepted source into primitives/skills/raw/
Generated harness configuration	Regenerate from accepted sources into declared materializations
.soul-state/runtimes/	Inventory and validate reusable installations under opt/
.soul-state/home/ and tool homes	Classify into home/, external auth, caches, and other declared roles
Agent Space and retained memory	Map durable evidence and learned knowledge into memorial/ with source mappings and provenance
Existing knowledge graphs and retrieval indexes	Preserve authoritative assertions and corrections; migrate logical schemas and rebuild incompatible query indexes
Dream state beside host daemon state	Extract per-Soul durable records into dreams/; recreate active host state
worktrees/ and repository relationships	Map into workspaces/repos/ and workspaces/worktrees/
Revision journals and snapshots	Preserve under the portable history contract without changing old meanings

Migration first inventories and classifies the source. It preserves the source until the destination is validated, records path and source-identity mappings, relocates credential custody where required, encrypts protected content, rebuilds derived outputs, and activates the result only after consistency checks. Unsupported files are retained and reported for resolution. Any retained plaintext source backup remains explicitly identified and access-controlled; migration cannot claim to have removed earlier copies merely by encrypting the new package.

The Soul ID remains stable. Historical hashes, approval records, and dream receipts keep their original algorithms and schema interpretations. A new layout can create a new definition revision and a migration receipt; it does not recompute history as though older packages had always used the new format. Migration creates a separately authenticated identity binding for the new layout under the existing owner's authority. Historical identity and lineage records remain interpretable under their original schemas rather than being retroactively hashed as reusable definition content.

Readers preserve unknown optional data and clearly reject unsupported required semantics. Layout changes that alter the meaning of a path require an explicit migration. Host applications consume a versioned descriptor so a mixed-version rollout does not depend on duplicated filesystem guesses.

15 Verification and conformance

Independent results

A validator does not execute bundled code, start services, or provision dependencies. It can inspect the public envelope and ciphertext integrity without unlocking protected contents. Logical content conformance and definition verification require a separate authorized inspection through the trusted decryption boundary. A locked package receives a limited result, never a claim that unavailable plaintext has been verified. Format conformance, content integrity, signature trust, unlock state, and execution readiness remain distinct evaluations.

The conformance suite tests implementations using portable fixtures. It should include canonical byte preimages and expected digests, supported and unsupported manifests, path and mode cases, transfer inventories, and failure-recovery scenarios. A Node test runner can execute one implementation's suite without making the specification language-dependent.

Acceptance scenarios

Scenario	Required observable result
Accepted helper is changed	Definition-integrity verification detects the changed inventoried bytes
Existing runtime is substituted	Launch validation rejects bytes that conflict with the required artifact identity; any permitted unverified external executable is separately declared
Temporary files are removed during an interrupted dream	Inputs, candidates, checkpoints, and unfinished worktree references remain recoverable
A source changes at the same path	Earlier progress for a different digest does not credit the new contents

Scenario	Required observable result
A dream candidate has a stale base	Publication refuses the conflict or requires explicit reconciliation and revalidation
Harness output is regenerated	Canonical and authoritative native sources remain unchanged; provenance identifies the new rendering
Definition publication is interrupted	Readers observe a complete revision or an explicit recovery state
A worktree moves with its repository	Relationships and selected uncommitted work are restored or reported unresolved
A template and a life export are imported	The template creates identity; the life transfer preserves identity and its declared durable data
A credential capability is incompatible	Readiness reports the mismatch without exporting protected material
A required platform artifact is absent	The retained life stays inspectable and the missing execution requirement is identified
A service definition is transferred	Its definition persists; active PIDs, sockets, and leases are recreated locally
Retrieval caches are deleted before transfer	Selected evidence, accepted graph assertions, and lineage remain intact; supported indexes can be rebuilt
The embedding model changes	Incompatible vectors are not mixed; readiness reports rebuilding without rewriting accepted knowledge
A knowledge claim is corrected	Current retrieval exposes its status and replacement while preserving the declared history
Private sources are excluded from a template	Export selection also accounts for their summaries, assertions, chunks, embeddings, and protected metadata
Retrieval coverage is incomplete	Results identify missing or stale coverage instead of presenting absence as proof that no relevant knowledge exists
An ordinary agent runs a recursive file search	Protected primitive and source bytes are inaccessible through its filesystem; only granted task files and explicitly permitted projections (Section 6) are visible
A dream proposes a change to the control plane	Automatic acceptance does not apply; the proposal waits for the owner or a delegated authority that did not produce it
An agent calls the broker with the harness's credential	The broker enforces the session's grant, budgets, and output limits identically; nothing beyond the session's grant is reachable
Two sessions share one operating-system identity	The protected profile uses enforced isolation or verifies complete teardown and protected output handling before reuse; failed cleanup blocks the next session
A template is opened before its signer is trusted	No bundled helper, hook, runtime, or service runs; readiness is computed from manifest, lock, and digests
An archive entry resolves outside the content root	Import rejects the archive and names the offending entry
A quarantined version is selected for a release export	The export validator refuses the version and its derivatives
A retrieval index is rebuilt on a new host	Quarantined sources stay excluded because quarantine state is an index input
A dream finding targets SOUL.md or access/	The finding is escalated without rewriting control records; a separately authorized host policy may suspend execution while owner recovery remains available

