SCHEMA REFERENCE · PUBLIC EDITION

EVIDE Minimum Intake JSON

Evidentiary Intake Structure and Interoperability Model

Schema v2.1 · Public Specification v2.2 · 14 November 2026

This public edition describes the EVIDE evidentiary intake structure and interoperability model.
Operational access details are available to certified API partners only.

What EVIDE Is

EVIDE (External Evidentiary Deposit) is a structured evidentiary intake layer designed to anchor AI-assisted or human decisions to an independent, cryptographically verifiable record. Each intake transforms a closed decision into a portable evidentiary object that can be validated outside the originating system, without requiring operational access to the source system, its logs, or its infrastructure.

This independence is what allows the evidentiary object to be verified without reliance on the originating system.

> *"EVIDE does not standardise how a decision is made. It standardises the minimum evidentiary object that can survive outside the source system."*

EVIDE does not:

-- verify whether the declared authority was legitimate
-- validate the correctness of the decision
-- introduce admissibility logic
-- replace governance frameworks
-- operate inside the decision pipeline

Boundary Semantics

EVIDE operates at a specific and bounded layer. Understanding what it does and does not do prevents misuse and misinterpretation.

-- EVIDE records closure state, not runtime execution
-- EVIDE anchors the externally visible responsibility state at the moment a decision becomes final
-- EVIDE preserves admissibility visibility without enforcing admissibility
-- EVIDE is evidentiary, not supervisory
-- EVIDE does not replay decisions - it anchors closure objects
-- EVIDE does not verify declared oversight - it makes the declaration persistent, portable, and independently verifiable

Behavioral completion vs. Responsibility closure

A finalized decision without a bound authority represents behavioral completion only - the system produced an outcome, but no accountable actor is formally attached to it.

A finalized decision with a bound authority and declared human oversight represents responsibility closure - the outcome is not only produced, but explicitly attributable.

The Layer 2 → Layer 3 boundary requires responsibility closure, not behavioral completion.

Identity-Bound Access Model

EVIDE access provisioning requires DAPI identity verification. This requirement exists because EVIDE operates as an evidentiary responsibility layer. The intake infrastructure must be able to attribute API activity to a verifiable identity source, even when deposited payloads contain declared or non-verified authority states.

DAPI therefore functions as the identity-bound access layer of the EVIDE intake infrastructure, separating:

-- infrastructure access attribution
-- payload-declared authority
-- evidentiary responsibility visibility

Account binding at intake. On the API intake, authority.dapi_number is required and must match the DAPI identity bound to the API key. Missing value: 422 missing_field; non-matching value: 403 dapi_mismatch; key not bound to a user identity: 403 missing_user_binding. The check binds the deposit to the authenticated account; it does not verify the declared authority, role or powers.

Access provisioning details are confidential and available to certified API partners only. Contact info@certifywebcontent.com to request access.

1. Schema Structure

The EVIDE intake payload is a JSON object submitted via POST. Its decoded form is canonicalized and hashed deterministically at intake (section 4). The current schema version is "2.1".

REQUIRED FIELDS

-- `evide_schema` - must be "2.1", the current and only value accepted by the API. Earlier values ("1.8", "1.9", "2.0") are historical and no longer accepted.
-- `source_system` - name of the originating system
-- `source_reference` - unique decision identifier in the source system
-- `source_timestamp_utc` - ISO 8601 UTC timestamp of decision production
-- `decision.type` - decision category (e.g. candidate_evaluation, risk_classification)
-- `decision.status` - final state, use "finalized"
-- `decision.closure_timestamp_utc` - UTC timestamp of closure
-- `decision.summary` - natural-language description of the decision
-- `authority.id` - authority identifier in the source system
-- `authority.role` - role of the authority
-- `authority.dapi_number` - required on the API intake; must match the DAPI identity bound to the API key used for the request, cross-checked server-side at intake time (errors: see Account binding at intake). The examples in this document use 0000000000 as a placeholder: replace it with the DAPI number of the holder of the API key used for the request; any other value is refused with 403 dapi_mismatch.
-- `intervention.classification_context` - upstream governance structure visibility (v1.7). Must be a non-empty JSON object ({} is refused with 422 missing_field); its sub-fields are optional. The `intervention` object itself is required.

