

KP:1 — Knowledge Pack Format Specification

Timothy Kompanchenko

ORCID: <https://orcid.org/0009-0007-5355-2764>

v0.8.5-preview · Released Editor's Draft · 2026-10-13

Version DOI: <https://doi.org/10.5281/zenodo.23329922>

Concept DOI: <https://doi.org/10.5281/zenodo.19445262>

Abstract

KP:1 is an open, text-native format for representing epistemic state: assertions and explanations with their context, confidence, evidence, relationships and history. It helps people and AI carry articulated understanding between tasks and tools through an account that others can inspect, challenge, correct and continue. The specification defines a directory and claim contract, a JSON Schema and PEG grammar, and topic companions for evidence, history, views, composition, multilingual sources and archives. Source accounts remain distinct from adopted assessments; uncertainty and contrary evidence remain explicit. This released editor's draft includes reference validation and experimental graph compilation, with their coverage and limits documented separately. It does not certify source truth, calibrated confidence, recipient understanding or permanent custody, and it does not promise stable V1 compatibility.

Reading this edition

This PDF is a reading projection of all 24 specification documents, followed by the published PEG grammar and JSON Schemas. The released repository and its original source files remain authoritative. CORE takes precedence over overlapping legacy prose in SPEC; topic companions govern their declared surfaces. Informative, experimental and deferred material retains its stated status. Publication does not establish stable V1 compatibility, whole-format implementation coverage or standards-body ratification.

The reference fixture/example suite contains 29 cases. Twenty-two reference contract tests and 50 experimental graph compiler tests exercise bounded behavior. These checks do not certify source truth, confidence calibration, application workflows or archive cryptographic verification. See the Capabilities and Compatibility chapter for the detailed boundaries.

Specification text: CC BY 4.0. Code, schemas and examples: Apache License 2.0. The editor and acknowledgment of AI drafting assistance are recorded in `ACKNOWLEDGMENTS.md`.

Source edition: <https://github.com/tymofiy/kp/blob/v0.8.5-preview/spec/README.md>

Contents

KP:1 — Knowledge Pack Format Specification	1
KP:1 Core Specification	3
Knowledge Pack Specification — Editor's Draft	17
Authority, Capabilities and Compatibility	49
Authoring — Knowledge Pack Companion Spec	51
Voice Views — Knowledge Pack Companion Spec	62
Composition Specification	70
Lifecycle, Archival & Visibility Specification	81
Multilingual Support — Knowledge Pack Companion Spec	93
Pack Organization — Knowledge Pack Companion Spec	101
Cross-Pack Consistency — Knowledge Pack Companion Spec	104
Linguistic Conventions — Knowledge Pack Companion Spec	109
Storage Independence — Knowledge Pack Companion Spec	115
Bundle Format — Knowledge Pack Companion Spec	122
Archive Format — Knowledge Pack Companion Spec	125
Definition & Policy Kinds — Knowledge Pack Companion Spec	137
Extensions — Producer-Defined Manifest Metadata	148
Notes Metadata — Knowledge Pack Companion Spec	159
Self-Driving Presentation Playback — Knowledge Pack Companion Spec	162
Reconciliation — Knowledge Pack Companion Spec	168
KP:1 — Rationale and Positioning	170
KP:1 Mapping to RDF/JSON-LD/PROV	177
Independence — Knowledge Pack Companion Spec	185
KP:1 Specification	187
Knowledge Pack Specification — Changelog	189
kp-claims.peg	206
kp-pack.schema.json	212
kp-composition.schema.json	220
kp-signatures.schema.json	228
References and source documents	230

KP:1 Core Specification

Status: Released Editor's Draft — v0.8.5-preview **Latest citable release:** KP:1 — Knowledge Pack Format Specification (v0.8.5-preview, 2026-10-13) **Editor:** Timothy Kompanchenko **Draft updated:** 2026-10-12 **Derived from:** SPEC.md, conformance/grammar/kp-claims.peg, conformance/grammar/kp-pack.schema.json **Lane:** Implementer surface — see [README.md](#) for the three-lane structure (CORE = implementer, SPEC = full normative + rationale, companions = topic-authoritative for their domains).

If you are authoring a pack (rather than implementing a parser/validator), read [AUTHORING.md](#) first. CORE.md tells you what a syntactically valid pack looks like; AUTHORING.md tells you how to make epistemically sound choices when authoring one.

1. Introduction

KP:1 is a plain-text format for packaging epistemic state — claims with confidence, evidence, relationships, and contradictions. This document specifies everything required to **implement a conformant KP:1 parser and validator**. It is the normative core of the specification.

Scope

This document covers:

- Pack directory structure and required files
- PACK.yaml manifest schema
- claims.md syntax (Rosetta header, frontmatter, claims in dense and verbose forms)
- evidence.md structure
- Views contract
- Confidence model
- Relation types
- Validation rules and semantic constraints
- Conformance levels

This document does **not** cover: producer-side decision rubrics ([AUTHORING.md](#)), design-principles rationale and standards-comparison ([RATIONALE.md](#)), voice surfaces ([VOICE.md](#)), composition/ meetings ([COMPOSITION.md](#)), lifecycle management ([LIFECYCLE.md](#)), multilingual support ([MULTILINGUAL.md](#)), pack organization patterns ([ORGANIZATION.md](#)), consistency patrol ([CONSISTENCY.md](#)), naming conventions ([CONVENTIONS.md](#)), storage formats ([STORAGE.md](#)), bundling ([BUNDLE.md](#)), archive format and integrity ([ARCHIVE.md](#)), definition/policy document kinds ([DEFINITIONS.md](#)), AI note-taking ([NOTES.md](#)), the producer-defined `extensions.*` catalogue ([EXTENSIONS.md](#)), self-driving voice playback ([PLAYBACK.md](#), experimental), cross-pack reconciliation ([RECONCILIATION.md](#), stub), or RDF/JSON-LD/PROV-O mapping ([MAPPING.md](#), informative). These are specified in companion documents within `spec/`. Cold AI agents should start from [AGENTS.md](#) at the repo root for a task-indexed reading map.

Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Term	Definition
Pack	A directory of plain-text files representing epistemic state
Claim	A belief with confidence, evidence, and relationships
Evidence	A backing source referenced by claims
View	A pre-rendered markdown document derived from claims, for human consumption
Rosetta header	Self-describing HTML comment at the top of claims.md
Frontmatter	Compact metadata block between --- fences

2. Pack Structure

A Knowledge Pack is a directory. The `.kpack/` suffix is **RECOMMENDED** but not required.

```
{name}.kpack/
├── PACK.yaml           # REQUIRED – manifest
├── claims.md           # REQUIRED – current claims
├── evidence.md         # RECOMMENDED – backing sources
├── history.md          # OPTIONAL – superseded/retracted claims
├── entities.md         # DEPRECATED (since v0.7.7) – use extensions.entities; see EXTENSIONS.md
├── validation.yaml     # OPTIONAL – test questions (schema in SPEC.md §10)
├── signatures.yaml     # OPTIONAL – cryptographic integrity (see ARCHIVE.md)
├── views/
│   └── overview.md     # OPTIONAL – pre-rendered display content
```

File requirements depend on the manifest's `kind` (default `claim`):

Kind	Required files	Additional content
<code>claim</code>	<code>PACK.yaml</code> , <code>claims.md</code>	Evidence for every cited evidence ID
<code>definition</code>	<code>PACK.yaml</code> , at least one <code>definitions/*.yaml</code>	Optional claims, evidence and views
<code>policy</code>	<code>PACK.yaml</code> , at least one <code>policies/*.yaml</code>	Optional claims, evidence and views
<code>mixed</code>	<code>PACK.yaml</code> , <code>claims.md</code> , at least one definition or policy document	Evidence for every cited evidence ID

Definition and policy document formats are defined in [DEFINITIONS.md](#). They do not require probabilistic claims merely to represent deterministic rules. Whenever `claims.md` is present, its ordinary header, frontmatter and claim rules apply. `evidence.md` is optional as a file, but SC-03 still requires every active or historical evidence reference to resolve.

Composition-pack exception. The presence of `composition.yaml` requires `claims.md` and validation of the composition schema. The claims document **MAY** be narrative-only, retaining the Rosetta header and compact frontmatter. A composition **MAY** omit `evidence.md` when it has no evidence references. See [COMPOSITION.md](#). This exception does not relax metadata or reference rules for any claim bullets it does contain.

The default file names above are the interoperable directory surface. The schema's optional `files` object is reserved for application mappings; schema acceptance alone does not define override behavior. Producers targeting the generic reader **MUST** retain the canonical files. A profile that uses alternate carriers must define its mapping and declare the required reader capability.

3. PACK.yaml Manifest

`PACK.yaml` is a YAML file declaring pack identity and configuration. The normative schema is `conformance/grammar/kp-pack.schema.json` (JSON Schema 2020-12).

Required Fields

Field	Type	Pattern	Description
<code>name</code>	string	<code>^[a-z0-9][a-z0-9-]*\$</code>	Pack identifier, kebab-case
<code>version</code>	string	<code>^YYYY.MM.DD(-revN)?\$</code>	CalVer snapshot date
<code>domain</code>	string	<code>^[a-z0-9][a-z0-9-]*(/[a-z0-9][a-z0-9-]*)*\$</code>	Hierarchical domain path
<code>author</code>	string	non-empty	Maintainer name

Core Optional Fields

Field	Type	Description
<code>kind</code>	enum	<code>claim</code> (default), <code>definition</code> , <code>policy</code> , <code>mixed</code>
<code>description</code>	string	Human-readable summary

Field	Type	Description
spec_uri	URL	Onboarding pointer to the KP:1 specification this pack conforms to. Informational, not a runtime parse dependency. Default published location: https://github.com/tymofiy/kp . See "Spec Onboarding Pointer" below.
spec_version	string	Version label of the KP:1 spec this pack conforms to (typically a tag, e.g., v0.8.5-preview). Pairs with spec_uri to pin against a specific revision.
confidence	object	scale (string, required within object), normalize (boolean), labels (object, for custom scales)
freshness	date	Last substantive review date
license	string	Reuse terms (e.g., CC-BY-4.0)
sensitivity	enum	public, internal, confidential, restricted
visibility	enum	private, shared, public
channels	array	Distribution routing strings
display	object	short_title, abbreviation, tagline, hook
provenance	object	author (required), role (required), reviewed_by, review_date, signed
evidence_basis	string	Narrative description of the evidence foundation underpinning this pack
tags	array	Topical tags for discovery and classification. Strict kebab-case (<code>^[a-z0-9]+(-[a-z0-9]+)*\$</code>), unique
views	array	View declarations with name, file, purpose, display_as (all required), hint (optional). Voice views add voice (boolean), duration (string, e.g. ~90 seconds), pace (enum: brisk, measured, deliberate). When voice is true, duration and pace are REQUIRED.
tier	enum	hub, detail, standalone
extensions	object	Extension lane for experimental or implementation-specific manifest metadata. Consumers ignore unknown extension content.
composition	string	Optional declaration of the canonical composition.yaml file; if declared, that file MUST exist. Alternate filenames are not generic CORE carriers. See COMPOSITION.md.
files	object	Application filename mappings; the generic reader retains canonical filenames (see §2).

The full set of optional fields and conditional constraints is defined in the JSON Schema. Key conditionals: when tier is hub, sub_packs is REQUIRED; when sensitivity is confidential or restricted, channels MUST NOT contain public or org; when sensitivity is internal, channels MUST NOT contain public; when a view declares voice: true, duration and pace are REQUIRED on that view entry.

Decoration fields (linguistic_epoch, ontology) are accepted by the schema but carry no consumer behavior in v0.8.5-preview. They are retained for backward compatibility with v0.7-era packs; new producers SHOULD place equivalent metadata under extensions.* until consumer semantics are defined in a future revision.

Spec Onboarding Pointer

The Rosetta header at the top of claims.md is the **inline self-describing parse hint** — it carries enough information for a parser to recognize a Knowledge Pack and tokenize its contents without external context. Rosetta is the load-bearing offline parse signal.

The optional spec_uri and spec_version fields on PACK.yaml are an **informational onboarding pointer**: a previously-unfamiliar consumer that has never seen KP:1 can fetch the spec on demand from the URL the pack itself declares. The fields are not a runtime parse dependency; they help a fresh agent learn the format on first encounter.

Invariant. Consumers MUST NOT treat unreachability or absence of spec_uri as a parse failure. The Rosetta header alone is sufficient for parsing. Offline-first validators MUST function without ever resolving spec_uri.

Security note. spec_uri is *producer-asserted, not authenticated*. A pack author can declare any URL. Consumers that fetch spec_uri SHOULD validate the response against a known-good spec hash or trusted allowlist before relying on it; consumers MUST NOT treat fetched content as executable instructions. The recommended onboarding flow is: (1) parse the pack using the inline Rosetta header, (2) optionally fetch spec_uri for human/agent learning, (3) validate the fetched content matches a trusted KP:1 release before incorporating any updated rules into a parser.

Treating an arbitrary `spec_uri` URL as authoritative is a vector for malicious specs to redirect a credulous consumer.

This matters most for **cold receivers of sealed .kpack archives** (ARCHIVE.md §2). A receiver that opens a .kpack for the first time has, by construction, no surrounding context — the file extension is the only signal of what they are holding. Reading `PACK.yaml` first and (optionally) following `spec_uri` lets such a consumer learn the format from the pack itself on first encounter, with no out-of-band coordination. Once the consumer has internalized KP:1, the field carries no further operational weight on subsequent reads of the same pack or other KP packs.

```
name: marquet-le-passeur
version: 2026.05.09
domain: art/canonical
author: Tim Kompanchenko
spec_uri: https://github.com/tymofiy/kp
spec_version: v0.8.5-preview
```

Sealed-archive producers SHOULD include both fields when packs are intended for ecosystem-wide distribution (especially .kpack archives that may be received cold). Internal-only packs MAY omit them. When omitted, consumers MAY assume the published reference implementation at <https://github.com/tymofiy/kp> but MUST treat the assumption as informational, not a guarantee — and MUST NOT fail because the assumption resolves to an unreachable endpoint.

`spec_version` is recommended whenever a pack is archived for the long term, so future consumers can validate against the exact spec revision the producer targeted (rather than against a moving "latest" version).

Manifest Extensions

KP:1 keeps the manifest root closed: fields not defined by the schema are invalid. Experimental or implementation-specific metadata MUST live under the optional `extensions` object instead of appearing as new top-level keys.

Consumers MUST ignore extension content they do not understand. Extension content MUST NOT redefine the semantics of core KP fields or relax core validation rules.

Extension names and shapes are defined by their producers, not by KP:1. The `ai_brief` payload below is one such producer-defined example.

Example:

```
extensions:
  ai_brief:
    version: 1
    verdict: acceptable
    headline: "Strong base attribution, but provenance gap remains"
```

Example

```
name: solar-energy-market
version: 2026.03.18
domain: energy/market-analysis
author: Jane Chen
confidence:
  scale: sherman_kent
  normalize: true
extensions:
  ai_brief:
    version: 1
    verdict: acceptable
    headline: "Strong base attribution, but provenance gap remains"
```

4. claims.md — Rosetta Header

Every claims.md MUST begin with a self-describing HTML comment before the frontmatter. This is the **Rosetta header**.

Format: `<!-- KP:` followed by the spec version number, legend text, and `-->`. Parsers MUST check the spec version. Parsers MAY ignore the legend text.

```
<!-- KP:1 — Knowledge Pack Format
Claims: - [ID] assertion {confidence|type|evidence|date|depth|nature} context
        Positions 5-6 optional. Verbose named-field syntax also accepted.
Types: o=observed r=reported c=computed i=inferred
Depth: assumed | investigated | exhaustive (optional, position 5)
Nature: judgment | prediction | meta (optional, position 6; omitted=factual)
Relations: →supports @contradicts @!error @~tension ←requires ~refines @supersedes @see_also
Files: evidence.md=sources history.md=past entities.md=graph
-->
```

The Rosetta header is part of the **constitutional core** (see Section 14) and is frozen across all spec versions.

5. claims.md — Frontmatter and Heading

Frontmatter

The frontmatter is a **KP-specific compact format**, NOT arbitrary YAML (AR-08). It sits between `---` fences and has exactly two lines with fixed field order.

```
---
pack: {name} | v: {CalVer} | domain: {domain}
confidence: {scale} | normalized
---
```

Parsers MUST NOT attempt to parse this as standard YAML.

Document Heading

An H1 heading with an optional entity annotation in brackets, followed by an optional blockquote description.

```
# Solar Energy Market [market-analysis|SEM,solar]

> Utility-scale solar market trends, cost trajectories, and regional adoption patterns.
```

Entity annotation format (AR-09): `[entity_type]` OR `[entity_type|alias1,alias2,...]`. Entity type uses `[a-z0-9-]+` (lowercase alphanumeric and hyphens only). Aliases are comma-separated, each `[A-Za-z0-9-_]+` (mixed case, underscores permitted).

Sections

Claims are grouped under `##` headings. Section headings are freeform text. A section MAY open with a blockquote description before its claims, mirroring the document-level description (AR-17). A claims document MUST retain at least one section, but a section MAY contain zero claims. This represents an empty current state, including withdrawal of the final active claim with its original record retained in history, without inventing a replacement assertion. Apart from the optional blockquote and blank lines, no freeform prose or subheadings are permitted between a section heading and its claims.

6. Claim Syntax

Each claim is a markdown list item. A claim uses EITHER dense OR verbose form for its metadata — never both (AR-13).

6.1 Dense Form

```
- [C001] Claim headline
  {confidence|type|evidence_ids|since_date|depth|nature} Optional trailing prose.
  Additional prose continues here. →C002, @~C003
```

Claim ID: [`c` + one or more digits + optional -`v` version suffix + `]`. Bold wrapping (**[C001]**) is optional syntactic sugar with no semantic meaning (AR-03). Zero-padding is conventional but not required (AR-01). IDs need not be sequential (AR-12).

One identifier per claim. A claim's [`c...`] id is both its reference target (for relations and evidence cross-references) and its human-readable label — there is no second, separate id in the file form (see *Single claim identity (normative)* below for the normative rule). Any globally-unique `claim_id` an operational index keeps is assigned by the indexing process, not the pack author, and is never serialized into `claims.md` (see [STORAGE.md](#)). Rationale: [decisions/2026-06-22-claim-identity.md](#).

Metadata block: Curly braces with pipe-delimited positional fields:

Position	Required	Format	Values
1	Yes	0.0–1.0	Confidence (normalized float)
2	Yes	single letter	o (observed), r (reported), c (computed), i (inferred)
3	Yes	comma-separated	Evidence reference IDs (E + digits)
4	Yes	YYYY-MM-DD	Date established
5	No	keyword	assumed, investigated, or exhaustive
6	No	keyword	judgment, prediction, or meta

Positions 1–4 are REQUIRED. Positions 5–6 are OPTIONAL. Empty interior slots are valid: `{0.85|r|E020|2026-03-10||prediction}`.

Single claim identity (normative). The bracket id (`[C001]`) is a claim's sole serialized identity — relations, history entries, and cross-pack references (AR-16) all resolve to it. The dense metadata block carries EXACTLY the positional fields defined above and no others: a producer MUST NOT append fields beyond position 6, and a conforming reader MUST treat any non-empty token past them as a malformed claim (rejecting it) rather than silently discarding it. An opaque or storage-layer identifier a system keeps for a claim (e.g. a database key) is non-wire — it is never emitted into, nor reconstructed from, the serialized claim; preserve it out-of-band if it must survive a round-trip. Storage-layer identity is a non-normative concern (see [STORAGE.md](#)).

Context prose may follow the metadata on the same line (trailing prose) or on subsequent indented continuation lines (2+ spaces, AR-11).

Relations MAY appear on the metadata line after the closing `}` (separated from it by a space) — either starting the trailing text or following the context prose — and/or on subsequent continuation lines. In every position, relations form a comma-separated list that ends the line, and a relation symbol is parsed as a relation only when immediately followed by a valid claim-ID reference (`c` + digits or a cross-pack reference, per AR-10); other occurrences are prose. Relations use symbol syntax:

```
→C002, @~C003
```

Relations may appear on their own line or trailing after prose (AR-04).

6.2 Verbose Form

```
- [C030] Supply chain concentration
  `confidence: 0.90 | type: inferred | evidence: E001, E030 | since: 2026-03-01 | depth:
  exhaustive | nature: meta`
  Context prose lines.
  `relations: requires C020`
```

Metadata in a backtick-delimited line with named fields separated by `|`:

Field	Required	Values
confidence:	Yes	0.0–1.0

Field	Required	Values
type:	Yes	observed, reported, computed, inferred
evidence:	Yes	comma-separated E + digits
since:	Yes	YYYY-MM-DD
depth:	No	assumed, investigated, exhaustive
nature:	No	judgment, prediction, meta

Required fields **MUST** occur exactly once; optional fields **MUST** occur at most once. Field order is not significant. Unknown fields and duplicate fields **MUST** be rejected, rather than interpreted by first-value or last-value precedence.

Relations in a separate backtick-delimited line using verbose names:

```
`relations: supports C002, contradicts:tension C003`
```

6.3 Complete Dense Example

```
- [C001] Cost decline is structural, not cyclical – driven by manufacturing scale
  {0.95|i|E001,E002|2026-03-01|exhaustive|judgment} 10/10 analyses converged.
  Learning curve (22% cost reduction per doubling) has held for 40 years. →C002, Ⓢ~C003
```

6.4 Complete Verbose Example

```
- **[C030]** Supply chain concentration in China is the highest systemic risk
  `confidence: 0.90 | type: inferred | evidence: E001, E030 | since: 2026-03-01 | depth:
  exhaustive | nature: meta`
  80% of polysilicon, 95% of wafers. Tariff/geopolitical disruption could
  create 12-18 month supply gaps.
  `relations: requires C020`
```

7. Claim Types

Code	Verbose	Meaning
o	observed	Directly verified from primary source
r	reported	Stated by a credible source, not independently verified
c	computed	Output of automated analysis
i	inferred	Derived from other claims via reasoning

Verbose type names **MUST** map to their single-letter equivalents (SC-11).

8. Confidence Model

All confidence values are **normalized floats from 0.0 to 1.0** (SC-01). PACK.yaml declares the native scale via `confidence.scale`.

Built-In Scales

Sherman Kent (`sherman_kent`):

Normalized	Label
0.99+	Certain
0.93–0.99	Almost certain
0.87–0.93	Highly likely
0.75–0.87	Likely
0.63–0.75	Probable
0.37–0.63	Chances about even
0.25–0.37	Probably not
0.13–0.25	Unlikely

Normalized	Label
0.07–0.13	Highly unlikely
0.01–0.07	Almost certainly not
<0.01	Impossible

Simple (simple):

Normalized	Label
0.90+	Verified
0.70–0.89	High confidence
0.50–0.69	Moderate confidence
0.30–0.49	Low confidence
<0.30	Unverified

Custom scales use `confidence.scale: custom` with a `confidence.labels` mapping in `PACK.yaml`.

The numeric confidence is authoritative. These label tables are illustrative display bands, not a deterministic interval-selection algorithm: shared endpoints and values between printed two-decimal bands have no universal label assignment. A profile requiring consistent labels **MUST** declare its thresholds and endpoint policy; labels never replace the stored number.

Confidence records the producer's assessment. Decimal precision and labels such as "Verified" do not establish that a verification procedure occurred or that the values are empirically calibrated. Explaining the evidential basis and method helps consumers assess what the value represents; see [AUTHORING.md §5](#). This clarification does not change the wire value or scale rules.

Investigation Depth (Position 5)

Value	Meaning
assumed	Minimal investigation, taken at face value
investigated	Multiple sources consulted and cross-referenced
exhaustive	Thorough investigation; remaining uncertainty is genuine

When omitted, depth is unspecified.

Claim Nature (Position 6)

Value	Meaning
judgment	Interpretive conclusion drawn from evidence
prediction	Forward-looking claim with a resolution horizon
meta	Claim about the state of knowledge itself

When omitted, the claim is a factual assertion (the default).

9. Relations

Relation Symbols (Dense Form)

Symbol	Name	Meaning
→	supports	Provides evidence for another claim
⊗	contradicts	Conflicts with another claim (unqualified)
⊗!	contradicts:error	One claim is wrong — prioritize resolution
⊗~	contradicts:tension	Both informative — disagreement is knowledge
←	requires	Depends on another claim being true
~	refines	Adds detail to another claim
⊃	supersedes	Replaces a prior claim
↔	see_also	Cross-reference (cross-pack: pack_name#section)

Multi-character symbols **MUST** be matched before single-character symbols in parsing (ordered choice: `⊗~`, `⊗!`, `⊗` before `~`).

Tilde ~ is parsed as a relation ONLY when immediately followed by a valid claim ID reference — either c + digits (local) or a cross-pack reference (pack_name#section). Otherwise it is prose (AR-10).

Verbose Relation Names

The verbose relation vocabulary is a **closed enum** (AR-05):

supports, contradicts, contradicts:error, contradicts:tension, requires, refines, supersedes, see_also

Cross-Pack References

Format: pack_name#section_ref (AR-16). Pack name follows [a-z0-9-]+. Section ref is freeform text until the next delimiter (comma, space, newline, or end of line).

10. Evidence Document (evidence.md)

Evidence entries are referenced by ID from claims. Two formats are valid (AR-07); implementations MUST parse both.

Evidence entries begin with an H2 heading containing the evidence ID, optionally followed by – or -- and a title:

```
## E001
## E001 – Interview Notes
```

Blockquote Format

Field names MUST be bold-wrapped (**field:**) and separated by |:

```
## E001
> **type:** multi_source_synthesis | **captured:** 2026-03-01
> **source:** Internal analysis – 10 independent assessments

Description prose.
```

List Format

Field names are plain (not bold), each on its own - line:

```
## E001 – Interview Notes
- Type: multi_source_synthesis
- Captured: 2026-03-01
- Source: Internal analysis – 10 independent assessments

Description prose.
```

Evidence Fields

Field	Required	Description
type	Yes	Source type (open vocabulary — unknown types MUST be accepted, AR-06)
captured	Yes	ISO date when evidence was obtained
source	Yes	URI, file path, or description of the source

Optional fields:

- **reliability** — NATO Admiralty source reliability tier. Single uppercase letter A–F. A = completely reliable, F = reliability cannot be judged.
- **credibility** — NATO Admiralty information credibility for this specific fact. Single digit 1–6. 1 = confirmed by other independent sources, 6 = credibility cannot be judged. Orthogonal to reliability: an A-tier source can carry a 6-credibility statement when uncorroborated.
- **kp-content-encoding** — registered reversible encoding for record-shaped Markdown in an evidence body. The only KP:1 value is markdown-backslash-v1; see “Record-safe encoded bodies” below.
- **SHA-256 / sha-256**, **Notes / notes**, **Prepared / prepared**.

Required evidence fields **MUST** be nonempty and occur exactly once. Other metadata fields **MUST NOT** repeat. Each field value occupies one physical line; indented or folded metadata continuations are not defined. `captured` **MUST** be a valid calendar date in `YYYY-MM-DD` form. The initial metadata block is separated from body content by one blank line; additional blank lines belong to the body. Once body content begins, field-shaped source quotations are body text, not metadata.

Field names are case-insensitive in validation (AR-07), including `kp-content-encoding`; its encoding value is case-sensitive. The combined `reliability + credibility` pair (e.g. `A1`, `B2`, `F6`) is the standard intelligence-tradecraft Admiralty grade and renderers **MAY** surface it as such.

Evidence IDs use the pattern `E` + one or more digits (AR-02). IDs are stable — once assigned, an ID always refers to the same source material.

Record-Safe Encoded Bodies

An evidence excerpt can contain Markdown that looks like KP structure — for example, a source-local `## E001` heading or `- [E001]` bullet. Emitting those lines verbatim makes record boundaries ambiguous. A producer **MAY** make the body reversible and record-safe by placing this exact blockquote metadata line after the ordinary evidence fields and immediately before the body:

```
> **kp-content-encoding:** markdown-backslash-v1
```

Producers **MUST** use the canonical lowercase field spelling shown above. Consumers match the field name case-insensitively per AR-07, but **MUST** compare `markdown-backslash-v1` case-sensitively.

For `markdown-backslash-v1`, the producer **MUST** preserve leading spaces and tabs, then prefix one backslash after that indentation when the remaining logical body line:

1. begins with a backslash;
2. equals the encoding marker line above;
3. is exactly `##`, or begins with `##` followed by whitespace; or
4. begins with `-`, then one or more whitespace characters, then `[E + one or more digits + ]`, followed by whitespace or end of line.

All other body lines remain byte-for-byte unchanged. Escaping an existing leading backslash makes the transform reversible rather than treating source-authored escapes as encoder output.

Consumers **MUST** segment evidence records on the encoded text before decoding. Any unescaped, top-level H2 terminates the current evidence record; only a conforming `## E + digits` heading begins another evidence record. Escaped headings and evidence-shaped bullets remain body content. A consumer that exposes the logical body **MUST** remove exactly one post-indentation backslash when the remainder matches one of the four encoder cases above, and **MUST** leave every other leading backslash intact. The marker is metadata, not user-facing body content.

An implementation that does not recognize the declared value **MUST** preserve the encoded body as-is, **MUST NOT** reinterpret escaped lines as structural records, and **SHOULD** warn that logical-body decoding was skipped. Producers **SHOULD NOT** emit unregistered `kp-content-encoding` values (AR-19).

11. Views (views/*.md)

Views are pre-rendered GFM (GitHub-Flavored Markdown) documents for human consumption. Views are **OPTIONAL**.

Rules

1. Views introduce no knowledge absent from the canonical primary files for the pack's kind. In a claim-bearing pack, a corresponding claim **SHOULD** exist for each stated fact. Definition and policy views derive from their canonical definition/policy documents and any claims present. This is editorial discipline — the conformance runner does not detect unsupported view assertions. Authors are responsible; see [AUTHORING.md §11](#) anti-pattern #4.

- Views are **derived**; the canonical primary files are authoritative when a view disagrees. A composition may navigate to member sources. Combined member-content rendering requires the source and revision contract described in [COMPOSITION.md](#); a local view alone does not supply it.
- Each view is independently displayable — no cross-view dependencies.
- Views SHOULD NOT contain claim notation (`{0.95|i|E001}` metadata). They contain readable prose.

Source Traceability

Views MAY include HTML comments linking content to claims:

```
<!-- source: C001, C002 -->
```

Declaration

When views are present, they SHOULD be declared in `PACK.yaml` under the `views` array with `name`, `file`, `purpose`, and `display_as` fields.

12. Validation Rules

Semantic Constraints

These constraints MUST be validated after parsing succeeds. They are not expressible in the PEG grammar and are checked as a post-parse pass.

ID	Constraint
SC-01	Confidence MUST be in range [0.0, 1.0]
SC-02	All claim IDs MUST be unique across <code>claims.md</code> and <code>history.md</code> within a pack
SC-03	All evidence references in active and historical claims MUST have corresponding entries in <code>evidence.md</code>
SC-04	Supersession chains (ϕ) MUST be acyclic
SC-05	All claim ID targets in relations MUST exist (within active/history records or as valid cross-pack references)
SC-06	Spec version in Rosetta header MUST equal 1 for KP:1 conformance
SC-07	Frontmatter pack name MUST match <code>PACK.yaml</code> name field
SC-08	Frontmatter CalVer MUST match <code>PACK.yaml</code> version field
SC-09	Frontmatter domain MUST match <code>PACK.yaml</code> domain field
SC-10	Frontmatter scale name MUST match <code>PACK.yaml</code> confidence.scale
SC-11	Verbose type names MUST map to their single-letter equivalents: observed=o, reported=r, computed=c, inferred=i
SC-12	When nature is prediction, confidence MUST be ≤ 0.95 . Predictions about future states have irreducible uncertainty; the 0.99+ band is reserved for trivially-falsifiable claims per AUTHORING.md §5 .
SC-13	All evidence IDs MUST be unique within <code>evidence.md</code> . Duplicate serialized identities are ambiguous even when their bodies are identical.

Historical claim records

`history.md` retains each superseded or retracted claim's original bullet, metadata, prose and outgoing relations. Editorial headings identify the transition date, reason and successor (if any); they are not replacement claim identities. Historical claim bullets use the same dense or verbose syntax as active claims, without a second claims-document frontmatter. Local relation targets resolve across the union of active and historical claims. Supersession acyclicity is checked over that union, including multiple generations. Retraction does not imply a successor, and unresolved disputes remain active. See [SPEC.md §9](#) and the [lifecycle fixture](#).

Formal Grammar

The normative PEG grammar for `claims.md` is in `conformance/grammar/kp-claims.peg`. The normative JSON Schema for `PACK.yaml` is in `conformance/grammar/kp-pack.schema.json`. Implementations MUST accept all documents that match the grammar and MUST reject documents that fail to match, subject to conformance level.

13. Conformance

Strict Level

All three checks **MUST** pass:

Check	Scope	Criteria
Syntactic	claims.md	Document matches PEG grammar
Semantic	claims.md + evidence.md	All constraints SC-01 through SC-13 pass
Schema	PACK.yaml	Validates against JSON Schema

Permissive Level

Check	Scope	Criteria
Syntactic	claims.md	Document matches PEG grammar (MUST pass)
Semantic (errors)	claims.md + evidence.md	SC-01 through SC-06 and SC-13 MUST pass
Semantic (warnings)	cross-file	SC-07 through SC-12 produce warnings, not errors
Schema	PACK.yaml	Required fields validate; optional field types produce warnings

These are normative conformance levels. The reference runner currently reports SC-07 through SC-12 as errors in both CLI modes; `--strict` adds PEG syntax checks and is not the normative strict/permissive switch. See [the capability guide](#).

The distinction serves tooling: editors operate in permissive mode during drafting, switching to strict for publishing.

14. Constitutional Core

The following elements are **frozen across all KP spec versions** and **MUST NOT** change:

Element	Specification
Claim syntax	- [ID] headline {metadata} prose
Confidence range	Normalized 0.0–1.0
Evidence references	E + digits
Relation symbols	→ ⊗ ← ~ ⊗ ⇔
Rosetta header	<!-- KP:N ... -->

Everything outside the constitutional core **MAY** evolve across spec versions.

Appendix A: Quick Reference

Dense Claim Syntax

```
- [CID] Headline
  {confidence|type|evidence|date|depth|nature} Trailing prose.
  Continuation prose. →CID, ⊗~CID
```

Positions: {0.95|i|E001,E002|2026-03-01|exhaustive|judgment}

Verbose Claim Syntax

```
- [CID] Headline
  `confidence: 0.95 | type: inferred | evidence: E001, E002 | since: 2026-03-01 | depth:
  exhaustive | nature: judgment`
  Prose.
  `relations: supports C002, contradicts:tension C003`
```

Relation Symbol Table

Dense	Verbose	Meaning
→	supports	Provides evidence for

Dense	Verbose	Meaning
⊗	contradicts	Conflicts with (unqualified)
⊗!	contradicts:error	One is wrong
⊗~	contradicts:tension	Both informative
←	requires	Depends on
~	refines	Adds detail to
⊃	supersedes	Replaces
↔	see_also	Cross-reference

Confidence Scales

Sherman Kent: Certain (0.99+) > Almost certain (0.93) > Highly likely (0.87) > Likely (0.75) > Probable (0.63) > Even (0.37) > Probably not (0.25) > Unlikely (0.13) > Highly unlikely (0.07) > Almost certainly not (0.01) > Impossible (<0.01)

Simple: Verified (0.90+) > High (0.70) > Moderate (0.50) > Low (0.30) > Unverified (<0.30)

Appendix B: Ambiguity Resolutions

The following normative decisions resolve ambiguities identified during grammar formalization. Full details in `conformance/grammar/README.md`.

ID	Resolution
AR-01	Claim ID is <code>C[0-9]+(-v[0-9]+)?</code> . Zero-padding conventional, not required.
AR-02	Evidence ID is <code>E[0-9]+</code> . Same padding rules as claim IDs.
AR-03	Bold wrapping (<code>**[C001]**</code>) is optional syntactic sugar. No semantic meaning.
AR-04	Relations MAY appear on the metadata line after the closing <code>}</code> (separated by a space; starting the trailing text or following the context prose), and/or on subsequent continuation lines (2+ spaces). Relations form a comma-separated list ending the line. A relation symbol is parsed as a relation only when immediately followed by a valid claim-ID reference (per AR-10); other occurrences are prose.
AR-05	Verbose relation names are a closed enum of 8 values.
AR-06	Evidence type is open vocabulary. Unknown types MUST be accepted.
AR-07	Both blockquote and list evidence formats are valid. Field names are case-insensitive.
AR-08	Frontmatter is KP-specific compact format, NOT arbitrary YAML.
AR-09	Entity annotation: <code>[entity_type]</code> or <code>`[entity_type</code>
AR-10	Tilde <code>~</code> is a relation only when immediately followed by a valid claim ID reference (<code>C</code> + digits or cross-pack ref). Examples: <code>~C002</code> is the <i>refines</i> relation; <code>~refines</code> (the legend token in the Rosetta header) and <code>~the manuscript</code> (mid-prose tilde) are prose, not relations.
AR-11	Continuation lines use 2+ spaces of indentation.
AR-12	Claim ID gaps are permitted. Uniqueness is the only constraint (SC-02).
AR-13	A claim uses EITHER dense OR verbose metadata. No mixing within a single claim.
AR-14	<code>signatures.yaml</code> schema defined in ARCHIVE.md. <code>composition.yaml</code> schema is published at <code>conformance/grammar/kp-composition.schema.json</code> (normative for supported composition types); the conformance runner detects composition packs by <code>composition.yaml</code> presence (per SPEC.md §2, COMPOSITION.md) and validates <code>composition.yaml</code> against the published schema when the file is present.
AR-15	<code>tier</code> is optional. When <code>hub</code> , <code>sub_packs</code> is required.
AR-16	Cross-pack references: <code>pack_name#section_ref</code> . <code>#</code> is the separator.
AR-17	Blank lines are permitted between structural elements (frontmatter, headings, descriptions, sections, claims). A section MAY open with a blockquote description, mirroring the document heading. Whitespace-only lines are not blank lines.
AR-18	The bracket id is a claim's sole serialized identity; relations, history, and cross-pack references (AR-16) resolve to it. A claim's metadata block (dense §6.1 or verbose §6.2) carries exactly its defined fields — producers MUST NOT add fields beyond them, and readers MUST reject a non-empty undefined/trailing token rather than silently dropping it. Opaque or storage-layer identifiers are non-wire (storage is non-normative; see STORAGE.md).
AR-19	Evidence bodies that contain record-shaped Markdown use the exact <code>kp-content-encoding: markdown-backslash-v1</code> marker and reversible escape algorithm in §10. Record segmentation precedes decoding; an unescaped top-level H2 ends the current record; unknown encodings are preserved, never guessed.

Appendix C: Companion Specifications

Document	Covers	When you need it
AUTHORING.md	Producer-side decision rubrics — claim type, nature, contradiction qualifier, confidence calibration, granularity, content routing	Authoring a pack from source material (informative)
VOICE.md	Voice surface format, spoken delivery conventions, register axis	Building voice/audio interfaces
COMPOSITION.md	Meeting packs, composite packs, agenda overlays, <code>composition.yaml</code>	Composing packs from multiple sources
LIFECYCLE.md	Ephemeral/seasonal/permanent packs, archival, supersession cascade, reconciliation	Managing pack lifecycles
MULTILINGUAL.md	Locale subdirectories, translation workflow, register sub-distinctions, evidentiary multilingual exception	Supporting multiple languages
ORGANIZATION.md	Hub/detail hierarchy, working sets, repo structure	Organizing large pack collections
CONSISTENCY.md	Cross-pack patrol, contradiction detection, confidence decay	Maintaining consistency across packs
CONVENTIONS.md	Linguistic conventions, naming style	Standardizing prose style
STORAGE.md	Pack-as-master, serialization, index contract	Storing packs in databases/caches
BUNDLE.md	Export formats, clipboard format, sharing	Exporting and sharing packs
ARCHIVE.md	ZIP archive, content hashing, integrity chain, <code>signatures.yaml</code>	Transporting packs between systems
DEFINITIONS.md	Definition/policy YAML schemas, entity types, codegen	Building ontology layers
EXTENSIONS.md	Producer-defined extensions, <code>* blocks</code> catalogue, canonical entity ID format	Adding extensions, <code>* payloads</code> or interpreting them
NOTES.md	AI note-taking metadata, disclosure, consent	Recording meetings with AI
PLAYBACK.md	Self-driving voice playback, <code>PlaybackPlan</code> , <code>AudienceProfile</code> (experimental)	Building voice presentation runtimes
RECONCILIATION.md	Cross-pack reconciliation protocol (stub — design deferred to v0.9 / v1.0)	Multi-pack drift detection (design pending)
MAPPING.md	RDF / JSON-LD / PROV-O / Nanopublications field-by-field translation (informative)	Interoperability with semantic-web standards
RATIONALE.md	Design principles, style systems rationale, cognitive perception rationale, relationship to existing standards (informative)	Understanding <i>why</i> the format is shaped the way it is
INDEPENDENCE.md	Ownership & capture-resistance guarantees — zero-dependency readability, freedom to fork, no required registry, optional cryptography	Understanding the independence commitments and their limits

Knowledge Pack Specification — Editor's Draft

Status: Released Editor's Draft — v0.8.5-preview **Latest citable release:** KP:1 — Knowledge Pack Format Specification (v0.8.5-preview, 2026-10-13) **Editor:** Timothy Kompanchenko **Draft updated:** 2026-10-12 **Version history:** See [CHANGELOG.md](#) **Lane:** Full normative spec + rationale + ecosystem — see [README.md](#) for the three-lane structure (CORE = implementer surface, SPEC = comprehensive, companions = topic-authoritative for their domains).

Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#).

1. Overview

A **Knowledge Pack** is a portable representation of epistemic state — what someone believes, how strongly they believe it, based on what evidence, in tension with what other beliefs, as understood at a particular moment in time.

KP:1 makes uncertainty, differing accounts, evidence and history explicit parts of the knowledge record. A disagreement can itself be useful knowledge; resolving it should preserve the original accounts and the reason for the change. Confidence may need reassessment when evidence or circumstances change. Different readers can use views of the same record for reasoning, visual reading or spoken delivery; the canonical files remain authoritative.

The format combines these concepts in an authored, portable text record. Other representations can carry parts of this state, but preserving KP:1's meaning can require custom vocabulary and explicit mappings. [MAPPING.md](#) distinguishes shared representations from lossy mappings and concepts that need extensions.

A Knowledge Pack packages claims with confidence, evidence, relationships, and contradiction into a multi-file plain text format that any intelligence — human or AI — can read without special tooling.

Why This Matters

When an AI reasons about a domain, it needs more than facts. It needs to know: *How sure are we? Who said this? Does it contradict something else we believe? When was this last confirmed? What evidence supports it?* Making this context explicit gives applications material to inspect and reason from. Applications remain responsible for retrieval, reasoning and checking their conclusions; the format alone does not prevent unsupported output.

When a human transfers knowledge to another human, they naturally include this context: "I think the market is about \$2 billion, but that number is from last year's report and a colleague quoted a different figure." Context can be lost when that account is reduced to isolated records. Knowledge Packs provide explicit places to retain it.

Design Philosophy

The Voyager Principle: Knowledge Packs must function with zero infrastructure dependencies beyond "a device that can process text." Like the Voyager Golden Record — symbols understandable by any intelligence without the sending civilization's infrastructure. No internet, no registry, no parser, no compiler. A hard drive, a computer, and a language model is all you need. This principle is a generative constraint: by designing for the most extreme portability, we free ourselves from unconsciously assuming today's infrastructure. The practical result is a format that depends on nothing but text and intelligence.

Knowledge, not data: Only epistemic content belongs in a Knowledge Pack — beliefs with confidence, evidence, and relationships. Instructions, procedures, configurations, and runtime data have their own formats. A knowledge pack represents what is believed to be true. An instruction set tells an agent how to behave. These are different concerns. "Only knowledge is a knowledge pack."

Uncertainty is first-class: Confidence scores, evidence chains, and contradiction relations are not optional metadata — they are the core of the format. A claim without a confidence level is incomplete. A contradiction between claims is surfaced, not resolved. The epistemic state IS the content.

Three surfaces, one truth: Knowledge Packs serve three modes of engagement through three surfaces. The **reasoning surface** (claims.md) is dense and optimized for AI inference — it represents what is believed and why. The **display surface** (views/) is pre-rendered markdown optimized for human visual comprehension. The **voice surface** (views/voice/) is optimized for spoken delivery — short sentences, pronunciation hints, pause markers. These are not cosmetic variations. Reading, seeing, and hearing are epistemically distinct acts. But all three surfaces represent the same underlying beliefs — views are derived from claims, never authoritative.

Inspectability over readability: The reasoning surface is optimized for AI consumption and information density. A human CAN inspect and verify the contents (it's plain text, not binary), but the format does not waste tokens on reading comfort. LLMs are the natural interface for humans engaging with the reasoning surface. Display and voice views serve human perception directly.

What a Knowledge Pack Is

- A directory of plain text files (UTF-8 markdown + YAML) representing epistemic state
- Claims — beliefs with confidence, provenance, evidence, and relationships to other claims
- Contradictions preserved as knowledge, not eliminated as errors
- Zero-dependency — no tooling, parser, or infrastructure required to consume
- Portable across any AI system that can read text
- Versionable, diffable, reviewable via standard git workflows

What a Knowledge Pack Is NOT

- A database (it represents beliefs with uncertainty, not facts with certainty — but databases can store and index packs as derived representations; see `spec/STORAGE.md`)
- A RAG pipeline (it's the input to one, not the system itself)
- A fine-tuning dataset (it's consumed at inference time, not training time)
- An instruction set (that's Agent Skills / `SKILL.md`)
- A context configuration (that's the role of agent instruction files like `AGENTS.md`)
- A runtime data protocol (that's MCP Resources)
- A note-taking format (unstructured notes are not knowledge packs; however, packs CAN contain structured notes and action items alongside claims — see `spec/STORAGE.md` §6)

Position in the AI Stack

Layer	Standard	Packages
Context configuration	Agent instruction files (e.g., <code>AGENTS.md</code>)	Agent behavior, preferences
Capabilities	Agent Skills (<code>SKILL.md</code>)	Procedures, tools, scripts
Epistemic state	Knowledge Packs	Beliefs, confidence, evidence, contradictions
Runtime access	MCP Resources	Live data endpoints

Quick Start — Create a Knowledge Pack in 5 Minutes

1. Create a directory:

```
mkdir my-project.kpack && cd my-project.kpack
```

2. Write `PACK.yaml`:

```
name: my-project
version: 2026.03.19
domain: technology/saas
author: Your Name
confidence: { scale: simple, normalize: true }
```

3. Write `claims.md`:

```
<!-- KP:1 — Knowledge Pack Format
Claims: - [ID] assertion {confidence|type|evidence|date|depth|nature} context
  Positions 5-6 optional. Verbose named-field syntax also accepted.
Types: o=observed r=reported c=computed i=inferred
Depth: assumed | investigated | exhaustive (optional, position 5)
Nature: judgment | prediction | meta (optional, position 6; omitted=factual)
Relations: →supports @contradicts @!error @~tension ←requires ~refines @supersedes ↗see_also
Files: evidence.md=sources history.md=past entities.md=graph
-->
---
```

My Project [product|MyProject,MP]

> B2B SaaS platform for widget management.

Overview

- [C001] B2B SaaS for widget management, 50 enterprise customers
{0.99|o|E001|2026-01-01}
- [C002] AI-powered widget classification = primary differentiator
{0.85|i|E002,E003|2026-02-15} 40% misclassification reduction vs.
competitors' rule-based systems (12-account A/B test). →C001

The HTML comment at the top is the **Rosetta header** (CORE.md §4 — required, self-describing). It lets a previously-unfamiliar consumer recognize a Knowledge Pack and tokenize its contents without external context.

4. Write evidence.md:

Evidence

E001

> **type:** observed | **captured:** 2026-01-01
> **source:** Company records, CRM dashboard

Customer count and platform functionality verified directly.

E002

> **type:** research | **captured:** 2026-02-15
> **source:** Internal A/B test — Q4 2025, 12 enterprise accounts

40% reduction in misclassification. AI classification vs. rule-based.

E003

> **type:** web | **captured:** 2026-02-10
> **source:** Competitor documentation review (5 platforms)

All 5 competitors use rule-based classification, no ML/AI component.

(Field names are bold-wrapped per CORE.md §10 "Blockquote Format". The list-format alternative —
- Type: observed, etc. — is also valid.)