Scenario	Required observable result
A dream requests sources outside its grant	The broker denies the disclosure; an agent-supplied role or path cannot enlarge the grant
A protected package is opened without keys	Envelope inspection works and protected-content validation remains explicitly unavailable
A native harness needs unrestricted source access	The protected profile reports incompatibility unless a separately authorized exposure satisfies its declared rules
An export uses a new recipient key	Logical content identity is preserved while ciphertext integrity and recipient access metadata are recomputed
A dream flags a dangerous direction	The finding records its evidence, reference policy, affected versions, uncertainty, and separately authorized containment
A source is quarantined	Ordinary retrieval and activation exclude it and affected derivatives while authorized review retains access
A session is contained	The host reports revoked future grants, stopped or unresolved child work, and any required restart with fresh context
A finding is a false positive	A separate disposition restores permitted access without deleting the original evidence or its correction
A quarantined life is transferred	The declared restrictions survive import; moving the package does not imply release
A listing displays an evaluation	The result identifies the exact tested release, suite, conditions, date, and evaluator
A release changes after evaluation	Relevant earlier evidence is marked stale or scoped to the old release until new checks complete
A dream produces a failing or blocked evaluation	Failure, interruption, and missing coverage remain visible; configured release gates are enforced
A dream prepares a store update	Publication requires a separate explicit or standing grant and a reviewed disclosure view
A dream changes SOUL.md, a skill script, or .zprofile	Control-plane classification follows behavior; current trusted policy requires owner or independent delegated acceptance
A candidate changes its own acceptance rules	The currently trusted policy evaluates the change; candidate-provided rules cannot authorize it
A second session uses the proposer's approval credentials	A new session ID does not establish independent authority; self-acceptance is rejected
A source-granted dream leaves a child process or plaintext copy	The sequential fallback blocks reuse until descendants and capabilities are terminated and retained outputs are protected
Two instances use one unchanged template	Definition digests match; distinct authenticated identity bindings preserve separate Soul IDs
An identity key is rotated or recovered	Authorized continuity preserves the Soul ID and definition digest; an unverified replacement key is rejected
A summary mixes owner, third-party, and seed sources	All contributing lineage and classifications remain available; owner text alone cannot authorize automatic acceptance
A cached result predates a quarantine decision	Current restrictions exclude the affected source and derivatives before disclosure authorization
An old backup predates a known restriction	The host retains newer known restrictions and refuses rollback of its trusted policy epoch
An offline import cannot establish restriction freshness	Freshness is reported unknown; protected activity waits for a recorded authorized recovery decision

Scenario	Required observable result
Concurrent sessions exhaust one disclosure budget	Atomic accounting across those sessions denies further unreserved disclosure; token renewal does not reset the ledger
Publication fails after acceptance	A durable failure or interruption receipt identifies the candidate and recovery boundary; no success is inferred
A copied device exposes ciphertext sizes	The validator reports the profile's declared physical-metadata leakage; encrypted logical fields alone do not imply length hiding

These scenarios establish observable responsibilities. They do not claim measured reliability, security certification, or completed cross-platform coverage. Each implementation publishes the profiles and platforms it has actually verified.