NOTABLE OPTIONAL FIELDS

-- `authority.verification` - declared textual reference, not validated server-side. When present, evidentiary_profile.identity returns "claimed" instead of "declared", without verifying identity, role or powers; separate from the authority.dapi_number check.
-- `intervention.type` - override / approval / rejection / escalation
-- `content_hash` - lets the client declare, before submission, the hash it expects EVIDE to compute over the canonicalized payload. EVIDE does not compare this value against its own computation; the client can independently compare its declared `content_hash.value` with the `intake_hash` returned by EVIDE. Only `SHA-256` is accepted as the algorithm, since that is the only algorithm EVIDE's own intake_hash ever uses.

-- `fedis_requested` - requests evaluation and a separate quote for optional FEDIS forensic certification; it does not generate FEDIS (section 7)

-- `chain.parent_evide_id` - links to a previous intake, forming a verifiable evidence chain. Must reference an existing record in the same evidentiary domain, in an accepted status — validation is strict, with no silent fallback to a new chain

-- `chain.chain_type` - free text describing the continuation (e.g. litigation_hold, internal_investigation). No fixed enum

-- `chain.matter_reference` - optional; if declared on both sides it must match, and is inherited automatically from the parent when omitted

-- `handoff` - boundary state declaration for L3 intake. Required. boundary_readiness is a structured object with status, readiness_gate, visibility_surface, and unresolved_signals. See Section 3.

-- `evidence_references` - array of declared external artifact references (screenshots, video, sensor logs, documents), anchored without EVIDE storing the underlying file. All entry fields are optional (artifact_type, pointer, declared_origin, declared_relationship, declared_description, declared_retention_status). If a hash is declared, hash.algorithm, hash.value, hash_scope and hashed_by are required; EVIDE does not check the format of hash.value. (v2.1)

-- `declarations` - array of declared operational perimeter statements (environment, privileges, purpose, tools, prohibited operations, agent configuration) an agent was authorized within, before it acted (EVIDE ANCHOR). EVIDE preserves the declaration only, never its correctness. (v2.1)

-- `declared_uncertainties` - array of statements of uncertainty made by the depositing party about the decision (each item: statement, required; context, optional; at most 20 items, 1000 and 500 characters). Distinct from the unresolved_signals of a readiness gate; accepted with any readiness status; EVIDE preserves the statements only, does not evaluate them, and they do not enter FCC, DWC or FAC. Declared order is preserved. (v2.1 extension, October 2026)

TOP-LEVEL FIELD WHITELIST

Only the fields listed above are accepted at the top level of the payload. A field the server does not recognize is refused with `422 invalid_field` rather than silently ignored — an unrecognized field would otherwise be canonicalized into the intake hash without ever having been validated. Three fields described in earlier public materials — `human_oversight`, `object_class`, and `created_at_utc` — are accepted by the top-level field whitelist and preserved as part of the intake: included in the canonicalized intake representation, but not independently interpreted by the core engine as oversight, classification, or authority signals. Their presence does not establish the truth or legal significance of the declared information. They remain in the Schema Evolution history below as a record of when each was introduced. Validation runs in three tiers: the top-level whitelist above (tier 1); structural checks — required fields, types, enums, and the rules for content_hash (tier 2); and semantic checks that cross-reference something outside the payload itself — authority.dapi_number against the API key's owning identity, chain.parent_evide_id against the referenced record, and the ESB minimum observation window (tier 3). The two ESB intake fields are validated when the observation window is opened, after the intake has been recorded; inside the ESB buffer object only the declared stabilization source is read, and any other member is not validated, although it remains part of the recorded payload and of its hash.