5. Done. Any LLM can read claims.md and reason about your project. Any human can inspect it and verify what's claimed.

2. File Structure

A Knowledge Pack is a directory with a conventional name ending in .kpack/ (recommended but not required).

```
{name}.kpack/
├── PACK.yaml          # REQUIRED — Manifest
```

claims.md	# REQUIRED – Current truth (the primary file)
evidence.md	# RECOMMENDED – Backing sources and citations
history.md	# OPTIONAL – Superseded claims (audit trail)
entities.md	# OPTIONAL – Entity definitions (deprecated – see §7)
composition.yaml	# OPTIONAL – Meeting/composite pack references (see spec/COMPOSITION.md)
validation.yaml	# OPTIONAL – Test questions for verification
signatures.yaml	# OPTIONAL – Cryptographic integrity proof (see spec/ARCHIVE.md)
definitions/	# OPTIONAL – Deterministic domain definitions (kind: definition)
{topic}.yaml	# Entity types, attributes, relation types
policies/	# OPTIONAL – Domain rules and prioritization (kind: policy)
{topic}.yaml	# Routing rules, applicability, priorities
views/	# OPTIONAL – Pre-rendered display content
overview.md	# Display view – visual (recommended first view)
{facet}.md	# Domain-specific display views
voice/	# Voice views – spoken delivery
briefing.md	# Voice-optimized briefing
{locale}/	# Locale-specific views (e.g., uk/)
overview.md	# Translated display view
voice/	# Translated voice views
briefing.md	

File Roles

File	Loaded	Purpose
PACK.yaml	Always (metadata only)	Manifest: version, domain, dependencies, confidence scale
claims.md	Always — this is THE file	Current active claims ONLY. Maximally dense. The LLM's complete picture of current truth. (~2-8K tokens)
evidence.md	On demand	Backing data: source URIs, content excerpts, timestamps. Referenced by claim IDs
history.md	On demand	Superseded and retracted claims. The audit trail. Preserves every past state
entities.md	On demand	Entity definitions with types, aliases, URIs, relationship graph
views/*.md	On display	Pre-rendered markdown for human visual consumption. Derived from claims. See §16
validation.yaml	Tooling only	Test questions to verify an LLM can reason from the pack
signatures.yaml	Tooling only	Cryptographic hashes + signing key for integrity verification (see spec/ARCHIVE.md)
definitions/*.yaml	On demand	Deterministic domain definitions: entity types, attributes, relations. See §5
policies/*.yaml	On demand	Domain rules and prioritization logic. See §5

The Core Contract

claims.md gives the LLM everything it needs to reason AND everything it needs to know where to look for more. An LLM reading only claims.md should:

1. Understand the domain's current truth (all active claims)
2. Know that evidence.md exists and can be loaded for citations
3. Know that history.md exists and can be loaded for audit/temporal questions
4. Know that entities.md exists and can be loaded for relationship graphs
5. See ↔ cross-references to other packs for deeper exploration

The primary file is a **complete reasoning surface** AND a **navigation map** to everything else.

Composition-pack File Requirements

A pack whose `composition.yaml` is present is a **composition pack** (also called a *composite pack* — see `spec/COMPOSITION.md` for the full semantics). Composition packs reference claims in other packs via ↔ `see_also` relations rather than carrying their own evidence. The file requirements adjust as follows:

File	Standard pack	Composition pack
PACK.yaml	REQUIRED	REQUIRED

File	Standard pack	Composition pack
composition.yaml	ABSENT	REQUIRED (defines the composition)
claims.md	REQUIRED	REQUIRED — contents are intentionally minimal: claims describe the composition context itself (who, when, why, what changed), not the topics being composed (see COMPOSITION.md §3)
evidence.md	RECOMMENDED — required in practice whenever any claim cites an evidence ID	OPTIONAL — may be omitted entirely
Everything else (history.md, entities.md, views/, definitions/, policies/, signatures.yaml, validation.yaml)	unchanged	unchanged

A composition pack **MUST NOT** redefine the claims or evidence of a pack it references. Its `claims.md` contains only claims about the composition itself; the referenced packs remain the canonical source for their own content.

3. PACK.yaml — Manifest

Required Fields

```
name: solar-energy-market      # Identifier (kebab-case, unique within scope)
version: 2026.03.18          # CalVer (YYYY.MM.DD) — snapshot date
domain: energy/market-analysis # Hierarchical domain path
author: Jane Chen             # Maintainer
```

Optional Fields

```
kind: claim                  # claim (default) | definition | policy | mixed
                             # See §5 for document kinds

description: >               # Human-readable summary
  Solar energy market analysis – technology trends, cost trajectories,
  and regional adoption patterns for utility-scale photovoltaic systems.

confidence:                  # Confidence model declaration
  scale: sherman_kent        # Native scale identifier
  normalize: true            # Claims use 0-1 normalized values

freshness: "2026-03-18"     # Last substantive review date
license: CC-BY-4.0          # Reuse terms
sensitivity: public          # public | internal | confidential | restricted
visibility: private          # private | shared | public (see spec/LIFECYCLE.md)

channels: [team]             # Distribution routing (see §3.1 Channels)
                             # Spec-defined: team | org | public
                             # Custom: namespace/name (e.g., acme/demo)
                             # Default: [] (owner-only, no distribution)

linguistic_epoch: en-US-2026 # Language era for semantic anchoring
ontology: western-empirical  # Knowledge framework (freeform declaration)

dependencies:                # Other packs this builds on
- uri: github.com/example/energy-taxonomy.kpack
  version: ">=2026.01"
- uri: local:../shared-entities.kpack
- uri: local:../core-taxonomy.kpack
  version: ">=2026.1"
  kind: definition           # Ontology dependency (see §5)
                             # Claims validated against these definitions
                             # Declares this is an ontology dependency
```

```

conflicts:                                     # Known contradictions with other packs
- pack: market-research-v2
  note: "Uses different TAM methodology"
  basis: methodology                         # methodology | data_source | temporal | interpretation
| scope
  resolution: prefer-this                     # prefer-this | prefer-other | flag-for-review

provenance:                                   # Trust context (S-C1: Poisoned Pack)
  author: Jane Chen                          # Who created/maintains this
  role: independent                          # manufacturer | independent | regulator | academic |
individual | co-author
  reviewed_by: null                          # Independent reviewer's name, if any
  review_date: null                          # When independently reviewed
  signed: false                              # Cryptographic signature present?

vocabulary:                                  # Predicate conventions
  custom:                                    # Domain-specific predicates
- cost_trajectory
- adoption_rate

files:                                        # Application mappings; canonical generic files remain
required (CORE §2)
  claims: claims.md
  evidence: evidence.md
  entities: entities.md
  validation: validation.yaml

display:                                     # Cognitive perception layer (see §18)
  short_title: "Solar Market"                # Condensed title for constrained contexts
  abbreviation: "SEM"                        # 2-5 chars for badges, pills, breadcrumbs
  tagline: >                                 # One sentence: what this IS (recognition < 3s)
    Utility-scale solar market trends and cost analysis

views:                                       # Pre-rendered display views (see §16)
- name: overview                             # View identifier
  file: views/overview.md                    # Path within pack
  purpose: "Executive summary with key metrics and status"
  display_as: "Overview"                     # UI tab/section label
  hint: "Market size, growth rate, key players, cost trends" # Human navigation aid (see §18)
- name: technology
  file: views/technology.md
  purpose: "Technology comparison and efficiency trajectories"
  display_as: "Technology"
  hint: "Panel types, efficiency curves, emerging technologies"

style: null                                 # External style system reference (see §17)
                                           # style-id string | null = unstyled markdown

lifecycle:                                  # Pack lifecycle management (see spec/LIFECYCLE.md)
  type: permanent                            # permanent | seasonal | ephemeral | experience |
working | incident
  archive_after_days: null                   # Auto-archive after N days (ephemeral packs)
  archive_policy: reconcile                  # reconcile | auto | manual | keep
  reconciled_at: null                        # Timestamp of last reconciliation check
  orphan_claims: []                         # Unreconciled claims found during review

consolidation:                              # Knowledge consolidation targets (ephemeral/seasonal
packs)
  targets:                                  # Standing packs that should receive consolidated claims
- pack: renewable-strategy                  # Target pack name
  claims: [F001, F002, A001]                # Claims to consolidate (or "all-knowledge" for auto-
classify)
- pack: market-landscape
  claims: [F011, F012]
  status: pending                           # pending | in-progress | complete
  completed_at: null                        # ISO timestamp of consolidation completion

locales:                                    # Multilingual support (see spec/MULTILINGUAL.md)

```

```
canonical: en-US # Canonical language (always American English)
available: # Translated locale views
- locale: uk-UA
  display_name: "Українська"
  status: draft # draft | reviewed | verified
  views: [overview, briefing]
  derived_from: "claims@2026.03.22"
  translator: machine # machine | human | hybrid
  reviewed_by: null

notes: # AI note-taking metadata (see spec/NOTES.md)
  mode: "off" # active (structured extraction) | passive (full
transcript) | off
  participants: [] # Meeting participants
  disclosed: false # Was AI note-taking disclosed to participants?
  consent: null # Required for mode: passive – obtained | declined |
pending

extensions: # Experimental / implementation-specific manifest metadata
  ai_brief: # Example extension payload (not core KP:1)
    version: 1
    verdict: acceptable
    headline: "Strong base attribution, but provenance gap remains"
```

Version Semantics

Packs use **Calendar Versioning (CalVer)**: YYYY.MM.DD, optionally with a revision suffix (2026.03.18-rev2).

Rationale: Facts are snapshots in time, not API contracts. CalVer is natural for knowledge that evolves by date, not by breaking-change semantics.

3.1 Channels — Distribution Routing

The `channels` field declares which distribution cohorts may receive a pack. It is the third axis of access control, orthogonal to the other two:

Axis	Question	Values
sensitivity	How strictly must this data be handled?	public, internal, confidential, restricted
visibility	Who can discover/query this once deployed?	private, shared, public
channels	Which deployment cohorts receive this pack?	Set of strings (see below)

Syntax: An optional array of unique, lowercase strings with set semantics. Order is non-semantic. Default is `[]` (owner-only — no automated distribution).

```
channels: [] # Owner-only (default if omitted)
channels: [team] # Internal team environments
channels: [team, org] # Organization-wide
channels: [team, acme/investor] # Team + custom channel
channels: [public] # Openly distributable
```

Spec-defined vocabulary (bare names, reserved):

Channel	Meaning
team	Named collaborators and internal team environments
org	Broader authenticated organization environments
public	Eligible for unrestricted public distribution

Custom channels use namespaced format: `{namespace}/{name}`. Bare names not in the spec vocabulary SHOULD be rejected by linters. Examples:

```
channels: [acme/investor] # Investor demo environments
channels: [acme/stakeholder] # Stakeholder previews
channels: [acme/partner] # Partner-facing content
```

Validation rules:

Constraint	Rule
sensitivity: confidential	channels MUST NOT contain public or org
sensitivity: internal	channels MUST NOT contain public
channels contains public	visibility SHOULD be public
Unknown bare name	MUST warn (likely typo or missing namespace)
Duplicate entries	MUST reject

Inheritance: Sub-packs (tier: detail) do NOT inherit channels from their hub pack. Each pack declares its own channels explicitly. This prevents accidental over-distribution through implicit inheritance.

Deployment mapping: Channels are mapped to concrete environments by deployment configuration (outside this spec). A typical mapping:

```
# deployment/environments/alpha.yaml (NOT part of KP:1 spec)
name: alpha
channels: ["*"] # Everything
---
# deployment/environments/beta.yaml
name: beta
channels: [team, org]
---
# deployment/environments/production.yaml
name: production
channels: [org, public]
```

The spec defines the `channels` field and its vocabulary. How channels map to infrastructure is a deployment concern.

3.2 Manifest Extensions

The `PACK.yaml` root is intentionally closed. New or experimental manifest keys MUST NOT be added at the top level unless and until they are standardized by a future KP revision.

Implementation-specific or experimental metadata MUST live under the optional `extensions` object. Consumers MUST ignore extension content they do not understand. Extension content MUST NOT override or redefine the meaning of core KP fields.

The example payload shown earlier (`ai_brief`) is illustrative — extension names and shapes are defined by their producers, not by KP:1. This creates a narrow compatibility lane: tools can ship experiments without forcing a schema break or prematurely promoting the payload into the core standard.

4. claims.md — The Primary File

Structure

```
<!-- KP:1 — Knowledge Pack Format
Claims: - [ID] assertion {confidence|type|evidence|date|depth|nature} context
  Positions 5-6 optional. Verbose named-field syntax also accepted.
Types: o=observed r=reported c=computed i=inferred
Depth: assumed | investigated | exhaustive (optional, position 5)
Nature: judgment | prediction | meta (optional, position 6; omitted=factual)
Relations: →supports @contradicts @!error @~tension ←requires ~refines @supersedes ↗see_also
Files: evidence.md=sources history.md=past entities.md=graph
-->
---
pack: {name} | v: {CalVer} | domain: {domain}
confidence: {scale} | normalized
---

# {Pack Title} [{entity_type}|{aliases}]

> {One-line description}
```

```
## {Section}

- [{claim_id}] {Claim headline — single assertable proposition}
  {confidence|type|evidence_ids|since_date|depth|nature} Optional context prose.
  {relation_symbols}{claim_ids}
```

Rosetta Header (Mandatory)

Every `claims.md` MUST begin with a self-describing legend comment before the frontmatter. This serves three purposes:

1. **Self-describing:** Any intelligence encountering the file for the first time can decode the notation without external documentation
2. **Format versioning:** `KP:1` identifies the spec version — future versions use `KP:2`, `KP:3`, giving any parser an immediate signal
3. **Navigation map:** Lists the companion files and their roles — the LLM knows what else exists without reading `PACK.yaml`

Cost: ~50 tokens. Non-negotiable — the Voyager Principle demands the file explain itself.

Frontmatter

Compact YAML frontmatter using pipe-delimited fields. An LLM encountering just `claims.md` (without `PACK.yaml`) should still understand the pack identity and confidence model.

Claim Anatomy

Each claim is a markdown list item with two layers:

1. **Headline** — The claim itself. A single, independently verifiable assertion. Claim ID in brackets.
2. **Metadata + prose line** — Curly-brace metadata block followed by optional prose. Prose is minimal — only what adds meaning beyond the structured data. Many claims need zero prose.

Relation symbols follow the prose (or metadata if no prose):

Symbol	Meaning
→	supports
⊗	contradicts (unqualified — flag for review)
⊗!	contradicts:error (one is wrong — prioritize resolution)
⊗~	contradicts:tension (both informative — preserve as productive disagreement)
←	requires / depends on
~	refines / adds detail to
⊃	supersedes
↔	see_also (cross-pack: pack_name#section)

Contradiction qualifiers: bare `⊗` is the default when the nature of the disagreement hasn't been assessed. Use `⊗!` when investigation shows one claim is wrong. Use `⊗~` when both claims are informative and the tension itself is knowledge (different methodologies, different scopes, different perspectives).

Claim Fields (inside {})

Pipe-delimited, positional: `{confidence|type|evidence_ids|since_date|depth|nature}`

Positions 1-4 are required. Positions 5-6 are optional — omit trailing fields when not needed. To specify position 6 without position 5, use an empty slot: `{0.85|r|E020|2026-03-10||prediction}`. Empty interior slots are always valid and mean "unspecified."

Position	Required	Values	Description
1	Yes	0.0 - 1.0	Normalized confidence
2	Yes	o (observed), r (reported), c (computed), i (inferred)	Claim type (single letter)
3	Yes	Comma-separated IDs	Evidence references
4	Yes	YYYY-MM-DD	When established
5	No	assumed, investigated, exhaustive	Investigation depth (omit if unspecified)

Position	Required	Values	Description
6	No	judgment, prediction, meta	Claim nature (omit = factual assertion)

Claim Types (position 2)

Type	Meaning	Example
observed	Directly verified from primary source	"Codebase has 85K lines of TypeScript"
reported	Stated by a credible source, not independently verified	"CEO says revenue grew 40%"
computed	Output of automated analysis	"Match score: 0.73 for mentor-student pair"
inferred	Derived from other claims via reasoning	"Scaffolding is the product, not matching"

Investigation Depth (position 5, optional)

Investigation depth captures how thoroughly a claim has been examined. This is distinct from confidence — confidence measures how sure we are, depth measures how much we've looked into it.

Depth	Meaning
assumed	Taken at face value — minimal investigation, gut feel, placeholder
investigated	Multiple sources consulted, cross-referenced, actively researched
exhaustive	Thorough investigation — remaining uncertainty is genuine, not ignorance

A claim at 0.50 confidence with `exhaustive` depth is more valuable knowledge than a claim at 0.90 confidence with `assumed` depth. The former represents informed uncertainty — the ambiguity itself is what we know. The latter is an uninvestigated assumption that may collapse under scrutiny.

When omitted, depth is unspecified — the consumer cannot distinguish ignorance from informed uncertainty. New claims and claim updates should record depth when the distinction matters.

Claim Nature (position 6, optional)

Claim nature describes what KIND of knowledge the claim represents. This is distinct from claim type (position 2), which describes how the claim was ACQUIRED (provenance).

Nature	Meaning	Example
judgment	Interpretive conclusion drawn from evidence	"Scaffolding is the product, not matching"
prediction	Forward-looking claim with a resolution horizon	"Pilot launches September 2026"
meta	Claim about the state of knowledge itself	"Our market size estimates are unreliable"

When omitted, the claim is a factual assertion (the default). Only annotate claims that are NOT straightforward facts.

Predictions have temporal semantics: the `since` date is when the prediction was made, and the claim text implies a resolution date. There is no structured horizon field — patrol extracts resolution dates from claim text on a best-effort basis. If the prediction text does not contain a recognizable date, patrol cannot flag it automatically.

Meta-claims signal where the knowledge system is weakest. An agent loading a meta-claim like "our competitive analysis is shallow" `{0.60|i|E001|2026-03-01|assumed|meta}` knows to be cautious about competitive claims and to seek additional evidence.

Relation Verbs

Verb	Meaning
supports	This claim provides evidence for another
contradicts	This claim conflicts with another (unqualified — needs review)
contradicts:error	One of these claims is wrong — prioritize resolution
contradicts:tension	Both claims are informative — the disagreement itself is knowledge
requires	This claim depends on another being true
refines	This claim adds detail to another
supersedes	This claim replaces a prior claim

Verb	Meaning
see_also	Cross-reference to a related claim or pack

Alternative Verbose Syntax

The dense positional notation is the primary format, but KP:1 also accepts a verbose named-field syntax. Both are valid. Tooling **MUST** parse both forms.

Dense (positional):

```
- [C001] Scaffolding = product; matching = infrastructure
  {0.95|i|E001,E002|2026-03-01|exhaustive|judgment} →C002 @~C003
```

Verbose (named fields):

```
- **[C001]** Scaffolding = product; matching = infrastructure
  `confidence: 0.95 | type: inferred | evidence: E001, E002 | since: 2026-03-01 | depth:
  exhaustive | nature: judgment`
  Context prose goes here.
  `relations: supports C002, contradicts:tension C003`
```

The claim headline (first line) and prose context are identical in both forms. Only the metadata and relation notation differ. Verbose metadata is wrapped in backtick delimiters; relations use a separate backtick-delimited `relations:` line with comma-separated entries. Dense is more compact for context injection. Packs may mix forms — individual claims within the same file can use either syntax.

Sections

Claims are organized into sections (`##` headings) by topic. Section organization is freeform — the author chooses what grouping makes sense for the domain. Common patterns:

- By facet: `## Thesis`, `## Architecture`, `## Team`, `## Risks`
- By entity: `## Apple Inc.`, `## Microsoft Corp.`
- By time: `## Q1 2026`, `## Q2 2026`

Example

```
---
pack: solar-energy-market | v: 2026.03.18 | domain: energy/market-analysis
confidence: sherman_kent | normalized
---

# Solar Energy Market [market-analysis|SEM,solar]

> Utility-scale solar market trends, cost trajectories, and regional adoption patterns.

## Thesis

- [C001] Cost decline is structural, not cyclical – driven by manufacturing scale
  {0.95|i|E001,E002|2026-03-01|exhaustive|judgment} 10/10 analyses converged.
  Learning curve (22% cost reduction per doubling) has held for 40 years. →C002, @~C003

- [C002] Grid parity already achieved in 75% of global markets
  {0.90|i|E001|2026-03-01|investigated|judgment} IRENA/BloombergNEF data.
  Subsidy removal in mature markets has not reversed adoption. →C001

- [C003] Cost floor approaching – further declines will slow significantly
  {0.30|r|E003|2026-02-14|investigated|judgment} Silicon material costs
  impose theoretical floor. 3/10 analyses supported this framing.
  Tension with C001 is informative – reveals the timeline question. @~C001

## Technology

- [C010] Bifacial modules now 70% of new installations globally
  {0.99|o|E010|2026-03-18} Rear-side gain 5-25% depending on albedo.
  Bankability established. Monofacial declining to niche applications.
```

- [C011] Perovskite tandem cells: 33.7% lab efficiency (Oxford PV)
{0.99|o|E011|2026-03-18} Lab-to-fab gap remains. Commercial viability estimated 2028-2030 for utility scale. →C012
- [C012] Durability is the unsolved risk for perovskite commercialization
{0.75|i|E012,E013|2026-03-10} Moisture/heat degradation pathways known but not fully mitigated. 25-year warranty requirement is the hard gate. ←C011

Market & Adoption

- [C020] Global installed capacity will exceed 2 TW by end of 2027
{0.85|r|E020|2026-03-10||prediction} Depends on China production + India demand.
- ****[C021]**** Key markets: China (45%), US (15%), India (12%), EU (10%)
'confidence: 0.95 | type: observed | evidence: E020 | since: 2026-03-10'
China dominates manufacturing and deployment. US growth driven by IRA incentives with political risk. India accelerating from low base.

Risks

- ****[C030]**** Supply chain concentration in China is the highest systemic risk
'confidence: 0.90 | type: inferred | evidence: E001, E030 | since: 2026-03-01 | depth: exhaustive | nature: meta'
80% of polysilicon, 95% of wafers. Tariff/geopolitical disruption could create 12-18 month supply gaps. 8/10 analyses flagged this. Not a technology problem – a trade policy and diversification problem.
'relations: requires C020'

5. Document Kinds — Three Knowledge Types

Knowledge Packs contain three kinds of documents, distinguished by the `kind` field.

The Principle

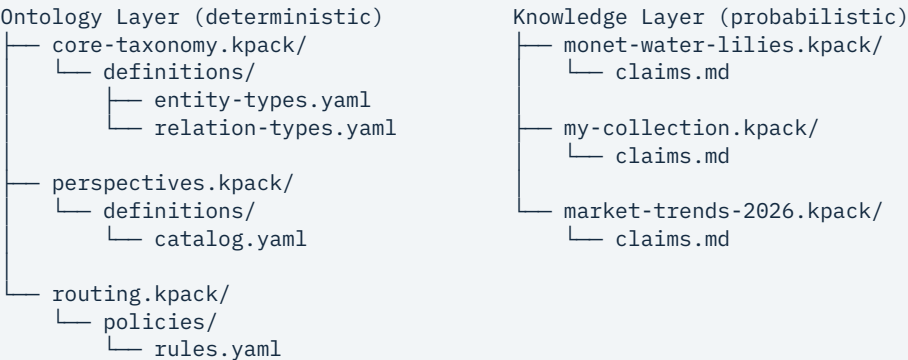
If it defines the domain, it's a definition. If it governs behavior within the domain, it's a policy. If it asserts something about a specific real-world entity, it's a claim.

Kind	Purpose	Epistemology	Key Fields
claim	Assert facts about entities	Probabilistic — confidence, evidence, contradictions	confidence, type, evidence_ids, since
definition	Define domain structure	Deterministic — what the domain IS	authority, version, status, effective_from
policy	Govern domain behavior	Deterministic — rules and priorities	authority, version, status, effective_from

Two-Layer Model

Claims are the **knowledge layer** — probabilistic assertions about specific real-world entities.

Definitions and policies are the **ontology layer** — deterministic facts about the domain itself.



The ontology layer defines what's possible. The knowledge layer says what's true. Definitions establish the vocabulary that claims operate within. When definitions are loaded, an LLM can validate claims against the domain model — flag unknown entity types, detect attribute mismatches, check relation validity. When definitions are absent, claims remain fully interpretable on their own. The constraint is **advisory**: it improves reasoning quality but is never required for comprehension.

Boundary Decision Table

Content	Kind	Why
"Painting is a subtype of Artwork"	definition	Structural — what the domain IS
"Art entities should prioritize authentication"	policy	Behavioral — what the system should DO
"This Monet is oil on canvas, 1906"	claim	Assertion about a specific entity
"The art market grew 12% in 2025"	claim	Assertion about the world (has uncertainty)
"Art entities have a medium attribute"	definition	Structural — what attributes exist
"High-value items require two authentication sources"	policy	Behavioral — a business rule
"Provenance chains link assets to owners over time"	definition	Structural — what relations exist
"Items over \$100K get expedited review"	policy	Behavioral — a threshold rule

Definitions

Ontological facts about the domain: entity types, attributes, subtypes, relation types. No confidence scores — these are not uncertain beliefs but structural decisions about how the domain is organized.

```
# definitions/entity-types.yaml
kind: definition
authority: domain-taxonomy-v1
version: "2026.1"
status: active           # active | deprecated | draft
effective_from: "2026-03-29"

entity_types:
- id: energy_source
  name: Energy Source
  description: "Primary energy generation technology or fuel type"
  subtypes: [solar_pv, solar_thermal, wind_onshore, wind_offshore, hydro, nuclear]
  key_attributes: [capacity_factor, lcoe_range, maturity, grid_impact]
```

Policies

Domain rules and prioritization logic: routing rules, applicability conditions, priority ordering. Encode expert judgment about what matters when.

```
# policies/perspective-routing.yaml
kind: policy
authority: domain-expert-review
version: "2026.1"
status: active
effective_from: "2026-03-29"

rules:
- id: energy-cost-priority
  condition: "entity.type == 'energy_source'"
  action: "prioritize perspectives: [cost_trajectory, grid_impact, maturity, scalability]"
  priority: 1
  scope: "analysis-intake"           # Which system context this rule applies to
```

Metadata Fields (definition and policy)

Field	Required	Values	Description
kind	Yes	definition or policy	Document kind
authority	Yes	Free text	Who established this (e.g., domain-taxonomy-v1)
version	Yes	CalVer or SemVer	Document version

Field	Required	Values	Description
status	Yes	active, deprecated, draft	Lifecycle status
effective_from	No	ISO date	When this takes effect
effective_to	No	ISO date	When this expires (open-ended if omitted)
superseded_by	No	URI or path	Replacement if deprecated
rationale	No	Free text	Why this definition/policy exists (deterministic does not mean uncited)

Pack-Level Kind Indicator

PACK.yaml declares the pack's primary kind:

```
name: core-taxonomy
kind: definition           # claim (default) | definition | policy | mixed
```

Packs with `kind: claim` (the default, may be omitted) use `claims.md` as their primary file. Packs with `kind: definition` use `definitions/` as their primary content. Packs with `kind: policy` use `policies/`. Packs with `kind: mixed` contain both ontological and claim content.

Mixed packs should be rare. When a pack contains both definitions and claims, the definitions establish local vocabulary and the claims operate within it. Versioning follows the pack's CalVer (not separate definition/claim versions). Validation applies to both layers. Prefer separate packs when the ontological and epistemic content have different change cadences or different audiences.

Ontology Dependencies

Claim packs that reference entity types, attributes, or relations from a definition pack should declare the dependency in `PACK.yaml`:

```
dependencies:
- uri: local:../core-taxonomy.kpack
  version: ">=2026.1"
  kind: definition           # Signals this is an ontology dependency
```

The `kind: definition` field on a dependency entry signals that this pack provides vocabulary constraints (not epistemic content). When this dependency is loaded, tooling can validate claims against the ontology. When it's absent, claims remain self-contained — the Voyager Principle is preserved.

Deprecation handling: When a definition pack deprecates an entity type, claim packs referencing that type are not invalidated. The claim's `since` date establishes when it was made. Tooling should warn ("entity type x deprecated since 2026-06-01") but not reject. Claims are historical assertions — retroactive invalidation by ontology changes would violate the append-only principle ([RATIONALE.md §1, Principle 5](#)).

Cross-Pack Reference Semantics (Future Work)

Formal namespace resolution, version pinning, and import mechanisms for cross-pack references are deferred to a future spec revision. Until then, the convention is: definition packs declare canonical IDs; claim packs use those IDs as strings; tooling validates the match via the `dependencies` declaration above.

Codegen Bridge

Definition and policy YAML are designed for domain expert review and AI consumption. For runtime type safety, a codegen step can produce TypeScript types and validation schemas from definition YAML. This pipeline is an implementation concern, not part of the KP:1 spec. See `spec/DEFINITIONS.md` for full schema documentation.

6. evidence.md — Backing Sources

Structure

```
# Evidence

### E001
> **type:** multi_source_synthesis | **captured:** 2026-03-01
> **source:** Internal analysis – 10 independent assessments across multiple AI models

Aggregate synthesis across 10 analyses and 8 distinct models. All converged
on cost decline being structural (manufacturing learning curve) rather than
cyclical (subsidy-driven).

### E002
> **type:** research | **captured:** 2026-03-01
> **source:** IRENA Renewable Power Generation Costs 2025

Comprehensive dataset on utility-scale solar PV LCOE trends across 40 countries.
Learning rate of 22% per capacity doubling confirmed across 1976-2025 data.

### E010
> **type:** observed | **captured:** 2026-03-18
> **source:** IEA Photovoltaic Power Systems Programme – Trends Report 2026

Global installed PV capacity figures and market share by technology type.
Bifacial module share crossed 70% in Q4 2025.

### E011
> **type:** observed | **captured:** 2026-03-18
> **source:** Oxford PV press release – certified tandem efficiency record

33.7% certified efficiency for perovskite-silicon tandem cell. Certification
by Fraunhofer ISE Callab. Commercial module target: 2028.
```

Evidence Fields

Field	Required	Values	Description
type	Yes	Source type string	document, web, expert, observed, research, multi_source_synthesis, interview, measurement, etc.
captured	Yes	ISO date	When the evidence was obtained
source	Yes	Free text	URI, file path, or description of the source
kp-content-encoding	No	markdown-backslash-v1	Reversible protection for record-shaped Markdown inside the evidence body (CORE §10 / AR-19)

Record-Safe Evidence Bodies

Source excerpts can contain their own `### E001` headings or `- [E001]` bullets. Producers **MUST NOT** emit those lines verbatim where a consumer could promote them into sibling KP evidence records. KP:1 registers the exact blockquote marker `> **kp-content-encoding:** markdown-backslash-v1` and the reversible line-escape algorithm in [CORE.md §10 “Record-Safe Encoded Bodies”](#). That CORE section is the implementer contract: record segmentation happens before decoding, existing leading backslashes are escaped for reversibility, and unknown encoding values are preserved rather than guessed.

Evidence IDs are unique serialized identities. A pack containing two unescaped top-level records with the same evidence ID is invalid (SC-13), even if the record bodies are identical.

Design Principles

- Evidence entries are **referenced by ID** from claims.md — the LLM doesn't need to read evidence.md to reason, only to cite

- Evidence content should be **excerpts and summaries**, not full documents — enough to verify the claim, not to reconstruct the source
- Evidence IDs are **stable** — once assigned, an ID always refers to the same source material
- Evidence descriptions should include enough context for an AI to **assess source reliability** — who provided this, what is their relationship to the claim, and how was it obtained. A CEO's self-reported revenue figure and an independent audit of the same figure have different epistemic weight even if both are type: expert

7. Deprecated: entities.md — Entity Definitions

Deprecated as of v0.7.7 (2026-04-28). New packs MUST NOT emit `entities.md`. Use `extensions.entities` for the typed entity graph and `extensions.relations` for typed edges. The successor surfaces carry stable canonical entity IDs, an external-ID vocabulary suitable for cross-pack resolution, per-claim `entity_refs`, and the `relation_types` schema from `DEFINITIONS.md §3`. A one-shot migration script in the reference producer implementation rewrites legacy packs into the new shape. Legacy `entities.md` packs continue to validate; the change is producer-side guidance, not a validation removal.

The following sections are preserved for legacy reference.

Structure

```
# Entities

### Utility-Scale Solar PV
> uri: org.irena.solar-pv | type: technology | aliases: solar, PV, photovoltaic

Photovoltaic electricity generation at utility scale (>1 MW). Dominant
renewable energy technology by installed capacity since 2023. Cost decline
driven by silicon manufacturing scale and cell efficiency improvements.

### Relationships
- `competes_with` → [Wind Onshore], [Nuclear]
- `depends_on` → [Polysilicon Supply Chain]
- `measured_by` → [LCOE], [Capacity Factor]

### Perovskite Tandem Cells
> uri: org.example.perovskite-tandem | type: technology | aliases: perovskite, tandem

Emerging PV technology layering perovskite on silicon for higher efficiency.
Lab record 33.7% (Oxford PV). Commercial viability estimated 2028-2030.

### Relationships
- `extends` → [Utility-Scale Solar PV]
- `blocked_by` → [Durability Challenge]
```

Entity Fields

Field	Required	Values	Description
uri	Recommended	Reverse-domain identifier	Disambiguates across packs
type	Yes	Free text	Entity type (person, product, organization, etc.)
aliases	No	Comma-separated	Alternative names

Entity Resolution Across Packs

When multiple packs are composed:

1. Match on `uri` if present (exact match)
2. Match on `type` + `name` if no URI (same-type, same-name = likely same entity)
3. Flag ambiguous matches as `potential_alias` for human review

8. Confidence Model

Canonical Representation

All confidence values in `claims.md` are **normalized floats from 0.0 to 1.0**.

Native Scale Declaration

`PACK.yaml` declares the native confidence scale. This tells consumers how to interpret the numbers and what human-readable labels to use.

Built-In Scales

Sherman Kent (intelligence analysis)

Normalized	Native (0-10)	Label
0.99+	10	Certain
0.93-0.99	9	Almost certain
0.87-0.93	8	Highly likely
0.75-0.87	7	Likely
0.63-0.75	6	Probable
0.37-0.63	5	Chances about even
0.25-0.37	4	Probably not
0.13-0.25	3	Unlikely
0.07-0.13	2	Highly unlikely
0.01-0.07	1	Almost certainly not
<0.01	0	Impossible

Simple (general purpose)

Normalized	Label
0.90+	Verified
0.70-0.89	High confidence
0.50-0.69	Moderate confidence
0.30-0.49	Low confidence
<0.30	Unverified

Custom scales can be declared in `PACK.yaml` via `confidence.scale: custom` with a `confidence.labels` mapping.

9. Claim Lifecycle

claims.md contains active claims only. It is the reasoning surface — what an agent needs to perceive the current epistemic state. Superseded and retracted claims move to `history.md`, which preserves the full audit trail.

After the final active claim is retracted, `claims.md` retains its header, frontmatter, H1 and at least one empty H2 section, optionally explaining the empty state in a blockquote. Historical records and their evidence remain. No sentinel or replacement claim is required. See [CORE.md §5](#).

Claims are never edited in place — they are superseded by new claims.

States

State	Location	Meaning
Active	<code>claims.md</code>	Current, best-known assertion (default)
Disputed	<code>claims.md</code>	Contradicted by another active claim; both remain — the dispute IS the current state
Superseded	<code>history.md</code>	Replaced by a newer claim (supersedes reference in the replacement)
Retracted	<code>history.md</code>	Withdrawn — found to be incorrect

Disputed claims stay in claims.md because both sides of the dispute represent the current epistemic state. Use **⚡** (error) for disputes where one side is wrong and resolution is needed, or bare **⚡** when unassessed. Patrol should flag **⚡** and **⚡** disputes older than 90 days without new evidence for review.

Productive tensions (**⚡~**) are NOT disputes — they are intentional. Both claims are informative and the disagreement itself is knowledge. Tensions are exempt from the 90-day flag and are skipped by patrol. Use **⚡~** only when you have assessed the contradiction and determined that both sides contribute understanding.

Supersession

When new evidence changes a fact:

1. The new claim is added to claims.md with a `supersedes` reference
2. The old claim is moved to history.md with its full metadata

```
<!-- In claims.md — the new claim (verbose form) -->
- **[C011-v2]** AI advisor migrated to Qwen 4.0 70B-A5B
  `confidence: 0.99 | type: observed | evidence: E040 | since: 2026-06-15`
  Upgraded from Qwen 3.5 35B-A3B after benchmarking showed 15% improvement
  in conversational quality with minimal cost increase due to larger MoE routing.
  `relations: supersedes C011`
```

The verbose-form relation line is ``relations: <name> <target>[, <name> <target>...]'` per [CORE.md §6.2 Verbose Claim Syntax](#). The dense equivalent is `⚡C011` on a continuation line. Either form is valid; mixing dense and verbose metadata within a single claim is forbidden by AR-13.

history.md — Audit Trail

history.md preserves every superseded and retracted claim with its original metadata, prose and outgoing relations, plus editorial context about what happened. Active and historical records share the pack-local ID namespace and semantic checks specified in [CORE.md §12](#).

```
# History

### Superseded

#### C011 (superseded 2026-06-15 by C011-v2)
- [C011] Perovskite tandem cells: 33.7% lab efficiency (Oxford PV)
  {0.99|o|E011|2026-03-18}
  Lab record for perovskite-silicon tandem. Commercial viability 2028-2030.
  Superseded after LONGi announced 34.2% certified efficiency in June 2026.

### Retracted

#### C003 (retracted 2026-04-01)
- [C003] Cost floor approaching — further declines will slow significantly
  {0.85|r|E003|2026-02-14}
  Original framing based on silicon material costs.
  Retracted after perovskite tandem results showed a new cost reduction
  pathway independent of silicon learning curve (see E011).
  Reason: New technology pathway invalidated the cost floor assumption.
```

Each entry preserves: the original claim text, original metadata, and a prose explanation of why it was superseded or retracted. The `since` date in the metadata is the original establishment date; the supersession/retraction date appears in the section header.

10. validation.yaml — Test Questions (Optional)

```
schema_version: "0.1"

tests:
- id: V001
  question: "What is driving solar cost decline?"
  expected_claims: [C001, C002]
```

```
expected_keywords: ["structural", "manufacturing", "learning curve"]
not_expected: ["subsidy", "temporary"]
difficulty: basic

- id: V002
  question: "What is the commercialization risk for perovskite cells?"
  expected_claims: [C011, C012]
  expected_keywords: ["durability", "degradation", "warranty"]
  difficulty: reasoning

- id: V003
  question: "What is the biggest systemic risk to solar deployment?"
  expected_claims: [C030]
  expected_keywords: ["supply chain", "China", "concentration"]
  not_expected: ["technology", "efficiency"]
  difficulty: inference

- id: V004
  question: "Are solar cost declines slowing down?"
  expected_claims: [C001, C003]
  expected_keywords: ["structural", "learning curve"]
  not_expected: ["yes", "approaching floor"]
  difficulty: reasoning
  note: "Tests whether the LLM correctly prioritizes C001 (0.95) over C003 (0.30)"
```

Test Fields

Field	Required	Description
id	Yes	Unique test identifier
question	Yes	The question to ask an LLM with the pack loaded
expected_claims	Yes	Which claims should inform the answer
expected_keywords	No	Words/phrases the answer should contain
not_expected	No	Anti-hallucination — words the answer should NOT contain
difficulty	No	basic (retrieval), reasoning (synthesis), inference (implied)
note	No	Explanation of what this test validates

Automated Extraction Confidence (Multi-Run Calibration)

When Knowledge Packs are generated automatically (via automated extraction or document processing), confidence levels should be empirically calibrated using multi-run extraction rather than relying on the LLM's self-reported confidence.

Primary intervention: Constrained decoding. Before voting, use logit-level constrained decoding (e.g., Outlines, SGLang) to a strict claim schema. This compresses the probability space at generation time, crushing non-determinism at the source at 1x cost. Voting is a secondary intervention for when constrained decoding alone doesn't achieve sufficient stability.

Voting method (when needed): Run extraction N=3 as a screening pass. If any disagreement, escalate to N=5. If 3-2 split at N=5, preserve both as contradicting claims (genuine ambiguity, not noise). For critical/regulated domains, use N=10+ with proper statistical tests.

The deterministic contract pipeline (preferred over voting):

Constrain (logit-level schema) → Canonicalize (normalize) → Validate (deterministic rules) → Escalate (only on failure)

This is cheaper (1x inference) and more principled than N=3-5 voting (3-5x inference). Voting becomes a fallback for when validation fails, not the primary strategy.

Claim comparison: Use **semantic comparison** (meaning-level consistency), not raw string matching. Two extractions that phrase the same fact differently should count as agreement. Semantic entropy (Kuhn et al., 2024) provides the theoretical basis.

Theoretical basis: Baldwin et al. (arXiv:2408.04667v5, 2025) demonstrate that LLMs at temperature=0 produce up to 15% accuracy variation across identical runs, with multi-step systems

compounding to ~77% stability over 5 steps. Non-determinism as a confidence signal: if the model can't agree with itself, it's telling you something about extraction difficulty.

Key caveats (from five-model analysis of Baldwin paper):

- Agreement is a **routing signal**, not a **truth signal** — models can be stably wrong ("confident idiot" problem)
- High agreement does NOT guarantee correctness when: input is ambiguous, evidence is incomplete, schema nudges a canonical-but-wrong answer, or error is an omission rather than a contradiction
- Calibration against labeled data (200+ examples) is a prerequisite before deploying agreement-as-confidence in production
- Track the **stable-wrong rate**: proportion of wrong outputs with high cross-run agreement — this is the failure mode that can't be detected by agreement alone
- Temperature is task-dependent: **T=0 to T=0.1 for claim extraction** (factual recall — higher T increases hallucination risk); **T=0.3-0.5 for narrative generation** from verified claims (diversity in expression is acceptable when facts are locked); **T=0.4 for reasoning** over claims (standard self-consistency range)
- Structured atomic claims are inherently more stable than narrative outputs (shorter outputs = higher agreement, paper's $\rho = -0.64$) AND more factual (Google long-form factuality study: factuality degrades with output length)
- **Hallucination is mathematically inevitable** (Liu et al. 2025, formal proof). Knowledge Packs do not eliminate hallucination — they make it detectable and traceable through confidence levels, evidence chains, and contradiction tracking. Design for tolerance, not elimination

Cross-Claim Consistency Validation

Beyond question-answer testing, `kpack validate` should run an LLM over the full `claims.md` to detect:

- **Arithmetic inconsistencies** (e.g., raise amount / burn rate $\neq$ stated runway)
- **Logical contradictions** not explicitly marked with ⚠
- **Temporal impossibilities** (claim dated before its evidence)
- **Confidence/evidence mismatches** (high confidence with weak evidence, or low confidence with strong evidence)

This was validated empirically: in blind testing of a fundraise knowledge pack, an independent AI model caught a runway calculation error (\$2.5M raise / \$120K burn $\neq$ stated runway) that the pack author hadn't flagged.

Harm-Weighted Validation

For regulated or high-stakes domains, not all errors are equal. A false claim is far more damaging than a missing claim (especially for legal, financial, or medical knowledge). Validation should support harm-weighted metrics:

```
# In validation.yaml
harm_weights:
  false_claim: 10      # A wrong claim actively misleads
  missing_claim: 1     # An omission is incomplete, not dangerous
  wrong_confidence: 3  # Overstated confidence erodes trust
  broken_evidence: 5   # Unsupported claim looks verified
```

The `kpack validate` tool should report harm-weighted error scores alongside standard precision/recall.

11. Sizing & Hierarchy

Context Budget

A Knowledge Pack's `claims.md` must fit within a practical LLM context budget alongside system prompts, conversation history, and other context. The format enforces this through recommended size tiers, not hard limits.

Size Tiers

Tier	Claims	Token Budget	When to use
Hub	10-20	2-4K tokens	Always loaded. Executive-level understanding of a domain.
Detail	20-40	4-8K tokens	Loaded on demand when conversation goes deep on a specific facet.
Standalone	15-30	3-6K tokens	Self-contained pack for a single, bounded topic.

The Split Rule

If claims.md exceeds ~40 claims or ~8K tokens, split the pack.

Ask: "What claims give an agent the framework to correctly interpret everything in the detail packs?" That's the hub. A good hub includes the domain thesis, key unresolved tensions, and the confidence landscape — not just the most important facts, but the interpretive context that makes the facts meaningful. Everything else becomes detail packs.

Pack Hierarchy

Large domains are organized as a hub pack with detail packs:

```
solar-energy-market.kpack/
├── PACK.yaml
├── claims.md
└── evidence.md
# Hub — always loaded
# tier: hub, declares sub_packs
# ~15 claims: thesis, market state, cost trends, outlook

solar-technology.kpack/
├── PACK.yaml
├── claims.md
└── evidence.md
# Detail — loaded on demand
# tier: detail, dependency: solar-energy-market.kpack
# ~25 claims: PV efficiency, perovskite, manufacturing

solar-policy.kpack/
├── PACK.yaml
├── claims.md
└── evidence.md
# Detail — loaded on demand
# tier: detail, dependency: solar-energy-market.kpack
# ~20 claims: tariffs, subsidies, permitting, grid access

solar-deployment-risks.kpack/
├── PACK.yaml
├── claims.md
└── evidence.md
# Detail — loaded on demand
# tier: detail, dependency: solar-energy-market.kpack
# ~15 claims: supply chain, grid capacity, financing
```

Hub Pack PACK.yaml

```
name: solar-energy-market
version: 2026.03.18
domain: energy/market-analysis
tier: hub

sub_packs:
- name: solar-technology
  uri: local:../solar-technology.kpack
  summary: "PV efficiency, perovskite advances, manufacturing trends, module costs"
  load_when: "questions about technology, panels, efficiency, manufacturing"

- name: solar-policy
  uri: local:../solar-policy.kpack
  summary: "Tariffs, subsidies, permitting timelines, grid access regulations"
  load_when: "questions about policy, regulation, tariffs, incentives"

- name: solar-deployment-risks
  uri: local:../solar-deployment-risks.kpack
  summary: "Supply chain concentration, grid capacity, financing gaps, labor shortages"
  load_when: "questions about risks, supply chain, deployment barriers"
```

The `load_when` field is a natural language hint — it helps the consuming system (or the LLM itself) decide when to pull in a detail pack.

Progressive Loading

The loading model applies at every level, with views as a separate display channel:

Level	Always loaded	Loaded on demand	Loaded for display
Within a pack	claims.md	evidence.md, entities.md	views/*.md
Across packs	Hub pack	Detail packs	Views from relevant pack
Across domains	KNOWLEDGE.yaml index	Relevant packs by topic	Views from selected pack

Views are loaded when the AI needs to **show** content to a human, not when it needs to **reason** about it. This keeps reasoning context lean while enabling rich display.

12. Navigation

Knowledge Packs are most powerful when connected. Navigation operates at three layers.

Layer 1: Root Index — KNOWLEDGE.yaml

A KNOWLEDGE.yaml file at the root of a knowledge collection serves as the sitemap — discovery and routing for every pack in the ecosystem.

```
# KNOWLEDGE.yaml
name: energy-research
description: Complete knowledge ecosystem for energy market research
updated: 2026-03-18

packs:
  # — Solar Cluster —
  - name: solar-energy-market
    path: solar-energy-market.kpack/
    tier: hub
    summary: "Utility-scale solar market trends, costs, and adoption"
    topics: [solar, PV, market, cost, adoption]

  - name: solar-technology
    path: solar-technology.kpack/
    tier: detail
    parent: solar-energy-market
    summary: "Panel types, efficiency trends, emerging technologies"
    topics: [technology, perovskite, bifacial, efficiency, manufacturing]

  - name: solar-policy
    path: solar-policy.kpack/
    tier: detail
    parent: solar-energy-market
    summary: "IRA incentives, tariffs, trade policy, grid regulations"
    topics: [policy, IRA, tariffs, grid, regulation]

  # — Wind Cluster —
  - name: wind-energy-market
    path: wind-energy-market.kpack/
    tier: standalone
    summary: "Onshore and offshore wind — costs, capacity, outlook"
    topics: [wind, onshore, offshore, turbine, market]

  - name: battery-storage-economics
    path: battery-storage-economics.kpack/
    tier: hub
    summary: "Storage cost trajectories, chemistry, deployment economics"
    topics: [battery, storage, lithium, deployment, cost]
```

An LLM reading this index knows every available knowledge pack, organized by cluster, with topic tags for routing. When someone asks about tariffs, the `topics` field routes to `solar-policy`. When someone asks about storage, it routes to `battery-storage-economics`.

The index is lightweight enough to be always available — it's metadata, not content.

Layer 2: Hub → Detail — sub_packs

Declared in the hub's `PACK.yaml` (see §11). The hub knows its children and provides `load_when` hints for contextual loading.

Layer 3: Inline Cross-References — ↻

Within `claims.md`, individual claims can point to other packs using the ↻ symbol:

```
- [C011] Perovskite tandem cells: 33.7% lab efficiency (Oxford PV)
  {0.99|o|E011|2026-03-18} Commercial viability estimated 2028-2030.
  →C012 ↻solar-technology.kpack#perovskite ↻solar-policy.kpack#manufacturing
```

The ↻ symbol uses the format ↻{pack_name}#{section}, where `#{section}` references a `##` heading in the target pack's `claims.md`. This acts as a hyperlink — the LLM knows where to go for more depth.