16 Decisions and implementation path

Proposed defaults and unresolved choices

The architectural direction is a persistent native package with explicit functional directories, durable workspaces, reusable primitives, versioned runtimes, protected source access, local retrieval, a portable knowledge graph, and dream workflows that depend on the layout. The following details remain decisions for the specification.

Decision	Proposed default	What remains to settle
Content root	Contents/ inside the .soul package	Confirm wrapper and reserved control paths
Identity anchor	Stable Soul ID with a separately authenticated identity binding; reusable definition excludes instance lineage	Specify identity envelopes, continuity, recovery authority, and key encoding
Control plane	Behavior-based inventory; current-policy acceptance by owner or independent delegate	Finalize dependency classification, delegation proof, and authenticated control-record schemas
Session isolation	Distinct sandbox or identity; sequential reuse only after verified teardown	Specify lifecycle enforcement, state reset, and evidence for each host profile
Source trust class	Separate provenance, confidentiality, evidential status, and authority; retain mixed-source lineage	Specify attribute schemas and claim-specific acceptance and validation gates
Raw skill semantics	Accepted canonical source	Confirm imported candidate and native override locations
Memorial semantics	Durable evidence, learned knowledge, and graph assertions	Define exact child paths, logical schemas, provenance, and correction rules
Retrieval portability	Durable corpus and logical graph with rebuildable indexes	Define query results, coverage metadata, compatible index profiles, and export selection
Protected content	Broker-mediated encrypted sources with explicit logical and physical metadata leakage	Define envelopes, padding if required, recovery, grants, and enforcement profiles

Decision	Proposed default	What remains to settle
Alignment review and quarantine	Current authenticated restriction state overrides stale snapshots and pinned definitions	Specify local policy epochs, freshness authorities, offline recovery, and revocation boundaries
Agent Store distribution	Versioned template releases with selected knowledge and scoped evaluation attestations	Define release manifests, result schemas, publisher trust, listing updates, licensing, and withdrawal policy
Shared scope	Reuse within one Soul	Define optional cross-Soul references and export closure
Revision activation	Pin definition content while applying current authorization and restrictions	Define transition boundaries and publication and recovery receipts
Home routing	A declared Soul home with adapter-reported support	Define full-home and per-tool compatibility profiles
Runtime installation ownership	Managed runtimes under opt/	Define compatibility with managers that own installations under home/
Credential prohibition	Managed custody and native auth remain external	Define the guarantee for arbitrary workspace content
Concurrent placements	No coordinated multi-host writers in the first profile	Define future replicas or retain explicit forks
Integrity and transport	Reusable definition digest plus authenticated identity, provenance, and transfer records	Select canonical encoding and final signature, archive, and encryption profiles
Disclosure accounting	Durable, atomic host-local ledger across sessions for a declared recipient and corpus scope	Specify charging units, renewal authority, derived-output handling, and any future cross-host accounting

A staged implementation

The first stage writes the format and layout contract, manifest definitions, content-classification rules, and language-neutral fixtures under an independently versioned specification. It builds a read-only envelope inspector and authorized logical-content validator. The protection profile receives an explicit enforcement and key-recovery design before claiming curated access is mandatory. The specification can remain in the existing repository until the agreed extraction criteria justify a separate home. [R1] The core inspection profile can validate layout, inventories, identity-record structure, and digest fixtures without claiming protected execution or knowledge-service support. Appendix C supplies a small example and expected results. Each additional capability declares its required schemas and tests explicitly.

The second stage extends the existing versioned environment descriptor to represent the new layout and explicit migration from current Souls. It proves retention of identity, workspaces, memories, learned assertions, revision history, and unknown content. A portable knowledge fixture demonstrates source lineage and correction history independently of the original database. Transfer and restore should be exercised before the new layout becomes the default for active work.

The third stage binds execution to declared profiles and consistent definition snapshots. It validates runtime artifacts, routes home and credential state, and connects harness tools to broker-enforced grants. It materializes only permitted primitives without changing another session's inputs and verifies that agent

shell access cannot bypass source restrictions. Each adapter states its supported capabilities and limitations.