MINIMAL PAYLOAD EXAMPLE

```
"evide_schema": "2.1",
"source_system": "AquariuOS",
"source_reference": "CDR-2026-00421",
"source_timestamp_utc": "2026-04-07T09:15:00Z",
"decision": {
  "type": "candidate_evaluation",
  "status": "finalized",
  "closure_timestamp_utc": "2026-04-07T09:15:00Z",
  "summary": "AI recommendation overridden by human authority"
},
"authority": {
  "id": "user_87421",
  "role": "HR Reviewer",
  "dapi_number": "0000000000"
},
"intervention": {
```

```
  "classification_context": {
    "threshold_status": "not_defined"
  }
},
"handoff": {
  "boundary_readiness": {
    "status": "candidate",
    "readiness_gate": null,
    "visibility_surface": null,
    "unresolved_signals": []
  },
  "reconstruction_independence": "declared",
  "submission_status": "not_submitted",
  "acceptance_status": "not_claimed"
}
```

Full schema reference: app.certifywebcontent.com/json

2. classification_context (v1.7 + v1.8)

The classification_context object exposes whether the classification act was supported by an upstream taxonomy and threshold structure at the moment of recording. It does not validate the threshold - it makes the presence or absence of such a structure externally visible.

THRESHOLD_STATUS VALUES

- -- `met` - threshold was defined and the decision satisfied it
- -- `not_met` - threshold was defined but not satisfied
- -- `unknown` - threshold may exist but state is not verifiable at deposit time
- -- `not_defined` - no threshold was formally defined for this classification

Governance gap signal: threshold_status: "not_defined" does not indicate an operational error. It signals that at the moment of decision, no formally defined governance structure existed - a governance gap made externally visible, not a system failure.

THRESHOLD_AUTHORITY (V1.8)

Exposes whether the threshold had a structurally attributable source of authority at closure. attribution_status is required when threshold_authority is present.

- -- `attributed` - single identifiable authority defined at closure
- -- `fragmented` - threshold defined across multiple sources, no single attributable authority
- -- `implicit` - threshold present in practice but not formally defined or attributable
- -- `unknown` - threshold authority not verifiable at the moment of closure

3. handoff (v2.0)

The handoff object defines the boundary state of a closure object before EVIDE accepts it for external anchoring. It is required. boundary_readiness is a structured object with four canonical states (promoted from a plain string in v2.0).

- -- `boundary_readiness` - in v2.1 (as since v2.0) a structured object with: status (candidate | verified | verified_partial | unverifiable), readiness_gate.identifier, readiness_gate.scope_reference, visibility_surface, unresolved_signals. Historical note: in v1.9, no longer accepted, it was a string (candidate or verified).
- -- `reconstruction_independence` - must be declared for EVIDE intake.
- -- `submission_status` - must be not_submitted when sent by the upstream system.
- -- `acceptance_status` - must be not_claimed when sent by the upstream system.

Boundary rule: no downstream step should require reconstruction of intent from audit logs, human memory, or system-specific context. The object must be self-contained and independently verifiable before submission.

After a successful intake, EVIDE returns both evidentiary_profile (dimensional read of the object) and a handoff confirmation object with submission_status: "submitted" and acceptance_status: "accepted" — confirming the upstream object crossed the EVIDE intake boundary. EVIDE does not alter the content of the submitted payload: it is stored as decoded and hashed on its canonical form (section 4) - the evidentiary fingerprint reflects what the upstream system declared, not what EVIDE computed after intake.

boundary_readiness and runtime conditions — architectural note

EVIDE anchors closure states, not runtime states. It does not monitor conditions after anchoring and does not introduce runtime governance.

The distinction between “candidate” and “verified” carries a precise architectural meaning: it declares whether the runtime conditions that grounded the closure state were independently confirmed as stable at the moment of boundary crossing.