Navigation in Practice

A conversation flow using navigation:

1. **Session start.** System loads `KNOWLEDGE.yaml`. LLM sees all available packs.
2. **Topic identified.** User says "tell me about solar energy costs." System loads `solar-energy-market.kpack/claims.md` (hub).
3. **Depth needed.** User asks "explain the perovskite opportunity." LLM sees `see_also: solar-technology.kpack#perovskite` on C011 and the hub's `sub_packs` entry. System loads the detail pack.
4. **Cross-domain.** User asks "how does this compare to wind?" LLM sees `wind-energy-market` in the root index, system loads it.
5. **Total context.** Hub (3K) + one detail pack (6K) + standalone pack (4K) = ~13K tokens of structured, relevant, navigable knowledge. Not a megabyte of raw documents.

13. Tooling (planned reference CLI)

Status of the `kpack` CLI (as of v0.8.5-preview): two subcommands are implemented in this repository — `kpack lint` (delegates to `python3 conformance/run.py --pack`; supports `--strict`) and `kpack new` (scaffolds a validated pack from the `hello-world` starter, the `--from hello-world` form of the contract below). Every other command in this section describes **planned reference tooling**. The CLI at `reference/kpack` doubles as a contract pointer — run `./reference/kpack` (or `./reference/kpack <subcommand>`) to see which spec section defines each unimplemented subcommand's contract. Producers can implement these contracts independently against `CORE.md`, the JSON Schema, and `AUTHORING.md`.

MVP Commands

```
# kpack lint ships today (delegates to the conformance runner, --strict
# supported); the rest of this list is planned reference tooling.
kpack lint [path]           # Validate structure and references
kpack test [path]          # Run validation.yaml against an LLM
kpack export [path] --format # Export to JSONL, JSON, or other formats
kpack render [path]         # Generate/regenerate views from claims
kpack render [path] --view X # Regenerate a specific view
```

Extended Commands

```
# kpack new ships today in its '--from hello-world' form; everything
# else below is planned reference tooling (see ../reference/kpack).

# Pack creation
kpack new [name]           # Create new pack (interactive)
kpack new --template meeting-prep [name] # Create from template

# Bundle & sharing (see spec/BUNDLE.md)
```

```

kpack bundle [path] # Full bundle → stdout
kpack bundle [path] --clip # Clipboard format → clipboard
kpack bundle [path] --url # Shareable URL (if published)

# Lifecycle & archival (see spec/LIFECYCLE.md)
kpack archive --dry-run # Preview what would be archived
kpack archive # Archive ephemeral packs past retention
kpack reconcile [path] # Check claims against standing packs

# Seal & transport (see spec/ARCHIVE.md)
kpack seal [path] -o [dir] # Seal pack directory → .kpack ZIP with signatures.yaml
kpack verify [path] # Verify archive integrity (hash + signature + ZIP safety)
kpack verify --chain v1 v2 v3 # Verify version chain integrity
kpack extract [archive] -o [dir] # Extract archive → pack directory
kpack info [archive] # Show archive metadata without extracting

# Multilingual (see spec/MULTILINGUAL.md)
kpack translate [path] --locale uk-UA # Generate translated views
kpack translate --check [path] # Report stale translations

# Consistency (see spec/CONSISTENCY.md)
kpack patrol [path] # Run cross-pack consistency checks
kpack validate [path] # Full validation suite

```

Lint Checks

- PACK.yaml has all required fields and is valid YAML
- Every claim has a unique ID matching pattern [C{id}]
- Every claim has confidence in range 0.0-1.0
- Every claim has at least one evidence reference
- Every evidence reference in claims.md has a corresponding entry in evidence.md
- Supersession chains are valid (referenced claims exist)
- No circular supersession
- Relation targets exist
- Cross-references in `see_also` point to existing packs and sections
- Hub packs have `sub_packs` declared
- KNOWLEDGE.yaml paths resolve to existing pack directories
- Dependency graph is acyclic (no circular references between packs)
- Dependencies use exact version pins (no "latest" or floating versions)
- Evidence provenance chains do not form cycles across packs
- Warning: claims with confidence >0.90 citing only self-authored evidence
- Views declared in PACK.yaml exist as files in views/
- View source comments (`<!-- source: C001 -->`) reference existing claims
- View generation timestamps are not older than claims.md modification date (staleness warning)
- Warning: view exceeding 80 lines (split signal — see §16 View Sizing)
- Warning: only 1 view for a pack with >15 claims (likely needs decomposition)
- Warning: overview view exceeding 35 lines (should be a router, not a summary)
- Channels array contains no duplicates and only lowercase strings
- Bare channel names are in spec vocabulary (`team`, `org`, `public`) — unknown bare names warn
- Custom channels use `namespace/name` format
- Channels respect sensitivity constraints (§3.1): no `public` on confidential packs

Export Formats

Format	Use Case
jsonl	Database import (one record per line)
json	API consumption (single structured document)
summary	Human-readable summary (claims without metadata)

14. Composition

Loading Multiple Packs

When a system loads multiple Knowledge Packs:

1. **Entity resolution** — Match entities across packs by URI (exact), then type+name (fuzzy)
2. **Claim merging** — Claims from all packs are available. Same-entity, same-predicate claims from different packs coexist.
3. **Conflict surfacing** — If claims contradict (different values for same entity+predicate), the contradiction is made visible to the LLM, not hidden.
4. **Confidence ordering** — When the LLM must choose, higher-confidence claims are preferred. But both are available for reasoning.

Dependency Resolution

Packs declare dependencies in `PACK.yaml`. The consuming system is responsible for ensuring dependencies are available. If a required pack is missing, the system should warn but not fail — graceful degradation.

15. Design Principles

The 25 numbered principles that informed KP:1's choices have been moved to [RATIONALE.md §1](#). They are reference material for understanding *why* the format is shaped the way it is; they are not themselves a conformance contract — the conformance contract lives in `CORE.md`, the JSON Schema, and the PEG grammar.

The principles include the Voyager Principle (zero dependencies beyond text-processing), append-only claim lifecycle, three-surfaces-one-truth, locale-first organization, knowledge-not-journal, and the constitutional-core list of frozen syntax elements. See [RATIONALE.md §1](#) for the full set with rationale.

16. Views — Pre-Rendered Display Content

Purpose

Views are pre-rendered markdown documents designed for **human visual consumption**. While `claims.md` is optimized for AI reasoning (dense, compressed, navigable), views are optimized for human perception (readable, formatted, displayable on Canvas or screen).

The Three-Surface Architecture

Surface	File	Audience	Optimization	When loaded
Reasoning	<code>claims.md</code>	AI/LLM	Max knowledge per token	Always — for thinking
Display	<code>views/*.md</code>	Humans (visual)	Max comprehension per glance	On demand — for showing
Voice	<code>views/voice/*.md</code>	Humans (auditory)	Max clarity per spoken sentence	On demand — for speaking

An AI assistant loading a Knowledge Pack uses `claims.md` to become knowledgeable about the domain. When it needs to *show* information to a human — on Canvas, in a UI panel, as a formatted response — it renders a pre-built display view. When it needs to *speak* information — in a voice conversation, as a briefing — it renders a pre-built voice view. Neither surface is composed on the fly from claims.

Voice views differ from display views in fundamental ways: shorter sentences (10-15 words), no tables or complex formatting, numbers spelled out, pronunciation hints for proper nouns, pause markers for pacing. See `spec/VOICE.md` for the full voice view specification.

This separation means:

- **Faster display** — No inference tokens spent on formatting
- **Consistent rendering** — Same view looks the same every time
- **Traceable content** — Every view statement traces back to a claim

- **Independent evolution** — Claims can be dense; views can be beautiful

View File Format

Each view is a standard GFM (GitHub-Flavored Markdown) document. Views MAY include:

1. **Source traceability comments** — HTML comments linking content back to claims:

```
<!-- source: C001, C002 -->
```

2. **Generation metadata** — When and from what version the view was generated:

```
<!-- generated: 2026-03-21 | claims: v2026.03.21 | style: default -->
```

3. **Semantic markers** — Optional hints that a style system interprets:

```
> [!metric] Grid parity achieved in 75% of global markets
> [!highlight] 22% learning rate sustained over 40 years
> [!risk] Supply chain: 80% polysilicon concentrated in one country
> [!status] Q1 2026 – bifacial modules reach 70% market share
```

Without a style system, markers render as standard blockquotes — graceful degradation.

PACK.yaml Declaration

```
views:
- name: overview
  file: views/overview.md
  purpose: "Router – what it is, why it matters, map to detail views"
  display_as: "Overview"           # UI label (tab name, section header)
  hint: "What this is, key takeaways, and where to go next"

- name: architecture
  file: views/architecture.md
  purpose: "Technical stack, package structure, key design decisions"
  display_as: "Architecture"
  hint: "Panel types, efficiency curves, manufacturing trends"

- name: market
  file: views/market.md
  purpose: "Regional adoption, market sizing, growth projections"
  display_as: "Market"
  hint: "Key markets by installed capacity and growth rate"

- name: risks
  file: views/risks.md
  purpose: "Supply chain, policy, and technology risk assessment"
  display_as: "Risks"
  hint: "Three systemic risks to global solar deployment"
```

The `purpose` field tells the AI **when** to display this view. The `display_as` field provides the **label** for UI navigation (tabs, menus, section headers). The `hint` field helps a human **decide** whether to open this view (see §18).

View Content Rules

1. **Markdown only** — Standard GFM: tables, headings, blockquotes, lists, code blocks. No HTML beyond traceability comments.
2. **Derived from claims** — Views contain no knowledge that isn't in `claims.md`. If a view states a fact, a corresponding claim exists. Source comments make this traceable.
3. **Self-contained** — Each view is independently displayable. No cross-view dependencies (a user can see "Overview" without having seen "Architecture").
4. **Consistent structure** — Views within the same style system use consistent formatting: same table styles, same heading hierarchy, same callout patterns.
5. **No claim notation** — Views contain readable prose and formatted data, not `{0.95|i|E001}` compressed metadata. The whole point is human readability.

6. **One view, one question** — Each view should answer one question a human would ask. If you cannot title the view as a clear question or noun phrase ("How does it work?", "Architecture", "Risks"), it is underspecified. If a view answers multiple distinct questions, split it.
7. **Inverted pyramid** — Lead with the answer, then supporting explanation, then nuance. A reader who stops after the first paragraph should still get the main point.

View Sizing & Decomposition

Views exist to be browsed, not scrolled. A view that requires excessive scrolling or contains the entire pack's knowledge has failed its purpose — it is a `claims.md` with better formatting, not a display surface. The guidance below ensures views serve human perception at Canvas/screen scale.

The Governing Principle: Progressive Disclosure

"Overview first, zoom and filter, then details-on-demand." — Shneiderman (1996)

A human encountering a pack should:

1. See the **overview** — understand what this is and why it matters (30-60 seconds)
2. **Choose** a detail view — based on clear labels and hints (< 10 seconds)
3. **Read** the detail — get a focused, complete answer to one question (1-3 minutes)

The overview is a **router**, not a summary of everything. It orients the reader and maps the territory. Detail views are the territory.

Size Guidelines

View type	Target lines	Target words	Reading time	When to use
Overview (router)	15-35	75-180	30-60 sec	Every pack — the entry point
Compact detail	15-35	150-250	1-2 min	Focused facet, single entity, narrow question
Standard detail	35-60	250-400	2-3 min	Most detail views — one topic, well-structured
Heavy detail	60-80	400-600	3-4 min	Evidence-dense views with tables, only when strongly structured

Split signal: A view exceeding **80 lines** or **600 words** should almost always be split.

Merge signal: A view under **10 lines** that cannot stand alone should be merged into a related view.

These are authoring heuristics, not hard limits. A 90-line view with excellent structure (tables, clear headings, scannable bullets) may be fine. A 50-line wall of prose is worse than a 90-line well-structured view. Structure determines whether length works.

Within views, body text should stay around **50-75 characters per line** for readability. WCAG 2.2 (SC 1.4.8) sets the accessibility ceiling at 80 characters.

View Count Guidance

Pack claims	Too few views	Sweet spot	Needs grouping	Too many
10-20	1-2	3-5	—	8+
20-40	1-3	5-9	10-12	13+
40+	1-4	6-10	11-15	16+

Heuristic: Aim for roughly **4-8 claims per detail view**. The overview references all claims but contains none of them in full — it states conclusions, not evidence.

Beyond **~12 sibling views**, introduce grouping or a two-level hierarchy (view folders, section prefixes) rather than a flat list. Navigating a flat menu of 15+ items exceeds working memory (Cowan, 2001: ~4 chunks).

The Overview View

The overview is the most important view and the most commonly miswritten. An overview that contains all the pack's knowledge is not an overview — it is the pack reformatted. A good overview does exactly four things:

1. **Says what it is** — One sentence identifying the subject (not the pack format)

2. **Says why it matters now** — The hook, the urgency, the "so what?"
3. **Surfaces 3-5 key takeaways** — The 20% of information that 80% of readers need
4. **Maps the pack** — What other views exist and what you'll find in each

An overview should be readable in **under 60 seconds**. If someone reads only the overview and nothing else, they should leave knowing what this is, why it matters, and where to go next.

What the overview must NOT do:

- Restate every claim in prose form (that's what detail views are for)
- Include detailed evidence, methodology, or technical specifications
- Contain sections that could be their own views
- Exceed 35 lines

Decomposition Principles

When splitting a pack into views, apply these principles in order of preference. Choose the one that best matches how a human would naturally browse the content:

Principle	Best for	Example
By question answered	Most packs (default)	"What is it?", "How does it work?", "Who is it for?"
By entity / actor	Personas, team, portfolio packs	One view per persona, per team member, per asset
By abstraction level	Research, decision, technical packs	Overview → thesis → evidence → provenance
By workflow step	Process, operational, meeting prep packs	Phase 1 → Phase 2 → Phase 3
By time slice	Roadmap, history, timeline packs	Now → Next → Later
By audience	Only when framing truly changes	Use sparingly — often creates duplication

"By question answered" is the default. If you're unsure which principle to use, structure each view as the answer to a question a reader would ask. This works for almost every domain.

Avoid arbitrary pagination. Never split on a fixed line count ("every 40 lines"). Split on meaning — a natural boundary where the question changes.

Default View Template

For a typical detail pack (20-40 claims), this decomposition works as a starting point:

```
views/
├── overview.md           # What it is, why it matters, view map (15-35 lines)
├── [core-thesis].md      # The main story or positioning (35-60 lines)
├── [how-it-works].md     # Architecture, process, mechanics (35-60 lines)
├── [who-or-what].md      # Entities, personas, segments, market (35-60 lines)
├── [risks-or-unknowns].md # Counterevidence, gaps, tensions (25-50 lines)
└── [state-or-next].md    # Current status, timeline, next actions (25-50 lines)
```

The bracketed names are placeholders — use domain-appropriate names. Not every pack needs all six. Some need more. The principle is: **each file answers one question**.

When to Split a View

Split when any of these are true:

- The first screen does not tell the reader if they're in the right place
- The view answers more than one distinct question
- Headings within the view could each stand alone as separate screens
- The reader must scroll significantly before reaching the key point
- The view exceeds 80 lines

When to Merge Views

Merge when any of these are true:

- View titles are hard to distinguish from each other
- Readers would almost always read both views together
- A view is too short (< 10 lines) to provide standalone value

Cross-View Claim References

A claim may be relevant to multiple views. The overview references many claims at summary level; detail views reference them in full. This is not duplication — it is progressive disclosure. The same claim appears as a one-line takeaway in the overview and as a paragraph with context in the detail view. Source comments (`<!-- source: C001, C002 -->`) track the mapping.

Deep Links and Self-Containment

A reader may arrive at any view directly (via search, deep link, or voice navigation). Every view — not just the overview — must include a brief framing sentence at the top that answers: "what pack is this from, and what question does this view answer?" This is implicit in the `<!-- generated: ... -->` metadata and the view's H1, but the first paragraph of prose should also orient a cold reader.

View Generation

Views can be created three ways:

1. **Hand-authored** — Written by a human for maximum editorial quality
2. **AI-generated** — `kpack render` uses an LLM to generate views from claims
3. **Hybrid** — AI generates a first draft; human refines

The `kpack render` command:

```
kpack render my-project.kpack/           # Generate all declared views
kpack render my-project.kpack/ --view overview  # Generate a specific view
kpack render my-project.kpack/ --style my-style # Apply style system
kpack render my-project.kpack/ --check         # Report stale views (claims newer than views)
```

View Staleness

When `claims.md` is updated, views may become stale (they reflect old facts). The generation comment tracks this:

```
<!-- generated: 2026-03-21 | claims: v2026.03.21 -->
```

`kpack lint` compares the view's `claims: version` against the current `claims.md` version. If they differ, it emits a staleness warning — not an error, since views may intentionally lag (e.g., during active editing of claims).

Views for Collections (Address Book Pattern)

For packs that represent collections (registries, portfolios, contact books), views can include:

- **Collection overview** — Summary of the entire collection (count, categories, health)
- **Entity cards** — Individual views per entity or entity group
- **Analytics** — Aggregate views (distribution, trends, comparisons)

The pack's `entities.md` defines the records; views present them for human consumption. Example:

```
asset-registry.kpack/
├── claims.md           # Claims about the collection itself
├── entities.md         # Entity definitions (the "contacts")
└── views/
    ├── overview.md     # Collection summary
    ├── by-category.md  # Entities grouped by type
    └── highlights.md   # Featured entities with detail
```

When individual entities have enough depth, they warrant their own packs (the hub → detail pattern applied to entities).

17. Style Systems — Visual Rendering Rules

Style systems are external to KP:1. A pack references a style system by name in its `PACK.yaml` `style` field; the style system itself is not part of the Knowledge Pack format. The rationale for this

separation, the recommended PACK.yaml `style` field shape, and a recommended style-system YAML format have been moved to [RATIONALE.md §2](#).

PACK.yaml Reference (kept here)

```
style: acme-professional      # Style system identifier
```

A null or omitted `style` field means the views render as plain markdown — which is always valid. The structure of a style system, the recommended YAML schema, and the renderer-pipeline rationale are at [RATIONALE.md §2](#).

18. Cognitive Perception Layer — `display` Block and `hint`

Knowledge Packs serve three surfaces: reasoning (AI), display (visual), and voice (spoken). The reasoning surface (`claims.md`) is dense and optimized for inference. The display and voice surfaces are optimized for human perception. The `display` block in PACK.yaml and the per-view `hint` field carry the structured metadata renderers need to support human perception across rendering contexts (web, mobile, terminal, voice, PDF, etc.).

The rationale for these fields — the four perception stages with timed budgets, why each field exists, the Stranger Test, the Good/Bad authoring tables — is in [RATIONALE.md §3](#). This section specifies the fields themselves; [RATIONALE.md §3](#) explains why.

Manifest: `display` Block

The `display` block sits at the pack level in PACK.yaml. All fields are OPTIONAL. When absent, renderers fall back to parsing the overview view's markdown (H1 for title, first blockquote for tagline). Pre-articulated fields are always preferred over parsed fallbacks — they represent **authored intent**, not extracted approximation.

```
display:
  short_title: "Solar Market"      # 2-4 words. Recognition in constrained contexts (lists,
  tabs, breadcrumbs, mobile headers).
  abbreviation: "SEM"              # 2-5 characters. Tightest contexts (badges, pills,
  keyboard shortcuts). Pronounceable or recognized acronym.
  tagline: >                       # One sentence: what the subject IS. Comprehension < 3
  seconds. Not pack format; subject statement.
  Utility-scale solar market trends and cost analysis
  hook: >                          # The single most compelling sentence in the pack.
  Provocation / insight / promise. Engagement < 5 seconds.
  The cost decline is structural, not cyclical – and it has held for 40 years.
```

Field	Type	Required	Purpose
<code>short_title</code>	string	optional	Recognition (Stage 1, < 1s). 2–4 words.
<code>abbreviation</code>	string	optional	Tight-context label. 2–5 chars.
<code>tagline</code>	string	optional	Comprehension (Stage 2, < 3s). One sentence: what the subject IS.
<code>hook</code>	string	optional	Engagement (Stage 3, < 5s). The pack's most compelling single sentence.

View-Level: `hint` Field

Each view entry in PACK.yaml may include a `hint` — a single sentence that helps a human decide whether to open this section. Max ~15 words.

```
views:
- name: risks
  file: views/02_risks.md
  purpose: "Risk register – mentor supply, FERPA, faculty adoption"
  display_as: "Risks"
  hint: "Three risks that could block the Sep 2026 pilot"
```

The `hint` differs from `purpose` in audience and intent:

Field	Audience	Intent
purpose	AI	When to display this view
display_as	Human	Section label (recognition)
hint	Human	Should I open this? (navigation, Stage 4, < 10s)

When `hint` is absent, renderers may extract one from the view's markdown (first blockquote, first paragraph sentence). Extracted hints are approximations — pre-authored hints are always better.

Fallback Hierarchy

Renderers **MUST** resolve display metadata in this order:

Need	Primary source	Fallback
Pack title	<code>display.short_title</code> or pack name in <code>PACK.yaml</code>	H1 from overview view
Pack tagline	<code>display.tagline</code>	First blockquote from overview view
Section label	<code>display_as</code>	View name with title case
Section hint	<code>hint</code>	First blockquote or paragraph from view

This hierarchy ensures packs work without display metadata (backward-compatible), while rewarding authors who provide it with better human perception.

For the rationale behind each field — *why* `short_title` is 2–4 words rather than something else, what cognitive job each field does, the Stranger Test for verifying authored fields, and Good/Bad worked examples — see [RATIONALE.md §3](#).

19. Companion Specifications

The following topics are specified in companion documents within the `spec/` directory. Each extends the core spec without increasing the size of this document.

Document	Topic	Decisions
<code>spec/CORE.md</code>	The implementer surface — minimum required to be KP:1-conformant	—
<code>spec/AUTHORING.md</code>	Producer-side decision rubrics (informative) — how to choose claim type, nature, contradiction qualifier, confidence value, depth, granularity	—
<code>spec/RATIONALE.md</code>	Design principles, style systems rationale, cognitive perception rationale, relationship to existing standards (informative)	—
<code>spec/MULTILINGUAL.md</code>	Locale subdirectories, translation workflow, drift detection, status tracking, evidentiary multilingual exception	D1, D2
<code>spec/VOICE.md</code>	Voice view format, spoken delivery conventions, duration/pace metadata, register axis	D13
<code>spec/COMPOSITION.md</code>	Meeting pack composition, agenda overlay, <code>pre_load/on_demand</code> , <code>composition.yaml</code>	D14
<code>spec/BUNDLE.md</code>	Export format (full bundle + clipboard), KP:1 markers, CLI commands	D11, D19
<code>spec/LIFECYCLE.md</code>	Pack types (ephemeral/seasonal/permanent), archival, supersession cascade, intelligent reconciliation, visibility	D4, D15, D19
<code>spec/NOTES.md</code>	AI note-taking metadata, disclosure vs consent, active vs passive modes	D16
<code>spec/CONSISTENCY.md</code>	Cross-pack patrol, claim contradiction detection, real-time alerting, confidence decay	D18
<code>spec/CONVENTIONS.md</code>	Linguistic conventions — American English, Merriam-Webster, 30-rule table, fallback hierarchy	D1
<code>spec/ORGANIZATION.md</code>	Nested pack categories, working set, migration strategy, repo structure	D3, D12
<code>spec/STORAGE.md</code>	Pack-as-master principle, serialization formats, index contract, storage independence	—
<code>spec/DEFINITIONS.md</code>	Definition and policy document kinds, YAML schemas, codegen concept, migration guidance	—
<code>spec/EXTENSIONS.md</code>	Producer-defined extensions. * blocks catalogue, canonical entity ID format	—
<code>spec/ARCHIVE.md</code>	ZIP archive format, content hashing, integrity chain, <code>signatures.yaml</code>	—
<code>spec/PLAYBACK.md</code>	Self-driving voice playback, <code>PlaybackPlan</code> , <code>AudienceProfile</code> (experimental)	—
<code>spec/RECONCILIATION.md</code>	Cross-pack reconciliation protocol (stub — design deferred to v0.9 / v1.0)	—

Document	Topic	Decisions
spec/MAPPING.md	RDF / JSON-LD / PROV-O / Nanopublications field-by-field translation (informative)	—

Topic companions carry authority for their declared surfaces. Informative, experimental and stub documents retain their stated limits; producer-profile catalog rules are conditional on profile support. See [COMPATIBILITY.md](#) for precedence and reference-tool coverage.

20. Relationship to Existing Standards

The positioning of KP:1 against [AGENTS.md](#) / [llms.txt](#) / [SKILL.md](#) / MCP Resources / JSON-LD / RDF / Nanopublications has been moved to [RATIONALE.md §4](#). For the field-by-field translation analysis between KP:1 and RDF / JSON-LD / PROV-O / Nanopublications, see [MAPPING.md](#).

Appendix A: Changelog

See [CHANGELOG.md](#) for version history.

Authority, Capabilities and Compatibility

Status: Informative implementation guide for the released v0.8.5-preview editor's draft

Authority and versions

CORE.md defines the generic directory/claim contract. Its normative PEG and manifest schema define syntax and shape; semantic checks apply after parsing. Topic companions govern their declared surfaces, including **ARCHIVE**, **COMPOSITION** and **DEFINITIONS**. **CORE** takes precedence over overlapping legacy prose in **SPEC**. Informative documents (**AUTHORING**, **MAPPING**, **RATIONALE**, **STORAGE** and this guide) explain practice, not extra universal wire requirements. **EXTENSIONS** catalogs producer profiles: a profile's rules apply when an implementation claims support for that profile, and cannot relax **CORE**. **PLAYBACK** is experimental; **RECONCILIATION** is a design stub.

KP:1 in the Rosetta header identifies the format epoch. A manifest `spec_version` identifies a specification revision or release tag; it is not the pack's CalVer `version`, the archive's `seal_version`, or an application profile version. Pin a tag or commit when reporting validation. `spec_uri` remains an offline-safe informational pointer, never a required network dependency.

Reference tooling identifies this release as **v0.8.5-preview** (the Python package uses **0.8.5a0**). Verification should record the release tag and source commit.

The latest citable release is **v0.8.5-preview (2026-10-13)**, DOI [10.5281/zenodo.23329922](https://doi.org/10.5281/zenodo.23329922). The **changelog** records its changes from **v0.8.4-preview**. A released editor's draft is public and citable; preview status does not promise stable V1 compatibility. Cite a released snapshot for that snapshot's requirements and a commit for any newer requirements.

Capability boundaries

Surface	Reference coverage	What a passing check does not establish
Manifest, claims, history and evidence	<code>conformance/run.py --strict</code> , fixture suite and contract tests	Source truth, confidence calibration, complete coverage of every clause, cross-pack target resolution
Definition/policy kinds	Manifest and required-file matrix	Validation of the companion's definition/policy YAML document shapes
Composition	Published schema, narrative claims exception and selection fixture	Resolver availability, collection membership, historical member snapshots
Archives	Signatures schema and fixed content-hash vectors	ZIP extraction behavior, digest verification of a supplied archive, cryptographic signature verification or chain ancestry
Voice/locales and views	Companion rules; editorial review	Automatic drift detection, provenance completeness or implemented rendering commands
Producer extensions	Manifest's open extensions lane	Support for a particular profile, unknown-payload interpretation or portable operational claim associations
Graph compiler	Experimental claim-bearing canonical-file projection and its unit tests	Whole-format conformance, full history reconstruction or byte-preserving archival
Reference CLI	<code>lint</code> delegates to the validator; new scaffolds a starter	Other commands: they print contract pointers, not implemented behavior

The default runner uses diagnostic regex parsing. `--strict` additionally executes the PEG for `claims.md` and the Claim subrule for historical bullets and claim bullets embedded in composition narrative. Narrative-only compositions remain valid. Neither CLI mode implements **CORE**'s normative permissive warning policy: SC-07 through SC-12 are errors in both. A suite pass is evidence for the exercised cases, not certification of every format surface. Missing declared fixtures fail the suite.

Compatibility changes in this draft

- **Empty current state:** A claim document may contain a section with no active claims after withdrawal. The header, compact frontmatter, H1 and at least one H2 remain required; history retains the original claim.
- **History:** IDs are pack-local across active and historical records. Existing successors pointing into `history.md` now resolve in the validator. Duplicate active/history IDs are rejected; retain original historical metadata and relations.
- **Composition declaration:** If `PACK.yaml` declares `composition`, its value must be `composition.yaml` and that file must exist. An existing canonical file is detected without the field. Alternate filenames are outside generic reference support.
- **Metadata vocabulary and dates:** Invalid calendar dates and unknown depth/nature values now return diagnostics instead of slipping through a regex-only parse.
- **Evidence:** Both metadata layouts are parsed, including blockquotes with or without a space after `>`. Required fields, registered field names and real captured dates are checked. Field-shaped source quotations after the metadata separator remain body text. `markdown-backslash-v1` decodes only declared, registered bodies after segmentation; unmarked bodies are never heuristically decoded. Unknown encodings preserve the encoded text and expose a warning.
- **Seal payload v2:** Quoted `seal_version: "2"` selects the identity-binding payload in ARCHIVE. Absence selects legacy v1. This does not select file-exclusion/hash rules or guarantee the authenticity of referenced packs.
- **Archive exclusions:** Component matching at every depth and AppleDouble exclusions can change a pack hash compared with a historical policy. Do not rewrite accepted pack bytes or reissue a seal silently. When historical verification is requested, use its pinned historical rules and record that revision/mode in the receipt. This does not require a retrospective scan, migration or new compatibility implementation. If a verifier offers a historical compatibility mode, selection must be explicit; never turn a current-rule mismatch into an unqualified success through automatic fallback. No new hash-policy wire field is defined here.
- **Selection:** `scope` is now required and nonempty, agreeing with the prose contract. Each declared member must have a nonempty pack reference; an explicitly scoped empty member list remains valid. New or revised selection manifests must state their authored scope; this does not require rewriting retained historical packs. Core references remain live pack names; dated reproducibility requires a declared snapshot profile and retained member revisions.
- **Bundles and storage:** A reading bundle, clipboard projection or semantic JSON round-trip does not preserve all pack files or sealed bytes. Keep the original directory or archive for lossless export. Re-sealing changed bytes creates a new receipt, not preservation of the old signature.

Reporting support

Report the exact specification revision, supported kinds, syntax modes, companions and application profiles, then the fixtures/checks actually exercised. Distinguish schema acceptance, semantic validation, logical-text round-trip, file-byte equality, content-hash verification and signature/chain verification. Do not collapse these into one "KP:1 verified" claim.

Unresolved design remains explicit: generic cross-pack reconciliation; a versioned file carrier for per-claim operational entity associations; a multi-transcript-per-locale extension shape; profile-specific revision resolution; and full compiler history support. None is supplied by adding an undocumented claim metadata field.

Authoring — Knowledge Pack Companion Spec

Date: 2026-05-09 **Status:** Draft (informative — producer-side decision rubrics) **Audience:** producers (human or AI) authoring a Knowledge Pack

1. Purpose

CORE.md tells you what a syntactically valid pack looks like. This document tells you **how to make epistemically sound choices** when authoring one.

CORE.md provides closed vocabularies — `o/r/c/i` for claim type, `⊗/⊗!/⊗~` for contradictions, Sherman Kent bands for confidence. CORE.md does not provide the decision rule for picking among them. Without rules, every author invents their own, packs become incomparable across producers, and the format degrades to "Markdown with extra symbols."

This file fills that gap. Each section gives you:

1. **The decision question** — what you are actually choosing between.
2. **A short test** — how to answer the question quickly.
3. **Worked examples** — positive and negative.
4. **Anti-patterns** — common ways producers (especially LLM agents) get this wrong.

If you implement an authoring agent, treat this document as the producer-side authoring guide — the discipline an author should hold themselves to. AUTHORING.md is informative, not normative: the parser does not check it, and `conformance/run.py` will pass packs that violate AUTHORING.md as long as they are syntactically valid. CORE.md is what the parser checks; AUTHORING.md is what the author should check.

Note on examples in this document. Worked examples below are shown in *compact form* — `[C001] headline {metadata} relations` on a single line, without the `-` list marker — for readability. Real KP:1 claims follow the two-line CORE.md §6.1 grammar: `- [C001] headline\n {metadata} optional context`. When copying these examples into a real pack, prepend `-` and break the line after the headline. See `conformance/fixtures/valid/maximal.kpack/claims.md` for the canonical real-pack form.

2. Claim type — `o / r / c / i`

Decision question

Where did this claim's *truth status* come from?

Test

You answer	The type is
"I (or this pipeline) directly observed, measured, or recorded it."	<code>o</code> (observed)
"A credible external source asserts it, and I am relaying that assertion on the source's authority."	<code>r</code> (reported)
"It is the result of applying a deterministic operation to other data."	<code>c</code> (computed)
"It is the result of reasoning over other claims, evidence, or world knowledge."	<code>i</code> (inferred)

Worked examples

```
√ [C001] The 2024 SEC filing for Acme Corp contains the string "going concern" {0.99|o|E001|2025-03-15}
  o: you can open the file and verify the string is present.

√ [C002] BloombergNEF estimates utility-scale solar LCOE at $0.029/kWh in Q1 2025 {0.85|r|E002|2025-03-15}
```

```
r: BloombergNEF is the asserting source; you are relaying their claim.

✓ [C003] Mean reported price across 12 vendor quotes is $0.044/kWh {0.95|c|E003|2025-03-15}
c: deterministic average over E003's quoted values.

✓ [C004] The market is approaching grid parity {0.70|i|E001,E002,E003|2025-03-15}
i: reasoned from the LCOE trend (C002), the spread (C003), and grid-tariff context.
```

The o vs r boundary

This is the most-confused boundary. The rule:

- If the claim is *about the source itself* — its contents, what it says, that it exists — that is **observed**. You are reporting your direct perception of the source.
- If the claim is *what the source asserts about the world*, treated as true on the source's authority — that is **reported**. You are relaying.

Example contrast:

```
[C001] The Reuters article dated 2026-03-15 contains the phrase "voluntary recall" {0.99|o|E001|2026-03-15}
o: about the article's contents.

[C002] Acme Corp issued a voluntary recall in March 2026 {0.85|r|E001|2026-03-15}
r: about the world; Reuters is the source asserting it.
```

Both are valid claims. They differ in what is being asserted, and the right type follows from that.

Anti-patterns

- **Defaulting to o for everything you "saw" in training data.** An LLM's training data is not direct observation — it is reported. Use r and cite the source, or use i and acknowledge it as inferred from training.
- **Using i (inferred) when you mean c (computed).** A deterministic aggregate (sum, mean, count) is c. Reasoning across qualitative claims is i. Don't mix.

3. Claim nature — factual default vs. judgment / prediction / meta

Decision question

What KIND of statement is this — about empirical reality, about your evaluation of it, about the future, or about other claims?

Test

You answer	Annotate as
"It is true or false based on observable evidence in the world as it exists."	(omit — factual is the default)
"It is my evaluation, opinion, or interpretation; another reasonable analyst could disagree."	nature: judgment
"It is about a future state of the world."	nature: prediction
"It is about claims, the pack itself, or the methodology."	nature: meta

Worked examples

```
✓ [C001] LVMH Q3 2025 revenue was €19.1B {0.99|o|E001|2025-10-22}
factual default, no annotation needed.

✓ [C002] LVMH's Q3 result is a strong buy signal {0.65|i|E001,E002|2025-10-22||judgment}
judgment: this is an evaluation; another analyst could read the same data differently.

✓ [C003] LVMH Q4 2025 revenue will exceed €20B {0.55|i|E001,E002,E003|2025-10-22||prediction}
prediction: about a future state.
```

✓ [C004] C002 may be revised when Q4 actuals are released {0.90|i|E001|2025-10-22||meta}
meta: about another claim.

The factual / judgment boundary

The most common authoring failure is **over-classifying empirical statements as judgments** to satisfy a hedging instinct. RLHF-tuned models in particular default to "I think" / "in my view" framing for assertions that are simply factual.

Test: if the claim could be falsified by a single piece of disconfirming evidence, it is factual.

✗ [C001] LVMH revenue was likely around €19B {0.65|i|E001|2025-10-22||judgment}
Misuse of judgment. The revenue is reported; cite it directly.

✓ [C001] LVMH Q3 2025 revenue was €19.1B {0.95|r|E001|2025-10-22}
factual; confidence captures any source-quality concern.

Use *confidence* to express uncertainty about factual claims. Use *nature: judgment* only when the claim is intrinsically evaluative — when reasonable analysts with the same evidence would disagree on the conclusion.

Anti-patterns

- **Annotating standard empirical statements as judgments to seem epistemically careful.** This degrades the pack's factual baseline.
- **Using prediction for past-tense statements that happen to mention a future date.** "Acme announced 2027 capacity expansion" is factual (the announcement happened); "Acme will hit 2 GW in 2027" is a prediction.

4. Contradiction qualifier — ☉ / ☉! / ☉~

This is the most novel piece of vocabulary the format offers, and the most often misused.

Decision question

When two claims are in conflict, which kind of conflict is this?

Test

The disagreement is	Use
Genuine, but you have not yet determined which side is right.	☉ (bare — should be temporary)
Investigated; one side is demonstrably wrong; the other side is correct.	☉! (error)
Both sides are informative; the disagreement itself is knowledge that should be preserved.	☉~ (productive tension)

Worked example — the same conflict, three ways

Two sources disagree about a market size: \$2B vs \$3B.

Case 1 — ☉! Both assertions concern the same 2025 market and definition. Source A used an incorrect unit conversion; E002 documents the inputs and correction, and the pack recomputes the result using the corrected conversion. The old world assertion is wrong, even though the claim "Source A printed \$2B" would remain true.

- [C001] The 2025 global widget market is approximately \$2B.
{0.85|r|E001|2025-12-31|investigated}

- [C002] The 2025 global widget market is approximately \$3B.
{0.93|c|E002|2026-01-15|investigated} Corrected conversion. ☉!C001

Once the correction is adopted, C002 supersedes C001 and C001 moves to history with its original record. By contrast, "the market was \$2B in 2020" and "it was \$3B in 2025" do not contradict; both dated facts may stand. Neither a changed date nor a newer source alone proves an error.

Case 2 — Two defensible models estimate the same 2025 market differently. Their assumptions remain unresolved and both estimates inform the decision.

- [C001] The 2025 global widget market is approximately \$2B under the adoption-led model. {0.75|i|E001|2025-12-31|investigated|judgment}
- [C002] The 2025 global widget market is approximately \$3B under the demand-led model. {0.75|i|E002|2025-12-31|investigated|judgment} Competing estimate. ~C001

This is assessed tension between estimates of one target, not a declaration that both incompatible exact values are verified facts. B2B and B2B+B2C figures concern different populations and can coexist without a contradiction relation; preserve their definitions and use a cross-reference when helpful.

Case 3 — Two equally credible 2025 sources, same definition, no obvious methodology difference, both well-researched.

```
[C001] Widget market is $2B {0.70|r|E001|2025-12-31|investigated}
[C002] Widget market is $3B {0.70|r|E002|2025-12-31|investigated} ~C001
```

Bare ~ says "they conflict and I haven't resolved it yet." This should be a *transitional* state. Either:

- Investigate further and upgrade to ! (with one claim moved to history.md) or ~ (with both kept).
- If after exhaustive investigation you still cannot resolve, keep the uncertainty explicit. Use ~ only if you have assessed both sides as informative; unresolved disagreement alone does not prove productive tension.

Anti-pattern: defaulting to ~ for everything

LLMs trained with RLHF tend to default to ~ for any contradiction because it requires no commitment. "Both perspectives are valuable" is a path-of-least-resistance answer.

This **degrades the format**. The ! qualifier exists precisely to record that one belief was wrong. If you've investigated and one side is wrong, mark it ! even though it requires asserting the wrongness. Preserving false-then-corrected beliefs is the audit trail; that is the format's value proposition.

A useful self-check: in a sealed pack with N contradictions, what fraction are !? The count is a review prompt, not a target ratio: zero established errors can be the honest result. Inspect the propositions and evidence before changing qualifiers.

Anti-pattern: hostile-reading "Agreeable Pack"

An agent optimizing for speed may tag every conflict with ~ to bypass the harder reasoning of ! triage. Conformance passes, but the pack contains zero actual epistemic rigor.

If your pack has many ~ and zero ! and zero , ask: did I really investigate, or did I optimize for "acceptable to all sources"?

5. Confidence calibration

CORE.md §8 declares two scales:

- **Sherman Kent:** Certain (0.99+), Almost certain (0.93), Highly likely (0.87), Likely (0.75), Probable (0.63), Even (0.37), Probably not (0.25), Unlikely (0.13), Highly unlikely (0.07), Almost certainly not (0.01), Impossible (<0.01).
- **Simple:** Verified (0.90+), High (0.70), Moderate (0.50), Low (0.30), Unverified (<0.30).

The bands give a vocabulary for expressing confidence. The rubrics below guide a producer's assessment from its evidential basis; they are heuristics, not a demonstrated calibration procedure. Empirical calibration requires comparing assessed confidence with outcomes across relevant cases. A display label such as "Verified" does not certify that a verification procedure occurred.

Within-band calibration

Inside a Sherman Kent band, assess confidence using **evidence depth and diversity**, with the method and its limits explained:

Evidence profile	Position within band
Single source	Low end of band
Two corroborating sources, similar methodology	Mid band
Three+ corroborating sources OR direct observation OR cross-method validation	High end
Source quality concerns (anonymous, advocacy-shaped, undated)	Notch one band lower

Example: the "Highly likely" band is 0.87–0.93; 0.87 is its lower boundary, not its center. Source independence and shared upstream material matter as much as citation count.

[C001] Acme Corp acquired Widget Inc in March 2025 {0.87|r|E001|2025-03-22}
One credible report. Highly likely band; low end (0.87).

[C002] Acme Corp acquired Widget Inc in March 2025 {0.90|r|E001,E002|2025-03-22}
Two independently sourced reports. Mid band (0.90).

[C003] Acme Corp acquired Widget Inc in March 2025 {0.91|r|E001,E002,E003|2025-03-22}
Independent reports plus the original transaction record. High band (0.91); three links alone would not justify an increase.

When to use 0.99+

Reserve confidence ≥ 0.99 for **trivially-falsifiable claims** — assertions an unfamiliar reviewer can verify in seconds.

✓ [C001] The 10-K filing for Acme Corp at SEC EDGAR contains the phrase "going concern" {0.99|o|E001|2025-03-15}
Falsification: open the filing, search the phrase. <30 seconds.

✓ [C002] The mean of {3.1, 4.2, 3.8, 4.0, 3.5} is 3.72 {0.99|c|E002|2025-03-15}
Falsification: compute the mean.

✗ [C003] Acme Corp will increase capacity in 2027 {0.99|i|E001|2025-03-15||prediction}
Predictions cannot be 0.99. Future-state claims have irreducible uncertainty.

Anti-patterns

- **The 0.95 ceiling.** Assigning every claim a high value because it "feels high" hides differences in the evidence. Each value needs its own reasons; a broad spread of scores is not a goal in itself.
- **Unsupported precision.** Choose a value on stated evidential grounds. Two decimal places can express distinctions an assessment method supports, but neither extra digits nor avoiding round numbers establishes calibration.
- **The Confidence Hallucination.** Emitting 1.0 for every claim because it is "easy to write." Conformance passes; the calibration is meaningless. The schema accepts but the format is degraded.

6. Depth — assumed / investigated / exhaustive

Decision question

How thorough was the verification effort behind this claim?

Test

Effort profile	Depth
Trusted training data, single quick lookup, or unverified working assumption.	assumed
Read the relevant primary sources; corroborated key facts.	investigated
Searched the field; no contradiction found in available primary sources.	exhaustive

Worked examples

```
[C001] Solar PV global capacity exceeded 1 TW in 2022 {0.85|r|E001|2025-03-15|assumed}
Single trade-press source; not corroborated.

[C002] Solar PV global capacity exceeded 1 TW in 2022 {0.91|r|E001,E002|2025-03-15|investigated}
IEA + IRENA both report. Cross-checked.

[C003] Solar PV global capacity exceeded 1 TW in 2022 {0.95|r|E001,E002,E003|2025-03-15|
exhaustive}
IEA + IRENA + BloombergNEF + national grid statistics from top-5 markets. No contradicting
source found.
```

Pack-level honesty

Most packs in early drafts have many `assumed` claims. That is acceptable as long as the depth is **declared honestly**. A pack that marks every claim `exhaustive` to look authoritative is misleading; conformance passes but the producer is lying about effort.

Anti-pattern

- **Depth inflation.** Marking all claims `investigated` because "investigated" sounds professional. Declared depth should match actual effort.

7. Supersede vs edit-in-place vs split

When a claim needs to change after a pack version exists, distinguish a new assessment, a refinement, a split and a record revision. They are not interchangeable.

Decision tree

1. **The new information OPPOSES the old (contradicts the truth value).** → `⊗` supersede. The old claim moves to `history.md`; the new claim takes its slot.
2. **The new information REFINES the old (more specific, narrower scope, additional context, no truth-value change).** → keep both, link the new claim to the old with `~` (refines).
3. **The old claim conflated two distinct assertions, and review now wants them separately addressable.** → split. The old claim is superseded (`⊗`) by two new claims, each with its own ID.
4. **The old claim has a typographical or formatting error and the meaning is unchanged.** → edit in place — but **only if the pack has not been sealed**. Once a `signatures.yaml` has been issued for a version, edit-in-place is forbidden; use `⊗` even for typos so the trail is preserved.
5. **An issued claim's context or outgoing relations need a semantic revision.** → preferably record the changed interpretation as a separately identified assessment referencing the original. If the claim record itself must be replaced, issue a successor with `⊗` and retain the complete predecessor, including its original relations, in `history.md`. Explain what changed. Replacing a support assessment does not mean that the underlying historical observation became false; do not add `⊗!` unless there is an actual error.

Worked examples

Supersede (truth-value change):

```
# claims.md (version 2026.05.09)
[C001-v2] Acme Corp Q3 2025 revenue was $4.7B {0.91|r|E007|2026-05-09|investigated} ⊗C001-v1

# history.md
[C001-v1] Acme Corp Q3 2025 revenue was $5.1B {0.85|r|E001|2026-03-15|assumed}
Superseded 2026-05-09: revised figure from audited 10-K. See C001-v2.
```

Refine (narrowing or specification):

```
[C001] Acme Corp acquired Widget Inc in 2025 {0.87|r|E001|2025-03-15|investigated}
[C002] Acme Corp acquired Widget Inc on March 18, 2025 for $312M {0.91|r|E002|2025-03-15|investigated} ~C001
```

C001 stays; C002 refines the date and price without contradicting the broader fact.

Split:

```
# Old (in history.md)
[C001-v1] Acme Q3 2025 revenue was $4.7B and operating margin was 18% {0.85|r|E001|2025-03-15}

# New (in claims.md)
[C002] Acme Q3 2025 revenue was $4.7B {0.91|r|E007|2026-05-09|investigated} @C001-v1
[C003] Acme Q3 2025 operating margin was 18% {0.91|r|E007|2026-05-09|investigated} @C001-v1
```

Both new claims supersede C001-v1; each is independently falsifiable.

Anti-pattern: silent edit

Editing a sealed claim in place to fix a discovered error destroys the audit trail — the very property the format exists to preserve. Always supersede; the cost is one entry in `history.md`.

8. Granularity — sizing a claim

A claim should be the **smallest unit that is independently falsifiable**.

Test

Ask: "Can I disprove just one part of this claim and have the rest still stand?"

- If yes → the claim contains multiple independent assertions; **split**.
- If no → the claim is one indivisible assertion; **right size**.

Worked examples

```
x Too big — multiple independent assertions:
[C001] The market is large, growing at 26% YoY, dominated by China (80% supply), with cost down 89% since 2010 {0.85|i|E001|2025-03-15}
    Four claims compressed into one. If 2010 cost trajectory is later revised, the entire claim must be superseded.

v Right-sized — one assertion each:
[C001] Global widget market is $24B in 2025 {0.85|r|E001|2025-03-15}
[C002] Global widget market grew 26% YoY in 2025 {0.85|r|E002|2025-03-15}
[C003] China produces 80% of global widget supply {0.83|r|E003|2025-03-15}
[C004] Global widget cost has fallen 89% since 2010 {0.87|r|E004|2025-03-15}
```

The other failure mode — too small

```
x Too small — atomic dust:
[C001] The market exists {0.99|o|E001|2025-03-15}
[C002] The market is large {0.85|i|E001|2025-03-15}
[C003] The market is growing {0.85|i|E001|2025-03-15}
    Each is too vague to falsify or build on. Combine into specific assertions.
```

A useful guideline: a right-sized claim usually fits in a single sentence of 8–25 words, names a specific entity, predicate, and value, and survives the "what would falsify this?" question with a concrete answer.

Anti-pattern: hostile-reading "Atomic Dust"

An agent that turns every sentence of a source document into an isolated `o` claim with no relations produces a fragmented, queryless cloud of data. Conformance passes; the pack is unusable. Granularity is not just about size — it is about each claim being a load-bearing assertion.