The fourth stage connects local retrieval and graph queries to sessions and moves dream discovery and durable records onto the layout contract. It verifies source identities, coverage, staged candidates, knowledge corrections, alignment findings, quarantine enforcement, and recovery after interruption. Transfer tests remove indexes and prove that retained knowledge can be queried again while access and quarantine restrictions survive. Background-service integration follows the same distinction between portable definitions and active host state.

The fifth stage establishes routine evaluation policies, release manifests, and portable result attestations. A reference Agent Store listing shows which release was tested and under which conditions. It proves that failed gates, stale results, private evaluation evidence, and unauthorized listing updates are handled according to the declared policy before claiming store-ready distribution.

Freeze the first profile only after an independent reader demonstrates preservation and conformance. Specify or explicitly exclude the remaining profile choices.

References

Repository documentation below was examined at main commit 31619aaebbcfb19df92257851f6aed6ed93a326e, dated 19 October 2026. Sources were accessed on 20 October 2026. References to the repository describe documented contracts; they do not substitute for implementation validation.

[R1] qwts. Spec soul as opaque soul bundle package. Issue 1147, including its review discussion.

[GitHub issue 1147](#)

[R2] qwts. ADR 0583 The soul root owns the environment.

[Repository / decisions/ADR-0583-the-soul-root-owns-the-environment.md](#)

[R3] qwts. Soul packages formats 1 and 2.

[Repository / soul-package.md](#)

[R4] qwts. Soul environment schema 1.

[Repository / soul-environment.md](#)

[R5] qwts. Soul runtimes and harness installs.

[Repository / soul-runtimes.md](#)

[R6] qwts. Soul tool homes and sign in adoption.

[Repository / soul-tool-homes.md](#)

[R7] qwts. Soul credentials and Soul providers.

[Repository / soul-credentials.md](#) | [Repository / soul-providers.md](#)

[R8] qwts. Soul builder.

[Repository / soul-builder.md](#)

[R9] qwts. Soul revisions format 1.

[Repository / soul-revisions.md](#)

[R10] qwts. Dream maintenance implementation contract.

[Repository / soul-skill-dream-design.md](#)

References continued

- [R11] Apple. Document Packages in Bundle Programming Guide.
[Apple Developer / Bundle Programming Guide](#)
- [R12] Git project. git worktree documentation.
git-scm.com/docs/git-worktree
- [R13] Agent Skills. Specification.
agentskills.io/specification
- [R14] Patrick Lewis and colleagues. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS 2020; arXiv version 4, 2021.
arxiv.org/abs/2005.11401v4
- [R15] Darren Edge and colleagues. From Local to Global A Graph RAG Approach to Query-Focused Summarization. arXiv version 2, 2025.
arxiv.org/abs/2404.16130v2
- [R16] W3C. PROV-DM The PROV Data Model. W3C Recommendation, 2013.
www.w3.org/TR/prov-dm/
- [R17] John X. Morris, Volodymyr Kuleshov, Vitaly Shmatikov, and Alexander M. Rush. Text Embeddings Reveal (Almost) As Much As Text. EMNLP 2023; arXiv:2310.06816.
<https://arxiv.org/abs/2310.06816>
- [R18] Anders Rundgren, Bradley Jordan, and Samuel Erdtman. JSON Canonicalization Scheme (JCS). RFC 8785, June 2020.
<https://www.rfc-editor.org/rfc/rfc8785>

Appendix A Changes from version 0.1

Version 0.2 is a security and consistency revision of the 0.1 draft. It does not change the architectural direction. Section numbers refer to this version.

