RANKIGI × EVIDE

Interface Mapping (v0.3)

Execution Proof, Responsibility Closure, and Evidentiary Portability

Authors

Wesley Snow

Founder, RANKIGI - Independent Execution Proof Layer for AI Agents

Emanuel Celano

Protocol </AI> Founder | EVIDE – The Missing Evidentiary Layer for AI | CertifyWebContent.com | DAPI-Certification.com | Digital Evidence & AI Governance

Purpose

This document defines a minimal interoperability boundary between an execution-layer system (RANKIGI / KYA) and an evidentiary responsibility layer (EVIDE), enabling the generation of a portable, independently verifiable closure record.

The objective is not system integration, but boundary definition:

- execution proof (what happened)
- responsibility closure (who is accountable)
- evidentiary portability (can it be proven outside the system)

This mapping was developed through direct technical collaboration between independent systems and validated at both the architectural and evidentiary level

Scope

This document defines only the interface layer. It does not modify the native RANKIGI or EVIDE schemas.

The objective is to establish a stable handshake between:

- execution-level evidence (RANKIGI / KYA)
- responsibility closure object (EVIDE)
- independent evidentiary export (FEDIS-compatible)

Key Architectural Insight

Execution proof and responsibility proof are not the same object.

- RANKIGI (KYA) provides a tamper-evident execution chain.
- EVIDE defines a structured responsibility closure object.
- FEDIS enables independent verification and legal admissibility.

The closure object does not exist inside the execution chain. It is anchored alongside it, and remains portable outside the originating system.

1. Field Mapping (RANKIGI ↔ EVIDE Interface Layer)

EVIDE reference: External Evidentiary Deposit (EVIDE)

https://app.certifywebcontent.com/

The following represents a minimal alignment between systems at the closure boundary.

RANKIGI → EVIDE Mapping

- `chain_id / event_id → source_event_ref.substrate_id`
- `event_hash_chain.prev_hash → source_event_ref.previous_hash`
- `actor_external_ref → actor.actor_external_ref`
- `actor_provider → actor.actor_provider`
- `policy_ref → actor.authority_basis`
- `closure_kind → closure.closure_kind`
- `occurred_at → closure.declared_at_utc`
- `closure_kind values → intervention.type`
- `chain_subjects.subject_type → subject.subject_type`
- `chain_subjects.external_ref → subject.subject_external_ref`
- `chain_subjects.lifecycle_state → subject.lifecycle_state`
- `timestamp_proofs → integrity.timestamp_token_ref`
- `passport_certificate → actor.authority_proof (external authority verification)`

Note

Field names represent an interface mapping, not native schema structures.

The passport_certificate is not the identity reference itself, but the cryptographic proof that allows external verification of the authority bound to actor_external_ref.

2. Closure Object Normalization

Before hashing and anchoring, the closure object must be normalized.

Requirements

- deterministic field ordering
- UTF-8 encoding
- no environment-dependent fields
- no runtime-dependent metadata
- stable serialization format (JSON canonicalization)

The normalized object is the only valid input for hashing. This ensures:

- reproducibility of the hash
- independence from system implementation
- consistency across verification environments

Compatibility note: the canonical JSON rules defined in KYA Section 6.1 are fully compatible with this normalization model. No divergence exists between the two specifications.

3. Integrity Model (Dual Anchoring)

The architecture relies on two independent integrity layers.

RANKIGI Layer

- event-level hash chaining

-- RFC 3161 timestamping at execution level

EVIDE Layer

-- independent closure object hashing
-- RFC 3161 timestamping at closure level

Key principle:

Execution proof is not responsibility proof

Both must exist independently.

The closure object sits alongside the KYA chain, not inside it. The source_event_ref field bridges the two layers without merging them.

4. Portable Closure Record (Export Format)

The exported closure record must be self-validating. It must allow a third party to verify the record without:

-- access to RANKIGI infrastructure
-- access to EVIDE systems
-- privileged APIs or internal logs

Minimum required structure

-- reference to originating chain event (event_hash) as recorded at the closure moment
-- normalized closure object
-- normalization rules used (see Section 2)
-- closure object hash
-- RFC 3161 timestamp proof (closure level)
-- authority reference (external identity baseline)
-- authority proof (e.g. passport_certificate or equivalent external verification artifact)

Optional but recommended

-- previous_hash reference
-- subject lifecycle snapshot

Self-validation test: a qualified third party must be able to verify the complete closure record using standard tools, without calling back to either RANKIGI or EVIDE. If that condition is not met, the record is not considered portable.

5. Validation Constraints

A closure record is considered valid if:

-- the closure object is complete, declared, and explicitly attributed
-- the hash matches the normalized object
-- the RFC 3161 timestamp is valid and verifiable
-- the authority reference is externally resolvable
-- the source_event_ref correctly links to the execution chain

Failure in any of these conditions results in:

-- invalid evidentiary closure
-- non-portable record
-- dependency on originating system

6. KYA Conformance Requirement

The originating execution substrate must conform to KYA Standard or KYA Extended. KYA Basic conformance is insufficient for FEDIS compatibility.

Rationale: KYA Basic leaves the X.509 fields optional, which means the authority reference is not guaranteed to be externally resolvable. KYA Standard makes those fields required, satisfying the authority resolution requirement at the FEDIS output stage.

policy_ref MUST reference an external governance framework or seal-bound policy context, not a field derived from the KYA execution chain.

The seal reference bound to actor.authority_basis MUST be externally resolvable at the time of closure and remain verifiable after the fact. This includes the ability to verify the authority basis using standard tools without reliance on the originating system.

Reference: KYA specification Section 3.5 (Issuer Authority and External Verifiability) and Section 9 (Conformance Levels).

7. FEDIS Compatibility

The portable closure record must be compatible with the FEDIS evidentiary model.

Reference: https://www.certifywebcontent.com/fedis-forensic-evidence-declaration-integrity-statement/

FEDIS defines how the closure record becomes:

-- independently verifiable
-- legally presentable
-- usable in audit, regulatory, and judicial contexts

Important:

FEDIS is not a default system artifact. It is a formal evidentiary output generated through a certification process.

The interface defined in this document ensures that the closure record can be used as a valid input for FEDIS generation.

The KYA passport_certificate + chain export (KYA Section 8, compliance table) constitutes the defined entry point for FEDIS closure record input.

Conclusion

The integration model is based on:

-- RANKIGI / KYA as execution substrate and identity layer
-- EVIDE as responsibility closure layer
-- FEDIS as evidentiary output layer

Two systems

Two anchors

One bridge

This interface defines that bridge.