9. Content routing — `claims.md` / `evidence.md` / `views/`

File	Contains	Question it answers
<code>claims.md</code>	The assertions. <i>X is Y because Z.</i>	What is believed?
<code>evidence.md</code>	The sources. <i>Document W on date D says X.</i>	Where did it come from?
<code>views/</code>	Pre-rendered presentations of claims for human consumption.	How should it be presented?

Routing rules

1. **A direct quote belongs in `evidence.md`.** The claim it backs belongs in `claims.md`.

```
# evidence.md
## E001
> **type:** research | **captured:** 2025-03-15
> **source:** BloombergNEF Q1 2025 Renewables Report, p. 14
"Utility-scale solar LCOE in low-cost markets has fallen to $0.029/kWh, with continued
downward pressure from polysilicon overcapacity."

# claims.md
[C001] Utility-scale solar LCOE was $0.029/kWh in low-cost markets in Q1 2025 {0.91|r|E001|
2025-03-15|investigated}
```

2. **A view never asserts knowledge that is not in `claims.md`.** Views derive from claims, not the other way around.

```
x Wrong – view contains a fact that is not in claims:
# views/overview.md
"The market is poised for consolidation, with Acme positioned as the dominant acquirer."
(No corresponding claim in claims.md.)

✓ Right – view is derived from claims:
# views/overview.md
"Three claims (C001, C002, C003) together suggest the market is poised for consolidation, with
Acme as a likely acquirer."
```

3. **Keep process context when it affects meaning.** A decision made at a meeting, a participant's role or the method used for an observation can be part of the knowledge. State the attributed account or decision as a claim when it matters; retain the fuller narrative in evidence. Omit incidental workflow chatter only when doing so leaves the attribution, scope, interpretation and reasons for change intact. Views retain those necessary qualifications.
4. **Scope absence findings to what was checked.** "No matching entry in the six databases searched on this date" is a search result. It does not establish that no claim exists anywhere. Distinguish no match found, sources not searched and questions still unknown in assertion/context prose and the supporting search record. No extra metadata vocabulary is required.

Anti-pattern: view laundering

The most common form of unsupported assertion is a view that introduces facts the claims file does not contain. Conformance passes (no SC checks views against claims for completeness). The pack is degraded.

A useful self-check: for every assertion in a view, name the claim ID it derives from. If you cannot, remove it from the view or add it to `claims.md`.

10. Self-check before sealing

Before you seal a pack version (issue a `signatures.yaml`, distribute, or stop authoring), run this checklist.

Type distribution

- Do you have a mix of `o`, `r`, `c`, and `i` types? A pack of all one type usually means you have not done the routing work.

- Are `r` claims tied to specific named sources in evidence?
- Are `i` claims tied to the lower-level claims they reason from (via `←requires` or in the prose)?

Confidence distribution

- Can you explain each confidence value from its evidence and assessment method? Neither a clustered nor a diverse histogram establishes calibration by itself.
- Have you used confidence ≥ 0.99 only for trivially-falsifiable claims?
- Have you used confidence ≤ 0.5 for any claim that is genuinely speculative?

Contradiction discipline

- For every contradiction in the pack, did you correctly choose `⊗`, `⊗!`, or `⊗~`? In particular, are there any conflicts you marked `⊗~` only because the choice was hard?
- If your pack has zero contradictions in a domain that has real disagreements, you have flattened the epistemic state.

Granularity

- Pick three random claims. For each, ask: "What single piece of evidence would falsify this?" If you cannot answer in one sentence, the claim is probably too big.
- Pick another three. For each, ask: "Is this claim load-bearing — does anything else depend on it?" If not, consider whether it should exist.

Routing

- For every assertion in any view file, can you name the claim ID it derives from?
- For every quote-shaped fragment in `claims.md`, ask: should this be in evidence instead?

Lifecycle

- For every superseded claim, is the prior version in `history.md` with a supersession reason?
- Are any claims edited-in-place since the last sealed version? If so, supersede instead.

11. Hostile-reading sanity checks

These are the "look-fine but degraded" patterns to actively avoid:

1. **The Agreeable Pack.** Many `⊗~`, zero `⊗!`, zero `⊗`. → You defaulted to non-commitment. Re-triage.
2. **Atomic Dust.** Hundreds of tiny `∘` claims with no relations and no `c` / `i` claims building on them. → No reasoning structure; the pack is data, not knowledge.
3. **Confidence Hallucination.** Every claim at 0.95 or 1.0 without supporting reasons. → Reassess each claim against its evidence and the method's limits; do not manufacture a diverse distribution.
4. **View Laundering.** A polished `views/overview.md` that contains assertions absent from `claims.md`. → The view is making up knowledge. Move assertions to claims or remove from view.
5. **Self-Referential Evidence.** A claim cites `E001`; the `E001` entry's `source` field is "see `C001`" (the same claim). → Circular justification with no external grounding. The evidence-claim chain must terminate at something outside the pack (a document, a measurement, an interview, a database snapshot). If the source field can only be filled by pointing back at a claim ID, the claim has no real evidence and should either be marked `nature: judgment` or removed. Note: as of v0.8.5-preview, this pattern is editorial-only — the runner does not detect circular evidence; `AUTHORING.md` flags it.
6. **Future-Date Confidence Inflation.** A claim with `nature: prediction` and `since: set` to a future date carries confidence 0.99+ — the runner now rejects this via SC-12 (predictions MUST have confidence ≤ 0.95). The pattern existed in v0.7 and was caught only editorially; v0.8.0-preview makes it a normative semantic constraint. If you genuinely believe a future-state claim at 0.99+, you are either wrong about confidence or wrong about it being a prediction.

If your pack has any of these patterns, conformance may pass (depending on the pattern — SC-12 catches #6) and the pack may still be epistemically degraded. The format protects against syntactic errors and a small set of named semantic ones; only the author can protect against the rest.

12. Worked end-to-end transformation

A short example of "raw observation → KP claims" to anchor the rubrics above.

Source material (raw)

A Reuters article dated 2025-03-15 reports that Acme Corp's Q3 2025 revenue was \$4.7B, up 12% year-over-year, exceeding analyst consensus of \$4.5B. The company's CFO stated on the earnings call that Q4 revenue was "very likely" to surpass \$5B. Bloomberg's separate analysis of the same earnings suggests the operating margin compression to 16% (from 18% in Q3 2024) reflects a pricing-driven slowdown.

As KP claims

```
[C001] Acme Corp Q3 2025 revenue was $4.7B {0.91|r|E001,E002|2025-03-15|investigated}
Reuters and Bloomberg both report. Two-source corroboration; high band.

[C002] Acme Corp Q3 2025 revenue grew 12% year-over-year {0.91|r|E001|2025-03-15|investigated}

[C003] Acme Corp Q3 2025 revenue exceeded analyst consensus of $4.5B {0.91|c|E001|2025-03-15}
Computed: 4.7 > 4.5.

[C004] Acme Corp Q4 2025 revenue is likely to exceed $5B {0.65|r|E001|2025-03-15||prediction}
Reported on the basis of the CFO's statement; downgraded from 0.85 because forward statements
on earnings calls have known biases. Prediction nature.

[C005] Acme Corp Q3 2025 operating margin was 16% {0.85|r|E002|2025-03-15|investigated}

[C006] Acme Corp Q3 2024 operating margin was 18% {0.85|r|E002|2025-03-15|investigated}

[C007] Acme Corp's operating margin has compressed by 2 percentage points YoY {0.85|c|E002|
2025-03-15}
Computed from C005 - C006.

[C008] The margin compression reflects a pricing-driven slowdown {0.55|r|E002|2025-03-15||
judgment}
Reported as Bloomberg's analysis (judgment-shaped). Judgment nature; downgraded confidence
because Bloomberg's analyst opinion is one read of the data.
```

Note the result:

- Five `r` claims, one `i` would have been wrong here (we are reporting, not reasoning).
- Two `c` claims for arithmetic operations.
- Confidence values span 0.55 to 0.91, assessed with reasons tied to evidence quality and claim nature.
- One `prediction`, one `judgment`, six factual defaults.
- C003 and C007 are computed and tied to their inputs by evidence ID.
- No contradictions in this pack — both sources agree on the underlying numbers.

A pack like this is what the rubrics aim to produce: an inspectable account with evidential reasons and appropriate routing between claims and evidence. Its confidence values remain assessments; the example does not demonstrate empirical calibration.

13. Relationship to CORE.md

CORE.md is **normative** for syntax. AUTHORIZING.md is **informative** for judgment. Where they appear to disagree, CORE.md wins on the implementable surface — but if a pack passes CORE.md and violates AUTHORIZING.md, the pack is syntactically valid and epistemically degraded.

A future revision of the spec may promote some of the rubrics here to enforced semantic constraints (sc-12 and beyond). For v0.8.5-preview, AUTHORIZING.md is the producer-side contract; the conformance suite does not check against it.

This file is companion to [CORE.md](#). Where CORE specifies form, AUTHORIZING specifies judgment. Both are needed for a sound pack.

Voice Views — Knowledge Pack Companion Spec

Date: 2026-04-12 **Status:** Draft **Decisions:** D13 (voice view format), D22 (locale-before-surface ordering)

kpack CLI invocations in this document describe planned reference tooling. `python3 conformance/run.py, ./reference/kpack lint` and `./reference/kpack new ship today`. See [SPEC.md §13](#) and the reference launcher at [reference/kpack](#) for status.

1. Purpose

Voice views are the third surface of the three-surface architecture. While `claims.md` serves AI reasoning and display views serve human reading, voice views serve human listening. The auditory channel is fundamentally different from the visual channel — a listener cannot scan ahead, re-read a sentence, or glance at a table. Voice views account for this by producing text optimized for spoken delivery.

Voice views are NOT transcripts, audio files, or TTS instructions. They are plain text documents written the way a knowledgeable briefer would speak to a listener. An AI reading a voice view aloud — or feeding it to a speech synthesis pipeline — should produce natural, clear, paced speech.

2. Why a Separate Surface

Display views cannot be read aloud effectively. The following table captures the seven dimensions where auditory and visual perception diverge:

Dimension	Display View	Voice View
Sentence length	Complex, nested clauses acceptable	Short sentences, 10-15 words maximum
Structure	Headers, bullets, tables, blockquotes	Sequential narrative flow, no tables
Numbers	"\$2,000,000" — scannable at a glance	"two million dollars" — speakable
Emphasis	Bold , <i>italic</i> , code	Pause markers, stress cues
Density	High — reader controls pace via scanning	Low — listener hears one sentence at a time
Proper nouns	Written once, recognized visually	Need pronunciation hints for unusual terms
Context	Reader sees entire page simultaneously	Listener hears only the current sentence

A display view read aloud sounds like someone reading a document. A voice view read aloud sounds like someone briefing you. The difference is the same as the difference between a written report and a radio news segment.

3. Directory Convention

Voice views live inside `views/voice/` within the pack directory:

```
{name}.kpack/
├── PACK.yaml
├── claims.md
├── evidence.md
├── views/
│   ├── overview.md           # Display view (English)
│   ├── pitch.md             # Display view (English)
│   ├── voice/               # Voice views (English)
│   │   ├── briefing.md      # Voice view – executive briefing
│   │   ├── elevator.md      # Voice view – 30-second pitch
│   ├── uk/                  # Ukrainian locale
│   │   └── overview.md      # Translated display view
```

```
|
└─ voice/
   └─ briefing.md      # Translated voice views
                        # Ukrainian voice briefing
```

Locale-Before-Surface Ordering (RATIONALE.md §1, Principle 23)

Locale is the higher-order grouping. A Ukrainian voice briefing lives at `views/uk/voice/briefing.md`, not `views/voice/uk/briefing.md`. Rationale: a locale directory contains everything for that language — display views and voice views together. A consumer requesting "show me everything in Ukrainian" finds it all under `views/uk/`.

4. File Format

Voice view files are plain text with minimal markup. The content is what gets spoken. Everything else is metadata in HTML comments.

4.1 Metadata Header

Every voice view begins with HTML comment metadata:

```
<!-- voice: briefing -->
<!-- duration: ~90 seconds -->
<!-- pace: measured -->
<!-- register: curatorial -->
<!-- generated: 2026-03-22 | claims: v2026.03.22 -->
<!-- source: C001, C002, C010, C020, C030, C040, C050 -->
```

Field	Required	Values	Purpose
voice	Yes	View identifier	Matches the name field in PACK.yaml voice view declaration
duration	Yes	~N seconds or ~N minutes	Approximate spoken duration at natural pace
pace	Yes	brisk, measured, deliberate	Speaking tempo hint for TTS or human reader
register	Recommended	plain, curatorial, technical, investor	Voice register — diction and emphasis profile (see §4.2 below)
generated	Recommended	ISO date + claims version	Staleness detection (same mechanism as display views)
source	Recommended	Comma-separated claim IDs	Traceability back to claims.md

4.2 Register

`register` is an orthogonal axis to `pace`: `pace` controls speed, `register` controls diction and emphasis profile. The four spec values:

- **plain** — straightforward declarative narration. Default for general-audience packs and for first-encounter explanations.
- **curatorial** — careful, balanced, evidence-forward. The default register for art / heritage / scholarship packs where the voice is the curator-as-narrator: confident on what is known, explicit about uncertainty, restrained on emphasis.
- **technical** — precise, terminology-honest, minimal hedging. For internal technical briefings to listeners already fluent in the domain.
- **investor** — verdict-forward, risk-explicit, decision-shaped. Opens with a one-line takeaway; closes with what matters for action.

`register` composes orthogonally with locale and persona surfaces: the same pack may emit `views/uk/voice/briefing.md` at `register: curatorial` and `views/voice/briefing.md` at `register: investor` from the same underlying claims. It is also distinct from app-side `AudienceProfile.register` (PLAYBACK.md §2 — the session-time selector across this axis) — the `register` defined here is a producer-side spec field on the voice view itself, governing how the prose was authored.

For language-specific register sub-distinctions (e.g., Western Ukrainian vs literary Ukrainian, Quebec French vs Parisian French) see MULTILINGUAL.md §3.3. The four values above are the primary axis; locale-specific sub-registers refine within them rather than replacing them.

4.3 Body Conventions

The body is written as spoken text. The following rules govern its format:

1. **No markdown formatting.** No bold, italic, headers, code blocks, or bullet lists. The file is what gets spoken — formatting characters would be read aloud or cause TTS artifacts.
2. **Numbers spelled out.** Write "two million dollars", not "\$2,000,000". Write "fifty percent", not "50%". Write "March twenty-second", not "March 22nd". Write "nine hundred and six tests", not "906 tests". Exception: years remain numeric when they would sound unnatural spelled out — "twenty twenty-six" is acceptable but "2026" is also fine since TTS engines handle years well.
3. **Short sentences.** Target 10-15 words per sentence. Never exceed 20. If a sentence needs a comma, consider splitting it into two sentences.
4. **Pause markers.** Use (pause) on its own line between sections or where a speaker would take a deliberate breath. This signals a 1-2 second silence in speech synthesis or a natural break point for a human reader.
5. **Pronunciation hints.** For terms an English TTS engine might mispronounce, add hints in parentheses after the first occurrence: Qwen (pronounced "chwen"), Hono (pronounced "hoh-no"), Neon (pronounced "nee-on"). Subsequent occurrences do not need the hint.
6. **Sequential narrative flow.** Information builds linearly. A listener cannot skip ahead. Open with context ("Here is a briefing on..."), build through the key points, close with what matters next. No forward references — do not say "as we will discuss later."
7. **Source traceability.** HTML comments `<!-- source: C001, C002 -->` may appear between sections to trace content back to claims. These are not spoken.
8. **Repetition for retention.** Unlike display views where a reader can glance back, voice views may restate a key term or number if it is referenced later. A display view says "906 tests" once; a voice view may say "those nine hundred tests" again when referencing them in a risk section.

5. Full Example

```
<!-- voice: briefing -->
<!-- duration: ~75 seconds -->
<!-- pace: measured -->
<!-- generated: 2026-03-22 | claims: v2026.03.22 -->
<!-- source: C001, C002, C004, C010, C020, C030, C040, C050 -->
```

Here is a briefing on the solar energy market.

Utility-scale solar is now the cheapest new electricity source in most markets. The levelized cost has fallen eighty-nine percent since twenty ten. This decline is driven by three factors. Manufacturing scale, cell efficiency gains, and financing maturity.

(pause)

The market is growing at twenty-six percent annually. China produces over eighty percent of global polysilicon supply. This concentration is the single largest supply chain risk. Trade policy changes could disrupt module availability within months.

```
<!-- source: C001, C002 -->
```

(pause)

Perovskite tandem cells are the leading next-generation technology. Lab efficiency has reached thirty-three point seven percent. Commercial deployment is expected between twenty twenty-seven and twenty twenty-nine. The manufacturing challenge is long-term stability under field conditions.

```
<!-- source: C010 -->
```

(pause)

Three systemic risks dominate the outlook.
 First, polysilicon supply concentration in a single country.
 Second, grid interconnection queues averaging four point one years.
 Third, permitting timelines that vary by a factor of ten across jurisdictions.

```
<!-- source: C020, C040 -->
```

(pause)

The biggest opportunity is in emerging markets.
 Sub-Saharan Africa and Southeast Asia have the highest unmet demand.
 Financing, not technology, is the primary barrier in these regions.

```
<!-- source: C050 -->
```

(pause)

That concludes this briefing on the solar energy market.

6. PACK.yaml Declaration

Voice views are declared alongside display views in the `views` array, distinguished by their path within `voice/`:

```
views:
  # Display views
  - name: overview
    file: views/overview.md
    purpose: "Executive summary – what it is, where it stands, key risks"
    display_as: "Overview"

  - name: pitch
    file: views/pitch.md
    purpose: "Stakeholder pitch – messaging, objection handling, demo strategy"
    display_as: "Pitch & Demo"

  # Voice views
  - name: briefing
    file: views/voice/briefing.md
    purpose: "Spoken executive briefing – key facts, status, top risk"
    display_as: "Voice Briefing" # Required on all views
    voice: true # Marks this as a voice view
    duration: "~75 seconds" # Expected spoken duration
    pace: measured # brisk | measured | deliberate

  - name: elevator
    file: views/voice/elevator.md
    purpose: "30-second spoken pitch for casual conversation"
    display_as: "Elevator Pitch" # Required on all views
    voice: true
    duration: "~30 seconds"
    pace: brisk
```

The `voice: true` field distinguishes voice views from display views in the manifest. Consumers that do not support voice can skip entries with `voice: true`. Consumers that only want voice views can filter to them.

Locale Voice Views

Translated voice views follow the locale-before-surface pattern:

```
locales:
  canonical: en-US
  available:
```

```
- locale: uk-UA
  display_name: "Українська"
  status: draft
  views: [overview, briefing] # Both display and voice
  derived_from: "claims@2026.03.22"
  translator: machine
```

The `views` list uses view names, not file paths. The tooling resolves `briefing` for locale `uk-UA` to `views/uk/voice/briefing.md` using the `locale-before-surface` convention.

7. When Voice Views Are Loaded

Voice views serve a different consumption context than display views. They are loaded when the AI needs to speak, not when it needs to show.

Context	What loads	Why
Voice conversation (OpenAI Realtime, etc.)	Voice views	The AI is speaking; it needs speakable content
Briefing mode ("brief me on solar energy")	Voice views	The user expects a spoken-style summary even in text
Display context (Canvas, UI panel, chat)	Display views	The user is reading, not listening
AI reasoning about the domain	<code>claims.md</code>	Neither view type — the AI reads claims

Progressive Loading for Voice

The same progressive loading model applies:

Level	Always loaded	Loaded on demand
Within a pack	<code>claims.md</code>	voice views when speaking
Across packs	Hub pack claims	Voice views from relevant pack
Across locales	Canonical (en-US) voice view	Translated voice view by locale

An AI in a voice session loads `claims.md` to become knowledgeable, then renders the appropriate voice view when it is time to deliver information. It does not compose speech from claims on the fly — the voice view exists precisely to avoid that.

8. Voice View Generation

Manual Authoring

Voice views can be hand-written. This produces the highest quality output — a human author naturally writes for the ear. Hand-authored voice views should still include the metadata header and source traceability comments.

`kpack render --voice`

The `kpack render` command generates voice views from claims:

```
# Generate all declared voice views
kpack render my-project.kpack/ --voice

# Generate a specific voice view
kpack render my-project.kpack/ --voice --view briefing

# Generate voice views for a specific locale
kpack render my-project.kpack/ --voice --locale uk-UA

# Check which voice views are stale
kpack render my-project.kpack/ --voice --check
```

The `--voice` flag instructs the renderer to apply voice conventions (short sentences, spelled-out numbers, pause markers, pronunciation hints) instead of display conventions (headers, tables, formatting). The renderer reads `claims.md`, selects claims relevant to the view's declared purpose, and produces spoken-style text.

Generation Prompt Pattern

When an LLM generates a voice view from claims, it should receive instructions equivalent to:

Write this as if you are briefing someone aloud. Use short sentences — fifteen words maximum. Spell out all numbers. No markdown formatting. Add pause markers between topic transitions. Include pronunciation hints for unusual proper nouns. The listener cannot see any text — they hear only your words, one sentence at a time.

Hybrid Workflow

The recommended workflow mirrors display views: generate with `kpack render --voice`, then refine by hand. AI-generated voice views tend to be correct but stilted. A human pass adds natural pacing and conversational rhythm.

9. Staleness Detection

Voice view staleness uses the same mechanism as display views. The generation metadata comment tracks the claims version:

```
<!-- generated: 2026-03-22 | claims: v2026.03.22 -->
```

Planned lint integration would compare the `claims:` version against the current `claims.md` version and emit a staleness warning if they differ. The current reference `kpack lint` does not perform this comparison; voice-view freshness requires editorial checking (see [capability boundaries](#)). Proposed diagnostic:

```
WARN: views/voice/briefing.md is stale (claims: v2026.03.21, current: v2026.03.22)
```

Staleness is a warning, not an error. Voice views may intentionally lag during active claims editing. The planned `--check` flag on `kpack render --voice` would report stale voice views without regenerating them; `render` remains a contract-pointer command.

Staleness Priority

Voice views are more sensitive to staleness than display views. A display view with a slightly outdated number is still scannable and correctable by the reader. A voice view with an outdated number is stated as fact to a listener who cannot cross-reference. Tooling should surface voice view staleness at higher priority than display view staleness.

10. Pace Guidance

The `pace` field provides a tempo hint for speech synthesis or human delivery:

Pace	Words per minute	Use when
brisk	~170 wpm	Elevator pitches, time-constrained updates, action-oriented summaries
measured	~140 wpm	Standard briefings, executive summaries, most voice views
deliberate	~110 wpm	Complex topics, risk discussions, bad news, content requiring absorption time

Pace is advisory. A TTS engine may map it to a speech rate parameter. A human reader uses it as a cue for how much silence to leave between sentences.

Duration Estimation

Duration is estimated from word count and pace:

```
duration = word_count / words_per_minute * 60
```

The `duration` metadata field records the estimate. Authors should verify by reading the view aloud — estimated and actual duration should agree within ten seconds. If they diverge, the content is either too dense (listener cannot absorb at the stated pace) or too sparse (unnecessary padding).

11. Pronunciation Hints

Voice views include pronunciation hints for terms that a standard English TTS engine might mispronounce. The format is the term followed by its pronunciation in parentheses:

```
Qwen (pronounced "chwen")
Hono (pronounced "hoh-no")
Kompanchenko (pronounced "kom-pan-CHEN-ko")
pnpm (pronounced "pee-en-pee-em")
CalVer (pronounced "kal-ver")
```

Rules

1. **First occurrence only.** The hint appears once, at the first mention. Subsequent uses of the term do not include the hint.
2. **English phonetic approximation.** Hints use informal English phonetics, not IPA. The audience is AI systems and non-specialist human readers, not linguists.
3. **Stress marking.** Use ALL CAPS for the stressed syllable when stress placement is non-obvious: "kom-pan-CHEN-ko".
4. **Skip common terms.** Do not add hints for terms that TTS engines handle reliably: common English words, well-known brand names (Google, Microsoft, Amazon), standard technical terms (API, PostgreSQL, JavaScript).
5. **Locale-specific hints.** Translated voice views may need different pronunciation hints. A Ukrainian voice view does not need hints for Ukrainian words but may need them for English technical terms.

12. Relationship to Display Views

Voice views and display views coexist as peer surfaces. Neither is derived from the other. Both are derived from claims.md.

```
claims.md (reasoning surface)
├── views/overview.md (display surface – visual)
└── views/voice/briefing.md (voice surface – auditory)
```

The same knowledge produces fundamentally different artifacts for each channel. A claim like:

```
- [C010] 7 packages, ~85K lines TS, pnpm monorepo
  {0.99|o|E010|2026-03-18} types(6.3K) db(6.6K) api(31K) demo(33.2K)
  matching(4.4K) scaffolding(3.3K) knowledge(650)
```

Becomes this in a display view:

```
| Layer | Technology |
|-----|-----|
| Codebase | ~85K lines TypeScript, 7 packages |
| Monorepo | pnpm |
```

And this in a voice view:

```
The codebase has about eighty-five thousand lines of TypeScript across seven packages.
It is organized as a monorepo using pnpm.
```

Same fact. Three representations. Each optimized for its channel.

When to Create Voice Views

Not every pack needs voice views. Create them when:

- The pack's knowledge will be delivered in voice conversations
- The pack serves a briefing function (meeting prep, status updates)

- The domain includes terms requiring pronunciation guidance
- The pack is used with speech synthesis (OpenAI Realtime, etc.)

Display-only packs (technical architecture, code documentation) rarely need voice views. Briefing-oriented packs (fundraise, project status, meeting prep) almost always do.

13. Bundle Integration

Voice views are included in bundles with the `--include-voice` flag (see `spec/BUNDLE.md`):

```
kpacak bundle my-project --include-voice
```

In the bundle format, voice views use the section marker `--- VOICE: {view-name} ---:`

```
--- VOICE: briefing ---
<!-- voice: briefing -->
<!-- duration: ~75 seconds -->
...spoken content...

--- VOICE: elevator ---
<!-- voice: elevator -->
<!-- duration: ~30 seconds -->
...spoken content...
```

Voice views are excluded from bundles by default. They are also excluded from clipboard bundles (`--clip`), which include only claims and the primary display view. The rationale: clipboard bundles target web chat interfaces where voice is not the consumption mode.

14. Antipatterns

Antipattern	Why it fails	Correct approach
Reading a display view aloud	Tables become incomprehensible, numbers are unspeakable, density overwhelms	Use the voice view
Adding markdown formatting to voice views	Bold markers are spoken or cause TTS glitches	Keep voice views as plain spoken text
Writing voice views longer than three minutes	Listener attention degrades; they cannot rewind	Split into multiple views or reduce scope
Composing speech from claims on the fly	Inconsistent, slow, spends inference tokens on formatting	Render voice views in advance
Using voice views for visual display	Missing structure, too sparse, no tables or formatting	Use display views for visual contexts
Omitting pronunciation hints for domain terms	TTS mispronounces key terms, breaking listener trust	Add hints at first occurrence
Putting claim notation in voice views	<code>{0.95</code>	<code>i</code>

Composition Specification

Status: Draft **Date:** 2026-03-22 **Editor:** Timothy Kompanchenko **Parent:** SPEC.md; design rationale in RATIONALE.md §1, Principle 18. **Implements:** Decision D14 (Meeting pack composition, agenda overlay, pre_load/on_demand)

kpack CLI invocations in this document describe planned reference tooling unless named here: `python3 conformance/run.py`, `./reference/kpack lint`, and `./reference/kpack new` ship today. See SPEC.md §13 and `reference/kpack` for status.

1. Overview

Composition defines how **composite packs** reference standing packs rather than duplicating their knowledge. A composite pack is a lens -- an agenda-shaped, purpose-shaped, or narrative-shaped view over existing packs, augmented with context-specific additions.

The canonical example is a **meeting prep pack**: it references the standing packs relevant to the meeting's agenda, adds meeting-specific claims (who, when, where, what to discuss), and defines a loading strategy that matches the meeting's conversational flow.

The Core Insight

Knowledge lives once, in its canonical pack. A meeting about supply chain diversification does not need a copy of the supply chain claims -- it needs a reference to `supply-chain-diversification.kpack` with a focus on the three claims most relevant to today's agenda. When the meeting produces new knowledge, that knowledge promotes back to the standing pack, not buried in a meeting archive.

What Composition Solves

Problem	Without composition	With composition
Meeting prep duplicates standing knowledge	Pack author copies claims into meeting pack	Meeting pack references standing pack by name
Post-meeting knowledge is trapped	New facts stay in an ephemeral meeting pack	New claims promote to standing packs via reconciliation
Voice assistant has no navigation plan	LLM loads everything or guesses what's relevant	<code>composition.yaml</code> declares <code>pre_load</code> , <code>on_demand</code> , and agenda structure
Context budget is wasted	Full standing packs loaded regardless of meeting focus	<code>focus_claims</code> narrows what gets loaded from each pack

2. composition.yaml

The canonical `composition.yaml` file in a `.kpack/` directory declares composition structure. Its presence signals a composite pack. `PACK.yaml` MAY declare `composition: composition.yaml`; when declared, the file MUST exist. Omitting that manifest field does not prevent detection of a present canonical file. Generic readers do not resolve alternate composition filenames.

Meeting Prep Example (Full)

```
type: meeting-prep

meeting:
  date: 2026-03-24T14:00:00-04:00
  duration: 60m
  participants: [Jane, Dr. Webb]
  location: video-call
  context: >
    Quarterly advisory review. Dr. Webb is a senior energy
    sector advisor. He needs thorough updates on all
```

active research threads and market developments.

agenda:

- topic: "Market forecast update"
reference: "solar-energy-market"
focus_claims: [C001, C003, C030]
talking_points:
 - "Cost floor thesis debate"
 - "Supply chain concentration risk update"
- topic: "Perovskite technology assessment"
reference: "perovskite-assessment"
talking_points:
 - "Oxford PV durability results"
 - "Timeline for commercial viability"
- topic: "GridScale partnership"
reference: "gridscale-partnership"
focus_claims: [C001, C010]
talking_points:
 - "Proposed co-development structure"
 - "Due diligence timeline"
- topic: "Regulatory landscape"
reference: "regulatory-landscape"
talking_points:
 - "IRA incentive reclassification"
 - "EU Carbon Border Adjustment impact"

pre_load:

- solar-energy-market
- perovskite-assessment

on_demand:

- gridscale-partnership
- regulatory-landscape
- supply-chain-diversification
- competitor-analysis
- solar-storage-economics

Field Reference

Top-Level Fields

Field	Required	Type	Description
type	Yes	enum	Composition type. See section 7 for all types
meeting	If type = meeting-prep	object	Meeting metadata
agenda	If type = meeting-prep	list	Ordered agenda items
pre_load	Yes	list of strings	Pack names to load before the session begins
on_demand	No	list of strings	Pack names available for loading when topics arise

meeting Object

Field	Required	Type	Description
date	Yes	ISO 8601 datetime	Meeting start time with timezone
duration	Yes	duration string	Expected duration (30m, 60m, 2h)
participants	Yes	list of strings	People in the meeting
location	No	string	video-call, in-person, phone, or address
context	No	string	Background context for the AI about this meeting

agenda Items

Field	Required	Type	Description
topic	Yes	string	Human-readable agenda item title

Field	Required	Type	Description
reference	No	string	Pack name this topic draws from. Uses name from KNOWLEDGE.yaml
focus_claims	No	list of claim IDs	Specific claims from the referenced pack to emphasize
talking_points	No	list of strings	Key points to cover under this topic

Loading Lists

`pre_load` and `on_demand` contain pack names (the `name` field from KNOWLEDGE.yaml or PACK.yaml). The consuming system resolves these to pack paths.

- **pre_load** packs are loaded into context before the session begins. The AI starts the meeting already knowledgeable about these domains.
- **on_demand** packs are available but not loaded until the conversation enters their territory. This preserves context budget for what matters now.

3. claims.md in Composite Packs

A composite pack's `claims.md` is intentionally minimal. It contains claims about the **composition context** itself -- not about the topics being composed.

What Belongs in a Meeting Pack's claims.md

- Claims about the meeting: who, when, why, what changed since last meeting
- Claims about relationships between participants and topics
- Claims about desired outcomes or decisions needed
- Claims about constraints (time pressure, sensitivity, pending deadlines)

What Does NOT Belong

- Facts about the solar energy market (those live in `solar-energy-market.kpack`)
- Facts about perovskite technology (those live in `perovskite-assessment.kpack`)
- Facts about the GridScale partnership (those live in `gridscale-partnership.kpack`)

Example: Meeting Pack claims.md

```
<!-- KP:1 - Knowledge Pack Format
Claims: - [ID] assertion {confidence|type|evidence|date|depth|nature} context
  Positions 5-6 optional. Verbose named-field syntax also accepted.
Types: o=observed r=reported c=computed i=inferred
Relations: →supports @contradicts @!error @~tension ←requires ~refines @supersedes ↗see_also
Files: evidence.md=sources history.md=past entities.md=graph composition.yaml=references
-->
---
```

pack: webb-advisory-2026-03-24 | v: 2026.03.24 | domain: meetings/advisory
confidence: simple | normalized

Webb Advisory Review [meeting|2026-03-24]

> Quarterly advisory session. Thorough update on market outlook, technology bets, and partnerships.

Meeting Context

- [C001] Quarterly advisory meeting with Dr. Webb, scheduled March 24 2:00pm
{0.99|o|E001|2026-03-22}
- [C002] Last meeting was February 10 -- six weeks since last sync
{0.95|o|E002|2026-03-22} Longer-than-usual gap. Extra context on Q1 developments needed.
- [C003] GridScale partnership timeline shifted from March to May
{0.85|r|E003|2026-03-20} Webb may not be aware of the delay.
↗gridscale-partnership#timeline

- [C004] Decision needed: GridScale co-development terms and due diligence scope
 {0.90|i|E003|2026-03-22} Webb's input on deal structure before GridScale meeting.
 ↻gridscale-partnership
- [C005] Polysilicon supply concentration is the outstanding strategic risk
 {0.90|i|E004|2026-03-22} Webb has contacts at alternative suppliers -- ask about sourcing options.
 ↻supply-chain-diversification#risks

Notice: every claim is about the meeting situation, not about the referenced domains. The ↻ (see_also) cross-pack references point to where the domain knowledge lives. The AI loads those packs to become knowledgeable; the meeting claims tell it what to do with that knowledge in this context.

4. Voice Assistant Integration

Composition is designed to work with the three-process voice architecture described in §4 below ("The Three Processes") and in NOTES.md §7.

The Three Processes

Process	Role in composition	What it reads
Voice LLM	Conversational partner, uses composition for navigation	composition.yaml (agenda, talking points), pre_load pack claims
Recording LLM	Background structured capture, writes new claims	composition.yaml (agenda structure for organizing captures)
Intelligence LLM	On-demand deep analysis when dispatched	Relevant pack claims + conversation context

Pre-Meeting: Loading

1. System reads composition.yaml
2. All pre_load packs are loaded into the Voice LLM's context
3. For each agenda item with focus_claims, those specific claims are highlighted (the Voice LLM knows these are the most relevant points)
4. on_demand pack names are noted -- the Voice LLM knows they exist but their claims are not yet in context
5. The meeting.context field provides situational awareness

During Meeting: Navigation

The Voice LLM uses the agenda list as a navigation structure:

1. **Topic tracking.** As conversation moves between topics, the Voice LLM maps utterances to agenda items. This is approximate -- conversations are not linear.
2. **On-demand loading.** When conversation enters an on_demand topic, the system loads the referenced pack. The Voice LLM signals: "Loading the supply chain diversification details now."
3. **Talking point coverage.** The Voice LLM can track which talking points have been covered and which remain. Near the end of the meeting, it can note: "We haven't discussed the regulatory landscape updates yet."
4. **Focus claim emphasis.** When discussing a topic with focus_claims, the Voice LLM prioritizes those specific claims in its responses. Other claims in the referenced pack are available but secondary.
5. **Cross-topic connections.** When a claim in one agenda topic relates to another topic (via see_also or shared entities), the Voice LLM can surface the connection: "That connects to what we discussed about supply chain concentration earlier."

During Meeting: Recording

The Recording LLM operates as a background process, receiving the transcript stream. Composition gives it structure:

1. **Agenda-organized capture.** New claims are tagged with the agenda item they arose from. The recording LLM uses the agenda list to categorize captures.

2. **Standing pack awareness.** The Recording LLM knows which standing packs exist (from `pre_load` and `on_demand`). When it captures a new claim, it can annotate which standing pack the claim should eventually promote to.
3. **Decision vs. discussion.** The Recording LLM distinguishes between decisions (high confidence, actionable) and discussion (lower confidence, exploratory). Decisions get immediate structured capture; discussion is noted but not over-structured.
4. **Contradiction detection.** When a participant states something that contradicts a claim in a loaded standing pack, the Recording LLM flags it. This is a signal, not an interruption -- the Voice LLM decides whether and when to surface it.

Post-Meeting: Promotion

After the meeting ends, the Recording LLM's captures go through promotion:

1. **Per-agenda-item review.** For each agenda item, review captured claims.
2. **Standing pack routing.** Each new claim is routed to its canonical standing pack based on the `reference` field in the agenda item.
3. **Claim creation.** New claims are drafted for the standing pack with:
 - Evidence referencing the meeting (date, participants, recording if consented)
 - Confidence appropriate to the claim type (decisions get high confidence; impressions get lower)
 - `see_also` back to the meeting pack for provenance
4. **Meeting pack archival.** Once promotion is complete, the meeting pack is a historical record. Its lifecycle type is `ephemeral` -- it archives according to the schedule in its `PACK.yaml`.

Example: Post-Meeting Promotion Flow

Meeting claim captured by Recording LLM:

GridScale timeline moved to June. Webb recommends waiting until after their Q2 earnings report. Decision: Jane will propose June 15 meeting date.

This promotes to `gridscale-partnership.kpack/claims.md` as:

```
- [C015] GridScale meeting target date: June 15, after Q2 earnings release
  {0.90|r|E040|2026-03-24} Decision from Webb advisory session.
  Webb recommended delay; Jane to propose. ~C012
```

With evidence in `gridscale-partnership.kpack/evidence.md`:

```
## E040
> **type:** meeting | **captured:** 2026-03-24
> **source:** Webb advisory session, 2026-03-24 (Jane, Dr. Webb)

Decision during quarterly advisory meeting. Webb noted GridScale
is releasing Q2 earnings in early June. Consensus to target
June 15 for next meeting.
```

The meeting pack retains its original claims as a historical record. The standing pack gains the new knowledge with full provenance.

5. PACK.yaml for Composite Packs

Composite packs declare their composition nature in `PACK.yaml`:

```
name: webb-advisory-2026-03-24
version: 2026.03.24
domain: meetings/advisory
author: Jane Chen
tier: standalone

description: >
  Quarterly advisory review with Dr. Webb. Covers solar market
```

```

forecast, perovskite assessment, GridScale partnership,
and regulatory landscape.

confidence:
  scale: simple
  normalize: true

freshness: "2026-03-24"
sensitivity: internal

lifecycle:
  type: ephemeral
  archive_after_days: 30
  archive_policy: reconcile

notes:
  mode: active
  participants: [Jane, Dr. Webb]
  disclosed: true
  consent: obtained

# Composition reference -- see composition.yaml for full structure
composition: composition.yaml

```

The `composition` field in `PACK.yaml` points to the composition file. Its presence tells tooling and consumers that this is a composite pack.

Lint Rules for Composite Packs

- Every pack name in `pre_load` and `on_demand` must resolve via `KNOWLEDGE.yaml`
- Every reference in agenda items must be either a `pre_load` or `on_demand` pack
- `focus_claims` must reference valid claim IDs in the referenced pack (warning, not error -- claims may have been superseded)
- Meeting date must be a valid ISO 8601 datetime
- Participants list must not be empty

6. Relationship to Standing Packs

Composition is Not Dependency

The `composition.yaml` references are distinct from `PACK.yaml` dependencies:

	dependencies (PACK.yaml)	<code>pre_load</code> / <code>on_demand</code> (composition.yaml)
Meaning	This pack builds on that pack's claims	This pack views that pack's claims through a lens
Loading	Dependency must be available for this pack to be valid	Referenced pack enhances but is not required
Direction	Structural (this pack's claims assume the other pack's claims)	Navigational (this pack points to the other pack for context)
Graceful degradation	Missing dependency = warning, reduced validity	Missing referenced pack = reduced context, pack still usable

No Duplication Rule

A composite pack **MUST NOT** duplicate claims from its referenced packs. If a fact exists in `solar-energy-market.kpack`, the meeting pack references it -- it does not restate it.

Exceptions:

1. **Meeting-specific framing.** A meeting claim may restate a standing claim with meeting-specific context: "Cost floor thesis revised based on new manufacturing data (Webb hasn't seen the latest analysis yet)." This is a claim about the meeting situation, not a duplicate of the cost thesis claim.
2. **Contradictions discovered during meeting.** If a participant states something that contradicts a standing claim, the meeting pack records both the participant's statement and the contradiction reference. This is new knowledge, not duplication.

3. **Temporal snapshots.** A meeting pack may note "As of meeting date, market forecast was X" -- this is provenance, not duplication. The standing pack's claim may have since changed.

7. Composition Types

Meeting prep is the primary composition type, but the `composition.yaml` format supports other composite pack patterns.

meeting-prep

The canonical composition type. Covered in full above.

```
type: meeting-prep
meeting: { ... }
agenda: [ ... ]
pre_load: [ ... ]
on_demand: [ ... ]
```

briefing

A briefing pack composes knowledge for a recipient who needs to get up to speed. Unlike a meeting, there is no interactive agenda -- the briefing is a one-directional knowledge transfer.

```
type: briefing

briefing:
  recipient: "Elena Vasquez"
  purpose: "Quarterly market intelligence briefing"
  date: 2026-03-22
  format: voice           # voice | document | presentation
  duration: 15m           # Target delivery time

sections:
  - heading: "Market forecast"
    reference: "solar-energy-market"
    focus_claims: [C001, C003, C030]
    narrative_hint: "Lead with cost floor debate"

  - heading: "Technology outlook"
    reference: "perovskite-assessment"
    focus_claims: [C011, C012]
    narrative_hint: "Oxford PV durability update"

  - heading: "Competitive landscape"
    reference: "competitor-analysis"
    focus_claims: [C040, C041, C042]

pre_load:
  - solar-energy-market
  - perovskite-assessment
```

Key differences from meeting-prep:

- sections instead of agenda (one-directional, not interactive)
- narrative_hint guides the voice or document renderer on framing
- format declares the intended delivery surface
- Typically no on_demand -- briefings are pre-composed, not exploratory (the schema permits it for every type)

research

A research composition assembles knowledge from multiple packs to investigate a question. The composition defines the research question, relevant packs, and the analytical frame.

```

type: research

research:
  question: "Is perovskite tandem commercially viable by 2028 given current durability
constraints?"
  hypothesis: "Commercial viability achievable if moisture degradation pathway is resolved by
mid-2027"
  date: 2026-03-22
  depth: thorough          # quick | thorough | exhaustive

sources:
- pack: "perovskite-assessment"
  relevance: "Primary subject -- lab efficiency and degradation data"
  focus_claims: [C001, C005, C010, C012]

- pack: "solar-energy-market"
  relevance: "Market context and cost trajectory baseline"
  focus_claims: [C001, C011]

- pack: "webb-advisory-2026-03-24"
  relevance: "Webb's perspective on commercialization timeline"

pre_load:
- perovskite-assessment
- solar-energy-market

on_demand:
- competitor-analysis      # For comparison

```

Key differences from meeting-prep:

- question and hypothesis define the analytical frame
- sources with relevance annotations explain why each pack matters
- depth hints at the level of analysis expected
- Output is analytical claims, not meeting minutes

presentation

A presentation composition assembles knowledge for a slide deck, pitch, or visual narrative. It maps pack claims to presentation structure.

```

type: presentation

presentation:
  title: "Solar Market Outlook 2026-2030"
  audience: "Industry conference attendees"
  date: 2026-03-25
  slide_count: 12          # Target slide count
  style: corporate

slides:
- title: "Cost Trajectory"
  reference: "solar-energy-market"
  focus_claims: [C001, C002]
  visual_hint: "Learning curve chart -- 22% per doubling"

- title: "Technology Roadmap"
  reference: "solar-energy-market"
  focus_claims: [C010, C011, C012]

- title: "Market Adoption"
  reference: "solar-energy-market"
  focus_claims: [C002, C003]
  visual_hint: "Grid parity map by region"

- title: "Systemic Risks"
  reference: "solar-energy-market"

```

```

focus_claims: [C030]

pre_load:
  - solar-energy-market

```

Key differences from meeting-prep:

- slides instead of agenda -- each entry maps to a presentation slide
- visual_hint guides the presentation renderer
- slide_count constrains scope
- style reference for visual consistency
- Typically references fewer packs (focused narrative)

selection

A selection composition is a lens over items that already live in **exactly one** collection. An exhibition catalogue, a loan agreement, an insurance schedule and a valuation report share this shape: a subset of items chosen for a stated purpose, where the items do not move and nothing is duplicated.

Unlike the other four types, a selection has no occasion. `meeting-prep` has a date and participants, `briefing` has a recipient, `presentation` has an audience; a selection has a purpose and a scope.

Selections also overlap and persist -- the same item may sit in a loan, an insurance schedule and a catalogue at once, for different durations, without conflict.

```

type: selection
selection:
  purpose: "Group exhibition at the fictional Example Gallery"
  collection: "example-gallery"
  scope: "Works displayed in the east room"
  prepared_for: "Example Gallery exhibition team"
  date: "2026-08-22"
members:
  - reference: "example-work-001"
  - reference: "example-work-002"
  - reference: "example-work-003"
pre_load:
  - "example-work-001"
  - "example-work-002"
  - "example-work-003"

```

`purpose`, `collection` and `scope` are REQUIRED nonempty strings. `purpose` remains free text, not a closed purpose enum. `members` is required but MAY be empty when the scoped selection has no matches; explain that result in the composition's narrative.

Resolution and time. This core shape is a live lens: member references name packs, not sealed revisions. `pre_load` and `on_demand` control loading, not membership; `members` defines the selection. The `collection` string names a collection but does not authenticate ownership or prove that a member belongs to it. A supporting resolver must use its declared collection authority, report missing or ambiguous references and membership mismatches, and must not silently substitute another pack. The schema check alone establishes none of these facts.

`selection.date` records preparation, not a snapshot of member contents. A historical insurance schedule, loan record or valuation needs an application profile that pins member revisions and retains the referenced artifacts. That profile must declare how references resolve, including unavailable revisions and concurrent branches. This core shape alone cannot reproduce what was known on the stated date. See [COMPATIBILITY.md](#).

Key differences from the other types:

- members instead of agenda/sections/slides -- each entry is an item, not a segment of a narrative
- No occasion field. `purpose` and `scope` carry why the selection exists and what it covers
- `scope` is load-bearing: it states what the selection *excludes*, which is what a third party reading an insurance schedule or a loan agreement needs
- Members are drawn from one collection. A composition spanning several collections is not a selection

The No Duplication Rule is the type's definition, not a constraint on it. For other composite types, referencing rather than restating is a discipline. For a selection it is the whole point: the works remain held by their collection, and the selection pack carries claims *about the selection* -- its scope, its date, what it leaves out and why -- while restating nothing from the works themselves.

8. Context Budget Management

Composition's loading strategy is fundamentally a context budget optimization.

Budget Allocation

For a typical meeting with a 128K-token model:

Component	Budget	Notes
System prompt + instructions	~4K	Fixed overhead
Conversation history	~20-40K	Grows during meeting
pre_load packs (claims only)	~4-12K	2-3 hub packs
on_demand packs (when loaded)	~4-8K	1-2 detail packs at a time
Meeting pack claims.md	~1-2K	Minimal by design
composition.yaml	~0.5K	Agenda and structure
Available for reasoning	~60-90K	Comfortable margin

Loading Priority

When context budget is tight:

1. **Always load:** composition.yaml and the meeting pack's claims.md
2. **Always load:** All pre_load pack claims.md files
3. **Load on trigger:** on_demand packs when conversation enters their territory
4. **Load on citation:** Evidence files when specific citations are needed
5. **Never pre-load:** History files, entity files, views (unless displaying)

Focus Claims Optimization

When a pre_load pack is large (detail tier, ~6-8K tokens), focus_claims provides a secondary optimization. The system can:

1. Load the full claims.md (preferred -- the AI has complete context)
 2. Or, if budget is tight, load only the sections containing focus_claims plus the pack header
- Option 1 is always preferred. Option 2 is a graceful degradation, not the default.