Area	Change	Reason
Section 1	Added a threat model naming five adversaries	Several 0.1 properties had no named adversary, which made their strength unclear
Sections 3, 7, 9	Defined a control plane that is never auto-accepted and never auto-quarantined	0.1 allowed automatic acceptance without excluding the files that govern acceptance, grants, and execution, which permitted self-escalation and injected lockout
Section 2	Soul ID excluded from the digest preimage and bound to an identity key	0.1 said an instance received a new revision only if covered bytes changed, but the ID is covered; self-asserted IDs also made attribution unverifiable
Section 2	Inspection of an untrusted package runs no bundled code	Readiness evaluation on open could otherwise execute helpers or runtimes before trust is established
Section 3	Path safety made a 0.1 requirement; run/ renamed host/	Zip-slip class defects are not a deferred decision; run/ collided with durable run records under dreams/runs/
Section 3	Curated memory removed from Resources/	Resources/ is definition and ships in templates by default, contradicting the confidentiality rule in Section 6
Section 3	Generated wrappers in bin/ regenerated from a covered recipe and verified at launch	0.1 left wrappers outside the inventory while requiring every executed input to be covered
Sections 3, 6	cache/, logs/, and tmp/ carry a protection class; protected-profile indexes are broker-private	Embeddings and logs of protected sources were unprotected derived read paths
Section 4	Secret-prone home files are external credentials by default	.npmrc and sign-in state would otherwise enter revisions and exports
Section 5	Materialization isolation declared dependent on home routing	0.1 promised per-session isolation that unrouted harnesses cannot provide
Section 6	Harness and agent shell treated as one trust domain; per-session tokens; broker-side enforcement	Any harness-held broker credential is readable by agent code
Section 6	Broker-managed executor defined as a separate sandbox	Running Soul-authored code inside the broker exposed its keys
Section 6	Source-granted dreams run under a distinct identity or the profile is single-session	Isolation between sessions sharing one identity was asserted but unenforceable
Section 6	Encrypted storage mapping and metadata coverage	Names, sizes, and counts of protected objects leaked structure
Section 8	Source trust class recorded at ingestion and propagated	Retrieval had no ingestion-time defense against memory poisoning

Area	Change	Reason
Section 8	Quarantine state added to index manifest inputs	A rebuild on a new host would re-index held content
Section 9	Release profile refuses quarantined versions; life transfer carries restrictions	Section 13 excluded quarantine evidence from store uploads while Section 9 required restrictions to travel
Section 11	Inference is a brokered operation under the protected profile; secret scanning in validator and export	A readable provider key is exfiltrable through any tool call
Sections 12, 13	Store life transfer retires the previous placement or is a fork	Section 13 allowed a life transfer that Section 12's single-placement rule could not represent
Section 15	Nine acceptance scenarios added	Cover the new requirements with observable results
Section 16	Identity anchor, control plane, session isolation, and source trust class added as decisions	Make the new defaults and open choices explicit

Appendix B Changes from version 0.2

White paper version 0.3 applies the peer-review corrections to the existing architecture. These are proposed requirements and examples for review; they do not establish that an implementation has passed conformance.

Sections 2 3 7 and 14. Separate reusable definition content from authenticated Soul identity, acceptance, and lineage records.

Sections 3 4 7 and 9. Define control-plane membership by behavior, require current-policy acceptance, and distinguish independent authority from a second session.

Section 6. Require verified teardown for sequential session reuse; declare physical metadata leakage and persistent disclosure accounting.

Section 8. Preserve mixed-source lineage and separate provenance, confidentiality, evidential status, and authority.

Sections 7 8 9 and 12. Apply current quarantine restrictions to cached results and restored snapshots; state offline freshness limits.

Section 9. Require durable outcomes for successful, failed, and interrupted publication, including representation and activation results.

Sections 15 and 16. Extend acceptance scenarios and distinguish the core inspection profile from optional execution and knowledge capabilities.

References and Appendix C. Add embedding-inversion and canonicalization references, a concrete digest fixture, and explicit validator outcomes.

Appendix C Worked core package example

This example exercises the smallest inspection contract: three covered files, a deterministic definition inventory, separate identity and checkpoint records, and a declared transfer selection. It uses the local identifier soul-core-example-1 solely for this appendix. It is not a registered format or a complete cryptographic profile. The example has no executable profile, protected corpus, or signatures, and therefore establishes neither trusted identity nor execution readiness.

Covered files and exact bytes

All paths below are relative to Contents/. File contents use UTF-8. SoulInfo.json and SoulLock.json contain exactly the displayed compact JSON with no trailing newline. Resources/SOUL.md contains the displayed sentence followed by one LF byte. Each covered file has the logical portable mode 0644; this is an inventory value rather than a claim about host-specific access controls.

Resources/SOUL.md

You are a careful assistant.