A system cannot evaluate its own boundary readiness and declare it “verified”. That evaluation must come from a gate that is independent of the system that produced the decision. If no such independent confirmation exists, “candidate” is the correct and complete value — not a fallback.

This distinction is what prevents EVIDE from anchoring a closure state that was already in runtime degradation at the moment of submission.

Note: “verified” does not guarantee correctness of the decision. It confirms only that boundary readiness conditions were independently assessed as stable at the moment of crossing.

EPISTEMIC STABILIZATION BUFFER (ESB) - V2.1

For situations that keep evolving after a decision has been anchored, EVIDE offers an additional intake profile: the Epistemic Stabilization Buffer. The intake is accepted, hashed and finalized exactly as with the standard intake, and, under normal operation, its first Evidentiary Artifact Record (EAR) version is generated immediately afterwards. ESB then opens a declared observation window on that already-final record, during which the source system can record declared observations and, at the end, a declared verdict - stable, unstable or deferred. The window is declared, not enforced, and it never delays or conditions the intake. EVIDE records the verdict as a declaration; it does not establish whether the observed system was stable. Under normal operation, closing the window generates a new EAR version, and earlier versions are not modified. ESB is part of EVIDE Schema v2.1 and does not require a separate payload schema. Operational endpoint details for ESB are available to certified API partners only - see the Identity-Bound Access Model section above.

IDEMPOTENT REQUESTS (IDEMPOTENCY-KEY) - API V2.2

A network timeout does not tell the source system whether EVIDE received a request. Since API v2.2 the intake and ESB endpoints accept an optional Idempotency-Key header: a request repeated with the same key, through the same API key, to the same endpoint and with the same body (compared in canonical form) is executed at most once, and every retry receives the original outcome, marked with the response header Idempotency-Replayed: true. Without the header nothing changes.

-- the key is 1 to 128 characters from A-Z a-z 0-9 . _ : - ; a UUID v4 generated once per event is recommended, and the key must not contain personal data

-- the same key with a different body or endpoint is refused (409 idempotency_key_reused); a request still in progress returns 409 idempotency_request_in_progress with Retry-After

-- every retry repeats the full API key checks: a revoked or expired key never receives a stored response

-- a key is recognised for 7 days from its first request; after that, a request sent with it is processed as new and may create a new deposit

-- if the intake created with a key is deleted by an administrator, retries receive 410 idempotency_record_deleted, without any data of the record: this outcome is final

-- a replayed response does not create a deposit and is not counted as a new one

Idempotency prevents duplicates caused by retrying the same request. It does not merge deposits sent with different keys, with different bodies, without a key or after 7 days, and the key never enters the intake record,

the intake hash, the EAR or a FEDIS. It is an API feature, not a schema change: evide_schema remains "2.1". Full behaviour and error codes: app.certifywebcontent.com/api-reference#idempotency

4. Canonicalization and Hashing

Before computing the SHA-256 intake hash, the server canonicalizes the decoded payload as specified in the technical note Payload Canonicalization (app.certifywebcontent.com/docs/payload-canonicalization/). Changing only JSON whitespace outside string values, or using alternative escape sequences that decode to the same string value, does not change the hash. Object key order is normalized by the canonicalization rules described below; this is not an equivalence for every semantically identical payload.

ALGORITHM

-- Decode the request body: the hash covers the decoded value, not the bytes sent
-- Remove the top-level content_hash node if present (nested ones are kept)
-- Sort object keys recursively at every nesting level, byte-wise (uppercase before lowercase)
-- Sort extensions by value and evidence_references by artifact_type, pointer, declared_origin, hash.value; items with equal sort keys keep their submitted order; swapping them can change the hash if their canonical representations differ
-- Keep the submitted order of every other array
-- Serialize to minified JSON (UTF-8, no escaped slashes) and compute SHA-256 of the resulting string

Key ordering inside classification_context: taxonomy_reference → threshold_authority → threshold_reference → threshold_status. Decoding also normalizes some values ({} becomes [], numbers are re-serialized by PHP under its serialize_precision setting): a client computing content_hash has to reproduce these effects.