9. Tooling

CLI Commands

```
# Create a meeting prep pack from template
kpack new --template meeting-prep "webb-advisory-2026-03-24"

# Validate composition references
kpack lint webb-advisory-2026-03-24.kpack/
# Checks: pack references resolve, focus_claims exist, dates valid

# Preview what would be loaded
kpack compose webb-advisory-2026-03-24.kpack/ --dry-run
# Shows: total token budget, pre_load packs, on_demand packs

# Generate post-meeting promotion suggestions
kpack reconcile webb-advisory-2026-03-24.kpack/
# Scans meeting pack claims for promotable content
# Outputs: suggested claim additions to standing packs

# Archive after reconciliation
```

```
kpack archive webb-advisory-2026-03-24.kpack/  
# Moves to archive after confirming all claims reconciled
```

Composition Validation (kpack lint)

The shipping conformance runner (python3 conformance/run.py, also reachable as kpack lint) already validates composition.yaml against the published kp-composition.schema.json whenever the file is present (per CORE.md AR-14). The checks below are the planned superset for composite packs:

Check	Severity	Description
Pack references resolve	Error	Every name in pre_load, on_demand, and reference must exist in KNOWLEDGE.yaml
Focus claims exist	Warning	Referenced claim IDs should exist in the referenced pack (warning because claims may have been superseded)
Meeting date is future or recent	Warning	Linting a meeting pack dated >30 days ago suggests it should be archived
No claim duplication	Warning	Claims in the composite pack should not restate claims from referenced packs
Context budget estimate	Info	Estimated total tokens for all pre_load packs

10. Design Principles (Composition-Specific)

These extend the core design principles in RATIONALE.md §1.

1. **Composition over duplication** (RATIONALE.md §1, Principle 18). The foundational rule. Knowledge lives once. Composite packs are lenses, not copies.
2. **Minimal claims.** A composite pack's claims.md should be the smallest file in the ecosystem. 5-10 claims about the composition context, not 40 claims restating domain knowledge.
3. **Agenda as navigation.** The agenda is not documentation -- it is a runtime navigation structure that the Voice LLM uses to track conversation and manage context loading.
4. **Pre_load for certainty, on_demand for possibility.** Pre-load what you know you will discuss. On-demand what you might discuss. When in doubt, on_demand -- context budget is finite.
5. **Promote, don't hoard.** Post-meeting, new knowledge flows outward to standing packs. The meeting pack is a temporary vessel, not a permanent home.
6. **Standing packs are canonical.** If a composite pack's claim conflicts with a standing pack's claim, investigate -- the standing pack is the default source of truth unless the meeting produced genuinely new information.
7. **Composition types share structure.** All composition types use pre_load and on_demand. The type-specific fields (meeting, briefing, research, presentation) are overlays on this shared loading model.

Lifecycle, Archival & Visibility Specification

Status: Draft **Date:** 2026-03-22 **Editor:** Timothy Kompanchenko **Decisions:** D4 (meeting pack lifecycle), D15 (archive-never-delete), D19 (reconcile-before-archive)

kpack CLI invocations in this document describe planned reference tooling unless named here: `python3 conformance/run.py`, `./reference/kpack lint`, and `./reference/kpack new ship today`. See [SPEC.md §13](#) and [reference/kpack](#) for status.

1. Lifecycle Types

Every Knowledge Pack has a lifecycle type that governs how long it remains in the active working set and what happens when it leaves.

Type	Description	Auto-archive	Default retention	Examples
permanent	Standing knowledge. Never auto-archived. Evolves in place through claim supersession.	No	Indefinite	Market analysis, sector coverage, domain expertise
seasonal	Relevant for a bounded period. Archived when the period ends.	Yes, on expiry	90 days (configurable)	Quarterly outlooks, research initiatives, conference prep
ephemeral	Short-lived. Created for a specific event or moment. Archived after retention period.	Yes, on expiry	14 days (configurable)	Meeting prep, call notes, weekly briefings
experience	Interactive runtime / keepsake pack — a narration or voice "room." A runtime class rather than an archival duration; retained as a standing artifact.	No	Indefinite	Narration rooms, voice keepsakes, personal experience packs
working	Active in-progress knowledge for an ongoing engagement. Reconciled in place; consolidates to a standing home when the engagement closes.	No	Until engagement closes	Diligence working notes, project assessments
incident	Time-anchored findings or incident record. Reconciled in place rather than auto-archived.	No	Indefinite	Security findings, incident reports

Default: `permanent`. A pack without an explicit lifecycle type is standing knowledge.

`permanent/seasonal/ephemeral` are the canonical archival-duration trio. `experience`, `working`, and `incident` are reference-implementation-emitted types reconciled into the vocabulary on 2026-06-14 (they were already in active use; the schema enum was widened to match). New producers **SHOULD** prefer the canonical trio unless they target the reference runtime's `experience-pack` behavior.

1.1 Memory Architecture

Standing packs and ephemeral packs serve fundamentally different cognitive functions, analogous to long-term and short-term memory.

Long-term memory (standing packs) is organized by semantic domain. Everything about supply chain risk clusters together regardless of which meeting surfaced it. Everything about a technology trajectory clusters together regardless of which research cycle analyzed it. The organizing principle is *what it's about*. Standing packs grow through claim supersession and evidence accumulation. They are the canonical home of knowledge.

Short-term memory (ephemeral packs) is organized by proximity of operational need. A meeting prep pack clusters claims from multiple domains -- market forecasts, technology assessments, regulatory updates, competitive intelligence -- because they're all needed *together* in the next 30

minutes. The organizing principle is *when and where it's needed*. This is a legitimate cognitive architecture, but it is temporary.

Consolidation makes knowledge available to later work. Useful findings from meeting prep or research can be added to a standing pack with their attribution, scope and references to the retained source record intact. The original event record remains available through archival. This organization is an analogy to memory, not a claim about biological or model memory. Material that remains in the event pack may include:

- **Event procedures** -- scripts, assigned roles, timing and fallback plans. A task instruction for one event need not become standing knowledge; a reusable procedure may belong in a definition or policy pack.
- **Event context** -- attributed experiences, impressions, process notes and decisions. These can remain useful after the event. Keep them in the retained event record, and reference or consolidate them when later work depends on them.

Content Classification Test

These categories guide placement and retention, not whether material has knowledge value. Preserve qualifications when consolidating an event-specific account:

Category	Test	Destination
Lasting subject knowledge	Does it help explain a fact, decision, procedure, relationship or interpretation after the event?	Consolidate where useful, with attribution and a reference to the retained source record
Event instruction	Is it a task, script or procedure specific to this event?	Retain in the event pack; a reusable procedure may have a standing home
Event account	Does its meaning depend on who experienced or decided something, when and under which conditions?	Retain those conditions in the event record; reference or consolidate the account when useful

Examples:

- A sourced production-share estimate → **Lasting subject knowledge**, qualified by its source, scope and period.
- "Jane presents the cost thesis first; Webb responds" → **Event instruction**, retained with that meeting's plan.
- "Elena seemed skeptical about the perovskite timeline" → **Event account**, attributed as the observer's impression, not Elena's established belief. It may inform a later conversation without losing that distinction.

When to Use Each Type

Permanent packs are the backbone of your knowledge graph. They represent things you know -- about markets, technologies, domains, organizations. They grow through claim supersession (old claims get superseded, new claims take their place). They are never "done."

Seasonal packs represent time-bounded knowledge that is authoritative during its period and historical afterward. A quarterly market outlook is the definitive source during the quarter; afterward, its claims should be absorbed into permanent packs (market analysis, technology assessment, regulatory coverage) and the pack archived. Seasonal packs have a natural lifespan measured in weeks or months.

Ephemeral packs bring knowledge together for a specific event. Their consolidation step makes useful findings available in standing packs without losing attribution, context or the route to the original account. The retained event pack may still hold valuable event-specific knowledge. Archiving changes its place in the active working set; it does not make that knowledge disposable. See §3 for retention and discoverability.

2. PACK.yaml Fields

```

lifecycle:
  type: permanent          # permanent | seasonal | ephemeral | experience | working |
incident
  archive_after_days: null  # Auto-archive trigger (null = never for permanent)
  archive_policy: reconcile # reconcile | auto | manual | keep
  reconciled_at: null       # ISO timestamp of last reconciliation check
  orphan_claims: []         # Claim IDs with no match in standing packs

consolidation:             # Knowledge consolidation targets (ephemeral/seasonal packs)
  targets:                 # Standing packs that should receive knowledge claims
    - pack: solar-energy-market # Target pack name (by semantic domain)
      claims: [F001, F002]      # Source claim IDs to consolidate
    - pack: perovskite-assessment
      claims: [F011, F012]
  status: pending          # pending | in-progress | complete
  completed_at: null       # ISO timestamp of consolidation completion

visibility: private        # private | shared | public
  
```

Field Reference

Field	Type	Required	Description
lifecycle.type	enum	No	permanent (default), seasonal, ephemeral, experience, working, incident
lifecycle.archive_after_days	integer or null	No	Days after freshness date before auto-archive triggers. Defaults: null (permanent), 90 (seasonal), 14 (ephemeral)
lifecycle.archive_policy	enum	No	What happens when archive triggers. reconcile (default), auto, manual, keep
lifecycle.reconciled_at	ISO datetime or null	No	Timestamp of last reconciliation. Set by tooling
lifecycle.orphan_claims	list of strings	No	Claim IDs flagged as unreconciled. Set by tooling
consolidation.targets	list of objects	No	Standing packs that should receive knowledge claims. Each entry: pack (name), claims (list of source claim IDs)
consolidation.status	enum	No	pending, in-progress, complete. Tracks consolidation progress
consolidation.completed_at	ISO datetime or null	No	Timestamp when consolidation finished
visibility	enum	No	private (default), shared, public

Archive Policies

Policy	Behavior
reconcile	Default. Run intelligent reconciliation before archiving. Archive only if all claims are reconciled. Flag orphans for human review.
auto	Archive immediately when retention expires. No reconciliation. Use for packs where all claims are already reflected in standing packs by design (e.g., generated summaries, derivative views).
manual	Never auto-archive. User must explicitly archive. For packs that may need indefinite active access despite being time-bounded.
keep	Never auto-archive, and not surfaced for manual-archive prompts. Retain indefinitely as a standing artifact — stronger than manual. Pairs with type: experience for keepsake / runtime packs.

3. Archival

The Rule

Archive, never delete. Packs are archived, not destroyed. Archived packs remain searchable, loadable, and present in git history. Archival moves a pack out of the active working set -- it does not remove it from existence.

Archive Location

Archived packs move to an `_archive/` subdirectory within their category:

```

packs/
├── meetings/
│   ├── webb-advisory-2026-03-22.kpack/    # Active
│   ├── vasquez-briefing-2026-03-25.kpack/ # Active
│   └── _archive/
│       ├── park-supply-review-2026-03-10.kpack/ # Archived
│       └── chen-irena-debrief-2026-03-01.kpack/ # Archived
├── research/
│   ├── solar-energy-market.kpack/          # Active
│   └── _archive/
│       └── solar-market-q4-2025.kpack/     # Archived
└── analysis/
    └── competitor-analysis.kpack/          # Permanent -- never archived

```

What Archival Changes

Aspect	Active	Archived
Location	<code>packs/{category}/</code>	<code>packs/{category}/_archive/</code>
Discoverable via <code>KNOWLEDGE.yaml</code>	Yes	Yes (marked <code>archived: true</code>)
Loadable via <code>load_pack</code>	Yes	Yes
Included in search results	Yes	Yes (lower ranking)
Included in working set for context	Yes	No
Editable	Yes	Read-only by convention
In git history	Yes	Yes

What Archival Does NOT Do

- Does not delete files
- Does not remove from git
- Does not strip claims
- Does not break cross-pack `see_also` references (references to archived packs resolve normally)
- Does not require re-indexing (the pack is the same content at a new path)

4. Consolidation & Reconciliation

The lifecycle of an ephemeral pack has two distinct phases: **consolidation** (the primary purpose) and **reconciliation** (the safety net).

4.1 Consolidation -- The Primary Purpose

Consolidation makes useful findings from an event pack available in standing packs organized by subject. Preserve their attribution, scope and route to the original record. Event accounts can remain in the retained event pack; a standing pack can reference them when later work needs them. Consolidation improves access without making the original account disposable.

When consolidation happens: Ideally, knowledge should be created in standing packs *during prep* (before the event) and referenced by the ephemeral pack. For knowledge discovered during or after the event, consolidation runs as soon as practical -- not deferred until the archive trigger fires.

The consolidation process:

1. **Review each claim's placement** using Section 1.1's lasting subject knowledge, event instruction and event account tests. Event-specific knowledge can still have lasting value.
2. **For material useful in a standing pack**, identify the target by subject; retain event-specific accounts with their conditions intact.
3. **Create or revise the standing record** with proper evidence and relations, following the ordinary claim revision rules. Preserve issued predecessors when their meaning or relations change.

4. **Record the consolidation** using an ordinary `↔target-pack#claim-id` cross-reference plus prose describing where the knowledge was copied. `->consolidated` is not a registered KP:1 relation and **MUST NOT** be emitted as wire syntax
5. **Update PACK.yaml** `consolidation.status` to reflect progress

Material useful in standing packs:

- New facts learned during research or the event → relevant standing pack
- Intelligence about organizations, technologies, markets → domain packs
- Strategic assessments and analyses → strategy packs
- Commitments and decisions → project packs
- Changed assumptions → updates to existing claims in standing packs

Material that may remain in the retained event pack:

- Meeting-specific choreography ("Jane presents cost thesis first; Webb responds")
- Temporal observations ("Elena seemed skeptical about the perovskite timeline")
- Process notes ("We ran 10 minutes over; shorten the technology section next time")
- Fallback plans and behavioral instructions

These examples are placement choices, not a rule that event accounts have no knowledge value. Reference them from standing packs when later work depends on them.

4.2 Reconciliation -- The Safety Net

Reconciliation runs before archival to catch knowledge claims that were missed during consolidation. It solves a psychological problem: **people resist archiving because they fear losing information.**

The Psychology

Information hoarding is a well-documented cognitive pattern. People keep files, emails, and notes they will never read again because the act of discarding triggers anxiety -- "what if I need this?" The Zeigarnik effect compounds this: unfinished cognitive tasks (like "did I capture everything from that meeting?") create persistent mental load until explicitly closed.

The standard advice -- "just delete what you don't need" -- fails because it requires a keep-vs-delete decision for every item. This decision is cognitively expensive and emotionally aversive, so people defer it indefinitely.

Our solution draws from the PARA method insight: reframe "keep vs delete" as "active vs archived."

Both states preserve the information. The only question is whether it's in your working set. This eliminates the loss anxiety.

Reconciliation checks that useful findings have been made available to later work and that event-specific accounts remain retained. A missing standing-pack match calls for a placement decision; it does not establish that the account is worthless. When consolidation was skipped or incomplete, reconciliation identifies material that still needs that decision.

The Three-Step Process

Reconciliation is a three-step process that runs before any ephemeral or seasonal pack is archived:

Step 1: Post-Event Consolidation Review

Verify that consolidation (Section 4.1) was completed. If `consolidation.status` is not complete, run consolidation first. This step should happen soon after the event, not weeks later when context has decayed.

Already consolidated → skip to Step 2.

Not yet consolidated:

- New facts learned during the meeting → relevant standing pack
- Commitments and action items → project or entity packs
- Changed assumptions → updates to existing claims in standing packs
- New relationships discovered → entity updates

What stays in the ephemeral pack:

- Meeting-specific context ("Dr. Webb seemed optimistic about the perovskite timeline")

- Temporal observations ("Polysilicon prices were volatile this week")
- Process notes ("We ran 10 minutes over; shorten the technology section next time")

Step 2: Reconciliation Check

When the archive trigger fires (retention period elapsed), an AI reviews every claim in the ephemeral pack against all standing packs. For each claim, one of three outcomes:

Outcome	Meaning	Action
Reconciled	Claim is reflected in a standing pack (identical, superseded, or refined)	None -- safe to archive
Promoted	Claim was promoted during post-event review (Step 1)	None -- already handled
Orphan	Claim has NO match in any standing pack	Flag for human review

Step 3: Archive with Receipt

If all claims are reconciled or promoted: Auto-archive. Move the pack to `_archive/`. Update `KNOWLEDGE.yaml`. Done.

If orphan claims exist: Do NOT archive. Notify the user with the specific orphan claims listed. The user decides: promote them to a standing pack, or acknowledge that they are meeting-specific context and clear the flag.

Reconciliation Output Example

When `kpack reconcile meetings/webb-advisory-2026-03-22.kpack/` runs, it produces a report like this:

```
Reconciliation Report: webb-advisory-2026-03-22.kpack
=====
Pack: Dr. Webb – Quarterly Advisory Review
Type: ephemeral | Retention: 14 days | Created: 2026-03-22

Claims reviewed: 8
  ✓ Reconciled: 5
  ✓ Promoted: 1
  △ Orphan: 2

— Reconciled —

[C001] Dr. Webb advises three utility-scale solar developers
  → MATCHED in contacts/marcus-webb.kpack [C003]
  "Webb advises SolarFirst, GridScale, and Helios Energy"

[C002] GridScale's pipeline exceeds 2 GW across four states
  → MATCHED in research/gridscale-partnership.kpack [C004]
  "GridScale: 2.1 GW pipeline – Texas, Arizona, Nevada, Colorado"

[C003] Perovskite durability results from Oxford PV were positive
  → MATCHED in research/perovskite-assessment.kpack [C015]
  "Oxford PV Q1 2026 field test: <5% degradation at 18 months"

[C004] Webb suggested connecting us with Helios Energy's CTO
  → MATCHED in research/solar-energy-market.kpack [C046]
  "Warm intro to Helios Energy CTO via Webb (pending)"

[C005] Webb's advisory clients include three thin-film manufacturers
  → MATCHED in contacts/marcus-webb.kpack [C007]
  "Advisory clients: NanoSolar, ThinFilm Corp, FlexPV (thin-film)"

— Promoted (during post-meeting review) —

[C006] Webb wants to see a live comparison of bifacial vs monofacial yield data
  → PROMOTED to research/solar-energy-market.kpack [C047]
  "Bifacial vs monofacial field yield comparison requested by Webb (prepare by Apr 5)"

— Orphan (not found in any standing pack) —

△ [C007] Webb mentioned IRA incentive reclassification expected Q3
```

```
2026 -- may affect GridScale's project economics
{0.70|r|E003|2026-03-22}
NOT FOUND in any standing pack.
Suggested target: research/regulatory-landscape.kpack § IRA Incentives
```

```
△ [C008] GridScale is considering a dedicated perovskite pilot
project, timeline late 2027
{0.60|r|E004|2026-03-22}
NOT FOUND in any standing pack.
Suggested target: research/gridscale-partnership.kpack § Technology Pilots
```

```
Result: BLOCKED -- 2 orphan claims require review.
Run: kpack promote C007 --to research/regulatory-landscape.kpack
Run: kpack promote C008 --to research/gridscale-partnership.kpack
Or:  kpack archive --force webb-advisory-2026-03-22.kpack
      (acknowledges orphans as meeting-specific context)
```

Why This Works

The reconciliation report closes every open loop:

1. **For each claim, the user sees proof** that it landed somewhere permanent. Not "trust us, it's fine" -- actual matched claim IDs in actual packs.
2. **Orphan claims get specific routing suggestions.** The system doesn't just say "2 claims not found." It says "this claim about IRA incentive reclassification should probably go in your regulatory landscape section."
3. **The user can always force-archive.** Sometimes a claim genuinely is meeting-specific context that doesn't belong in any standing pack. The `--force` flag acknowledges this explicitly rather than silently dropping it.
4. **1-2 click retrieval.** Research on personal information management (Jones, 2007; Bergman & Whittaker, 2016) consistently shows that archival systems are trusted only when retrieval is fast and reliable. Archived packs are loadable with the same `load_pack` command, searchable through the same search index, and visible in `KNOWLEDGE.yaml1`. The archive is not a graveyard -- it is a quieter shelf.

5. CLI Commands

```
# Check what would be archived (dry run)
kpack archive --dry-run
# Output: lists all packs past retention with reconciliation status

# Archive all reconciled packs past retention
kpack archive
# Archives packs where all claims are reconciled. Skips packs with orphans.

# Archive a specific pack
kpack archive meetings/webb-advisory-2026-03-22.kpack/

# Force-archive despite orphan claims
kpack archive --force meetings/webb-advisory-2026-03-22.kpack/
# Logs acknowledgment that orphan claims were accepted as-is

# Run reconciliation without archiving
kpack reconcile meetings/webb-advisory-2026-03-22.kpack/
# Produces the reconciliation report shown in Section 4

# Reconcile all ephemeral packs
kpack reconcile --all

# Promote a specific claim to a standing pack
kpack promote C007 --to research/regulatory-landscape.kpack
# Adds the claim to the target pack, marks it as promoted in the source
```

```
# Restore a pack from archive
kpack restore meetings/webb-advisory-2026-03-22.kpack/
# Moves from _archive/ back to active directory
```

The `kpack` commands above describe planned reference tooling. As of v0.8.5-preview, the commands that ship in this repository are `python3 conformance/run.py`, `./reference/kpack lint` (which delegates to it), and `./reference/kpack new` (scaffolds a validated pack from the starter). See [SPEC.md §13 Tooling](#) for status.

6. Claim Supersession Cascade

When a claim is superseded with $\emptyset$, the original moves to `history.md` and a successor claim takes its slot in `claims.md`. Other claims in the same pack may have linked to the superseded claim — typically via `→supports` (this claim provides evidence for the parent), `←requires` (this claim depends on the parent's truth), or `~refines` (this claim adds detail to the parent).

The cascade rule:

Rule (L1). When a claim is superseded with $\emptyset$, every dependent claim that links to it via `→supports`, `←requires`, or `~refines` **MUST** be reviewed by the editor for whether it still stands under the new parent. Dependent claims are NOT automatically invalidated. Review does not authorize removing or re-pointing relations in an issued record. Record a changed support judgment as a separately identified assessment, or issue a successor and retain the complete predecessor in history if its serialized context or relations must change. The original relation still names the original target, including after that target moves to history.

This is the conservative default. It treats the parser as authoritative for syntax and the editor as authoritative for epistemic state.

Why not auto-invalidate

A stricter rule — automatically marking every `←requires` dependent as invalid when its parent is superseded — was considered and rejected for the v0.8 preview line. The reasons:

1. **Many supersessions narrow rather than negate.** A successor claim that refines the parent (e.g., from "the work was made between 1955 and 1965" to "the work was made in 1962") may leave most dependent claims standing under the new parent's truth. Auto-invalidation would force unnecessary supersession of claims that remain accurate.
2. **The graph topology is the editor's responsibility.** Whether a dependent's relation is still meaningful under the new parent is an epistemic call, not a structural one. The format gives the editor the primitives ($\emptyset$, $\sim$, supersession in `history.md`); the editor composes them.
3. **The parser stays simple.** A parser that auto-invalidates a transitive closure across multiple files would need graph-traversal semantics that the current PEG grammar deliberately avoids. The supersession cascade rule lives at the editorial layer, not the parsing layer.

Worked example

Suppose the editor has adopted a dating assessment and used it to choose technical and market comparisons:

```
- [C001] The work was made in 1957.
  {0.83|i|E001|2025-11-15|investigated|judgment}

- [C002] The recorded X-ray fluorescence result is consistent with the artist's documented
  palette for 1955-1959.
  {0.87|o|E002|2025-12-04|investigated} →C001
```

```
- [C003] Recorded lots dated 1955-1959 sold in 2024-2025 at €820,000-€1,180,000.
  {0.99|r|E003|2026-05-09} ⇨C001
```

After a catalog raisonné review, the editor adopts C001-v2 and moves the original C001, unchanged, to `history.md`:

```
- [C001-v2] The work was made in 1962.
  {0.91|i|E004|2026-05-10|investigated|judgment} ⊙C001, ⊙!C001
```

Per Rule L1, the editor reviews C002's support relation and the use of C003 as a comparison:

- **C002** remains an accurate statement of the recorded examination. Retain its issued bullet, including the original `→C001` relation; that target remains the historical C001. Record the changed relevance in a new, separately identified assessment citing C002 and the adopted dating. Do not infer support for C001-v2 from the old edge. If C002's serialized context or relation must change, issue a successor and retain the complete predecessor in history. Explain that the support assessment changed, rather than treating the recorded examination as false. A new examination finding needs its own claim and evidence.
- **C003** remains an accurate report of those lots and their prices. Whether they are suitable comparisons for a work made in 1962 is a separate judgment. Retain the report and record a new comparison assessment or a new dataset if needed. Do not supersede an accurate historical sale report merely because its relevance changed.

A source-content claim such as “the specialist's report dated the work 1957” also remains true after the adopted dating changes. Keep it separate from the adopted world assessment. Record the cascade decisions and their reasons in `history.md` alongside the transition for C001.

Cross-pack supersession

Supersession is local to a pack. A claim in pack A superseded with `⊙` does not propagate to claims in pack B that referenced the original via `⇨` (cross-pack see-also). Cross-pack effects are a `RECONCILIATION.md` concern (currently a stub; full design deferred to v0.9 / v1.0).

The conservative reading for v0.8.5-preview: a cross-pack `⇨` reference is a navigational pointer, not a binding dependency. Pack B's claim with `⇨packA#section` does not require pack A to be reconciled before pack B is published. A section reference (`packA#section`) remains navigation to that section, whose contents may change. A claim-identity reference (`packA#C001`) still names that original claim, including when it has moved to history; it **MUST NOT** be silently retargeted to a successor. Neither form automatically reconciles pack B's conclusions.

7. Visibility

Visibility controls who can access a pack and how it behaves during bundling and sharing operations.

Level	Who can access	Bundle behavior	Sync behavior
private	Pack owner only	Requires <code>--force</code> flag to bundle	Never syncs to shared systems
shared	Named collaborators	Bundles normally; private annotations stripped	Syncs to authorized systems only
public	Anyone	Full content bundled	May be published and distributed freely

Private Packs

Private packs contain knowledge that should not leave the owner's systems. Financial details, personal contacts, health information, legal matters.

Constraints:

- `kpack bundle` on a private pack fails with a warning unless `--force` is passed
- Private packs are excluded from shared search indexes
- Private packs never sync to collaborative platforms (e.g., team knowledge bases)
- Cross-pack `see_also` references TO private packs are allowed (the reference is visible; the content is not accessible to unauthorized consumers)

Shared Packs

Shared packs are accessible to named collaborators. During bundling, **private annotations** (inline notes prefixed with `<!-- PRIVATE: ... -->`) are stripped from the output. The claims and evidence remain; the author's private commentary does not.

```
- [C045] GridScale requested access to the full market analysis dataset (2026-03-22)
  {0.95|o|E045|2026-03-22} Follow-up meeting scheduled for next week.
  <!-- PRIVATE: Elena seemed very interested. Strong partnership signal. -->
```

In a shared bundle, the `<!-- PRIVATE: ... -->` comment is removed.

Public Packs

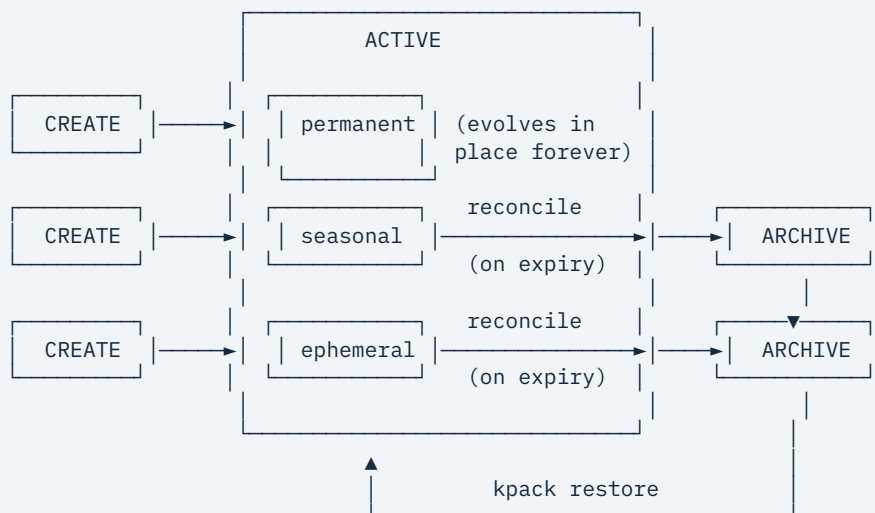
Public packs have no access restrictions. They can be bundled, published, distributed, and referenced without constraint. Open-source project documentation, published research, public market reports.

Interaction with sensitivity

visibility and sensitivity are independent fields. sensitivity (from PACK.yaml) describes the classification of the content. visibility describes who is allowed to access it.

Combination	Valid	Example
sensitivity: confidential + visibility: private	Yes	Proprietary market intelligence
sensitivity: confidential + visibility: shared	Yes	Research shared with co-analysts
sensitivity: confidential + visibility: public	No	Lint error -- confidential content cannot be public
sensitivity: public + visibility: private	Yes	Public data you haven't published yet

8. Lifecycle Transitions



Lifecycle Events

Event	Trigger	Effect
Create	kpack new or manual	Pack enters active working set
Evolve	Claim supersession	Pack stays active, claims update
Archive trigger	archive_after_days elapsed since freshness	Reconciliation check begins
Reconcile	kpack reconcile	Claims reviewed against standing packs
Archive	All reconciled, or --force	Pack moves to _archive/
Restore	kpack restore	Pack moves from _archive/ back to active

Permanent Pack Lifecycle

Permanent packs do not have an archive trigger. They evolve through claim supersession:

1. New information arrives
2. New claim is added (or existing claim superseded with `supersedes`)
3. `freshness date` is updated
4. `version` bumps to new CalVer date

A permanent pack may be manually archived if the domain itself becomes irrelevant (e.g., a superseded technology sector). This is a deliberate human decision, not an automated process.

Seasonal Pack Lifecycle

1. Pack is created for a bounded period (e.g., "Q2 2026 market outlook")
2. Pack is actively used during the period
3. Retention period elapses
4. Reconciliation runs -- valuable claims are checked against standing packs
5. Pack archives to `_archive/`

Ephemeral Pack Lifecycle

1. **Activation** -- Pack is created for a specific event. Research generates knowledge claims. Ideally, novel knowledge is created in standing packs first and referenced by the ephemeral pack.
2. **Event** -- The event occurs. New claims are captured.
3. **Consolidation** -- Make useful findings available in standing packs with their source context and references to original event accounts intact (Section 4.1). Do this while context is fresh.
`consolidation.status` tracks progress.
4. **Retention** -- The active retention period elapses (default: 14 days). The event pack still retains its original accounts, instructions and context.
5. **Reconciliation** -- Review placement of findings and retained event accounts, resolving any outstanding decisions before archival (Section 4.2).
6. **Archive** -- Pack moves to `_archive/` with receipt. Useful findings remain accessible through standing packs and the retained original event record.

9. KNOWLEDGE.yaml Integration

The root `KNOWLEDGE.yaml` index tracks lifecycle and archive status:

```
packs:
- name: solar-energy-market
  path: packs/research/solar-energy-market.kpack/
  lifecycle: permanent
  visibility: shared

- name: webb-advisory-2026-03-22
  path: packs/meetings/webb-advisory-2026-03-22.kpack/
  lifecycle: ephemeral
  visibility: private
  archive_after: 2026-04-05          # Computed: created + 14 days

- name: park-supply-review-2026-03-10
  path: packs/meetings/_archive/park-supply-review-2026-03-10.kpack/
  lifecycle: ephemeral
  visibility: private
  archived: true
  archived_at: 2026-03-25
  reconciliation: complete          # all claims reconciled
```

10. Design Principles

These principles are normative. Tooling that implements lifecycle management **MUST** adhere to them.

1. **Archive, never delete.** (RATIONALE.md §1, Principle 19.) Deletion requires explicit, deliberate action and should be exceptional. Archival is the default lifecycle outcome.
2. **Reconcile before archive.** (RATIONALE.md §1, Principle 21.) Every claim in an ephemeral pack represents knowledge that someone thought was worth capturing. Before burying it in an archive, verify it landed somewhere permanent. Orphan claims are the system catching what humans miss.
3. **Close the loop explicitly.** The reconciliation report is not a formality -- it is the mechanism that makes archival psychologically safe. Users trust the system because it proves nothing was lost, not because it promises nothing was lost.
4. **1-2 click retrieval.** An archived pack must be loadable with the same command as an active pack. Search must include archived packs (with lower ranking). If retrieval from the archive is difficult, users will resist archiving -- and the working set will grow without bound.
5. **Defaults favor safety.** The default lifecycle type is `permanent` (never auto-archived). The default archive policy is `reconcile` (check before archiving). The default visibility is `private` (no accidental sharing). Unsafe behavior requires explicit opt-in.
6. **Visibility is access control, not classification.** `visibility` says who can see the pack. `sensitivity` says what kind of information it contains. These are independent axes. A pack can be publicly accessible but low-sensitivity, or highly sensitive but shared with co-analysts.

Multilingual Support — Knowledge Pack Companion Spec

Date: 2026-03-22 **Status:** Draft

kpack CLI invocations in this document describe planned reference tooling. `python3 conformance/run.py, ./reference/kpack lint` and `./reference/kpack new` ship today. See [SPEC.md §13](#) and the reference launcher at [reference/kpack](#) for status.

1. Scope

This companion spec defines how Knowledge Packs support multiple languages. It covers `PACK.yaml` locale metadata, directory layout for translated views, translation workflow, drift detection, and tooling integration.

Out of scope: The canonical language conventions themselves (American English rules, Merriam-Webster hierarchy, etc.) are specified in `spec/CONVENTIONS.md`.

2. Core Principles

P1. Claims are always English. The reasoning surface (`claims.md`) is written in American English. It is never bilingual, never translated. Translations exist only at the view layer. *Exception:* see §12 "Evidentiary multilingual exception" for an optional structured audit-trail annotation when underlying evidence is inherently multilingual (e.g., field-collected witness statements in Russian or Ukrainian, foreign-language press, recorded interviews). This does not relax P1. Original evidence and attachments keep their language; `extensions.translations` may add structured annotations without making them canonical claims.

P2. Translations are derived views, never authoritative. If a translated view disagrees with the canonical primary files for the pack's kind, those files govern. Claim-bearing packs use `claims.md`; definition/policy packs use their companion documents. Translations carry no independent authority — they are rendered presentations of the canonical knowledge, not alternative sources of truth.

P3. Locale first, then surface. Multilingual views are organized by locale, then by surface type. This makes locale the higher-order grouping, which is cleaner for discovery, grep, and deletion.

P4. Non-English proper nouns preserved in original script. Per standard linguistic conventions, proper nouns (people, companies, places) retain their original-language rendering. A Ukrainian view refers to "Timothy Kompanchenko", not a transliterated form. Ukrainian-origin names use Ukrainian script where that is the original form.

3. `PACK.yaml` — `locales` Section

The `locales` block declares the canonical language and enumerates available translations.

```
locales:
  canonical: en-US          # Canonical language (always American English)
  available:
    - locale: uk-UA        # BCP 47 language tag
      display_name: "Українська" # Native-language display name
      status: draft        # draft | reviewed | verified
      views: [overview, briefing] # Which views have been translated
      derived_from: "claims@2026.03.22" # Pack version the translation was generated from
      translator: machine    # machine | human | hybrid
      reviewed_by: null      # Human reviewer (null if unreviewed)
```

3.1 Field Reference

Field	Required	Type	Description
<code>canonical</code>	Yes	BCP 47 tag	The canonical language. Always en-US.
<code>available</code>	No	List	Translated locales. Omit if monolingual.
<code>available[].locale</code>	Yes	BCP 47 tag	Language tag for the translation (e.g., uk-UA).
<code>available[].display_name</code>	Yes	String	Name of the language in its own script.
<code>available[].status</code>	Yes	Enum	Translation maturity: draft, reviewed, verified.
<code>available[].views</code>	Yes	List	View names that have been translated.
<code>available[].derived_from</code>	Yes	String	Pack version string the translation was built from. Format: <code>claims@{CalVer}</code> .
<code>available[].translator</code>	Yes	Enum	Who performed the translation: machine, human, hybrid.
<code>available[].reviewed_by</code>	No	String	Name of human reviewer. Required when status is reviewed or verified.

3.2 Language Tags

All language tags use **BCP 47** format (IETF BCP 47 / RFC 5646):

Tag	Language
en-US	American English (canonical)
en-GB	British English (en-US sibling — see §3.3)
uk-UA	Ukrainian
de-DE	German
fr-FR	French
ja-JP	Japanese
ar-SA	Arabic

The subtag after the hyphen is the region. Use the most specific applicable tag. The short form (e.g., uk without -UA) is acceptable in directory names for brevity.

en-GB is treated as a *mechanical sibling* of en-US, not a translation in the full sense: the two share semantics and structure, differing in spelling (colour/color, organise/organize), idiom (boot/trunk, lift/elevator), and a small set of dialect-specific terms. Tooling MAY auto-derive an en-GB view from en-US via spelling normalization plus a curated idiom list; full re-translation is not required. Other en-* tags (en-CA, en-AU, en-IN, etc.) are reserved for future use; producers SHOULD NOT mint custom en-* tags without filing a spec note.

3.3 Register Sub-distinctions

Within a single locale, voice content may legitimately need to vary by **register** beyond the four primary registers defined in [VOICE.md §4.2](#). Locale-specific sub-registers refine within that primary axis rather than replacing it.

Examples:

- **Ukrainian.** Literary Ukrainian (the standard reviewed by Kyiv-based editors) and Western Ukrainian / Halychyna register (Lviv-area diction, vocabulary differences, occasional loanwords from Polish or German) sound noticeably different in voice surfaces. A pack targeting a Western Ukrainian audience SHOULD declare the sub-register in voice view metadata so producers and reviewers can converge.
- **French.** Parisian / Métropolitain French and Quebec French differ in vocabulary, idiom, and prosody. A French voice view should be explicit about which.
- **Spanish.** European (peninsular) Spanish and Latin American Spanish differ in pronunciation (vosotros use, s/z/c distinctions), vocabulary, and rhythm.
- **Arabic.** Modern Standard Arabic (MSA) is the literary register; spoken dialects (Egyptian, Levantine, Gulf, Maghrebi) are markedly different and not mutually intelligible by voice.

When sub-register matters for a pack, producers SHOULD record it in the voice view's metadata header as a free-form annotation alongside `register`. Example:

```
<!-- voice: briefing -->
<!-- duration: ~90 seconds -->
<!-- pace: measured -->
<!-- register: curatorial -->
<!-- sub_register: uk-halychyna -->
```

The `sub_register` field is informational, not a normative enum; values are agreed locally by producer and reviewer. The four primary `register` values from VOICE.md remain the authoritative axis.

`sub_register` adds locale-specific nuance without expanding the spec-level vocabulary.

Reviewer responsibility. A pack declaring `sub_register: uk-halychyna` **MUST** be reviewed by a native speaker of that sub-register — generic Ukrainian review is insufficient. Producers without access to a qualified reviewer **SHOULD** ship the pack at literary-Ukrainian register and tag the sub-register field as `null` or omit it, rather than ship unverified locale-specific prose.

4. Directory Structure

Translated views live in locale subdirectories under `views/`. The locale directory mirrors the structure of the root `views/` directory.

```
{name}.kpack/
├── PACK.yaml
├── claims.md
├── evidence.md
├── views/
│   ├── overview.md
│   ├── architecture.md
│   ├── voice/
│   │   ├── briefing.md
│   │   └── uk/
│   │       ├── overview.md
│   │       ├── architecture.md
│   │       └── voice/
│   │           ├── briefing.md
```

Always English – never translated

English display view

English display view

English voice view

Ukrainian locale directory

Ukrainian display view

Ukrainian display view

Ukrainian voice view

Layout Rules

1. **Locale directory name** **SHOULD** use the short language code when it uniquely identifies a declared locale. If two locales share that language (for example `pt-BR` and `pt-PT`), use their full BCP 47 tags. A reader resolves an exact declared-tag directory first, then a short-language directory only when the mapping is unambiguous; it must report ambiguity rather than silently choosing a sibling. The same mapping applies to declared view and voice paths.
2. **Locale before surface type.** Ukrainian voice briefing is `views/uk/voice/briefing.md`, not `views/voice/uk/briefing.md`. Rationale: locale is the higher-order grouping. You can `rm -rf views/uk/` to remove all Ukrainian content. You can `ls views/uk/` to see everything available in Ukrainian.
3. **Mirror structure, not filenames.** The translated `views/uk/overview.md` corresponds to `views/overview.md`. No filename suffixes (not `overview.uk.md`). The directory provides the locale signal.
4. **Selective translation.** Not every view needs to be translated. The `views` list in the `locales` section declares which views exist for each locale. Unlisted views are not available in that language.
5. **Canonical assertions stay English.** Translated presentation lives under `views/`; active and historical claim assertions remain English. Original evidence excerpts and source artifacts retain their original language. See §12 for transcripts and audit fidelity.

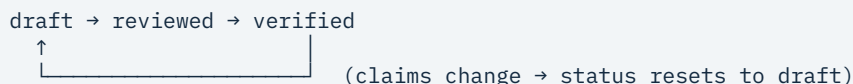
5. Translation Status

Each locale has a `status` field tracking translation maturity.

Status	Meaning	Requirements
draft	Machine-generated or initial human translation	<code>translator</code> field set. No human review.

Status	Meaning	Requirements
reviewed	A human has read the translation and confirmed accuracy	reviewed_by field set. Factual accuracy verified against claims.
verified	A domain-fluent native speaker has approved the translation	reviewed_by field set. Linguistic quality and domain terminology confirmed.

Status Transitions



When `claims.md` is updated, all translations that derived from the previous version reset to `draft`. The `derived_from` field determines whether a translation is current or stale. Status never auto-upgrades — it requires explicit human action.

6. Translation Drift Detection

Translation drift occurs when `claims.md` has been updated but translated views still reflect a previous version.

Detection Mechanism

Each translated view includes a generation metadata comment:

```
<!-- translated: 2026-03-22 | derived_from: claims@2026.03.22 | locale: uk-UA | translator:
machine -->
```

The `derived_from` value is compared against the current pack version in `PACK.yaml`. If they differ, the translation is stale.

derived_from Format

```
claims@{CalVer}
```

Example: `claims@2026.03.22` means the translation was generated from the `2026.03.22` version of `claims.md`.

Staleness Rules

1. **Stale translation** = `derived_from` version does not match the current pack version in `PACK.yaml`.
2. **Stale translations are warnings, not errors.** A translation may intentionally lag during active editing of claims.
3. **Stale translations reset status to draft.** A previously `verified` translation becomes `draft` when `claims` change, because the content it was verified against no longer matches.
4. **Planned lint integration reports stale translations.** It would compare each locale's `derived_from` against the current pack version and emit a warning for mismatches. The current reference `kpack lint` does not perform this check; translation freshness requires editorial checking (see [capability boundaries](#)).

7. Translation Workflow

Generation

```
kpack translate my-project.kpack/ --locale uk-UA
```

1. Reads `claims.md` to understand the domain knowledge.
2. Reads the English view files listed for the target locale.
3. Generates translated view files in `views/{locale}/`.
4. Inserts the generation metadata comment in each translated file.

5. Updates `PACK.yaml` — sets `derived_from` to the current pack version, `status` to draft, `translator` to machine.

Review

A human reviewer reads the translated views, corrects errors, and updates the locale entry:

```
- locale: uk-UA
  status: reviewed           # upgraded from draft
  derived_from: "claims@2026.03.22"
  translator: hybrid        # machine + human corrections
  reviewed_by: "Olena Shevchenko"
```

Verification

A domain-fluent native speaker confirms both linguistic quality and domain-specific terminology:

```
- locale: uk-UA
  status: verified          # upgraded from reviewed
  reviewed_by: "Olena Shevchenko"
```

Re-translation on Claims Change

When `claims.md` is updated (new pack version):

1. Run `kpack translate --check my-project.kpack/` to identify stale translations.
2. Run `kpack translate my-project.kpack/ --locale uk-UA` to regenerate drafts.
3. Human review cycle repeats.

Check Command

```
kpack translate --check my-project.kpack/
```

Output:

```
my-project.kpack/
uk-UA: STALE (derived_from: claims@2026.03.18, current: 2026.03.22)
  views/uk/overview.md — needs re-translation
  views/uk/voice/briefing.md — needs re-translation
```

8. Lint Integration

The following multilingual checks are planned for `kpack lint`. They are not implemented by the current reference validator; check these obligations manually when authoring or reviewing translations (see [capability boundaries](#)).

Check	Severity	Description
Locale directory exists	Error	Every locale in available must have a corresponding <code>views/{locale}/</code> directory.
Declared views exist	Error	Every view listed in a locale's <code>views</code> array must exist as a file in <code>views/{locale}/</code> .
<code>derived_from</code> present	Error	Every locale entry must have a <code>derived_from</code> field.
Translation freshness	Warning	<code>derived_from</code> version differs from current pack version.
Generation comment present	Warning	Translated view files should contain the <code><!-- translated: ... --></code> metadata comment.
Reviewer required	Warning	Locales with <code>status: reviewed</code> or <code>status: verified</code> should have <code>reviewed_by</code> set.
Orphan locale directories	Warning	<code>views/{locale}/</code> directories that are not declared in <code>PACK.yaml</code> <code>locales.available</code> .

9. Translated View File Format

Translated views follow the same format as English views (standard GFM markdown) with two additions:

Generation Metadata Comment

Every translated view begins with a generation metadata comment:

```
<!-- translated: 2026-03-22 | derived_from: claims@2026.03.22 | locale: uk-UA | translator:
machine -->
```

Field	Description
translated	Date the translation was generated or last updated.
derived_from	Pack version the translation was built from.
locale	BCP 47 tag of the target language.
translator	Who performed the translation: machine, human, hybrid.

Source Traceability

Translated views preserve the same `<!-- source: C001, C002 -->` comments as English views. Claim IDs are language-independent — they always reference the English `claims.md`.

Proper Noun Handling

Non-English proper nouns are preserved in their original script:

- **Ukrainian names in Ukrainian text:** Use Ukrainian script (e.g., "Тимофій Компанченко").
- **English names in Ukrainian text:** Keep Latin script (e.g., "Timothy Kompanchenko" remains as-is — do not transliterate to Cyrillic).
- **Company/product names:** Retain original form unless an established localized name exists (e.g., "Google" stays "Google", not "Гугл").
- **Technical terms:** Preserve English technical terms where no standard translation exists in the target language. Provide a parenthetical gloss on first use if the term may be unfamiliar.

10. First Target Locale: Ukrainian (uk-UA)

Ukrainian is the first non-English locale for Knowledge Packs. Implementation notes:

- **Script:** Cyrillic. No RTL considerations.
- **BCP 47 tag:** `uk-UA`
- **Directory name:** `uk`
- **Display name:** Українська
- **Use case:** Knowledge packs readable by Ukrainian-speaking family members and collaborators.
- **Initial translator:** `machine` (LLM-generated), with human review planned.

Example Pack Structure

```
solar-energy-market.kpack/
├── PACK.yaml
├── claims.md                                # English
├── views/
│   ├── overview.md                        # English
│   ├── voice/
│   │   └── briefing.md                    # English
│   └── uk/
│       ├── overview.md                    # Ukrainian
│       └── voice/
│           └── briefing.md                # Ukrainian
```

Example PACK.yaml

```
name: solar-energy-market
version: 2026.03.22
domain: energy/market-analysis

locales:
  canonical: en-US
  available:
    - locale: uk-UA
      display_name: "Українська"
      status: draft
      views: [overview, briefing]
      derived_from: "claims@2026.03.22"
      translator: machine
      reviewed_by: null
```

11. Future Considerations

These are not specified in this version but inform the design:

- **Right-to-left scripts (Arabic, Hebrew).** The directory convention and metadata format support RTL languages without changes. Rendering is a platform concern, not a pack concern.
- **CJK languages.** No structural changes needed. Character encoding is UTF-8 throughout.
- **Claim-level language anchoring.** Some claims contain language-specific content (quotes, proper nouns). A future spec version may add per-claim language markers. For now, the pack-level `linguistic_epoch` field in `PACK.yaml` provides sufficient anchoring.
- **Translation memory.** Repeated translations of similar packs could benefit from shared glossaries or translation memory. Not specified here — this is a tooling optimization, not a format concern.
- **Automated quality scoring.** Machine translations could carry a quality confidence score. Deferred until there is enough translation volume to calibrate such scores.

12. Evidentiary multilingual exception (normative)

Principle P1 — *claims are always English* — is the default rule for KP:1 packs. Original evidence may retain its language. The optional annotation below does not create translated canonical claims or relax P1.

When a pack's underlying evidence is **inherently multilingual and evidentiary** — handwritten witness statements in Russian or Ukrainian, foreign-language press reports, recorded interviews, court records, or any source whose canonicalization to English would lose evidentiary fidelity — the pack MAY carry the original-language transcripts as audit trail. The English claim remains the single normative assertion.

Optional structured location. Original source artifacts may remain in evidence records or attachments. Structured original-language transcript annotations MAY live in `extensions.translations` (per `EXTENSIONS.md` §2.8). They MUST NOT be promoted to `claims.md`, MUST NOT appear as parallel claim text, and MUST NOT trigger a "co-canonical" reading. The translations block is evidence-shaped, not claim-shaped:

- The English claim in `claims.md` is the assertion.
- The original-language transcript in `extensions.translations` is the audit trail backing it.
- The claim is what the pack currently asserts; disagreement with an original source is a finding requiring review, not permission to replace or discard the source. Preserve the original and flag translation drift.