Byte length: 29. SHA-256:

6055fd8b575e7ae95ac3e52ab2590a9a2a586b93154dede1353929573f6d56af

SoulInfo.json

```
{"format":"soul-core-example-1","layout":"contents-1"}
```

Byte length: 54. SHA-256:

4b9c215f61322588219a75aa952b757af25660ffbf085bc24b8ff3b3ad12d241

SoulLock.json

```
{"artifacts":[]}
```

Byte length: 16. SHA-256:

bf551427bb93b2762e29fc326bedde7a8e2cac4e25b3126cea8fc66f46d2e55b

Definition digest

Sort the inventory by path using ascending ASCII byte order. Each entry has exactly the keys mode, path, and sha256. Serialize the resulting array using RFC 8785 JSON canonicalization. The example uses only ASCII strings and arrays, so the compact, sorted-key serialization below produces the same canonical bytes. Prepend the UTF-8 bytes of SOUL-DEFINITION-EXAMPLE-1 followed by one LF byte, and hash the concatenation with SHA-256. The prefix separates this example from other signed or hashed record types. [R18]

```
inventory = sorted(inventory, key=lambda entry: entry["path"])
encoded = json.dumps(inventory, sort_keys=True,
    separators=(",", ":"), ensure_ascii=False).encode("utf-8")
preimage = b"SOUL-DEFINITION-EXAMPLE-1\n" + encoded
definition_revision = hashlib.sha256(preimage).hexdigest()
```

The expected preimage length is 376 bytes. The expected definition revision is:

36a0351bd2ac3db3b091b65c85def71064632889c34215e75a97ae88595ce768

The inventory is a record of these three files, not an additional covered file. Soulidentity.json and the records below are excluded from this reusable definition digest by the example's fixed rules. An implementation must not allow a candidate inventory to expand those exclusions. Adding or changing any covered file requires rebuilding the inventory and digest.

Separate identity and provenance

Instance A uses Soul ID urn:uuid:655fc577-804b-4e75-a5d8-70f9a21a19ce. Instance B uses urn:uuid:2d36cd18-b647-4b71-b673-f0c4ecec5827. Each Soulidentity.json names its own ID, an externally resolved identity-key reference, and the definition revision computed above. The statements are authenticated separately in a complete profile. Both instances can reference the same definition because none of their identity fields changes the three covered files.

A template-instantiation receipt records the source template identity and revision. A fork receipt records the parent Soul ID and inherited checkpoint while assigning a new Soul ID. Rotation and recovery

receipts preserve a Soul ID only when authorized under the established continuity policy. Ordinary Git author text is attribution metadata; it does not replace verification of those records. This fixture intentionally supplies no signing keys or signatures, so a validator must report identity trust as unverified.

Checkpoint and transfer receipt

The example checkpoint identifies instance A, the computed definition revision, the three covered files, and an empty selected durable-life inventory. It declares no managed credential material, no live processes, and no retrieval cache. It records the source host's local restriction epoch as 0 and freshness as unknown to an independent recipient. That local number is not globally comparable and does not establish that no later restriction exists.

A template transfer receipt selects the three definition files and the template provenance statement; destination instantiation creates a new identity binding. A life-transfer receipt additionally selects instance A's authenticated identity and checkpoint records and declares the retirement or unresolved status of the prior placement. Neither receipt exports credentials or grants a destination permission to execute. A recipient verifies the selected bytes, authenticates the relevant records when possible, and resolves required authorization and freshness separately.

Expected validator outcomes

Definition inventory and byte digests: pass when the three exact files, paths, modes, and expected digest match.

Identity authentication: unverified because the example provides no identity signature or trust binding.

Protected-content status: not applicable to this unencrypted core fixture; it makes no protected-execution claim.

Restriction freshness: unknown to a new recipient; protected activity requires the declared authorization or recovery decision.

Execution readiness: unavailable because no executable profile or required local grants are declared.

Changed SOUL.md byte: inventory verification fails. Recomputed content receives a different definition revision and requires the applicable acceptance decision.

Different instance ID or an authorized key rotation: definition digest stays the same, but the separate identity statement and authentication evidence change.

Known newer local restriction with an older checkpoint: retain the newer restriction and reject rollback, even when historical content integrity passes.

A complete conformance release must add positive and negative signature fixtures, malformed and duplicate-key manifests, path and mode cases, interruption recovery, and the host-enforcement tests required by each claimed capability. This small example establishes byte-level agreement and explicit limited results without substituting for those suites.