Reference value: the intake hash returned in the API response is the fingerprint computed by EVIDE. The source-declared content_hash is optional and EVIDE never compares it with its own computation: the client compares it with the returned intake hash. Detecting a later change requires a reference digest kept separately and reliably; if content and stored digest are altered together, an internal comparison can still match.

4b. Evidentiary Object Profile

The API response (v2.0) includes evidentiary_profile — a server-computed object with profile_version: "1.1". It is not part of the submitted payload and is not included in the canonical hash.

EVIDE does not reduce an evidentiary object to a single validity state. It exposes the known, partial, degraded, or unverifiable dimensions of that object at the moment of deposit.

Dimension	Question	Source field	Possible states
`identity`	Who is linked to the record?	`authority.verification presence`	claimed / declared
`authority`	Who assumes decision responsibility?	`authority.id + authority.role`	declared / null
`classification`	Is the classification stable?	`intervention.classification_status`	stable / provisional / contested
`threshold`	Was a	`classification_context.threshold_sta`	met / not_met / unknown /

	decision threshold defined?	tus	not_defined
threshold_authority	Who owns the threshold?	threshold_authority.attribution_status	attributed / fragmented / implicit / unknown
boundary_readiness	Was the object stable at crossing?	handoff.boundary_readiness.status	candidate / verified / verified_partial / unverifiable
runtime_visibility	Was the runtime condition observable?	handoff.boundary_readiness.visibility_surface, mapped: declared_complete → confirmed, partial → partial, insufficient → unverifiable	confirmed / partial / unverifiable / null
trace_reference	Can the upstream trace be located?	intervention.trace.access (value declared by the source system, reproduced as-is)	As declared in intervention.trace.access, e.g. available / restricted / unavailable; null if absent. Free text, not a server-enforced list, and not proof that the trace is accessible.
continuity	Does the declared observational visibility structurally support the classification?	inferred: classification x runtime_visibility (Forensic Cross-Check)	{mode: inferred, state: stable / degraded / broken / unknown}
dwc	Decision Wave Compression: is oversight throughput exceeded?	inferred (DWC)	{mode: inferred, state: not_detected|detected|critical|unknown}
fac	Formal Accountability Collapse: is accountability attributable?	inferred (FAC)	{mode: inferred, state: not_detected|detected|critical|unknown}

5. Schema Evolution

EVIDE follows a strict backward-compatible evolution model. Each version adds optional fields or response attributes without breaking existing integrations.

Version	Addition
1.0	Base schema - decision, authority, intervention, human_oversight, chain
1.1	created_at_utc, object_class, content_hash
1.2	intervention.rationale_type