Cardinality and custody. The current translations map holds only one transcript per locale. It cannot represent several independent statements in that language; retain distinct original evidence records/attachments and their IDs instead of overwriting or merging transcripts. A structured multi-transcript profile requires its own versioned mapping.

Producer responsibility. When a producer chooses this path, the pack SHOULD also include an explicit narrative-prose acknowledgment in `evidence.md` that the underlying source material is non-English, naming the language(s). This makes the pack's evidentiary character legible to readers without requiring them to inspect the manifest.

Scope limit. This exception exists for *evidentiary multilingual domains* — content where the original language is the source of truth and the English claim is a derived translation. It does NOT extend to:

- Convenience translations of internally-authored prose (those belong in `views/{locale}/`).
- Co-canonical multilingual claims (rejected in v0.8.0 per Decision Log D1; revisit only if a new pack class genuinely warrants it).
- Bilingual reasoning surfaces (also rejected in D1).

Inaugural producer. Witness statements collected in the field were authored in Russian and Ukrainian; English claims are derived from them. The transcripts in `extensions.translations.{ru,uk}` are the audit trail; the English `claims.md` is what the pack asserts.

13. Decision Log

#	Decision	Alternatives Considered	Rationale
D1	Canonical language is American English; claims are never bilingual	Bilingual claims, per-claim language tags	Complexity explosion. Claims are the reasoning surface — one language keeps them unambiguous. Translations belong at the view layer.
D1a	Evidentiary multilingual exception: original-language transcripts MAY ship in <code>extensions.translations</code> for evidentiary domains where canonicalization to English loses fidelity (witness statements collected in the field, foreign-language court records, recorded interviews)	Co-canonical claims (rejected per D1), per-claim language tags (rejected per D1), requiring every source to fit a single transcript map	Optional <code>extensions.translations</code> annotations (EXTENSIONS.md §2.8) add machine-readable context while preserving original evidence/ attachments and D1's "claims are English" rule. The English claim remains the assertion; the transcript is evidence-shaped. See §12.
D2	Translation storage uses locale subdirectories (<code>views/{locale}/</code>)	Filename suffixes (<code>overview.uk.md</code>), flat directories	Subdirectories are cleaner for discovery (<code>ls views/uk/</code>), removal (<code>rm -rf views/uk/</code>), and grep. Filename suffixes create visual clutter and complicate glob patterns.
D2a	Locale before surface type (<code>views/uk/voice/</code>)	Surface before locale (<code>views/voice/uk/</code>)	Locale is the higher-order grouping. A locale is a complete translation; a surface type is a rendering format. You add/remove entire locales, rarely individual surface types within a locale.
D2b	Short language codes when unambiguous; full BCP 47 tags for same-language siblings	Always-short directories	Preserve readable defaults without collapsing regional/script variants. Full tags remain in <code>PACK.yaml</code> metadata; exact tag paths take precedence.
D2c	Translations are derived views, never authoritative	Co-equal translations, authoritative translations	Canonical primary files remain authoritative. Translated views are presentations; disagreement requires review of the view while preserving the original evidence.

Pack Organization — Knowledge Pack Companion Spec

Date: 2026-03-22 **Status:** Draft **Decisions:** D3 (spec location), D12 (nested categories)

1. Repository Structure

A knowledge pack workspace separates the format specification, the pack content, and the supporting tooling. The example below shows one reasonable layout — implementations are free to organize differently as long as the pack format itself is preserved.

```

your-knowledge-repo/
├── spec/
│   ├── CORE.md
│   ├── SPEC.md
│   ├── README.md
│   ├── VOICE.md
│   ├── COMPOSITION.md
│   ├── LIFECYCLE.md
│   ├── MULTILINGUAL.md
│   ├── ORGANIZATION.md
│   ├── CONSISTENCY.md
│   ├── CONVENTIONS.md
│   ├── STORAGE.md
│   ├── BUNDLE.md
│   ├── ARCHIVE.md
│   ├── DEFINITIONS.md
│   ├── EXTENSIONS.md
│   ├── NOTES.md
│   ├── PLAYBACK.md
│   ├── RECONCILIATION.md
│   ├── MAPPING.md
│   ├── RATIONALE.md
│   └── CHANGELOG.md
│   # THE FORMAT — defines what a pack is
│   # Implementer surface (parser/validator contract)
│   # Full normative spec + rationale + ecosystem
│   # Lane overview (CORE / SPEC / companions)
│   # Voice view format
│   # Meeting pack composition
│   # Pack lifecycle management
│   # Locale conventions
│   # This file (pack categories and repo layout)
│   # Cross-pack consistency patrol
│   # Linguistic conventions
│   # Storage formats and serialization
│   # Export/sharing format
│   # ZIP archive, integrity chain, signatures
│   # Definition and policy document kinds
│   # Producer-defined `extensions.*` blocks catalogue
│   # AI note-taking metadata
│   # Self-driving presentation playback (experimental)
│   # Cross-pack reconciliation (stub — design deferred)
│   # Mapping to RDF/JSON-LD/PROV-O (informative)
│   # Design principles, positioning, philosophy (informative)
│   # Version history
├── packs/
│   ├── KNOWLEDGE.yaml
│   ├── <domain-a>/
│   │   └── <pack-1>.kpack/
│   ├── <domain-b>/
│   │   ├── <pack-2>.kpack/
│   │   └── <pack-3>.kpack/
│   ├── meetings/
│   │   └── _archive/
│   └── (additional categories as needed)
│   # THE CONTENT — actual knowledge packs
│   # Pack index (discovery and routing)
│   # Domain-grouped standing packs
│   # Ephemeral meeting packs
│   # Auto-archived past meetings
├── templates/
│   ├── meeting-prep.template/
│   └── project.template/
│   # Pack scaffolds by category
├── styles/
├── inbox/
└── tools/
    # Narrative style definitions
    # Unprocessed incoming material
    # CLI tooling (e.g., a `kpack` command)
  
```

The categories under `packs/` (`meetings/`, plus whatever domain folders fit your work) are illustrative. The only requirement is that each `.kpack/` directory follows the file structure defined in `SPEC.md`.

2. The Two-Zone Model

The repository has two zones:

Zone	Contents	Rule
packs/	Clean KP:1 knowledge packs only	Everything here is structured, validated, trustworthy
Everything else	Legacy data, tooling, process files	Untouched until ready to migrate

The Amnesia Test

If you looked at this repository with no context, you should be able to tell what's what:

- `spec/` → "this defines a format"
- `packs/` → "this contains structured knowledge"
- `packs/energy/solar-energy-market.kpack/` → "this is a knowledge pack about the solar energy market, filed under energy"
- `entities/` → "this is older, unstructured data"

The `.kpack` directory extension is the ultimate differentiator. Any directory ending in `.kpack` is a knowledge pack. Nothing else uses this extension.

3. Nested Categories

Packs are organized into category subdirectories within `packs/`. Categories provide cognitive clarity — you know where to look.

Working Set (Not Canonical)

These categories are a starting point. They will evolve through use as more packs are created and migrated:

Category	Purpose	Example packs
<code>energy/</code>	Market analysis, technology, policy	<code>solar-energy-market</code> , <code>wind-energy-market</code>
<code>meetings/</code>	Ephemeral, per-meeting, auto-archived	<code><topic>-<YYYY-MM-DD></code>
<code>creative/</code>	Personal creative projects	<code><your-pack-name></code>
<code>technical/</code>	Domain-agnostic technical knowledge	<code>ai-models</code> , <code>architecture-decisions</code>

Additional categories will emerge as migration surfaces the need. The principle is:

- **Resist proliferation** — When in doubt, use an existing category
- **Split when cognitive load demands it** — If a category has 20+ packs spanning clearly different domains, split
- **Categories can be renamed** — Renaming is cheap (git move). Don't agonize over names early.

What Doesn't Go in Categories

- The `KNOWLEDGE.yaml` index stays at `packs/` root (not inside a category)
- Templates stay in `templates/` (not inside `packs/`)
- The `spec` stays in `spec/` (not inside `packs/`)

4. Migration Strategy

Legacy data migrates from the old zone to `packs/` gradually, on demand:

Migration Workflow

1. Pick something from the old zone (e.g., `entities/companies/acme-solar.md`)
2. Determine which pack it belongs to (e.g., business pack, or a contacts pack)
3. Convert the information into claims and evidence
4. Add to existing pack or create new pack in `packs/{category}/`
5. Delete the source from the old zone (or leave it — no pressure)

Migration Principles

- **No rush** — Migrate when you touch something, not all at once
- **Don't restructure the old zone** — Leave legacy directories exactly as they are
- **One-way migration** — Knowledge flows from old to new, never back

- **The old zone shrinks over time** — Eventually it's empty and can be deleted
- **Not everything migrates** — Some legacy content is truly obsolete and can be left or deleted

5. KNOWLEDGE.yaml Updates

The pack index must reflect the category structure. Paths are relative to `packs/`:

```
packs:
  # — Energy —
  - name: solar-energy-market
    path: energy/solar-energy-market.kpack/
    tier: hub
    category: energy
    summary: "Utility-scale solar market trends, costs, and adoption"
    topics: [solar, PV, market, cost, adoption]

  # — Technical —
  - name: battery-storage-economics
    path: technical/battery-storage-economics.kpack/
    tier: hub
    category: technical
    summary: "Battery storage cost trajectories, chemistry, deployment economics"
    topics: [battery, storage, lithium, cost, deployment]

  # — Meetings —
  - name: vendor-review-2025-Q4
    path: meetings/vendor-review-2025-Q4.kpack/
    tier: standalone
    category: meetings
    lifecycle: ephemeral
    summary: "Vendor review – supply chain status, module pricing"
    topics: [meeting, vendor, supply-chain]
```

The `category` field is informational (the directory structure is authoritative), but it enables filtering: "show me all business packs" without directory traversal.

Cross-Pack Consistency — Knowledge Pack Companion Spec

Date: 2026-03-22 **Status:** Draft **Decisions:** D18 (automated consistency checking)

kpack CLI invocations in this document describe planned reference tooling unless named here: `python3 conformance/run.py`, `./reference/kpack lint`, and `./reference/kpack new` ship today. See [SPEC.md §13](#) and [reference/kpack](#) for status.

1. Purpose

As the number of knowledge packs grows, the same entities appear across multiple packs. Claims about the same entity in different packs can drift, contradict, or become stale independently. Cross-pack consistency checking detects these issues automatically.

2. Issue Types

Issue	Description	Example	Severity
Contradiction (unqualified ☹)	Same entity, conflicting claims — not yet assessed	Market pack says "TAM \$50B", proposal says "TAM \$2B"	High
Contradiction (error ☹!)	One claim is wrong — prioritize resolution	Market size claim uses outdated methodology	Critical
Contradiction (tension ☹~)	Both informative — productive disagreement, do NOT flag	Different methodologies give different TAM; both valid	Skip
Stale claim	Claim's temporal anchor is old, likely outdated	Team size claim from 6 months ago	Medium
Stale dispute	Disputed claims (☹ or ☹!) unresolved for >90 days	Two contradicting claims with no new evidence since January	Medium
Broken reference	↪pack_name points to nonexistent pack	↪competitor-analysis but pack doesn't exist	High
Confidence decay	Low-confidence claim older than 90 days, unreviewed	{0.70} from 4 months ago	Low
Orphan pack	Pack not referenced by any other pack or KNOWLEDGE.yaml	Disconnected pack with no inbound links	Low
Duplicate claims	Same fact in two packs with different confidence	Both market pack and deployment pack claim team size	Medium
Evidence link rot	Evidence references a URL or document that no longer exists	Source URL returns 404	Medium
Unexplained process references	Incidental process narration or a request/tool name used as sufficient justification for a conclusion	A claim cites a meeting without explaining its relevant account or decision	Medium
Canonical duplication	Same concept independently defined in multiple packs without cross-reference	Six-layer stack fully restated in pitch pack instead of referencing hub pack	High

3. Pack Patrol

Cross-pack consistency checks integrate with the existing patrol system. A new `knowledge` category of checks runs against the `packs/` directory.

Configuration

```
# config/patrol.yaml (addition)
categories:
```

```
knowledge:
  path: packs/
  checks:
    - cross-pack-consistency      # Same entity, conflicting claims
    - claim-staleness            # Temporal anchor + freshness check
    - reference-integrity        # ⇔ cross-references resolve
    - confidence-decay           # Old low-confidence claims
    - orphan-detection           # Packs with no inbound references
    - evidence-link-check        # Evidence URLs/paths exist
    - duplicate-detection        # Same fact in multiple packs
    - provenance-contamination  # Process language in claims or views
    - canonical-duplication      # Same concept defined without cross-reference
```

CLI

```
kpack patrol packs/                # Run all consistency checks
kpack patrol packs/ --check cross-pack  # Run specific check
kpack patrol packs/ --pack solar-energy-market  # Check one pack against all others
kpack patrol packs/ --fix            # Interactive fix mode
```

Output Format

```
Pack Patrol Report — 2026-03-22

CONTRADICTIONS (2 found)
  ▲ solar-energy-market.kpack C007 vs competitor-analysis.kpack C003
    market: "TAM is $50 billion" {0.85}
    competitor: "Addressable market is $2 billion" {0.70}
    Likely: different scope (TAM vs SAM). Consider adding scope qualifier.

  ▲ solar-energy-market.kpack C021 vs solar-deployment-risks.kpack C005
    market: "Team: 7 analysts" {0.95}
    deployment: "Core team: 3 analysts" {0.90}
    Likely: different definition (full team vs core team). Consistent.

STALE CLAIMS (5 found)
  i solar-energy-market.kpack C030 — "polysilicon prices stable" dated 2026-03-01 (21 days)
    Confidence: 0.90. Recommend refresh.

BROKEN REFERENCES (0 found)
  ✅ All cross-references resolve.

ORPHAN PACKS (1 found)
  i solar-storage-economics.kpack — Not referenced by KNOWLEDGE.yaml
    Action: Add to KNOWLEDGE.yaml or link from hub pack.
```

4. Real-Time Consistency Checking

Beyond scheduled patrol scans, the system supports real-time consistency checking during voice conversations. This is specified in `spec/NOTES.md §8`.

Architecture

The recording LLM (from the three-LLM architecture) continuously cross-references spoken claims against loaded knowledge packs:

1. A claim is made in conversation (by any participant)
2. Recording LLM matches it against the nearest claim in loaded packs
3. If confidence-weighted distance exceeds threshold → alert
4. Alert delivered via visual channel (Canvas, notification)

Alert Design

Alerts are **non-accusatory**. They do not say "this is wrong." They say "this doesn't match what we have." The user determines the correct interpretation:

- "Maybe the pack is stale and the spoken claim is the update"
- "Maybe the speaker misspoke"
- "Maybe they're talking about a different scope"

```
△ Pack: solar-energy-market C007 says "TAM is $50 billion" {0.85}
  Spoken: "The market is about $500 million"
  Possible: different scope? Worth clarifying.
```

Alert Threshold

Only claims with confidence above the configured threshold trigger alerts. Low-confidence claims are already uncertain — alerting on them creates noise.

```
# In PACK.yaml notes section
notes:
  alert_threshold: 0.70    # Only alert when contradicting claims ≥ 0.70
```

5. Consistency Scan Schedule

Scan Type	Frequency	Scope	Cost
Real-time	During conversations	Loaded packs only	Per-token (recording LLM)
On-demand	Manual trigger	Selected packs	Per-scan
Nightly	Automated (patrol)	All packs	Per-run
Weekly deep	Automated (deep patrol)	All packs, all checks	Per-run

6. Suppression

Some contradictions are known and intentional (different methodologies, different scopes). These can be suppressed to avoid repeated alerts:

```
# config/patrol-suppressions.yaml (addition)
knowledge:
  - check: cross-pack-consistency
    packs: [solar-energy-market, competitor-analysis]
    claims: [C007, C003]
    reason: "Different scope — TAM vs SAM. Both correct."
    expires: 2026-06-22    # Re-check in 3 months
```

Suppressions expire. Knowledge changes; a suppression valid today may mask a real problem in six months.

7. Author Quality Checklist

Run before committing any pack. Six checks, in order:

1. **Preserve meaningful context** (RATIONALE.md §1, Principle 24). Keep names, roles, dates, methods and decision context when they affect a claim's attribution, scope, interpretation or reasons for change. Remove incidental process narration only when those distinctions remain intact. Fuller source narratives belong in evidence; their necessary qualifications also belong with the claim.
2. **Preserve qualifications in views** (RATIONALE.md §1, Principle 24). A view may explain who said something, under which conditions, or why an assessment changed. Check that these details are supported by the record. Process-language matches are prompts for editorial review, not reasons for automatic deletion or conformance failure.

3. **Canonical ownership** (RATIONALE.md §1, Principle 18). Every concept is defined in exactly one pack. If a claim restates content from another pack, it must carry an explicit derivation marker ("Pitch rendering. Canonical: pack#claim") and a `see_also` link to the canonical home.
4. **Meeting/pitch composition**. Meeting and pitch pack claims contain only meeting- or pitch-specific context. Standing knowledge (product architecture, competitive positioning, technical details) is cross-referenced to standing packs, not independently defined.
5. **Display block stranger test** (RATIONALE.md §3 "The Stranger Test"). Can someone who has never heard of this product understand the tagline and hook in one reading? Explain necessary terms and keep the context needed to understand the subject. Omit incidental workflow details from short introductions without removing meaningful attribution or temporal scope.
6. **Hub routing, not restating**. Hub pack overview claims summarize and route to detail packs via `see_also` links. If a hub claim contains >3 sentences of detail about a concept that lives in a detail sub-pack, move the detail down.

8. Pattern Definitions for Automated Checks

Review patterns for the planned `provenance-contamination` and `canonical-duplication` patrol checks. Matches identify material to inspect; they do not establish a defect or authorize removing context.

8.1 Process Reference Review

Scope: `claims.md` context prose after the metadata block and `views/*.md` content. A name, date, method, event or tool can be material knowledge in any pack. Review whether the reference explains attribution, scope, a method, a decision or a change; preserve it when it does. Fuller process narratives may remain in `evidence.md` with a reference from the claim. No blanket exemption or prohibition is needed for meeting claims.

In `claims.md` context prose

Pattern Category	Examples	Regex
Person-as-trigger	"[Name] asked", "[Name] requested", "[Name] wanted"	<code>\b[A-Z][a-z]+\s+(asked</code>
Session attribution	"crystallized during", "decided in the [date] session", "from the [date] meeting"	<code>\b(crystallized</code>
Tool attribution	"generated by [tool]", "via [framework]", "using NotebookLM"	<code>\b(generated by</code>
Framework-as-source	"from Positioning Pyramid", "established via [framework]"	<code>\bfrom\s+[A-Z][a-zA-Z]+\s+(Pyramid</code>
Date-process coupling	"the March 2026 redesign", "the Q1 pivot"	<code>\bthe\s+w+\s+d{4}\s+(redesign</code>

In `views/*.md` content

Pattern Category	Examples	Regex
Process verbs as descriptors	"redesign", "pivot", "shifted from", "transitioned to"	<code>\b(redesign</code>
Tool names in non-comment text	"NotebookLM", tool-specific names	<code>\b(NotebookLM</code>
Session references	"positioning session", "strategic session", "March 24 session"	<code>\b\w+\s+session\b</code>
Temporal process markers as headers	"## Something (Month Year)" where date marks a decision	<code>^#\s+.*\s+([A-Z][a-z]+\s+d{4})</code>

In `PACK.yaml` display block

Pattern Category	Examples	Regex
Version/sprint details	"v0.49", "sprint", "wave", release identifiers	<code>\bv\d+\.\d+\b sprint wave\s+d</code>
Process words	"redesign", "migration"	<code>\b(redesign migration overhaul pivot)\b</code>
Person attribution	"[Name]'s ask"	<code>\b[A-Z][a-z]+'s\s+(ask request requirement)\b</code>

8.2 Canonical Duplication

Detection requires cross-pack comparison. These patterns identify candidates for manual review.

Cross-pack claim similarity

Pattern	Detection Method
Same concept, different packs, no derivation marker	Two claims in different packs with the same noun phrase in the assertion headline AND no derivation marker in context ("Pitch rendering", "Hub-level summary", "Meeting-specific")
Verbose cross-reference without link	A non-hub pack claim containing >3 sentences of descriptive context about a concept with a canonical home in another pack, without a <code>see_also</code> link
Standing knowledge in meeting packs	Meeting pack claims that define product architecture, competitive positioning, or technical details rather than meeting-specific context (who said what, what was decided, what actions were agreed)

Single-pack self-duplication

Pattern	Detection Method
Hub restating detail	A hub pack claim with >3 sentences of detail about a concept that should live in its detail sub-pack

Derivation markers (valid — do not flag)

Claims containing these phrases in their context prose are legitimate cross-references:

- "Pitch rendering. Canonical: ..."
- "Hub-level summary. See ..."
- "Meeting-specific context. Standing knowledge at ..."
- Any `see_also` link to the canonical pack

8.3 Formatting Hygiene

Grep-able patterns for text hygiene checks across `claims.md` files.

Check	Pattern	Description	Exceptions
Orphaned relations	<code>^\s{2}[\rightarrow\leftarrow\sim\leftrightarrow]\s</code>	Standalone relation line that should be merged onto preceding context line	Lines where <code>~</code> means "approximately" (e.g., ~8,400), prose flow arrows (e.g., <code>→portfolio→certificate</code>), or pipeline descriptions (e.g., <code>→ Provenance → Compliance</code>)
Mid-sentence breaks	<code>^\s{2}[a-z]</code>	Continuation line starting with a lowercase letter that isn't a relation symbol — likely a broken sentence	Legitimate multi-line context prose (most continuation lines are valid; flag only when the preceding line ends mid-word or mid-phrase)
ASCII arrows in claims	<code>-></code> or <code><-</code> in claims context	Should use Unicode <code>→</code> and <code>←</code> for KP:1 relation symbols	Prose flow descriptions (e.g., <code>Discovery -> Definition -> Execution</code>), environment variable cascades, code examples
Verbose <code>see_also</code>	<code>*see_also\s</code>	Should use compact <code>*pack#claim</code> format	None — always normalize

Linguistic Conventions — Knowledge Pack

Companion Spec

Date: 2026-03-22 **Status:** Draft **Decisions:** D1 (canonical language — American English, Merriam-Webster)

Canonical conventions for American English text in Knowledge Pack claims, evidence, entities, and views. These conventions govern all human-language content in the Knowledge Pack system. Machine-readable identifiers (IDs, slugs, URIs) follow their own conventions defined in the schema. Source of truth: this document.

Design Principles

1. **English-canonical, multilingual-derivable.** All claims are authored in American English. Other language views are derived, never authoritative.
2. **Dual audience.** Claims are consumed by LLMs (reasoning surface) and displayed to humans (view surface). When conventions conflict between the two audiences, the reasoning surface wins and the view surface adapts.
3. **Unambiguous over elegant.** Prefer explicit, literal phrasing over idiomatic or culturally dependent expression. This improves both LLM parsing and translation.
4. **Consistent over correct.** When multiple conventions are defensible, pick one and apply it everywhere. Consistency reduces parsing variance in LLMs more than any specific format choice.
5. **Atomic over compound.** One assertion per claim. Compound claims obscure verification scope and degrade confidence scoring.

Conventions Table

#	Area	Convention	Example	Notes
1	Spelling authority	Merriam-Webster (first listed entry)	"canceled" not "cancelled"	M-W is the de facto standard referenced by AP, Chicago, Google, Microsoft, Apple
2	Style fallback	Chicago Manual of Style, 18th ed.	—	For questions M-W doesn't answer (hyphenation edge cases, etc.)
3	Locale tag	en-US	PACK.yaml: linguistic_epoch: en-US-2026	BCP 47 language tag; year suffix anchors semantic drift. Note: linguistic_epoch is a decoration field per CORE.md §3 (no consumer behavior in v0.8.5-preview; new producers SHOULD prefer extensions.★)
4	Dates	ISO 8601: YYYY-MM-DD	2026-03-22	Never "March 22, 2026" in claims. Views may localize
5	Date ranges	ISO 8601 with "to"	2026-01-01 to 2026-03-31	Not en-dash ranges in claims; views may use en-dash
6	Times	ISO 8601 with UTC	14:30 UTC or 2026-03-22T14:30:00Z	UTC always; IANA zone in parentheses if local matters: (America/New_York)
7	Large numbers	Numeral + word	\$2 million, 3.5 billion	Not \$2M (too informal) or \$2,000,000 (noisy). Full form in evidence when precision matters
8	Percentages	Symbol, no space	50%	Not "50 percent" or "50 %"

#	Area	Convention	Example	Notes
9	Numeric ranges	"to" connector	2 to 5, \$10K to \$50K	Not 2-5 (ambiguous hyphen) or 2–5 (en-dash requires typography). Views may upgrade to en-dash
10	Capitalization: headings	Sentence case	### Team and timeline	Not "Team and Timeline". Only proper nouns capitalized
11	Capitalization: claims	Sentence case	Platform supports 7 AI capabilities	Claims are assertions, not titles
12	Serial comma	Always (Oxford comma)	types, db, and api	Non-negotiable in claims. Eliminates a class of parsing ambiguity
13	Dashes: hyphen	Compound modifiers only	well-known, single-GPU	Standard hyphen character - (U+002D)
14	Dashes: en-dash	Ranges in views only	pages 128–134	En-dash – (U+2013). Never in claims.md; use "to"
15	Dashes: em-dash	Parenthetical breaks	scaffolding – not matching – is the product	Em-dash — (U+2014) with spaces. Sparingly in claims; acceptable in views and evidence
16	Acronyms	Expand on first use per section	Model Context Protocol (MCP)	Subsequent uses: MCP. Skip expansion for universals: API, URL, USD, AI
17	Tense: current facts	Simple present	Platform uses Qwen 3.5	Present tense = "true as of pack version date"
18	Tense: historical facts	Simple past	Founded in 2024	Clear temporal anchoring; avoid perfect tenses
19	Hedging	External to claim text	Claim: Team has 30 mentors / Confidence: 0.40	Avoid redundant hedging, but preserve source attribution, allegations, modality and quotation qualifications when they change the proposition. Confidence does not replace attribution
20	Negation	Explicit assertion	Platform does not store PII locally	Never rely on absence of a claim to imply negation. State negative facts explicitly
21	Pronouns	Minimize; repeat nouns	The platform supports... not It supports...	Especially across sentence boundaries. Improves LLM parsing and translation accuracy
22	Granularity	One verifiable assertion per claim	One [C###] = one fact	Compound claims must be decomposed. "A and B" may become two claims with valid distinct IDs. Splitting does not itself justify a support relation
23	Product names	Official capitalization, always	Qwen 3.5, OpenRouter, Fireworks AI	First use: full name. No trademark symbols in claims (too noisy)
24	Measurement units	SI / metric preferred, with context	3B active params, 85K lines	Use K/M/B suffixes for informal magnitude in claims; precise numbers in evidence
25	Currency	ISO 4217 code or \$ for USD	\$2 million, EUR 500K	Dollar sign for USD (dominant context). ISO code for all others
26	Phone numbers	E.164 with spaces	+1 555 012 3456	Canonical storage format. Views may add parentheses
27	Quotation marks	Straight double quotes	"scaffolding" not "scaffolding"	ASCII " (U+0022). Curly quotes in views only

#	Area	Convention	Example	Notes
28	Lists in prose	Semicolons for complex items	types (6.3K); db (6.6K); api (31K)	Commas for simple items; semicolons when items contain internal commas
29	Non-English terms	Preserve original script + romanization	Київ (Kyiv)	Follow MULTILINGUAL.md conventions. Never anglicize proper nouns
30	Sentence length	Target 15-25 words	—	Aligns with controlled language standards (ASD-STE100). Improves translation and LLM parsing

Justifications

1. Merriam-Webster as spelling authority

Merriam-Webster is the most widely adopted spelling reference in American English. The AP Stylebook explicitly defers to M-W for words it doesn't cover. Apple's style guide references M-W as its spelling standard. Google and Microsoft developer documentation teams use M-W as their baseline. The "first listed entry" rule (when M-W lists multiple spellings) follows AP and Chicago convention.

Key M-W vs. alternatives:

- M-W is descriptive and comprehensive (documents actual usage)
- American Heritage Dictionary is more conservative but less widely adopted institutionally
- AP Stylebook is prescriptive but narrow (journalism-focused, not a dictionary)

2. Chicago Manual of Style as style fallback

Chicago is the dominant style authority for non-journalistic American English writing. It covers edge cases (complex hyphenation, citation format, number styling) that M-W does not address. Apple explicitly uses Chicago as its style authority with M-W for spelling. Most technical writing organizations follow the same two-tier hierarchy.

3-6. ISO 8601 dates/times

ISO 8601 is unambiguous, machine-parseable, and culture-neutral. LLMs show 8-12% fewer parsing errors with ISO dates than written-out dates in multilingual contexts. "to" connector for ranges avoids the hyphen/en-dash ambiguity problem that plagues mixed human/machine consumption. UTC as default eliminates seasonal timezone confusion (EST/EDT problem). IANA identifiers (America/New_York) handle DST automatically.

7-9. Numbers, percentages, ranges

"Numeral + word" (\$2 million) is recommended by Google, Microsoft, and AP style guides for technical documentation. The % symbol is universally preferred in technical/scientific contexts. "to" for ranges is clearer for international audiences and more reliably parsed by both LLMs and regex. These choices optimize the dual-audience requirement: scannable by humans, unambiguous for machines.

10-11. Sentence case

Sentence case is the consensus standard across Google Developer Docs, Microsoft Writing Style Guide, Apple Style Guide, and Wikipedia Manual of Style. It mirrors natural reading patterns, reduces cognitive load, and is unambiguous (no debates about which words to capitalize in title case). Schema.org uses TitleCase for programmatic type names but sentence case for human-readable labels and descriptions.

12. Serial comma (Oxford comma)

The Oxford comma eliminates a class of parsing ambiguity without any downside. Most academic and technical style guides recommend it. AP is the notable exception (journalism context, not applicable here). In structured claims where precision matters, the Oxford comma is non-negotiable.

13-15. Dash hierarchy

Hyphens for compounds, en-dashes for ranges, em-dashes for breaks. These are distinct Unicode codepoints that tokenize differently in LLMs. Using the correct character improves machine parsing. In claims.md (the reasoning surface), we avoid en-dashes for ranges because "to" is unambiguous even in plain ASCII contexts. Views can use proper typographic en-dashes.

16. Acronyms

Expand-on-first-use is universal across Google, Microsoft, Apple, and Wikipedia. Skip expansion for universally understood acronyms (API, URL, etc.). Per-section expansion (not per-document) accounts for the fact that claims.md sections may be loaded independently.

17-18. Tense

Simple present for current facts (true as of version date) and simple past for completed events. This follows Wikipedia's convention and aligns with Wikidata's temporal qualifier pattern. Avoiding perfect and continuous tenses reduces translation complexity and ambiguity.

19. Hedging externalized

Embedding "possibly" or "might" in claim text creates dual ambiguity: is the uncertainty about the fact or about the phrasing? Knowledge Packs have a dedicated confidence system (0.0-1.0 normalized scores). The claim text states the assertion; the confidence score carries the uncertainty. This separation is validated by fact-checking systems (FEVER, Full Fact) which universally separate claim text from truth-value metadata.

20. Explicit negation

Following Wikidata and open-world assumption principles. The absence of a claim does not imply its negation. Negative facts ("X does not Y") must be stated explicitly with their own evidence and confidence scores.

21. Pronoun minimization

Controlled language standards (ASD-STE100, Caterpillar Technical English) and translation-readiness research agree: repeating nouns instead of using pronouns reduces ambiguity by 15-40%. LLMs are more sensitive to ambiguous pronoun resolution than human readers who use real-world context.

22. Atomic claims

Validated by FEVER dataset methodology, Full Fact practice, and atomic fact extraction research (AFEV framework). Compound claims create ambiguity in verification and confidence scoring. The Knowledge Pack relation system (`→supports`, `←requires`) captures relationships between atomic claims.

23-25. Product names, units, currency

Official capitalization for product names follows the ACID test (International Trademark Association). SI/metric preferred for internationalization. Dollar sign for USD (dominant context in this corpus) with ISO 4217 codes for other currencies follows AP and financial industry convention.

26-28. Phone, quotes, lists

E.164 for phones is the international standard and machine-parseable. Straight quotes avoid Unicode encoding issues in plain text and are consistent with Markdown conventions. Semicolons for complex list items follows Chicago Manual of Style.

29-30. Non-English terms, sentence length

Follows the MULTILINGUAL.md conventions. Sentence length target of 15-25 words aligns with ASD-STE100 Simplified Technical English and improves both human comprehension and machine translation accuracy.

Fallback Hierarchy

When a linguistic question arises that is not covered by this document:

1. This document (CONVENTIONS.md)
↓ not covered
2. Merriam-Webster Dictionary (first listed spelling)
↓ not covered (style question, not spelling)
3. Chicago Manual of Style, 18th edition
↓ not covered (domain-specific terminology)
4. ISO 704:2022 (Terminology work – Principles and methods)
↓ not covered (controlled language question)
5. ASD-STE100 Simplified Technical English principles
↓ not covered (knowledge representation question)
6. Wikidata conventions (temporal, multilingual, negation)
↓ truly novel
7. Author judgment, documented as a decision in this file

The chain reflects a deliberate narrowing: general English → technical English → terminology science → controlled language → knowledge representation. Each level addresses questions the previous level cannot.

Knowledge-Pack-Specific Conventions

These conventions are specific to Knowledge Packs and would not appear in a standard style guide.

Claim text vs. metadata separation

Claim text is a pure declarative assertion. All epistemic qualification (confidence, evidence type, temporal scope) lives in the `{confidence|type|evidence|date}` block. Never mix these layers.

Wrong: Platform probably uses Qwen 3.5 (based on code review in March) **Right:** Platform uses Qwen 3.5 `{0.99|o|E011|2026-03-18}`

Temporal anchoring

All claims are implicitly anchored to the pack version date. A claim Platform has 906 tests means "as of the version date in PACK.yaml." If a claim has a different temporal anchor, make it explicit: Platform had 400 tests at v0.30.0.

Entity reference style

When referencing entities in claims, use the canonical display name on first mention, then aliases for brevity. Cross-pack references use `↔pack_name#section` notation.

Confidence-appropriate language

Even though hedging is externalized to confidence scores, claim text should not overstate. A claim with 0.60 confidence should not be phrased as an absolute:

Poor: The market is \$50 billion at `{0.60}` **Better:** The addressable market is estimated at \$50 billion at `{0.60}`

The word "estimated" is not hedging — it describes the claim type (estimation vs. measurement). This is a factual descriptor, not epistemic qualification.

Density notation in claims.md

Claims.md uses compressed notation optimized for token efficiency. Conventions that apply only in claims.md (not views):

- Parenthetical abbreviations: `types(6.3K) db(6.6K) api(31K)` — no spaces
- Slash notation for alternatives: `GPT-3.5-class cost / single-GPU self-hostable`
- Pipe-delimited metadata: `{0.95|i|E001,E002|2026-03-01}`
- Relation symbols inline: `→C002, ⊗C003`

Views expand these into full prose with proper punctuation and spacing.

Claims across domains

When a Knowledge Pack spans multiple domains (business + technical + legal), each domain's terminology follows its own standards, but connective language follows this document. If a legal claim uses "shall" and a technical claim uses "must," both are correct within their domains. The linguistic conventions here govern the connective tissue, not domain jargon.

Translation-ready claim writing

Claims destined for multilingual views should additionally:

- Avoid idioms, metaphors, and cultural references entirely
- Use explicit rather than implicit causal chains (**Not:** Given the burn rate, runway is 18 months
Instead: Monthly burn rate is \$120K. At \$120K/month, \$2.1M gives 18 months runway)
- Make all quantifiers explicit (`several` → `3 to 5`)
- Avoid relative descriptions (`large`, `small`) — use absolute measures
- Keep subject-verb-object order; avoid inverted or embedded clauses

View-layer typographic upgrades

Views may upgrade plain claims.md notation to proper typography:

Claims.md	Views
"quoted"	"quoted" (curly quotes)
2 to 5	2–5 (en-dash)
\$2 million	\$2,000,000 (when precision needed)
--	— (em-dash)
2026-03-22	March 22, 2026 (when locale-appropriate)

The claims.md form is always canonical. Views are derived presentations.

Scope and Governance

This document governs all human-language content in Knowledge Packs:

- `claims.md` — claim text and prose annotations
- `evidence.md` — source descriptions and excerpts
- `entities.md` — entity labels, descriptions, and relationship labels
- `views/*.md` — pre-rendered display content
- `PACK.yaml` — description and other prose fields

It does **not** govern:

- Machine identifiers (IDs, slugs, URIs) — see the patterns in `CORE.md §3` and `conformance/grammar/kp-pack.schema.json`
- Metadata field names — see `conformance/grammar/kp-pack.schema.json` (the normative manifest schema)
- Name romanization — see `MULTILINGUAL.md`
- Privacy and sensitivity classification — see the `sensitivity / visibility` fields in `CORE.md §3`

Changes to this document should be versioned in git and noted in the `updated_at` frontmatter field.

Storage Independence — Knowledge Pack Companion Spec

Date: 2026-03-29 **Status:** Draft **Editor:** Timothy Kompanchenko

1. Purpose

This document formalizes the relationship between Knowledge Packs and the infrastructure that stores, indexes, and serves them. It establishes the **pack-as-master** principle: Knowledge Packs are the canonical source of truth for their contents. All operational infrastructure — databases, caches, indices — is derived from packs and rebuildable from them.

2. The Pack-as-Master Principle

Statement

A Knowledge Pack is the authoritative representation of its epistemic contents. Any external system that stores, indexes, caches, or projects pack data is a **derived representation** that serves operational needs. When a derived representation disagrees with the pack, the pack is correct.

Analogy

This is the relationship between a git object store and git refs/index. The objects (blobs, trees, commits) are the source of truth. Refs, the index, remote tracking branches, and merge state are operational infrastructure derived from objects. You can `git fsck` and rebuild the index from objects. You can never rebuild objects from the index.

Similarly:

- Knowledge Packs = git objects (the truth)
- Databases, caches, search indices = git index/refs (derived, rebuildable)

Implications

1. **Writes flow through packs.** When knowledge changes, the pack changes first. Indices update to reflect the pack, not the other way around.
2. **Indices are rebuildable.** Any index or database projection of pack data can be dropped and reconstructed by re-reading packs. This is the test of whether the pack is truly master.
3. **Packs are portable without infrastructure.** A pack exported to a USB drive, emailed, or placed on a blockchain functions identically — the Voyager Principle is preserved regardless of operational storage.
4. **No infrastructure-only knowledge.** If a piece of knowledge exists only in a database column and not in any pack, it is not part of the canonical knowledge base. It may be useful operational data, but it is not mastered knowledge.

What This Does NOT Mean

- **It does not mean you cannot have databases.** You need operational infrastructure for fast cross-pack queries, vector search, real-time session accumulation, and entity resolution. The principle says these are indices into packs, not replacements for them.
 - **It does not mean packs must be files on disk.** Packs can be stored in any medium (see §3). The logical model is what matters, not the physical storage.
 - **It does not mean everything must be a pack.** Task queues, UI state, authentication tokens, and operational configuration are not knowledge and should not be forced into pack format (see §5).
-

3. Serialization Formats

A Knowledge Pack is a logical model (manifest + claims + evidence + views + history). The SPEC.md defines the **file serialization** as the canonical human-readable form. This section defines additional serialization formats for operational storage.

3.1 File Serialization (Canonical)

The format defined in SPEC.md §2: a directory of plain text files (PACK.yaml, claims.md, evidence.md, views/).

- **When to use:** Curated packs, version-controlled repositories, export/import, long-term archival, Voyager-compliant distribution.
- **Properties:** Human-readable, git-diffable, zero-dependency. This is the format that satisfies the Voyager Principle.
- **Authority:** When a pack exists in file form, the files are authoritative.

3.2 JSON Serialization (Operational)

A single JSON document or a set of JSON records representing the pack's logical contents.

```
interface KPackJSON {
  manifest: KPackManifest; // PACK.yaml contents as JSON
  claims: KPClaim[];       // Structured claim records
  evidence: KPEvidence[];  // Evidence records
  views: KPView[];         // View records (name, surface, content)
  history: KPHistoryEntry[]; // Superseded/retracted claims
}
```

- **When to use:** Database storage (Postgres JSONB or relational), API responses, programmatic consumption.
- **Properties:** Machine-queryable, structured, parseable without markdown parsing. Loses some human readability but gains structured access.
- **Conversion:** Deterministic bidirectional conversion between file and JSON forms. Round-trip fidelity is required — file → JSON → file must produce semantically identical output.

Note — semantic fidelity and sealed packs. Semantic round-trip fidelity is the right requirement for operational storage, but it is weaker than what a seal requires. The content hash in ARCHIVE.md §3 is computed over each file's **raw bytes**, so a file → JSON → file round trip that is semantically identical while differing in any byte — key order, numeric formatting, quoting style, whitespace, or line endings — produces a different pack hash and invalidates an existing signatures.yaml.

The two requirements do not conflict; they apply to different situations. For packs that are not sealed, semantic fidelity is sufficient. **For any pack whose files are covered by a content hash, a round trip that is intended to preserve the seal must be byte-stable, not merely semantically equivalent.** Implementations have two ways to satisfy this:

1. **Preserve the original bytes** of hash-covered files across the round trip, treating the JSON form as a derived index rather than a source of truth for those files, in keeping with the pack-as-master principle of §2.
2. **Re-seal after conversion**, accepting that the pack hash changes and that any prior signature no longer applies.

Integers in declared units can reduce numeric precision ambiguity, but equal numeric values do not imply equal serialized bytes: quoting, key order, whitespace and line endings still matter. JSON consumers may also impose integer range/precision limits; RFC 8259 §6 identifies the interoperable binary64 integer range. Declare units and supported precision in the relevant profile, preserve original bytes for existing seals, and test conversions rather than inferring byte stability from an integer representation.

3.3 Compact Serialization (Ephemeral)

A minimal representation for space-constrained contexts (Redis capsules, context window injection, transfer).

```
interface KPackCompact {
  name: string;
  version: string;
  claims: Array<{
    id: string;
    text: string;
    confidence: number;
    type: string;
    status: string;
  }>;
  // Evidence, views, and history omitted — loadable on demand
}
```

- **When to use:** Redis restart capsules, context window injection, WebSocket transfer, in-flight session accumulation.
- **Properties:** Minimal token footprint. Intentionally lossy — strips evidence chains, views, and history. The compact form is sufficient for reasoning but not for full epistemic audit.
- **Conversion:** One-way: full pack → compact (lossy). Compact → full requires access to the evidence and history from the original source.

3.4 Format Equivalence

All three serializations represent the same logical model. The relationship:

File (canonical)	↔	JSON (operational)	→	Compact (ephemeral)
↑		↑		↑
authoritative		round-trippable		lossy projection
human-readable		machine-queryable		minimal footprint
Voyager-compliant		DB-storable		cache-friendly

4. The Index Contract

Operational indices enable capabilities that individual packs cannot provide: cross-pack search, entity resolution, statistical aggregation, vector similarity. This section defines what indices must guarantee.

4.1 What Indices Provide

Capability	Description	Why packs alone can't do this
Cross-pack claim search	"What do I know about Picasso?" across all packs	Requires reading all packs; too slow at scale
Entity resolution	Match entities across packs by name, alias, URI	Requires global entity registry
Vector similarity	"Find claims similar to X"	Requires embeddings and approximate nearest neighbor
Statistical aggregation	Market overviews, claim counts, confidence distributions	Requires structured query across many packs
Temporal queries	"What changed since last week?"	Requires timestamped index entries

4.2 Index Invariants

1. **Rebuildable.** The entire index can be dropped and reconstructed by reading all packs. This is the master test. If rebuilding the index from packs loses information that was in the index, that information was not properly mastered in a pack.
2. **Eventually consistent.** When a pack changes, the index may lag briefly. This is acceptable. The index is a cache, not a source of truth.
3. **Additive only.** Indices add capabilities (search, aggregation, speed). They never add knowledge. A claim that exists only in an index and not in any pack is an orphan that should be flagged and resolved.

4. **Pack-attributed.** Every index entry traces back to a specific pack and claim. The index knows where each piece of knowledge came from.

4.3 Index Schema (Reference)

This is a reference schema, not a mandate. Implementations may vary.

```
-- Pack registry
CREATE TABLE pack_index (
  pack_id TEXT PRIMARY KEY,
  name TEXT NOT NULL,
  version TEXT NOT NULL,
  domain TEXT,
  tier TEXT,
  lifecycle_type TEXT,
  storage_backend TEXT,      -- 'file', 'postgres', 'redis'
  storage_location TEXT,    -- file path, table name, redis key
  indexed_at TIMESTAMPTZ
);

-- Claim index (cross-pack searchable)
CREATE TABLE claim_index (
  claim_id TEXT PRIMARY KEY,      -- globally unique (ULID/UUIDv7)
  pack_id TEXT REFERENCES pack_index(pack_id),
  display_id TEXT,               -- human-readable alias (C001)
  assertion TEXT,
  confidence NUMERIC(3,2),
  claim_type TEXT,
  depth TEXT,
  nature TEXT,
  status TEXT,
  since DATE,
  valid_to DATE,
  indexed_at TIMESTAMPTZ
);

-- Claim relations (cross-pack)
CREATE TABLE claim_relation_index (
  from_claim_id TEXT,
  to_claim_id TEXT,
  relation TEXT,      -- supports, contradicts, supersedes, etc.
  PRIMARY KEY (from_claim_id, to_claim_id, relation)
);

-- Vector embeddings for semantic search
CREATE TABLE claim_embeddings (
  claim_id TEXT PRIMARY KEY,
  embedding vector(1536),
  model TEXT
);
```

5. Pack Boundaries — What Is and Isn't a Pack

Clear Knowledge Packs

These are unambiguously knowledge packs:

Content	Pack Type	Lifecycle	Example
Curated domain knowledge	standalone/hub	permanent	solar-energy-market.kpack
Entity intelligence	detail	permanent	picasso-provenance.kpack
Meeting notes	standalone	ephemeral	meeting-jackson-2026-04-02.kpack
Conversation capture	standalone	ephemeral	session-2YC6M-2026-03-29.kpack
Research findings	standalone	permanent	market-analysis-pe.kpack

Incremental Adoption Candidates

These may evolve into knowledge packs as the model matures:

Content	Current System	Path to KP:1	When Ready
Memory suggestions	Postgres flat rows	Claims in a staging pack with review workflow	When the pack model naturally accommodates the review lifecycle
Session notes ("take a note")	Not yet built	Notes in the session's conversation pack	When conversation packs are proven end-to-end
Action items	Extracted by passive analyzer	Already in conversation packs as view content	Already there — formalize as first-class content type

Not Knowledge Packs

These should NOT be forced into pack format:

Content	Why Not	What They Are
Task queues	Operational state, not epistemic content	Workflow infrastructure
UI state	Transient, per-client	Application state
Auth tokens	Security infrastructure	Configuration
Build artifacts	Computational output	DevOps
Raw event logs	Undifferentiated data stream	Operational telemetry

The test: **Does this content represent beliefs with confidence, evidence, and relationships?** If yes, it's a candidate for KP:1. If it's operational state, workflow data, or infrastructure configuration, it isn't.

6. Pack Content Types

6.1 Claims (Existing)

The primary content type, defined in SPEC.md §4. Structured assertions with confidence, type, evidence, depth, nature, and relations.

6.2 Non-Claim Content (New)

Packs may contain content that is not a formal claim but is epistemically relevant to the pack's domain. These are captured in views or in structured sections within claims.md.

Content Type	Description	Example	Where It Lives
Decision	A choice made, with rationale	"Decided to use Clerk for auth"	Claim with nature: judgment or views/decisions.md
Action Item	Something to be done, with owner/ deadline	"Team lead to contact supplier by Friday"	views/action-items.md
Note	User-directed capture ("take a note of this")	"Analyst quoted \$800M TAM"	Claim with type: reported if epistemic, or views/notes.md if operational
Question	Unresolved question worth tracking	"What's the actual TAM?"	views/questions.md or meta-claim

The rule: If content has epistemic value (confidence, evidence, relationships matter), it should be a claim. If it's operational (tracking who does what by when), it belongs in a view. Both live inside the same pack. The pack is the container; claims and views are the content types.

Notes specifically: When a user says "take a note," the system determines whether the note is:

- **Epistemic** ("The analyst said the TAM is \$800M") → becomes a claim: [N001] Analyst quoted \$800M TAM {0.70|r|session-transcript|2026-03-29}
- **Operational** ("Call the supplier tomorrow") → goes to views/action-items.md OR views/notes.md

This distinction matters because epistemic notes gain the full benefit of KP:1's confidence, evidence, and contradiction tracking. Operational notes don't need that overhead.

7. In-Flight Pack Accumulation

During live sessions (voice conversations, meetings, research), packs are accumulated in real time. The pack-in-progress is conceptually a pack even before it's finalized.

Lifecycle

Accumulating (Redis)	→	Materialized (Postgres/File)	→	Archived/Promoted
in-flight		durable		long-term
high write freq		queryable		indexed
session-scoped		room-scoped		tenant-scoped

Accumulation Rules

1. **In-flight packs use compact serialization** in Redis. Claims are appended as they're extracted.
2. **Session end triggers materialization.** The conversation-kpack materializer produces the full pack (all serialization formats).
3. **Materialized packs are indexed.** The index updates to include the new pack's claims.
4. **Ephemeral lifecycle.** Session packs have `lifecycle.type: ephemeral` with `archive_after_days` set. Before archival, the reconciliation step ([RATIONALE.md §1, Principle 21](#)) ensures valuable claims are promoted to standing packs.

Restart Capsules

Restart capsules (used for voice session recovery) are compact serializations of the in-flight pack. They contain enough information to resume accumulation but are not full packs. When a session restarts, the capsule seeds the new accumulation context.

8. Relationship to Entity-Claim-Evidence Indices

An entity-claim-evidence index — a database (relational, document, or graph) that stores entities, claims, and evidence as queryable rows or nodes — is a natural **operational index** for a knowledge pack corpus. It provides cross-pack entity resolution, claim querying, and evidence lookup at scales where scanning files is impractical.

This section is non-normative. It describes how such an index would relate to packs under the pack-as-master principle (§3), not how it must be implemented.

Index as derived view

In the pack-as-master model, an entity-claim-evidence index is a derived projection of the pack corpus:

Index Component	Role	Derived From
Entities	Global entity registry	<code>extensions.entities</code> (legacy: <code>entities.md</code>) across all packs
Claims	Cross-pack claim index	<code>claims.md</code> across all packs
Evidence	Evidence lookup	<code>evidence.md</code> across all packs
Entity Links	Cross-entity relationships	Claim relations + entity references

The index can be dropped and rebuilt from packs at any time. It is operational infrastructure, not a source of truth.

Structural alignment considerations

Any index schema that materializes packs must reconcile differences in vocabulary, ranges, and structure. Common alignment work includes:

Gap	Typical Resolution
Index uses an integer or enumerated confidence; KP:1 uses 0.0–1.0	Normalize at ingest using the source pack's <code>confidence.normalize</code> declaration
Index lacks depth or nature fields	Extend the index schema or store as opaque metadata

Gap	Typical Resolution
Index uses structured predicate + value; KP:1 uses prose text	Index keeps structured form; serializers generate prose for export
Index claim statuses do not match KP:1's four	Map index statuses onto KP:1's set; document the mapping
Index has only entity-level links; KP:1 has claim-level relations	Extend the index schema to support per-claim relations

These gaps are illustrative. Specific implementations will encounter their own.

Bidirectional flow

```
File pack → parse → index → query, resolve, aggregate
index → materialize → file pack (export, distribute)
```

Both directions should be deterministic given a fixed pack corpus and a stable index schema. Materializers that reconstruct packs from index rows are an existing pattern in conversation-derived pack tooling.

9. Design Principles for Storage Independence

1. **Pack is master.** When pack and index disagree, trust the pack. Rebuild the index.
2. **Serialize, don't reshape.** Packs stored in Postgres are serialized packs, not database-shaped data that happens to contain pack fields. The logical model comes first.
3. **Index for speed, pack for truth.** Indices exist because reading all packs for every query is slow. They are performance infrastructure, not knowledge infrastructure.
4. **One authority per pack.** Each pack has exactly one authoritative backend at any time. A file-authored pack is authoritative in its file form. A session pack is authoritative in Redis during the session, then in Postgres after materialization.
5. **Export is always possible.** Any pack in any backend can be exported to the file serialization (the Voyager-compliant form). This is the escape hatch that guarantees portability.
6. **Incremental adoption.** Not everything needs to become a pack today. Existing systems (memories, session ledgers) continue to work. As the pack model proves itself for each use case, content migrates naturally.

Bundle Format — Knowledge Pack Companion Spec

Date: 2026-03-22 **Status:** Draft **Decisions:** D11 (sharing format), D19 (bundle marker)

kpack CLI invocations in this document describe planned reference tooling. `python3 conformance/run.py, ./reference/kpack lint` and `./reference/kpack new` ship today. See [SPEC.md §13](#) and the reference launcher at [reference/kpack](#) for status.

1. Purpose

Knowledge Packs are multi-file directories. When sharing outside the IDE — pasting into a web AI chat, emailing to a collaborator, uploading to an external system — a single-file format is needed. The bundle format concatenates pack files into one markdown document with clear section markers. Any AI can parse it. Any human can read it.

2. Format Identifier

All bundles begin with a KP:1 format marker in an HTML comment:

```
<!-- KP:1 BUNDLE — {pack-name} v{version} -->
```

This reuses the existing KP:1 format identifier from the Rosetta Header. Bundle interpretation is a separate capability; a directory parser is not automatically a bundle parser. Any system that doesn't see a markdown comment and ignores it. The Voyager Principle applies — the marker is self-describing.

3. Reading Bundle Format

For reading or context sharing, as a projection of selected pack files:

```
<!-- KP:1 BUNDLE — {pack-name} v{version} -->
<!-- Generated: {ISO-8601 timestamp} -->
<!-- Files: {count} -->

--- PACK.YAML ---
{contents of PACK.yaml}

--- CLAIMS ---
{contents of claims.md}

--- EVIDENCE ---
{contents of evidence.md}

--- ENTITIES ---
{contents of entities.md, if present (deprecated — legacy packs only)}

--- VIEW: {view-name} ---
{contents of views/{view-name}.md}

--- VIEW: {view-name-2} ---
{contents of views/{view-name-2}.md}

--- END BUNDLE ---
```

Parsing Rules

- Section markers are lines matching the pattern `--- {SECTION_TYPE}(: {name})? ---`

- Content between markers belongs to that section
- The `--- END BUNDLE ---` marker is required — it signals clean termination
- Sections may appear in any order, but the recommended order is: `PACK.YAML`, `CLAIMS`, `EVIDENCE`, `ENTITIES`, then `VIEWS`
- Missing sections are valid — a pack without evidence simply has no `--- EVIDENCE ---` section

Optional Sections

Section	Included by default	Flag to include
PACK.YAML	Always	—
CLAIMS	Always	—
EVIDENCE	Yes	<code>--no-evidence</code> to exclude
ENTITIES	If present (deprecated — legacy packs)	<code>--no-entities</code> to exclude
VIEWS	All declared views	<code>--views overview,briefing</code> to select
HISTORY	No	<code>--include-history</code> to include
VALIDATION	No	<code>--include-validation</code> to include
VOICE VIEWS	No	<code>--include-voice</code> to include
LOCALE VIEWS	No	<code>--locale uk-UA</code> to include

4. Clipboard Bundle Format

For quick paste into web AI chat interfaces. Optimized for 8K-32K context windows typical of consumer AI interfaces. Strips everything except claims and the primary view:

```
<!-- KP:1 CLIP - {pack-name} -->
{contents of claims.md}

---
{contents of views/overview.md}
```

Evidence, entities, history, and secondary views are omitted. Claims + primary view is usually sufficient for a web chat session. The AI receiving this gets the full reasoning surface plus a human-readable summary.

5. Visibility Interaction

Bundle generation respects the pack's `visibility` field:

Visibility	kpack bundle behavior
private	Warning: "This pack is marked private." Requires <code>--force</code> flag.
shared	Allowed. Private annotations (if any) are stripped.
public	Allowed. Full content.

This prevents accidental sharing of sensitive knowledge. The friction is minimal (one flag) but intentional.

6. CLI Commands

```
# Full bundle to stdout
kpack bundle solar-energy-market

# Clipboard format, copied to system clipboard
kpack bundle solar-energy-market --clip

# Full bundle to file
kpack bundle solar-energy-market -o bundle.md

# Selective views
kpack bundle solar-energy-market --views overview,architecture
```

```
# Include locale views
kpack bundle solar-energy-market --locale uk-UA

# Include voice views
kpack bundle solar-energy-market --include-voice

# Shareable URL (requires publishing configuration)
kpack bundle solar-energy-market --url

# Force bundle of private pack
kpack bundle personal-contacts --force
```

7. Round-Trip

The proposed `kpack unbundle` command would reconstruct only explicitly represented sections. This marker format has no defined escape for a source line that itself matches a section marker, no general binary-attachment carrier, and no complete representation of every file or metadata profile. It does **not** guarantee a lossless directory round-trip, preserve an existing seal, or constitute an archive. Use the original directory or [ARCHIVE.md](#) ZIP representation for complete transfer and preservation. Clipboard bundles are intentionally lossy. A future lossless bundle needs a separately specified escaping and file-inventory contract; this draft does not define one.

8. QR Code / Deep Link (Future)

For mobile use: a QR code that encodes a URL pointing to a bundle file. Scan to load a knowledge pack into a voice session. Specification deferred until URL-based sharing infrastructure exists.

Archive Format — Knowledge Pack Companion Spec

Date: 2026-04-12 **Status:** Draft **Resolves:** AR-14 — `signatures.yaml` schema (formal artifact: `conformance/grammar/kp-signatures.schema.json`)

kpack CLI invocations in this document describe planned reference tooling. `python3 conformance/run.py`, `./reference/kpack lint` and `./reference/kpack new` ship today. See [SPEC.md §13](#) and the reference launcher at [reference/kpack](#) for status.

1. Purpose

A Knowledge Pack is a directory of plain text files. This is correct for editing, version control, and the Voyager Principle — any intelligence can read the files without tooling.

But directories cannot travel. When a pack moves between systems — client to server, server to client, backup to archive — a single-file format is needed. This companion spec defines the **archive format**: a sealed, hashed, versioned single-file representation of a Knowledge Pack.

Relationship to BUNDLE.md and STORAGE.md

Three companion specs address three questions about pack representation:

Spec	Question	Target	Properties
STORAGE.md	How does a pack <i>exist</i> ?	Databases, caches, file systems	File / JSON / Compact serializations
BUNDLE.md	How does a pack travel to an <i>intelligence</i> ?	AI chat windows, email, human readers	Markdown concatenation, AI-readable, Voyager-compliant
ARCHIVE.md	How does a pack travel between <i>systems</i> ?	Client-server transfer, backup, integrity chain	Zip container, hashed, signed, versioned

BUNDLE is infrastructure-free and human-readable. ARCHIVE is infrastructure-aware and machine-targeted. Both are single-file representations of the same directory. They serve different purposes and neither replaces the other.

2. Format Definition

A Knowledge Pack archive is a **ZIP file** containing the pack's directory contents.

File extension

The `.kpack` extension is used for both directory and archive forms:

Context	Form	Example
On disk, in git, during editing	Directory	<code>marquet-le-passeur.kpack/</code>
In transit, in storage, in backup	ZIP file	<code>marquet-le-passeur.kpack</code>

Tools distinguish the two forms by checking whether the path is a file or directory. This mirrors macOS bundle conventions (`.app/` is a directory that behaves as a unit).

Filesystem coexistence. A file and a directory cannot share the same path. When sealing a pack, the archive **MUST** be written to a different location than the source directory — either a specified output path (`-o`) or the current working directory. The naming convention describes how the artifact is *identified*, not that both forms coexist at the same path. See §7 for CLI examples.

Archive contents

The ZIP file contains the pack directory's contents at the root level — not nested inside a subdirectory:

```
marquet-le-passeur.kpack (ZIP)
├── PACK.yaml
├── claims.md
├── evidence.md
├── signatures.yaml      # REQUIRED in sealed archives – integrity metadata (absent in export
archives)
├── entities.md         # if present in pack (deprecated – legacy packs only)
├── history.md          # if present in pack
├── composition.yaml    # if present in pack
├── validation.yaml     # if present in pack
├── views/
│   ├── overview.md
│   └── briefing.md
├── definitions/        # if present in pack
│   └── *.yaml
├── policies/           # if present in pack
│   └── *.yaml
├── attachments/        # if present in pack
│   └── *
```

Requirements

1. The archive **MUST** be a valid ZIP file (PKZIP format, per APPNOTE.TXT).
2. Entries **MUST** be rooted at the pack directory level — no extra wrapper subdirectory. (Nested paths like `views/overview.md` and `attachments/photo.jpg` are expected; the rule prohibits wrapping the entire pack in a subdirectory like `marquet-le-passeur/PACK.yaml`.)
3. `PACK.yaml` and the files required for the declared kind **MUST** be present (per [CORE.md §2](#)). Claim-bearing and composition packs require `claims.md`; definition/policy packs use their companion documents.
4. `signatures.yaml` **MUST** be present in any archive that participates in an integrity chain (§4). It **MAY** be absent in archives created for convenience (e.g., manual export). See "Conformance levels" below.
5. Compression method **SHOULD** be DEFLATE. Store (no compression) is acceptable for small packs or when speed is preferred.
6. ZIP64 extensions **MUST** be used for archives exceeding 4 GB (archives without ZIP64 at this size are invalid, not merely suboptimal).
7. ZIP encryption **MUST NOT** be used. Access control is handled at the transport and storage layers, not the container layer. If confidentiality is required, encrypt the archive file itself (e.g., age, GPG) rather than using ZIP encryption.

Safety requirements

Archives are ZIP files and carry the same security risks as any ZIP container. Implementations **MUST** enforce:

8. **No path traversal.** ZIP entries containing `..` path segments, absolute paths (leading `/` or drive letters like `c:\`), or backslash separators **MUST** be rejected during extraction and verification.
9. **No symlinks.** ZIP entries marked as symbolic links **MUST** be rejected. Symlinks enable arbitrary file reads and are not portable across platforms.
10. **No duplicate entries.** ZIP entries with identical paths **MUST** be rejected (some ZIP implementations silently overwrite; the behavior is undefined and exploitable).
11. **No special files.** Device files, FIFOs, and other non-regular-file entries **MUST** be rejected.
12. **OS metadata exclusion.** The excluded set is exactly:

```
.DS_Store    Thumbs.db    desktop.ini    __MACOSX    .*
```

These **SHOULD** be stripped on archive creation and **MUST** be ignored during hash computation. **This requirement is the single normative statement of the excluded set; other sections refer to it rather than restating it.**

Matching is by path component, at any depth: a path is excluded if any of its components matches an entry in the set — `.DS_Store`, `Thumbs.db`, `desktop.ini` and `__MACOSX` by equality, and `.*` by prefix, meaning a component beginning with `._` (AppleDouble sidcar files). A file at `views/.DS_Store` is excluded; so is one at `._resources/notes.md`.

No open-ended prefix exclusions. Named OS-metadata patterns are enumerable, and the set above is closed and complete. `.*` is a named pattern with a bounded meaning — AppleDouble sidcars — not an open class such as "every dotfile". A file named `.provenance`, `.gitattributes` or `.anything-else` **is part of the pack and is included in the hash**.

Conformance levels

Level	signatures.yaml	Integrity chain	Use case
Sealed archive	REQUIRED	Participates	Client-server transfer, versioned storage, regulated contexts
Export archive	ABSENT	Does not participate	Manual sharing, convenience export, one-off backup

An export archive is a valid ZIP containing a valid pack. It is not sealed and does not participate in the integrity chain. Tools that expect sealed archives **MUST** reject export archives with a clear error rather than silently skipping verification.

Format discoverability

A receiver that encounters a `.kpack` archive cold — with no surrounding context, no documentation, no prior exposure to KP:1 — should be able to learn the format from the archive itself. Two pack-internal mechanisms support this, **in this order of priority**:

1. **Rosetta header (load-bearing).** For claim-bearing packs, the first line of `claims.md` is a self-describing HTML comment naming the format epoch and tokenization legend ([CORE.md §4](#)). Definition/policy kinds instead use the manifest and their companion document rules; they need not invent a claims file or header. The Rosetta header is the offline, network-free parse signal — it does not depend on any external resource.
2. **`spec_uri` / `spec_version` (informational onboarding pointer).** Sealed-archive producers **SHOULD** declare a discovery URL on `PACK.yaml` pointing to the full KP:1 specification ([CORE.md §3 "Spec Onboarding Pointer"](#)). When present, a previously-unfamiliar consumer can fetch the spec from the URL the pack itself names — useful for first-time onboarding of agents that have never encountered KP:1 before. **`spec_uri` is informational, not a runtime parse dependency.** Consumers **MUST NOT** treat unreachability of `spec_uri` as a parse failure; claim-bearing headers provide the inline parse hint; supported companion kinds use their local files and known companion rules. Once a consumer has internalized the format, `spec_uri` carries no further operational weight.

Consumers **SHOULD** read `PACK.yaml` first and, on initial encounter, follow `spec_uri` (when declared) to onboard against the spec for the pack's declared `spec_version`. When `spec_uri` is absent, consumers **MAY** assume the published reference implementation at <https://github.com/tymofiy/kp> but **MUST** treat the assumption as informational, not a guarantee. Offline-first validators **MUST NOT** fail on absent or unreachable `spec_uri` — the field's purpose is onboarding ergonomics, not integrity.

The file extension identifies the format family; a claim-bearing pack's header supplies its inline parse hint, and the manifest declares its kind. The optional onboarding pointer helps an unfamiliar receiver learn the format. Readers that implement the relevant companion rules can parse the supported local files without network access.

3. Content Hash

The integrity of an archive is verified by its **content hash** — a deterministic hash computed from the pack's file contents. The hash is of the *knowledge*, not the *container*: re-zipping the same files produces the same content hash even if the ZIP metadata differs.

Computation

1. Enumerate all files in the pack directory, excluding `signatures.yaml` **at the root level**. (A file at `attachments/signatures.yaml` is included normally.)

2. For each file, compute SHA-256 over its **raw bytes**. (Empty files hash as the SHA-256 of the empty string: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855.)
3. Normalize each file path:
 - Use / as the path separator (no \).
 - Use bare relative paths: no leading ./, ../, or /.
 - Apply **Unicode NFC normalization** to the path string. (macOS decomposes filenames to NFD; other systems use NFC. Without normalization, the same filename produces different path bytes on different operating systems.)
4. Express each file's SHA-256 digest as **lowercase hexadecimal**.
5. Sort the file list by normalized path (UTF-8 byte order, case-sensitive).
6. Concatenate entries as {normalized-path}\0{lowercase-hex-digest}\n (null byte separating path from digest, newline terminating each entry).
7. Compute SHA-256 over the concatenation. This is the **pack hash**.

Example

Given a pack with three files:

```
claims.md      → SHA-256: a1b2c3...
PACK.yaml     → SHA-256: d4e5f6...
views/overview.md → SHA-256: 789abc...
```

Sorted by path:

```
PACK.yaml\0d4e5f6...\n
claims.md\0a1b2c3...\n
views/overview.md\0789abc...\n
```

SHA-256 of this concatenation → pack hash.

Canonicalization notes

- **Line endings.** Hashing operates on raw bytes, so `\r\n` and `\n` produce different hashes. Implementations that use git SHOULD set `* text=auto eol=lf` in `.gitattributes` within the pack directory to prevent `core.autocrlf` from altering line endings across platforms.
- **OS metadata files.** Operating-system metadata is not part of the Knowledge Pack. Implementations MUST exclude it from hash computation and SHOULD exclude it from archives. **The excluded set is defined once, in §2 requirement 12** — see §10 (Security Considerations) for why these files are excluded rather than which they are.

Draft policy compatibility

The component matching and AppleDouble exclusions in this draft can produce a different content hash from earlier published rules. `seal_version` selects a signing payload, not an exclusion policy. Historical artifacts MUST NOT be silently reserialized or re-sealed to satisfy the new policy. A verification receipt must identify the exact spec revision/hash rules it applied. An explicitly selected historical compatibility mode may verify the historical rules; automatic fallback from a current-policy mismatch MUST NOT be presented as current-policy success. Keep historical bytes and receipts. See [COMPATIBILITY.md](#).

Fixed expected hashes in [archive-hash-vectors.json](#) cover component exclusions, retained ordinary dotfiles, root-only receipt exclusion, NFC filenames and raw line endings. The [reference helper](#) accepts an already enumerated regular-file map; it does not implement archive extraction or signature/chain verification.

Properties

- **Deterministic.** Same file contents and file paths always produce the same pack hash, regardless of filesystem ordering, ZIP implementation, creation timestamp, or operating system.
- **Content-addressed.** Changing any file's contents changes the pack hash. Adding or removing a file changes the pack hash.

- **Self-excluding.** Root-level `signatures.yaml` is excluded from the hash computation because it contains the hash. This avoids the circular dependency.

4. `signatures.yaml`

This section defines the schema for `signatures.yaml`, resolving SPEC.md §2's "Tooling only — Cryptographic hashes + signing key for integrity verification" and CORE.md AR-14. The normative JSON Schema artifact is `conformance/grammar/kp-signatures.schema.json`; the conformance runner validates `signatures.yaml` against it whenever the file is present.

Schema

```
# signatures.yaml — integrity metadata for a sealed Knowledge Pack

# Hash algorithm (currently only SHA-256; future-proofed for upgrades)
algorithm: SHA-256

# Stable pack identity across versions (OPTIONAL for legacy archives;
# SHOULD be present in new archives — see "Required fields" below)
pack_id: "0b3c1f9a-4d7e-4a1c-9f2b-3e4a5b6c7d8e"

# Self-describing receipts (OPTIONAL; mirrors PACK.yaml name/version)
pack_name: "solar-energy-market"
pack_version: "2026.03.18"

# Signing-payload version (§4.2) — omit for v1
seal_version: "2"

# Pack hash — computed per §3
pack_hash: "a1b2c3d4e5f6..."

# Per-file hashes — lowercase hex, must not include signatures.yaml itself
files:
  PACK.yaml: "d4e5f6..."
  claims.md: "a1b2c3..."
  evidence.md: "789abc..."
  views/overview.md: "def012..."

# Sealing metadata
sealed_at: "2026-04-12T14:30:00Z"      # ISO 8601 UTC timestamp
sealed_by: "capture-client"           # Identity of the sealing system

# Version chain (§5) — absent for v1 (first version)
parent:
  version: "2026.04.11"                # Parent pack version
  pack_hash: "fedcba987654..."       # Parent pack hash

# Optional digital signature (§4.2)
signature:
  method: "ed25519"                   # or "hmac-sha256", "rsa-pss-sha256"
  value: "base64-encoded-signature..." # Signature over the signing payload (§4.2)
  key_id: "my-signing-key-2026"       # Key identifier for verification
```

Field types

Field	Type	Format
<code>algorithm</code>	string	Algorithm identifier. Currently SHA-256.
<code>pack_id</code>	string	Stable identity of the pack across versions. A UUID is RECOMMENDED. Two sealed archives MUST share the same <code>pack_id</code> if and only if they represent versions of the same logical pack.
<code>pack_name</code>	string	Self-describing copy of <code>PACK.yaml.name</code> . Enables receipts and indexing without loading the manifest.
<code>pack_version</code>	string	Self-describing copy of <code>PACK.yaml.version</code> . Same rationale as <code>pack_name</code> .

Field	Type	Format
seal_version	string	OPTIONAL. Signing-payload version (§4.2). Absent means v1, whose payload does not cover pack_id. Present values: "2". Itself a signed field, so it cannot be stripped to select a weaker payload.
pack_hash	string	Lowercase hex-encoded SHA-256 digest (64 characters).
files	map<string, string>	Keys: NFC-normalized relative paths with / separator. Values: lowercase hex SHA-256 digests. MUST NOT include signatures.yaml.
sealed_at	string	ISO 8601 UTC timestamp (e.g., 2026-04-12T14:30:00Z).
sealed_by	string	Identifier of the sealing system (e.g., capture-client, analysis-service).
parent.version	string	Parent pack's version string from its PACK.yaml.
parent.pack_hash	string	Parent pack's pack_hash (lowercase hex).
parent.merge_parents	array<object>	OPTIONAL. Additional parent references for branch-merge lineage. Each entry has the same shape as parent itself (version + pack_hash). Absent in single-parent archives.
signature.method	string	Signing method identifier: ed25519, hmac-sha256, or rsa-pss-sha256.
signature.value	string	Base64-encoded (standard, padded) signature over the signing payload.
signature.key_id	string	Opaque key identifier for locating the verification key.

Required fields

Field	Required	When
algorithm	Always	
pack_hash	Always	
files	Always	
sealed_at	Always	
sealed_by	Always	
pack_id	New archives	SHOULD be present in archives sealed against v0.7.5 or later. Schema accepts absence for backwards compatibility with archives sealed before pack_id was documented. Readers MUST tolerate its absence.
pack_name	Never	OPTIONAL; aids self-describing receipts.
pack_version	Never	OPTIONAL; aids self-describing receipts.
parent	When version > 1	Absent for the first version of a pack
parent.version	When parent is present	Both sub-fields are REQUIRED when the parent block exists
parent.pack_hash	When parent is present	
parent.merge_parents	Never	OPTIONAL; populated only when a version results from merging two or more lineages
signature	When signing is configured	Optional; depends on deployment
signature.method	When signature is present	All three sub-fields are REQUIRED when the signature block exists
signature.value	When signature is present	
signature.key_id	When signature is present	

4.1 Sealing

Sealing is the act of computing the content hash and writing `signatures.yaml`. A sealed pack is one whose `signatures.yaml` is present and whose `pack_hash` matches the computed hash.

Sealing happens at every version transition in the pack lifecycle:

Transition	Sealer	Example
Capture → v1	Capture client (on-device)	User photographs a document
Analysis → v2	Analysis service	Service extracts claims and evidence
Research → v3	Research service	Research loop completes
Edit → v(n+1)	Analysis service	User edits are merged

4.2 Signing (optional)

Signing provides **tamper evidence** for the sealing metadata — proof that `pack_hash`, `sealed_at`, `sealed_by`, and the primary parent fields (`parent.version`, `parent.pack_hash`) have not been modified since sealing. It is OPTIONAL because:

- Single-user deployments gain little from signing.
- Multi-tenant deployments (auction houses, galleries) benefit from knowing *which system* sealed a version.
- Regulatory contexts (provenance disputes, legal discovery) may require it.

Signing payload. The signature is computed over a canonical byte string that binds the metadata, not just the pack hash. This prevents an attacker from rewriting sealing metadata while preserving a valid signature.

There are two payload versions. A seal declares which one it uses with the OPTIONAL top-level `seal_version` field in `signatures.yaml`; **its absence means v1**.

Payload v1 (`seal_version` absent) — the original form:

```
{algorithm}\n{pack_hash}\n{sealed_at}\n{sealed_by}\n{parent.version or empty}\n{parent.pack_hash or empty}\n
```

Payload v2 (`seal_version: "2"`) — adds `pack_id`, and adds the version declaration itself:

```
{seal_version}\n{algorithm}\n{pack_hash}\n{pack_id or empty}\n{sealed_at}\n{sealed_by}\n{parent.version or empty}\n{parent.pack_hash or empty}\n
```

In both, absent fields are the empty string, and all fields use their literal `signatures.yaml` values — no normalization or transformation.

The version declaration is itself a signed field, and MUST be. A verifier selects the payload from `seal_version`, so a marker held outside the signature would be a downgrade rather than a protection: an attacker strips it from a v2 seal, the verifier falls back to v1, the v1 payload does not cover `pack_id`, and the seal verifies — while carrying a claim of being fixed. Because the declaration is signed, stripping it yields a v1 payload that cannot match a v2 signature, and adding it to a v1 seal yields a v2 payload that cannot match a v1 signature. Neither direction forges; both fail closed. Verifiers MUST refuse a seal declaring a `seal_version` they do not implement, and MUST NOT fall back to a payload version the seal did not declare.

Producers SHOULD emit v2. Verifiers MUST accept both.

Scope note — what a v1 seal does and does not attest. `signatures.yaml` is excluded from the pack content hash (§3), so any field inside it that the payload does not name is covered by **neither** the signature **nor** the hash. Under v1 that includes `pack_id`: an attacker can add, change, or remove a pack's identity inside a signed archive without invalidating anything, undetectably. Since identity is what groups a pack's revisions into a family, a v1 pack can be made to claim another pack's identity and enter its lineage — so **verification of a v1 seal is not verification of the identity it carries**, and consumers MUST treat a v1 `pack_id` as transport metadata rather than as a signed assertion. v2 closes this for seals minted under it; **v1 seals remain identity-rewritable by construction, permanently**, because they were signed over a payload that never covered the field. Existing packs are repaired only by being re-sealed at v2.

`parent.merge_parents` entries (§4 field table) are outside **both** payload versions for the same reason and remain **unauthenticated lineage hints** under v1 and v2 alike: an attacker can add, remove, or alter them in a signed archive without invalidating anything. Verifiers MUST NOT treat merge-parent assertions as tamper-evident; only the referenced archives' own contents can be verified, never the assertion that they are parents of this pack. A future payload revision may extend coverage.

Signing procedure:

1. Compute the pack hash per §3.

2. Assemble the signing payload as defined above.
3. Sign the payload using the configured method.
4. Record the signature (base64-encoded), method, and key identifier in `signatures.yaml`.

Supported methods:

Method	Type	Properties
ed25519	Asymmetric	Non-repudiation. Compact signatures. Recommended for multi-party contexts.
hmac-sha256	Symmetric (shared secret)	Tamper detection only — does not provide non-repudiation (both parties hold the key). Suitable for single-tenant or internal deployments.
rsa-pss-sha256	Asymmetric	Non-repudiation. Compatible with existing PKI. Uses PSS padding (PKCS#1 v2.1).

Implementations SHOULD support at minimum Ed25519. HMAC-SHA256 MAY be supported for simpler deployments where non-repudiation is not needed.

Key distribution is out of scope for this spec. The `key_id` field identifies the key; how verification keys are discovered, distributed, and trusted is an implementation and deployment concern. Implementations MAY use well-known endpoints, key registries, or out-of-band exchange.

5. Version Chain

Each sealed version of a pack references its parent, forming an integrity chain. A verified chain establishes the exercised hash links among the available revisions; it does not prove that every historical revision or source artifact was retained.

Chain structure

```
v1 (capture) → v2 (analyzed) → v3 (researched) → v4 (edited) → v(n)
hash: aaa      hash: bbb      hash: ccc      hash: ddd
parent: ∅      parent: aaa     parent: bbb  parent: ccc
```

Each version's `signatures.yaml` contains the `parent` block with the previous version's `version` string and `pack_hash`. The first version has no parent.

Verification

To verify a chain:

1. Start with the earliest available version.
2. Compute its pack hash. Confirm it matches `signatures.yaml.pack_hash`.
3. Move to the next version. Confirm its `parent.pack_hash` matches the previous version's `pack_hash`.
4. Repeat until the current version.

A broken chain (parent hash mismatch) indicates tampering, data loss, or a version created outside the integrity system. Broken chains SHOULD be flagged but MUST NOT prevent the pack from being read — the knowledge is still valid even if the chain is broken.

Branching

Version chains may branch when multiple systems modify the same pack concurrently (e.g., two users editing simultaneously). Branch resolution is an implementation concern, not a spec concern. This spec defines the linear chain and the `parent.merge_parents` data shape (§4); branch conflict-resolution *policy* is deferred to a future version of this spec.

6. Archive Operations

6.1 Create (pack → archive)

```
Input:  name.kpack/ (directory)
Output: name.kpack   (ZIP file with signatures.yaml, written to -o path or cwd)
```

Steps:

1. Exclude OS metadata files from the file set, per §2 requirement 12
2. Compute file hashes for all remaining files in the directory
3. Compute pack hash per §3 (NFC normalization, lowercase hex, sorted paths)
4. Write signatures.yaml (with parent reference if this is not v1)
5. ZIP the directory contents (including signatures.yaml) → archive
6. Write the archive to the output path (MUST NOT overwrite the source directory)

6.2 Verify (archive → result)

Input: name.kpack (ZIP file)

Output: { status: "sealed" | "export" | "invalid", errors: string[] }

Steps:

1. Validate all ZIP entries against §2 safety requirements (path traversal, symlinks, duplicates)
2. Validate archive structure (§2 requirements):
 - PACK.yaml and the files required by CORE §2 for the declared kind MUST be present
 - Entries MUST be pack-rooted (no wrapper subdirectory)
 - No two entry paths may collide after NFC normalization
 - If any structural check fails: return status "invalid"
3. Check for signatures.yaml in the archive:
 - If absent: return status "export" (valid pack structure, not sealed)
 - If present: continue to step 4
4. Verify signatures.yaml.algorithm is a supported algorithm (currently: SHA-256)
5. Compute file hashes for all other files in the archive (excluding OS metadata)
6. Compute pack hash per §3
7. Compare computed pack_hash to signatures.yaml.pack_hash
8. Compare per-file hashes against signatures.yaml.files
9. Optionally verify digital signature if present (using signing payload from §4.2)
10. Optionally verify parent chain if previous version is available

Note: Empty directories in ZIP entries are ignored — only regular files participate in hash computation, consistent with §3.

6.3 Extract (archive → pack)

Input: name.kpack (ZIP file)

Output: name.kpack/ (directory, at -o path or cwd)

Steps:

1. Validate all ZIP entries against §2 safety requirements:
 - Reject entries with path traversal (../ segments)
 - Reject entries with absolute paths or backslash separators
 - Reject symlinks, device files, and duplicate entries
 - Strip OS metadata entries, per §2 requirement 12
2. Verify the archive (§6.2) — warn but do not block on failure
3. Extract ZIP contents to directory
4. signatures.yaml is preserved in the directory (it is part of the pack)

6.4 Transfer

When a pack archive moves between systems:

1. Sender computes or reads the pack hash.
2. Sender transmits the archive file and its pack hash (the hash MAY travel in an HTTP header, a sidecar file, or as part of an API response envelope).
3. Receiver verifies the archive against the declared hash.
4. On mismatch: reject the transfer. Do not silently accept corrupted packs.

7. CLI Commands

Note: kpack archive is reserved for lifecycle archival (see LIFECYCLE.md). The seal/transport commands use kpack seal to avoid ambiguity.

```

# Seal a pack directory → ZIP with signatures.yaml
kpack seal solar-energy-market.kpack/ -o ./archives/
# → creates ./archives/solar-energy-market.kpack (ZIP)

# Seal without -o (writes to cwd; fails if name collides with source directory)
kpack seal ../other-location/solar-energy-market.kpack/
# → creates ./solar-energy-market.kpack (ZIP) in the current directory

# Seal with parent reference (integrity chain)
kpack seal solar-energy-market.kpack/ -o ./archives/ --parent ./archives/solar-energy-market-
v2.kpack
# → creates archive with parent hash from the v2 archive

# Verify an archive's integrity
kpack verify solar-energy-market.kpack
# → checks pack hash, file hashes, optional signature, ZIP safety

# Verify a chain of archives
kpack verify --chain v1.kpack v2.kpack v3.kpack
# → checks each archive and verifies parent references

# Extract an archive to a directory
kpack extract solar-energy-market.kpack -o ./packs/
# → creates ./packs/solar-energy-market.kpack/

# Show archive metadata without extracting
kpack info solar-energy-market.kpack
# → prints: version, pack_hash, sealed_at, sealed_by, parent, file count

```

8. Interaction with Other Pack Features

Attachments

Binary attachments (images, PDFs, scanned documents) are included in the archive as-is. Their hashes are computed over raw bytes and included in `signatures.yaml.files`. Large attachments dominate archive size — implementations MAY use Store (no compression) for already-compressed formats (JPEG, PNG, PDF) and DEFLATE for text files.

Composition

Composite packs (see `COMPOSITION.md`) reference other packs by name, not by embedding them. A composite pack's archive contains only the composite's own files — it does not bundle the referenced packs. Each referenced pack has its own archive and its own integrity chain.

Lifecycle

Archive creation is a lifecycle event, not a lifecycle state. A pack's lifecycle type (permanent, seasonal, ephemeral) is orthogonal to whether it has been archived. All lifecycle types can be archived. Ephemeral packs SHOULD be archived before their retention window expires so the integrity chain is preserved even after the pack is no longer active.

Visibility

Sealing respects the pack's `visibility` field, following the same rules as `BUNDLE.md §5`:

Visibility	kpack seal behavior
private	Warning. Requires <code>--force</code> flag.
shared	Allowed.
public	Allowed.

9. Design Decisions

Why ZIP and not tar, tar.gz, or a custom format?

- ZIP is universally supported. Every operating system, programming language, and mobile platform can create and extract ZIP files without additional dependencies.
- ZIP supports random access — individual files can be read without extracting the entire archive. This matters for large packs with binary attachments.
- ZIP is the container format behind .docx, .xlsx, .epub, .jar, and .ipa. The pattern is well-established and well-understood.
- tar requires a separate compression step (gzip, zstd). ZIP handles both in one format.

Why hash the contents, not the ZIP bytes?

Two independent ZIP implementations archiving the same files may produce different byte sequences (different metadata, different ordering, different compression parameters). Hashing ZIP bytes would make verification implementation-dependent. Hashing file contents makes verification deterministic regardless of the ZIP implementation.

Why is `signatures.yaml` excluded from the hash?

`signatures.yaml` contains the hash. Including it in its own hash computation creates a circular dependency. Excluding it is the standard approach (cf. git's tree hashing excludes commit metadata; TLS certificates exclude the signature field).

Why is signing optional?

The integrity chain (hash verification) is valuable without signing. Signing adds tamper evidence for metadata and, with asymmetric methods, non-repudiation (proving *who* sealed the pack). This matters in multi-party contexts but adds key management complexity. Making signing optional keeps the spec useful for simple single-user deployments while supporting enterprise requirements when needed.

Why does the signature bind metadata, not just the pack hash?

`signatures.yaml` is excluded from the content hash to avoid circular dependency. But without binding, the metadata fields (`sealed_at`, `sealed_by`, the primary `parent` fields) could be rewritten without breaking verification — defeating the purpose of the integrity chain. The signing payload (§4.2) binds these fields so that any modification invalidates the signature. Exactly which fields are bound depends on the payload version: v2 additionally binds `pack_id`, which v1 left rewritable; `parent.merge_parents` is bound by neither. See the §4.2 scope note for what a v1 seal does and does not attest.

10. Security Considerations

ZIP extraction attacks (ZipSlip)

ZIP archives can contain entries with path traversal sequences (..) that, when extracted naively, write files outside the intended directory. This is a well-documented class of vulnerability (CVE-2018-1002200 and variants). The safety requirements in §2 mandate rejection of such entries. Implementations **MUST** validate every ZIP entry path **before** extraction. Validation **MUST** reject:

- Paths containing .. segments (e.g., ../../etc/passwd)
- Absolute paths (e.g., /etc/passwd, C:\Windows\system32\...)
- Paths using backslash separators (may be interpreted as directory separators on Windows)
- Symbolic link entries
- Duplicate entry paths (undefined behavior in most ZIP libraries)
- Device files and other non-regular entries

OS metadata contamination

Operating systems create hidden metadata files — for example `.DS_Store` on macOS, `Thumbs.db` on Windows, and `__MACOSX/` resource-fork directories produced by macOS ZIP tools. **These examples are illustrative; the normative set is §2 requirement 12.** Such files:

- Are not part of the Knowledge Pack.
- Vary by platform — including them in the hash makes seals non-portable.
- May leak filesystem metadata (directory structure, timestamps, thumbnail images).

AppleDouble sidecars (`._*`) are a special case worth stating plainly. These files carry resource-fork and extended-attribute data, and appear both inside `__MACOSX/` directories in ZIP archives and as bare `._`-prefixed files on disk when an archive is extracted by a tool that flattens that directory. The two shapes are the same mechanism.

They also present a portability problem the other entries do not: **on macOS, Foundation's directory-enumeration APIs do not report bare `._`-prefixed files to a caller.** An implementation built on those APIs therefore cannot observe such a file in order to hash it, while an implementation reading the directory by other means can. Requiring these files to be hashed would make the requirement unsatisfiable for one class of conformant implementation and produce seals that verify on one toolchain and fail on another, with the file present on disk in both cases. Excluding them by name removes the divergence rather than leaving it to be discovered.

Implementations **MUST** exclude the files named in §2 requirement 12 from hash computation and **SHOULD** strip them when creating archives.

Line ending normalization

The content hash operates on raw bytes. If the same text file is checked out with `\n` on one system and `\r\n` on another (common with git's `core.autocrlf` on Windows), the hashes will differ. This is by design — the hash reflects the actual bytes — but it creates a practical portability problem.

Packs that use version control **SHOULD** include a `.gitattributes` file with `* text=auto eol=lf` to ensure consistent line endings across platforms. This is a recommendation for pack authors, not a requirement for implementations.

Unicode normalization in filenames

macOS HFS+/APFS normalizes filenames to NFD (decomposed). Linux and Windows preserve the original form, typically NFC (composed). The content hash algorithm (§3) mandates NFC normalization of paths to ensure cross-platform determinism. Implementations that create archives on macOS **MUST** normalize paths to NFC before computing hashes.

Definition & Policy Kinds — Knowledge Pack Companion Spec

Parent: SPEC.md, §5 **Date:** 2026-03-29 **Status:** Draft **Editor:** Timothy Kompanchenko **Origin:** Cross-cutting architecture decision — knowledge packs as master knowledge

1. Purpose

This document specifies the `definition` and `policy` document kinds introduced in SPEC.md §5. These kinds represent **deterministic domain knowledge** — structural facts about how a domain is organized (definitions) and behavioral rules for operating within that domain (policies). Unlike claims, which are probabilistic and evidence-backed, definitions and policies are authoritative declarations that constrain the vocabulary and behavior of the knowledge system.

2. Relationship to Claims

The Two-Layer Model

Knowledge Packs operate on two layers:

Layer	Kind(s)	Epistemology	Purpose
Ontology	definition, policy	Deterministic — these ARE the domain	Define what exists and how it works
Knowledge	claim	Probabilistic — beliefs about entities	Assert what's true about specific entities

The ontology layer constrains the knowledge layer. A claim references entity types, attributes, and relations that are defined in the ontology. A research perspective is routed by policies that determine which perspectives apply to which entity types.

Constraint Is Informational

An LLM reading a claims pack can reason about claims without definitions loaded. The claims are self-contained. But with definitions loaded, the LLM can:

1. **Validate** — Is `painting` a valid entity type? Is `medium` a valid attribute for `art`?
2. **Complete** — What attributes should a complete `art` entity have? What perspectives are missing?
3. **Navigate** — What other entity types exist in this domain? What relations connect them?
4. **Detect anomalies** — A claim about a `painting`'s `horsepower` doesn't match the taxonomy.

3. Definition Schema

See also: entity *instances* (as opposed to entity *types* defined here) live in `extensions.entities` and use the canonical `ent_<type>_<6-hex>` ID format documented in EXTENSIONS.md §3.1. DEFINITIONS.md governs the *types* (vocabulary); EXTENSIONS.md §2.2 governs *instances* of those types attached to specific packs.

File Location

Definitions live in the `definitions/` subdirectory of a `.kpack/` directory:

```
core-taxonomy.kpack/
├── PACK.yaml           # kind: definition
├── definitions/
│   ├── entity-types.yaml # Entity type catalog
│   ├── attributes.yaml   # Attribute vocabulary
│   └── relation-types.yaml # Relation type catalog
```

Document Header (Required)

Every definition YAML file begins with these fields:

```
kind: definition
authority: platform-team      # Who established this
version: "2026.1"            # CalVer or SemVer
status: active                # active | deprecated | draft
effective_from: "2026-03-29" # Optional: when this takes effect
effective_to: null            # Optional: when this expires (null = open-ended)
superseded_by: null          # Optional: replacement path/URI if deprecated
rationale: null               # Optional: why this definition exists
```

Entity Type Schema

```
kind: definition
authority: platform-team
version: "2026.1"
status: active
effective_from: "2026-03-29"
rationale: >
  Core entity types for alternative asset verticals.
  Covers the primary collectible and luxury categories.

entity_types:
- id: art                      # Unique identifier (kebab_case)
  name: Artwork                # Human-readable name
  description: >                # What this type covers
    Fine art – paintings, sculptures, prints,
    drawings, photographs, and mixed media works
  subtypes:                     # Enumerated subtypes
    - painting
    - sculpture
    - print
    - drawing
    - photograph
    - mixed_media
    - installation
  key_attributes:               # Primary attributes for this type
    - medium
    - dimensions
    - period
    - movement
    - technique
    - condition
    - provenance_length
  relations:                    # Common relation types for this entity
    - created_by                # artist → artwork
    - exhibited_at              # artwork → venue
    - part_of_series            # artwork → series
# NOTE: Perspective coverage (which perspectives apply to this type)
# belongs in a POLICY pack, not in the taxonomy definition.
# See §4 Policy Schema for perspective-coverage rules.
```

Attribute Schema

```
kind: definition
authority: platform-team
version: "2026.1"
status: active

attributes:
- id: medium
  name: Medium
  description: "Materials and technique used to create the work"
  applies_to: [art]             # Which entity types use this
  value_type: string            # string | number | enum | date | boolean
```

```

indexed: true           # Should this be searchable?
required: false         # Is this attribute mandatory?
sensitive: false        # Contains PII or confidential data?
examples:
  - "Oil on canvas"
  - "Bronze"
  - "Lithograph on paper"

- id: movement_type
  name: Movement Type
  description: "Watch movement mechanism"
  applies_to: [watch]
  value_type: enum
  enum_values: [mechanical_manual, mechanical_automatic, quartz, spring_drive, hybrid]
  indexed: true
  required: true
  sensitive: false

```

Relation Type Schema

```

kind: definition
authority: platform-team
version: "2026.1"
status: active

relation_types:
- id: created_by
  name: Created By
  description: "Links a work to its creator/artist/maker"
  source_types: [art, watch, jewelry, instrument]
  target_types: [person, brand, workshop]
  cardinality: many_to_one           # many_to_one | one_to_many | many_to_many
  inverse: creator_of               # The reverse relation ID

- id: provenance_chain
  name: Provenance Chain
  description: "Ownership history – ordered sequence of possession events"
  source_types: [art, watch, jewelry, wine, numismatic]
  target_types: [person, institution, auction_house]
  cardinality: many_to_many
  ordered: true                     # Sequence matters
  temporal: true                    # Has date ranges
  edge_attributes:                  # Properties on the relation itself
    - id: start_date
      type: date
      description: "When this ownership period began"
    - id: end_date
      type: date
      description: "When this ownership period ended (null = current)"
    - id: acquisition_method
      type: enum
      enum_values: [purchase, inheritance, gift, auction, commission]
      description: "How the entity was acquired"
    - id: sale_price
      type: number
      description: "Transaction price if known"
      sensitive: true

```

Relation Edge Attributes

Relations with `temporal: true` or complex semantics often carry their own properties. The `edge_attributes` field defines the data model for properties on the relation itself (not on the source or target entities).

```

edge_attributes:
- id: start_date           # Attribute identifier

```

```

type: date                      # date | number | string | enum | boolean
description: "When this edge became active"
- id: sale_price
  type: number
  description: "Transaction value"
  sensitive: true                # Contains financially sensitive data

```

Edge attributes are particularly important for:

- **Temporal relations** — start_date, end_date for ownership periods
- **Transactional relations** — price, currency, terms for sales
- **Qualified relations** — role, capacity, authority for participation

Graph modeling note: A provenance_chain with edge attributes is equivalent to an event-based model where each ownership transition is a discrete event. The relation-with-attributes model is more compact for simple chains; for complex provenance with multiple parties per event, consider modeling OwnershipEvent as a separate entity type with dedicated relations. The right choice depends on domain complexity — start with edge attributes, promote to event entities when the edges become too heavy.

Consumer Pattern — extensions.relations in PACK.yaml

The relation_types schema above defines the *vocabulary*. Producers in the reference implementation write *instances* of relations under extensions.relations in PACK.yaml, where each instance references entities declared under extensions.entities:

```

# In PACK.yaml — instance of `provenance_chain` with edge_attributes
extensions:
  entities:
    - id: ent_a_artwork_xyz
      type: asset
      canonical: "Untitled (1962)"
    - id: ent_p_collector_a
      type: person
      canonical: "Sarah Whitfield"
  relations:
    - id: rel_001
      from: ent_a_artwork_xyz
      to: ent_p_collector_a
      type: provenance_chain
      since: "1989-04-12"
      until: "2003-11-08"
      attributes:
        acquisition_method: auction
        sale_price: 425000
        evidence_ids: [E007]

```

vocabulary above
maps to start_date edge_attribute
maps to end_date edge_attribute
enum from edge_attributes
sensitive — audit-only surface

Edge attributes carrying sensitive: true (e.g. sale_price) **MUST NOT** appear in user-facing dossier prose; they surface in audit views only. See EXTENSIONS.md §3 for the cross-cutting discipline.