1.3	intervention.taxonomy_version - enables classification replay in audit
1.4	fedis_requested - request for evaluation and quote for optional FEDIS certification (no automatic generation)
1.5	authority_verification_status (response) - claimed vs declared. Consolidated into evidentiary_profile.identity in v2.0.
1.6	intervention.classification_status - stable / provisional / contested
1.7	intervention.classification_context - upstream governance structure visibility
1.8	classification_context.threshold_authority - threshold ownership attributability
1.9	handoff - boundary-ready schema, universal handoff contract for L3 intake. Required for evide_schema 1.9. boundary_readiness as string: candidate | verified.
2.0	handoff.boundary_readiness promoted from string to structured object. Four canonical states: candidate, verified, verified_partial, unverifiable. Adds readiness_gate (identifier + scope_reference), visibility_surface, unresolved_signals. intervention.classification_context required. decision.status must be "finalized". Response includes evidentiary_profile (server-computed, profile_version 1.1) with eleven evidentiary dimensions. Dimension 9 is inferred via Forensic Cross-Check (classification x runtime_visibility) and expresses the structural coherence of a single closure — exposed under the historical API field name continuity, unrelated to Evidentiary Continuity across a chain.
2.1	extensions and evidence_references arrays - declaring external artifacts (screenshots, video, sensor logs, documents) anchored without EVIDE storing the underlying file. Any integrity hash is declared by the submitter and never independently verified by EVIDE. Adds chain.matter_reference and strict server-side validation of chain continuations. The extensions array is opt-in in both directions: an optional block present in the payload but not listed there is rejected, and an extension listed there with no corresponding content is rejected as well. The additions are non-breaking at the payload structure level. Also adds the declarations array (EVIDE ANCHOR) for declaring the operational perimeter an agent was authorized within, before it acted - same opt-in extensions mechanism, same non-verification principle. In October 2026 adds the declared_uncertainties extension: statements of uncertainty by the depositing party, distinct from the unresolved signals of a readiness gate - same opt-in extensions mechanism, same non-verification principle, evide_schema unchanged. "2.1" is the current and only evide_schema value accepted by the API.

API v2.2 (14 November 2026) adds the optional Idempotency-Key header (see Idempotent requests, after Section 3). It is an API change, not a schema change: evide_schema remains "2.1" and the payload, canonicalization and intake hash are unchanged.

6. Interoperability

EVIDE is designed to receive structured closure objects from any system - AI pipelines, workflow engines, governance platforms, runtime agent systems - without requiring integration inside the decision pipeline.

The minimal boundary requirements for the source system are:

-- an explicit closure state (not only behavioral completion)
-- an identified authority bound to that closure
-- an explicit responsibility declaration
-- a stable reference to the upstream trace

-- a coherent classification context (even if initially internal)

SOURCE IDENTITY FIELD MAPPING

-- `evidence_id` → source_reference (primary object identity)
-- `event_id` → intervention.trace.reference (runtime event identity)
-- `trace_reference` → intervention.trace.reference (upstream trace continuity)

Reference implementation: RANKIGI × EVIDE Interface Mapping (v0.4.1) - app.certifywebcontent.com/docs/rankigi-evide-interface/

7. FEDIS Compatibility

FEDIS (Forensic Evidence Declaration & Integrity Statement) is a forensic document that references an EVIDE intake record and is intended to support its later examination. It is not a default system artifact but the optional output of a separate certification process. Setting fedis_requested: true submits a request for evaluation and a separate quote. It does not generate or issue FEDIS. The platform generates the document only after the certifier explicitly initiates generation.

A FEDIS-ready EVIDE record must include:

-- cryptographically fixed closure state (SHA-256 intake hash)
-- independent UTC intake timestamp
-- declared authority bound to the closure
-- classification context sufficient for external interpretation
-- stable upstream trace reference

On the certifier's command, the platform prepares the FEDIS document, which states the SHA-256 fingerprints of the Canonical Intake Record and of the frozen Evidentiary Artifact Record (EAR) and documents the chain of custody. The certifier signs it with a qualified electronic signature (eIDAS Art. 25) and, as a distinct step, it receives a qualified electronic time stamp (eIDAS Art. 41) from a third-party qualified trust service provider. The time stamp attests the moment at which it is applied to the FEDIS document: it does not make the intake timestamp qualified, does not date the original event and does not prove that the content is true. FEDIS is intended to support examination in audit, regulatory and judicial contexts; it does not guarantee admissibility or evidentiary weight.

Architecture in three layers

-- Source system - produces the decision and surfaces the closure object
-- EVIDE - anchors the closure object externally with integrity and timestamp
-- FEDIS - documents the anchored record in a signed forensic statement intended to support later examination

One closure. Three layers. One independent record.

For operational access, contact
Dr. Emanuel Celano

Founder & Chief Architect at EVIDE
info@certifywebcontent.com