4. Policy Schema

File Location

Policies live in the policies/ subdirectory:

```

routing.kpack/
├── PACK.yaml
└── policies/
    ├── perspective-routing.yaml
    ├── priority-rules.yaml
    └── coverage-requirements.yaml

```

kind: policy

Document Header (Required)

Same metadata fields as definitions:

```
kind: policy
authority: domain-expert-review
version: "2026.1"
status: active
effective_from: "2026-03-29"
```

Rule Schema

```
kind: policy
authority: domain-expert-review
version: "2026.1"
status: active

rules:
  - id: art-provenance-priority          # Unique rule identifier
    description: >                      # What this rule does
      Art entities prioritize authentication and provenance
      perspectives – forgery risk makes these critical
    condition: "entity.type == 'art'"    # When this rule applies
    action: >                           # What to do
      prioritize perspectives:
        [authentication, provenance, condition, valuation]
    priority: 1                         # Lower = higher priority (for conflict resolution)
    rationale: >                        # Why this rule exists (domain justification)
      Art authentication is a prerequisite for meaningful
      valuation – an unverified work may be worth nothing

  - id: watch-authentication-first
    description: "Watches require authentication before valuation"
    condition: "entity.type == 'watch'"
    action: "require perspectives: [authentication] before [valuation]"
    priority: 1
    scope: research-intake              # Which system context this rule applies to
    rationale: >
      Watch counterfeiting is industrialized – authentication
      must gate all downstream analysis
```

Rule Scope

The `scope` field declares which system context a rule applies to. This prevents routing rules meant for research intake from accidentally affecting display rendering or voice briefings.

Scope	Meaning
research-intake	Rules for the research/analysis pipeline
display	Rules for view rendering and presentation
voice	Rules for voice delivery and briefing
global	Applies to all contexts (use sparingly)

Scope is freeform — implementations define their own scope vocabulary. The spec recommends but does not mandate specific scope values.

Conflict Resolution

When multiple rules match the same entity, conflicts are resolved by:

1. **Priority** — Lower number wins (priority 1 beats priority 2)
2. **Specificity** — More specific conditions win over general ones (`entity.subtype == 'painting'` beats `entity.type == 'art'`)
3. **Scope** — Context-specific rules win over `global` rules
4. **Recency** — Among equal priority/specificity, the rule with the later `effective_from` date wins

If a conflict cannot be resolved by these criteria, the consuming system should flag it for human review rather than silently choosing.

Condition Language

Conditions are freeform strings evaluated by the consuming system. The KP:1 spec does not mandate a condition language — implementations choose their own. Common patterns:

```
# Simple type match
condition: "entity.type == 'art'"

# Subtype match
condition: "entity.subtype in ['painting', 'sculpture']"

# Attribute-based
condition: "entity.estimated_value > 100000"

# Compound
condition: "entity.type == 'art' && entity.provenance_length < 3"
```

The condition is a **hint** for the consuming system, not executable code. An LLM reading the policy can interpret conditions in natural language. A code system can parse them into whatever evaluation engine it uses.

5. PACK.yaml for Definition and Policy Packs

Kind Declaration

```
# PACK.yaml for a definition pack
name: core-taxonomy
kind: definition # Required for non-claim packs
version: 2026.03.29
domain: platform/taxonomy
author: Timothy Kompanchenko

description: >
  Core entity type taxonomy for an alternative assets platform –
  entity types, subtypes, attributes, and relation types.
```

Kind Values

Value	Primary Content	Required Files	Optional Files
claim (default)	claims.md	PACK.yaml, claims.md	evidence.md, history.md, entities.md, views/
definition	definitions/*.yaml	PACK.yaml, definitions/	views/
policy	policies/*.yaml	PACK.yaml, policies/	views/
mixed	All of the above	PACK.yaml, claims.md, definitions/ or policies/	All optional files

Mixed Packs

A pack with `kind: mixed` contains both ontological content (definitions/policies) and probabilistic content (claims). This is useful when a domain's structural knowledge and entity knowledge are tightly coupled. However, prefer separate packs when possible — mixed packs are harder to version and review.

6. Versioning and Lifecycle

Authority

The `authority` field identifies who established the definition or policy. This is not a confidence signal (definitions are deterministic) but a provenance signal — who is responsible for this structural decision.

Authority	Meaning
platform-team	Core platform team decision
domain-expert-review	Reviewed by domain subject matter expert
industry-standard	Based on industry classification (e.g., Getty AAT)
regulatory	Required by regulation

Status Lifecycle

```
draft → active → deprecated
      ↓
      (superseded_by points to replacement)
```

- **draft** — Under development. May be incomplete. Not consumed by production systems.
- **active** — Current and authoritative. Consumed by production systems.
- **deprecated** — Still loadable but a replacement exists. `superseded_by` points to the replacement. Consuming systems should emit warnings.

Version Strategy

Definitions and policies use CalVer (YYYY.N) or SemVer, declared in the `version` field. Version changes signal:

- **Additive changes** (new entity type, new attribute): minor version bump
- **Breaking changes** (removed type, renamed attribute, changed cardinality): major version bump
- **Corrections** (typo in description, reworded rationale): patch or no version change

Effective Dates

The `effective_from` and `effective_to` fields mark the active window for a definition or policy:

- `effective_from` — When this takes effect (required for temporal queries)
- `effective_to` — When this expires. `null` or omitted means open-ended (no expiry). A non-null value means the definition/policy is time-bounded.

This enables:

- **Scheduled rollouts** — Author a new taxonomy version today, effective next quarter
- **Temporal queries** — "What entity types were active on 2026-01-15?"
- **Sunset planning** — Set `effective_to` on rules with known expiry dates
- **Audit trails** — When combined with git history, full evolution is traceable

7. Codegen Concept

The Bridge

Definition packs are YAML designed for domain expert readability and AI consumption. Many runtime systems need TypeScript types, Zod schemas, or enum maps. The codegen bridge connects these:

```
definitions/*.yaml → codegen script → generated/*.ts
      ↑                               ↓
    Domain experts                    Runtime consumers
    review YAML                       import TypeScript
```

What Gets Generated

Definition Content	Generated Artifact
Entity types + subtypes	TypeScript union types, enum maps
Attributes	Zod validation schemas, type interfaces
Relation types	Type-safe relation constructors
Policy rules	Lookup tables, routing maps

Example

From this definition:

```
entity_types:
  - id: art
    subtypes: [painting, sculpture, print]
```

A codegen script produces:

```
// GENERATED — DO NOT EDIT
// Source: core-taxonomy.kpack/definitions/entity-types.yaml

export const EntityType = {
  Art: "art",
  // ...
} as const;

export type EntityType = (typeof EntityType)[keyof typeof EntityType];

export const ArtSubtype = {
  Painting: "painting",
  Sculpture: "sculpture",
  Print: "print",
} as const;

export type ArtSubtype = (typeof ArtSubtype)[keyof typeof ArtSubtype];
```

Codegen Is Not Part of KP:1

The codegen pipeline is an **implementation concern**. KP:1 specifies the YAML schema; how consumers transform it is their business. The spec guarantees that the YAML is machine-parseable and that the schema is stable within a version.

8. Migration Path

From TypeScript to KP Definitions

For existing systems with hand-maintained TypeScript types (e.g., `timbuktu-registry/src/taxonomy/definitions.ts`):

1. **Extract** — Read existing TypeScript and produce equivalent YAML definitions
2. **Author** — Create the `.kpack/` directory with `PACK.yaml` and `definitions/`
3. **Generate** — Build codegen script that produces TypeScript from YAML
4. **Parity test** — Generated TypeScript must be functionally identical to hand-written
5. **Flip** — Switch consumers to import from generated artifacts
6. **Delete** — Remove hand-maintained source after a stable cycle

Migration Order (Recommended)

Phase	Content	Difficulty	Impact
1	Spec extension (this document)	Low	Foundation for everything else
2	Facet routing policies	Low	Already declarative YAML
3	Core taxonomy definitions	Medium	45+ entity types, many attributes
4	Perspective catalog	Medium	Prompt templates add complexity

Phase	Content	Difficulty	Impact
5	CI validation	Low	Ensures ongoing correctness

9. Validation Rules

Definition Validation

- Every definition file has `kind: definition` and `authority` and `version` and `status`
- Entity type `id` values are unique across all definition files in the pack
- Attribute `applies_to` references existing entity type IDs
- Relation type `source_types` and `target_types` reference existing entity type IDs
- Subtypes are unique within their parent entity type
- If `status: deprecated`, `superseded_by` should be set

Policy Validation

- Every policy file has `kind: policy` and `authority` and `version` and `status`
- Rule `id` values are unique across all policy files in the pack
- Rules have a non-empty `condition` and `action`
- Priority values are positive integers
- If a rule references entity types (in conditions), those types should exist in a definition pack

Cross-Pack Validation

- Claims referencing entity types should match types defined in the ontology layer
- Policies routing to perspectives should reference perspectives defined in a definition pack
- Deprecated definitions should not be referenced by active claims or policies without a warning

Deprecation and Orphan Handling

When a definition is deprecated, existing claims referencing it are not invalidated:

1. **Claims are historical assertions.** A claim made when `entity.type == 'widget'` was active remains valid even after `widget` is deprecated. The claim's `since` date establishes its temporal context.
2. **Tooling should warn, not reject.** When a claim references a deprecated entity type, attribute, or relation, emit a warning: "entity type `widget` deprecated since 2026-06-01 — consider updating to `gadget`."
3. **`superseded_by` enables migration.** If the deprecated definition has `superseded_by: gadget`, tooling can suggest the replacement. Automated migration is an implementation choice, not a spec requirement.
4. **Orphan detection.** A claim referencing an entity type that exists in NO definition pack (not deprecated, not active — simply absent) is an orphan. Tooling should flag orphans for review — they may indicate a missing dependency declaration or a data entry error.
5. **Retroactive invalidation is forbidden.** Changing a definition must never silently invalidate existing claims. This follows from the append-only principle ([RATIONALE.md §1](#), Principle 5): the past is preserved.

10. Design Principles

1. **Deterministic, not probabilistic.** Definitions and policies have no confidence scores. "A painting is a type of artwork" is a structural decision, not an uncertain belief.
2. **Authoritative, not evidenced — but not uncited.** Definitions cite authority (who decided) rather than evidence (what proves it). Domain structure is declared, not discovered. However, the optional `rationale` field documents *why* a definition exists — deterministic does not mean unexplained.
3. **YAML, not markdown.** Definitions are structured data (YAML) because they're consumed programmatically. Claims are markdown because they're consumed by LLMs. Match the format to the consumer.

4. **Constraining, not enforcing.** Definitions inform but do not block. An LLM can reason about a claim that doesn't match the taxonomy — the mismatch is a signal, not an error.
5. **Portable.** Definition packs follow the Voyager Principle. A `.kpack/` directory with YAML definitions is readable by any intelligence without infrastructure.
6. **Versionable.** Definitions evolve. The `version`, `status`, `effective_from`, `effective_to`, and `superseded_by` fields support graceful evolution without breaking consumers.

Boundary Decision Table

When classifying domain content, use this table to determine the correct kind:

Content	Kind	Why NOT the other?
Entity type catalog (art, watch, wine)	definition	Structural vocabulary, not a rule about what to do
"Art entities should prioritize authentication"	policy	Behavioral rule, not structural vocabulary
Attribute vocabulary (medium, dimensions)	definition	What attributes exist, not what to do with them
"Which perspectives apply to which entity types"	policy	Routing behavior, not domain structure
Relation types (created_by, provenance_chain)	definition	What relations exist, not when to use them
"High-value items require expedited review"	policy	Threshold rule governing behavior
"This specific Monet is oil on canvas"	claim	Assertion about a real entity (has uncertainty)
Industry classification standard (e.g., Getty AAT)	definition	External vocabulary adopted as structure

The test: Remove all verbs like "should", "must", "prioritize", "require." If the content still makes sense, it's a definition. If the verbs are the point, it's a policy. If it's about a specific entity instance, it's a claim.

Catalog-Level Entities (The Gray Area)

Alternative assets often have a hierarchy between the abstract type and the specific item:

Domain (Watch) → Brand (Rolex) → Model (Daytona) → Reference (116500LN) → Item (Serial #78432)

Where does the **definition** end and the **claim** begin?

Level	Kind	Rationale
Domain (watch)	definition	Structural — defines the vertical
Brand (rolex)	Edge case	Could be a definition (vocabulary) or a claim entity
Model (daytona)	claim entity	Specific enough to have disputed properties
Reference (116500LN)	claim entity	Has attributes that change over time (market price)
Item (serial #78432)	claim entity	Specific real-world thing with unique provenance

The rule of thumb: If it's stable enough to appear in a dropdown menu and never needs a confidence score, it can be a definition (entity type or subtype). If it has properties that different sources might dispute or that change over time, it's a claim entity.

For most domains: Brands and product lines sit at the boundary. The pragmatic choice is:

- **Subtypes in definitions** for broad categories (painting, sculpture) that are universally agreed
- **Claim entities** for specific brands, models, and references that have market-relevant properties

This keeps definitions stable (slow-changing vocabulary) and claims dynamic (fast-changing facts about specific things). A definition pack should not need to update when Rolex releases a new reference number — that's entity knowledge, not domain structure.

Future Work

The following are recognized gaps deferred to future spec revisions:

- **Attribute inheritance.** If `art` defines `dimensions` as a key attribute, subtypes like `painting` should inherit it. The schema currently has no inheritance or override mechanism. Until formalized, the convention is: subtypes inherit all parent attributes implicitly; overrides are documented in comments.
- **Temporal claim validation.** When a definition changes (`effective_from: 2026-06-01`), claims made under the old definition (`since: 2025-01-01`) should validate against the definition active at claim

time, not the current definition. This requires a temporal resolution mechanism that is not yet specified.

- **Local attribute constraints.** Entity-type-level overrides for attribute properties (e.g., `condition` required for `watch` but optional for `real_estate`) are not supported. The `attributes.yaml` schema defines global defaults; type-specific overrides require a mechanism for local constraint declaration.

Extensions — Producer-Defined Manifest Metadata

Status: Draft **Date:** 2026-04-28 **Editor:** Timothy Kompanchenko **Parent:** SPEC.md §3.2, CORE.md §3

1. Overview

KP:1 keeps the manifest root closed: fields not defined by the schema are invalid. Experimental or implementation-specific metadata **MUST** live under the optional `extensions` object instead of appearing as new top-level keys (CORE.md §3 "Manifest Extensions", SPEC.md §3.2).

This document is the catalogue of **producer-defined** extension blocks in active use across the reference implementation. It is informative — KP:1 does not require any extension to exist, and consumers **MUST** ignore extension content they do not understand. New extensions are listed here so producers and consumers can converge on shared shapes without forcing schema breaks or prematurely promoting a payload into the core standard. Requirements below are conditional profile contracts, not requirements for every KP:1 pack. Generic readers preserve source files and ignore unknown profile semantics; producers must name/version a profile when they rely on its additional capabilities. A catalog entry does not establish that all reference tooling implements it.

Reading Guide

Each extension below specifies:

- **Owner** — which subsystem produces or consumes the block.
- **Status** — experimental, active, or deprecated.
- **Shape** — the YAML/JSON structure under `extensions.<key>`.
- **Compatibility** — what consumers are expected to do when the block is absent, malformed, or carries unknown keys.

2. Producer-defined extension blocks

2.1 `extensions.ai_brief` — analyst-facing verdict line

Owner: reference producer-side enrichment implementation. **Status:** active.

The illustrative extension shipped with v0.7.4 (SPEC.md §3.2). A short headline and verdict produced at end-of-enrichment, surfaced by analyst tooling as a one-line summary above the dossier. See SPEC.md for the canonical example.

2.2 `extensions.entities` — typed entity graph

Owner: reference entity-extraction implementation. **Status:** experimental (introduced 2026-04-28).

Promotes the in-memory entity graph to first-class manifest metadata. Consumed by Decision Workspace materialization and the cross-pack `entity_index` in the reference relay.

```
extensions:
  entities:
    - id: ent_p_8a3f2c          # canonical reference ID, see §3.1
      type: person             # person | company | asset | event
      canonical: Alexei Mordashov # display name, single source
      aliases:
        - Алексей Александрович Мордашов
        - Mordashov
      external_ids:             # see §3.2; producers fill what they have
        opensanctions: Q123456
        companies_house: "01495478"
        ofsi: RUS0770
```

```
wikidata: Q1413288
first_seen_in: [E001, E007]
confidence: high
role_in_pack: subject

# evidence IDs that established this entity
# high | medium | low | unknown
# subject | counterpart | beneficiary
# | sibling | director | other
```

Cardinality. A pack MAY emit zero or more entities. The pack's foreground entity (when one exists) is conventionally first in the array but order is non-semantic; consumers MUST identify the foreground via `role_in_pack: subject` rather than position.

ID stability. Within a pack, an entity ID never changes once written to a sealed manifest. Across packs, the relay's `entity_index` resolves identity by `external_ids` first, then by (canonical, aliases) overlap. See §3.1.

Compatibility. Consumers that do not understand the block MUST ignore it. Producers MAY omit the block entirely; downstream consumers fall back to the H1 entity annotation in `claims.md` (CORE.md AR-09) when present.

2.3 `extensions.relations` — typed edges

Owner: reference entity-extraction implementation. **Status:** experimental (introduced 2026-04-28).

Typed edges between entities declared in `extensions.entities`. Vocabulary mirrors `relation_types` defined in DEFINITIONS.md §3.

```
extensions:
  relations:
    - id: rel_001
      from: ent_p_8a3f2c
      to: ent_c_4b1d9e
      type: director_of
      since: "2018-03-15"
      until: null
      attributes:
        confidence: 0.9
        evidence_ids: [E007, E012]

# vocabulary from DEFINITIONS.md §3
# ISO-8601 date, optional
# ISO-8601 date, optional (null = current)
# edge_attributes from DEFINITIONS.md §3
```

Edge attribute discipline. Numeric confidence on a relation is machine-internal — surfaces in audit views only, never in user-facing prose. User-facing dossier copy uses the high/medium/low/unknown labels defined elsewhere.

Compatibility. As with §2.2.

2.4 `extensions.intent` — decision frame

Owner: reference producer-side intent inference implementation. **Status:** experimental (introduced 2026-04-28).

The decision-frame tag selected for the pack. Used by the reference implementation's playbook module to choose (domain, intent) → Playbook.

```
extensions:
  intent: SanctionScreen
  intent_schema_version: "1"
  intent_derivation: explicit

# current reference vocabulary, v1
# pin against future intent-set expansions
# explicit | from_case_type | from_workspace
```

Intent vocabulary (reference implementation, v1): Buy, Consign, Comply, SanctionScreen, Advise, Cover, Discover. This is the producer-defined vocabulary used by the reference implementation; other producers MAY define their own values under `extensions.intent` per the informative-catalog disclaimer at the top of this document. Future reference-vocabulary expansions bump `intent_schema_version`; older clients detect-and-degrade.

Relationship to case_type. `case_type` (compliance | academic | canonical | journalism) is a stylistic / verdict-shape tag retained unchanged. `intent` is a sibling axis, not a replacement. A pack MAY carry both; defaults derive `intent` from `case_type` when absent.

2.5 `extensions.playbook` — runtime trace

Owner: reference producer-side enrichment loop, in concert with the reference intelligence pipeline.

Status: experimental (introduced 2026-04-28).

The structured trace of a playbook execution. Written incrementally as steps complete; the final manifest carries the full record. Consumed by the dossier renderer and the audit view.

```
extensions:
  playbook:
    id: sanction-screen.person           # playbook identifier
    intent_schema_version: "1"               # mirrors WorkspaceState.intentSchemaVersion
    started_at: "2026-04-28T15:00:00Z"
    completed_at: "2026-04-28T15:08:23Z"
    results:                                # one entry per executed step
      step_001_sanctions:
        status: complete                  # pending | running | complete | skipped | failed
        confidence: high
        entities: [ent_p_8a3f2c]           # entity IDs targeted by this step
        provider: opensanctions            # primary provider (canonical for findings)
        provider_attempts:                   # full provider sequence trail
          - { provider: opensanctions, status: success, latency_ms: 412 }
          - { provider: ofsi, status: success, latency_ms: 1180 }
        findings_by_entity:                 # per-entity verdict
          ent_p_8a3f2c:
            finding: present
            evidence_ids: [E007, E008]
        new_entities: []                    # entities surfaced by this step (see §2.2)
        new_relations: []                  # relations surfaced (see §2.3)
        gaps: []                           # per Gap shape — {field, why, suggestedAction}
```

Streaming. Producers MAY write `results[step_id]` entries incrementally during enrichment; consumers MUST treat partial blocks as in-flight rather than malformed.

2.6 `extensions.workspace` — Decision Workspace metadata

Owner: reference workspace materialization implementation. **Status:** experimental (introduced 2026-04-28).

Per-pack workspace bookkeeping. The wedge ships with single-pack workspaces persisted as a `workspace.yaml` sidecar in the `.kpack/` directory; a small pointer block in the manifest records which workspace this pack belongs to.

```
extensions:
  workspace:
    id: ws_lemos_2026_04_28                # workspace ID, opaque
    constructor_enabled: true              # kill switch — when false, Bernard Live
                                           # constructor tool is not registered
    role: foreground                       # foreground | evidence (multi-pack only)
```

Compatibility. Pre-reframe packs without the block continue to enrich through the existing pipeline.

2.7 `extensions.research` — research metadata

Owner: reference research-metadata implementation. **Status:** active (precedent for the producer-defined pattern; emitted by ~16 packs in the reference implementation as of v0.7.7).

Free-text anchor subject and surrounding research metadata. Predates the typed entity graph; coexists with it. New work targets `extensions.entities` as the structured channel; `extensions.research.anchor_subject` remains valid for legacy and lightweight cases.

2.8 `extensions.translations` — evidentiary multilingual transcripts

Owner: the reference handwritten-inventory intake pipeline + the reference relay's intake module. Consumed by the reference macOS viewer and downstream relays. **Status:** active (introduced 2026-05-09 with the inaugural field-collected corpus).

When a claim's canonical text is English (per [MULTILINGUAL.md §2 P1](#)) but the underlying evidence is in another language — handwritten Russian/Ukrainian witness statements, foreign-language press,

recorded interviews — the `translations` block carries original-language transcripts as audit trail without making them co-canonical. The English claim remains the single normative assertion; the transcripts are evidence-shaped, not claim-shaped.

```
extensions:
  translations:
    ru:
      transcript: |
        Оригинальный текст показаний свидетеля...
      transcript_format: verbatim          # verbatim | summary | excerpt
      source_evidence_id: E007            # which evidence this transcript backs
      transcribed_by: field-team-mariupol
      transcribed_at: "2026-04-22T14:30:00Z"
    uk:
      transcript: |
        Оригінальний текст свідчення...
      transcript_format: verbatim
      source_evidence_id: E007
```

Schema. `translations: { [locale: string]: { transcript: string, transcript_format?: "verbatim" | "summary" | "excerpt", source_evidence_id?: string, transcribed_by?: string, transcribed_at?: string } }`. Locale keys are BCP-47 strings (per [MULTILINGUAL.md](#)).

Cardinality. This legacy shape has one transcript object per locale. It cannot hold multiple independent transcripts in the same language. Preserve all original source artifacts in evidence/attachments, with distinct evidence IDs and source associations; do not concatenate them silently or overwrite an earlier transcript. A profile needing structured multi-transcript storage must define a versioned carrier. This catalog does not invent that carrier.

Producer expectations. Producers SHOULD include `source_evidence_id` whenever the transcript is excerpted from a larger evidence record, so audit-trail reconstruction can map transcript → evidence → claim. When multiple transcripts back the same evidence record, locales sit side-by-side under the same `translations` block keyed by BCP-47 locale.

Cross-reference. See [MULTILINGUAL.md §12](#) for the normative exception this block carries — it is the sanctioned location for non-canonical-language text whose canonicalization-to-English would lose evidentiary fidelity.

Compatibility. Consumers that do not understand the block MUST ignore it. The English claim remains valid and complete on its own; the `translations` block adds audit trail without creating a parallel canonical surface.

2.9 `extensions.collection` — pack-bundle membership

Owner: reference intake implementation + reference macOS viewer library organization. **Status:** active (introduced 2026-05-24).

Declares that a pack belongs to a user-facing **collection** — a bundle of related packs sharing a real-world origin (a private art collection, a watch collection, a wine cellar, a portfolio of holdings). Collections are an **orthogonal organizational axis** alongside `tier`: (claim-clustering hubs, ≤20 sub_packs) and the viewer-derived `category` (first segment of `domain`:). A collection MAY span multiple categories — e.g., a private art collection may contain both antiquities and fine art under the `art/*` root.

```
extensions:
  collection:
    name: "Example Antiquities Collection" # required — display name, grouping key
    id: example-antiquities-collection     # optional — stable identifier (see compat)
    role: member                          # optional — hub | member | standalone (default: member)
    item_kind: object                     # optional — see "Inventory taxonomy" below
    item_role: artist                     # optional — finer classification within item_kind
```

Field semantics.

- `name` — required. The display name shown in the viewer's scope chip and used today as the grouping key. Two packs with the same `name` are treated as belonging to the same collection.

- `id` — optional. A stable, opaque identifier (e.g., kebab-case slug). Reserved for future use when collections become a top-level spec field with rename-safe references. Today, consumers MAY ignore `id` and group by `name`.
- `role` — optional enum (`hub` | `member` | `standalone`), defaults to `member`. `hub` marks a spine pack that describes the collection itself (typically `domain: art/private-collection` or similar) — viewer renders it as the collection cover when present. `member` is a regular pack in the collection. `standalone` is reserved for packs that historically carried the block but stand alone in the viewer; treat as "no collection" for grouping.
- `item_kind` — optional controlled-vocabulary enum identifying what kind of thing this pack represents *within the collection's inventory*. Drives the viewer's inventory-dashboard layout when the user scopes to a collection. Generalizable across art, watches, wine, real estate, and auction inventory. See the table below.
- `item_role` — optional free-form string narrowing the `item_kind`. e.g. a pack with `item_kind: person` might carry `item_role: artist` | `scholar` | `dealer` | `family` | `owner` | `auctioneer`; a place might carry `item_role: room` | `gallery` | `auction_house` | `residence` | `region`; an event might carry `item_role: exhibition` | `auction` | `sale` | `acquisition` | `appraisal`. Consumers MAY surface these as filter pills within the kind's section. Unknown values are ignored gracefully.

Inventory taxonomy (`item_kind`)

Value	Meaning	Private-collection example	Auction-house analogue
<code>spine</code>	Collection's overview/hub pack (typically one per collection)	<code>example-antiquities-collection</code>	The auction-house catalog pack
<code>subcollection</code>	Category hub that groups objects (a sub-bucket within the collection)	<code>category-antiquities</code>	Department (Impressionist, Contemporary)
<code>object</code>	Individual inventory item	<code>category-antiquities-001-example-jade-figure</code>	Lot
<code>person</code>	Any human entity	Artists, scholars, family members, owners	Consigners, buyers, specialists
<code>place</code>	Physical location or organization-with-location	Rooms, residences, galleries, auction houses	Sale rooms, regions
<code>period</code>	Temporal classification (era, dynasty, vintage)	Tang Dynasty, Hellenistic period	Era, vintage year
<code>event</code>	Exhibition, auction, sale, acquisition, appraisal	1985 university loan exhibition	Auctions, sales
<code>document</code>	Records (appraisals, catalogs, certificates, invoices)	2022 specialist appraisal	Sale catalogs, condition reports
<code>narrative</code>	Story content (voice briefings, monographs, voice scripts)	<code>narration-collection-overview</code>	Sale-room essays

Item kinds are display-axes, not access-control or routing axes. A pack's `item_kind` controls how it appears in the collection inventory dashboard (which section, which filter pill). It does not affect search, embeddings, sealing, or any other system. Producers MAY omit `item_kind` entirely; the viewer falls back to name-prefix inference for legacy packs and otherwise shows the pack as a generic "other" entry.

One spine per collection (convention, not enforced). A collection should have at most one pack with `item_kind: spine` — the cover/overview pack. If multiple are present, the viewer picks one deterministically (typically the oldest-sealed or alphabetically first). This is a convention, not a hard constraint; the parser does not error.

Distinction from `tier: hub`. `tier: hub` (spec) is for small claim clusters (≤ 20 sub_packs) that declare their children explicitly via `sub_packs: .extensions.collection` is for large user-facing bundles (hundreds of items possible) where members share a `name` rather than being pre-declared. Both MAY coexist on the same pack; they describe different concerns.

Distinction from `category`. `category` is derived at scan time from the first segment of `domain:` and is not authored. A collection MAY span multiple categories. Producers MUST NOT use `extensions.collection` to override or canonicalize category derivation.

Compatibility. Consumers that do not understand the block **MUST** ignore it. Producers **MAY** omit the block entirely; packs without it appear in the viewer's "Uncollected" section (or the category section, depending on grouping mode). The reference macOS viewer treats `name` as authoritative for grouping today; if/when `collection:` is promoted to a top-level spec field with `id` semantics, this catalogue entry will be updated and an alias-and-migrate path documented.

Authoring expectations. When an intake pipeline detects sibling structure (shared origin, similar domains, shared creator/location/period metadata) it **SHOULD** propose a collection name to the user rather than emit individual standalone packs. The user accepts, edits, or rejects the proposal; on acceptance, every pack in the batch receives the same `name` and `role: member` (or `role: hub` for the spine pack if one exists).

2.10 `extensions.audience` — producer-side named recipients

Owner: authored manifests + the reference macOS viewer's audience badges and recipient gates.

Status: active (canonical location as of 2026-06-14; migrated from the legacy top-level `audience:` key).

A flat list of opaque producer-defined identifiers naming **who a pack is authored for** — the producer-side answer to "who is this for."

```
extensions:
  audience: [alice, bob]
```

Distinct from the two existing audience mechanisms. KP:1 already expresses audience two other ways; this field is neither:

- **Views** (CORE.md §11) — each view is an *aimed surface* (register/voice for a reader). `extensions.audience` names *people*, not surfaces.
- **PLAYBACK AudienceProfile** (PLAYBACK.md §3) — the *session-time* listener context (familiarity, purpose, domain_lens, register) chosen at playback. `extensions.audience` is *authoring-time* recipient identity — an orthogonal axis that deliberately keeps a separate vocabulary.

Shape. A flat array of string identifiers, mirroring `PackManifest.audience: [String]` in the reference runtime. A richer per-recipient form (`[{id, role, view}]` distinguishing recipient vs. operator) was considered and deferred: the surprise gate that would consume `role` is currently retired (see §2.11), so the enrichment is not yet load-bearing.

Compatibility. Consumers that do not understand the block **MUST** ignore it. The reference loader accepts the field in either location — `extensions.audience` (canonical) or the legacy top-level `audience:` key — so packs may be migrated one at a time without a flag day. New packs **SHOULD** write `extensions.audience`; the top-level `audience:` key is **deprecated** — it violated the closed manifest root (CORE.md §3 "Manifest Extensions").

2.11 `extensions.sharing` — gift-gating / do-not-share metadata

Owner: the reference macOS viewer's surprise-pack gate and library filtering. **Status:** active, gate-dormant (canonical location as of 2026-06-14; migrated from the legacy `lifecycle.surprise` / `lifecycle.do_not_share` keys).

Marks a pack as a **sealed gift** that must not surface in demos, pack browsers, screenshots, or shared contexts until a human has presented it to its recipient.

```
extensions:
  sharing:
    surprise: true
    do_not_share: >
      This pack is a surprise gift — treat it as a sealed envelope addressed to
      someone else.
```

Gate status. Per the reference implementation's 2026-05-20 "just on, singular solution" directive, `SurprisePackGate.isUnlocked` returns `true` unconditionally — the gate is **retired** and these fields are dormant metadata, preserved so a re-enable stays a one-line revert. Relocating them changes no live behavior.

Why not lifecycle. These were authored under `lifecycle.{surprise,do_not_share}`, but `lifecycle` is a *closed* core object (CORE §3 / the schema) reserved for archive-management fields. Gift-gating is implementation-specific viewer behavior, so it belongs under `extensions` per the closed-root rule.

Compatibility. Consumers that do not understand the block **MUST** ignore it. The reference loader accepts these fields in either location — `extensions.sharing.*` (canonical) or the legacy `lifecycle.surprise / lifecycle.do_not_share` keys — so packs may be migrated one at a time. New packs **SHOULD** write `extensions.sharing`; the `lifecycle.*` gift-gate keys are **deprecated**.

2.12 `extensions.case` — situation-anchored case composition

Owner: producer-side case materialization implementation (writer) + consuming library (case cards, roster surfaces); member resolution by the consuming library or relay at serve time. **Status:** experimental (introduced 2026-07-19).

A **case pack** is an ordinary pack whose single subject is a *situation* — a deal under review, a client-development campaign, a matter to be decided — rather than a person, company, or asset. The `case` block declares the situation's flavor (`kind`) and the case's **member references**: pinned pointers to the standalone packs the case draws on.

A case is not a new artifact class. It is a pack, and it inherits sealing, provenance, views, sharing, and supersession unchanged. The one-subject rule is also unchanged (CORE.md AR-09): a case pack has exactly one foreground entity — the situation itself, conventionally an event entity (`ent_v_`, §3.1; usage conventions in §3.5). People, companies, and assets involved in the situation are *participants*, declared under `extensions.entities` (§2.2) with roles and per-claim `entity_refs` (§3.3); they are never additional anchors. If a design appears to need a second anchor, model the situation that is actually the subject and anchor there.

```
extensions:
  case:
    kind: deal                                # deal | campaign — reference vocabulary v1
    members:
      - pack_id: "0b3c1f9a-4d7e-4a1c-9f2b-3e4a5b6c7d8e"
        revision: "2026.07.02"                # the member's PACK.yaml `version`, pinned
        role: subject                         # subject | evidence | inventory | prospect
        entity: ent_p_8a3f2c                  # optional — participant this member documents
      - pack_id: "3f9e2d61-8c4b-4f2e-b1a7-5d6e7f8a9b0c"
        revision: "2026.06.28"
        role: subject
        entity: ent_a_7e2f01
      - pack_id: "9d8c7b6a-5e4f-4d3c-a2b1-0f9e8d7c6b5a"
        revision: "2026.05.30"
        role: evidence
```

Case kinds (reference vocabulary v1).

- `deal` — a small-cast situation: two or three central participants plus a bridge of relationship questions between them. The canonical shape is a consignment review — consignor, work, and the institution's decision. `examples/auction-house-consignment-review.kpack` models the scenario (the example predates this block and does not carry it).
- `campaign` — a cohort situation: many prospective participants, an inventory of assets, and a purpose (an outreach program around an exhibition, fair, or season). Most of a campaign's cohort enters as in-pack **stubs**, not members — see "Members vs stubs" below.

The kind vocabulary is closed in the same spirit as the relation vocabulary (§3.4): new kinds are deliberate registrations in this catalogue, not ad hoc values. Consumers encountering an unknown kind **SHOULD** degrade to generic case handling (render members; skip kind-specific surfaces).

Member fields.

- `pack_id` — required. The member's stable identity across versions — the `pack_id` documented in ARCHIVE.md §4. This is reference *identity*, deliberately not content.
- `revision` — required. The member's `PACK.yaml` version string as sealed (the `pack_version` receipt in the member's `signatures.yaml`). References are **pinned, always**: when its supporting profile provides unambiguous resolution and retains the referenced artifacts, a case can record what was

known when the decision was made — the audit question is "what did you know when you advised this?", and a floating reference cannot answer it. Producers **MUST NOT** emit revision-less members. Freshness is a serving/UI concern — see the staleness signal below.

- **role** — required. One of the closed v1 vocabulary below. Consumers that do not recognize a role **MUST** still treat the entry as a well-formed member (resolvable, renderable) without role-specific behavior.
- **entity** — optional. The `ent_*` ID (§3.1) of the participant in this case's `extensions.entities` that the member pack documents. This is what ties a roster row to its dossier after promotion.

Role	Meaning	Typical kind
subject	Dossier for a central participant of the situation	deal
evidence	Supporting material the case's findings draw on	any
inventory	Member of the asset set the situation offers or matches against	campaign
prospect	Dossier for a cohort member under consideration	campaign

Reference pinning and verification (no member hashes — by design). A member reference pins *identity* (`pack_id` + `revision`) and deliberately does **not** carry the member's `pack_hash` or any content digest. This dissolves the composite signing-chain problem instead of solving it:

- *No hash cycles.* The case's seal ([ARCHIVE.md](#) §3–§4) is computed over the case's own files only. Since member entries embed no member digests, sealing order between packs never matters, and packs that reference each other cannot deadlock each other's seals.
- *No parent-deed / child-deed entanglement.* Re-sealing a member (a new dossier revision) never invalidates the case's seal, and a case's seal asserts nothing about member content. No custody chain flows through a case into its members.
- *A case verifies standalone.* "Verified" for a case archive means: its own seal is valid and its member references are well-formed. Verifiers **MUST NOT** require member archives to be present or resolvable to verify a case.
- *Member presence is a serving question, never a validity question.* Whether a pinned member is **present**, **absent**, or **superseded** (a verified descendant revision of the pinned member is known — [ARCHIVE.md](#) §5, [LIFECYCLE.md](#) §6) is answered by the consuming library or relay at serve time, and **MUST** be reported *alongside* verification results, never conflated with them.
- *Member authenticity is the member's own.* Because the reference is a name and version, not a checksum, content authenticity of a resolved member is established by the member's own seal at resolve time — never by the case.

Pinned resolution and staleness. `pack_id` plus `revision` is a logical reference, not a unique content digest. A supporting profile must make resolution unambiguous within its library, retain pinned artifacts, and report absent or ambiguous pairs without selecting an arbitrary candidate. Concurrent branches may carry the same version string; CalVer alone does not establish ancestry. A newer known revision may be a sibling, not a superseder. Report "newer revision available" separately from supersession established by verified lineage. Surfaces **SHOULD** offer re-pinning as a deliberate act producing a new case revision; they **MUST NOT** silently resolve a pin to another revision. The case's own seal proves none of these resolver facts.

Members vs stubs; promotion. Members are references to *real packs*. A campaign's cohort largely is not that: bulk intake produces **stubs** — participant entities and their claims *inside* the case pack (roster rows), with honest resolution-fidelity downgrades, not dozens of thin packs. Promotion is the deliberate act that turns a stub into a dossier: a precise identity resolution, a standalone pack created through the normal research flow (seeded from the stub's external IDs and the case's evidence), and a **new case revision** whose member list gains a pinned reference (`role: prospect` OR `subject`, with `entity` pointing at the participant). Claims never migrate between packs: the stub's claims are superseded in the case revision per the ordinary supersession rules, and the new dossier's claims carry their own provenance from birth. Fidelity metadata (stub vs promoted) **SHOULD** travel with any retrieval or roster surface, so bulk-resolved and precise-resolved knowledge are never presented as equivalent.

Distinct from the neighboring mechanisms. Four existing features solve adjacent problems; case replaces none of them:

- **COMPOSITION.md** `composition.yaml` — a *loading lens*: agenda-shaped references **by pack name**, un-pinned, chosen for a session (`pre_load` / `on_demand`). A case pins identity + revision for the decision record. A case pack MAY additionally carry `composition.yaml` (e.g., to brief a meeting about the case).
- `extensions.collection` (§2.9) — organizational bundling by shared `name`; no pinning, no per-member roles, no decision semantics.
- `tier`: `hub` + `sub_packs` — small claim-clustering hubs with pre-declared children.
- `extensions.workspace` `role`: `foreground` | `evidence` (§2.6) — per-pack workspace bookkeeping recorded *on the member side*. Case membership is recorded *on the case side*; only the case's own list is authoritative for the case.

Compatibility. Consumers that do not understand the block MUST ignore it; the pack remains an ordinary pack whose claims and views stand alone. Consumers that understand `case` but cannot resolve members (offline, a single shared archive, a partial library) MUST NOT report the pack as invalid or incomplete — member resolution is a serving feature layered on top of an already-complete pack. Producers MAY emit `members`: `[]` (a case that so far contains only stubs and in-case findings).

3. Cross-cutting concerns

3.1 Canonical entity IDs

Entity IDs in `extensions.entities[].id` follow the form:

```
ent_<type>_<6-hex>           # ent_p_8a3f2c, ent_c_4b1d9e, ent_a_7e2f01, ent_v_3ab9cc
```

Where `<type>` is `p` (person), `c` (company), `a` (asset), `v` (event — `v` chosen to avoid collision with evidence's `e` prefix).

IDs are deterministic SHA-256 of `(type, canonical, first_external_id_if_any)`, truncated to six hex characters. Re-runs of the extractor on the same source produce the same ID.

Cross-pack stability is provided by the relay's `entity_index` table, which maps `(canonical, external_ids)` triples to `[pack_id, ent_id]` pairs. Two packs with the same `external_ids.opensanctions: Q123456` resolve to the same entity even when the per-pack IDs differ.

3.2 External identifier vocabulary

`extensions.entities[].external_ids` is an open-keyed map. Producers fill what they have; consumers MUST tolerate unknown keys. The recommended vocabulary:

Key	Source	Format
<code>opensanctions</code>	OpenSanctions API	string ("Q123456")
<code>companies_house</code>	UK Companies House	8-digit string
<code>ofsi</code>	UK OFSI list	string ("RUS0770")
<code>ofac_sdn</code>	OFAC SDN list	integer
<code>eu_consolidated</code>	EU consolidated list	string
<code>un_consolidated</code>	UN consolidated list	string
<code>orcid</code>	ORCID API	"0000-0001-2345-6789"
<code>opencorporates_<jurisdiction></code>	OpenCorporates	path-style ID
<code>sec_cik</code>	SEC EDGAR	10-digit CIK
<code>wikidata</code>	Wikidata	"Q\d+"
<code>courtlistener_party</code>	CourtListener	integer
<code>lei</code>	GLEIF	20-character LEI

Producers MUST NOT emit malformed external IDs. The reference implementation's external-ID parsers validate format strictly; values that fail validation are dropped from the pack and recorded as a Gap in the playbook trace.

3.3 Per-claim entity references

An implementation MAY annotate which entities a structured operational claim describes via the `extensions` lane on the structured claim record (KPClaim, [STORAGE.md §3.2](#)):

```
- id: C014
  text: "Mordashov became Severstal CEO in 1996."
  extensions:
    entity_refs:                                # which entities this claim is about
      - id: ent_p_8a3f2c                        # subject | object | other
        role: subject
      - id: ent_c_severstal
        role: object
    relation_refs: [rel_001]                    # relations established by this claim
```

Portability limit. This structured-record example is implementation-local. KP:1 currently defines no file carrier that serializes these per-claim associations into `claims.md`; they cannot be reconstructed from a generic directory merely because the manifest has `extensions.entities`. A supporting profile must define a versioned file mapping before claiming portable reconstruction. Do not add an undocumented claim metadata field.

This complements the H1 entity annotation in `claims.md` ([CORE.md AR-09](#)). The annotation declares the pack's *one* foreground entity for legacy compatibility; per-claim `entity_refs[]` link specific claims to the typed entities under `extensions.entities`.

3.4 Relation vocabulary conventions (closed sets, extended per playbook)

`extensions.relations[].type` (§2.3) draws from the relation-type vocabulary defined in [DEFINITIONS.md §3](#). The growth convention: the vocabulary is a **closed set, extended per playbook** — each playbook family declares the relation types its steps may emit, and additions are deliberate registrations (a definition-pack change plus a row here), never ad hoc strings invented at emission time. Consumers MUST ignore relation instances whose type they do not recognize (§2.3 compatibility), so registration cost stays low while the graph stays queryable.

The case playbooks (§2.12) introduce the initial case vocabulary:

id	From → to	Semantics
consigns	person → asset	The person offers the asset for consignment (the deal bridge). Temporal (since/until); inverse <code>consigned_by</code> .
fits_budget	person → asset	The participant's documented budget range accommodates the asset's expected price. A matching <i>finding</i> — verdict-less, evidence-cited.
interested_in	person → asset person	Documented interest in a work, a maker, or a category. A matching <i>finding</i> — verdict-less, evidence-cited.
possibly_same	entity ↔ entity (same type)	Uncertain identity between two entities. A <i>question</i> , never an assertion. Symmetric; self-inverse.

Findings-shaped relations. `fits_budget` and `interested_in` are written by the matching steps of a campaign playbook (§2.5 `findings_by_entity` is the per-step trace shape; the durable edges land here). They MUST carry `attributes.evidence_ids`, and SHOULD cite both endpoints' evidence — the participant-side evidence (budget, stated interest) and the asset-side evidence (estimate, attribution). They are findings, not verdicts: the edge encodes no recommendation, and numeric confidence stays machine-internal per §2.3.

possibly_same discipline. Cross-pack identity merges happen only on *established* identity — matching `external_ids` (§3.2) or a resolved canonical (§3.1). When identity is merely suspected, producers MUST keep the entities separate and connect them with `possibly_same`. Consumers MUST NOT treat `possibly_same` as established identity (no index merging, no claim pooling across the edge); surfaces SHOULD present it as an open question. A silent merge is the graph-level wrong-subject dossier — the error class this discipline exists to prevent.

3.5 Event entities (`ent_v_`) — usage conventions

The `v` prefix has been reserved since this catalogue was introduced (§3.1); these conventions make event entities operational. An **event** is an occurrence that relates persons, companies, and assets

over time: an exhibition, an auction, a sale, an acquisition, an appraisal, a loan, a restitution — or a client-development campaign anchored on one (§2.12).

Projection fields. Consumers that compute typed per-entity projections SHOULD project three well-known fields for events:

Field	Shape	Projected from
date	ISO-8601 date or {start, end} range	Claims about when the event occurs, via entity_refs (§3.3)
location	string (venue / city)	Claims about where, via entity_refs
participants	list of ent_* refs with roles	Union of (a) entities co-referenced with the event on a claim (§3.3) and (b) entities linked to the event by a relation edge (§2.3)

Projections are **computed from claims, never authored**: each projected field carries the provenance of the claims it was computed from, disagreeing claims surface as disputes rather than silent overwrites, and a projection can be rebuilt only when its profile supplies a serialized association carrier and interpretation rules. The operational-only associations in §3.3 do not yet provide that portability. Projections are derived indexes; claims preserve assertions, including uncertainty and dispute, rather than certifying source truth.

Events as case anchors. A campaign case pack (§2.12) conventionally anchors on an event entity: the foreground entity (role_in_pack: subject) is the ent_v_ for the campaign or the occasion it targets, and cohort participants relate to it.

Why events matter to provenance. A provenance history is a chain of events — sale, exhibition, transfer, restitution — each relating persons to a work over time. Modeling those occurrences as first-class ent_v_ entities with date projections makes provenance graph-traversable (artwork → events → persons, with dates) instead of prose-bound; questions like "is there a gap in this ownership window?" become traversals over event sequences.

Distinct from item_kind: event. The collection taxonomy's event (§2.9) is a *display axis* — which dashboard section a pack renders in. An ent_v_ entity is *graph identity* — a node that claims and relations attach to. A pack about an exhibition may well carry both; they answer different questions.

4. Migration

4.1 From entities.md to extensions.entities

The entities.md file (SPEC.md §7) is **deprecated** as of v0.7.7 (2026-04-28). It saw use in five packs out of ~770 and never landed as a practical primitive — its free-text Relationships block was never machine-consumed.

New packs MUST NOT emit entities.md. Producers MUST write extensions.entities and extensions.relations instead. A one-shot migration script (scripts/migrate-entities-md-to-extensions.mjs) walks the legacy packs and rewrites their content into the new shape.

4.2 From free-text anchor_subject to typed entities

extensions.research.anchor_subject (a free-text string) coexists with extensions.entities for the wedge. New work targets the typed channel; the free-text field is retained for legacy packs and for lightweight cases that do not warrant the full entity record.

5. Status and stability

Every extension in this document is **producer-defined** — its shape and semantics are governed by the producing subsystem, not by KP:1. Extensions graduate to core fields only after a future KP revision standardizes them.

Consumers MUST ignore extension content they do not understand. Extension content MUST NOT redefine the semantics of core KP fields or relax core validation rules.

The catalogue in this document is informative: producers register their extensions here so consumers can converge on shared shapes, but the authoritative shape lives in producer code (cited at the top of each section).

Notes Metadata — Knowledge Pack Companion Spec

Date: 2026-03-22 **Status:** Draft **Decisions:** D16 (recording consent reframed as notes)

1. Purpose

When an AI assistant participates in meetings, it can take structured notes — capturing key decisions, action items, names, and facts in knowledge pack format. This document specifies the metadata that tracks note-taking mode, disclosure status, and participant awareness.

2. Terminology

The system uses **"notes"**, not **"recording."**

Term	Meaning	Connotation
Recording	Passive capture of everything, full transcript	Surveillance-adjacent
Notes	Selective capture of important information	Professional assistance

An AI assistant taking notes during a meeting is analogous to a human assistant who sits in and writes down key points, action items, and decisions. This is the correct frame for the system's behavior.

3. Modes

The `notes.mode` field in `PACK.yaml` controls what the AI captures:

Mode	Captures	Use case	Disclosure	Consent
active	Structured extraction: decisions, action items, claims, key facts	Normal meetings	Disclosure recommended	Not required
passive	Full transcript + structured extraction	Legal depositions, formal negotiations	Required	Required
off	Nothing	No AI note-taking	N/A	N/A

Active Mode (Default for Meeting Packs)

The AI extracts structured information in real-time:

- Decisions made
- Action items with owners and deadlines
- New facts (names, numbers, dates)
- Claims that update or contradict standing knowledge

The output is a knowledge pack with `claims.md` (structured) and optionally a summary view. No raw transcript is stored as a primary artifact.

Passive Mode

Full transcript capture plus structured extraction. This mode is subject to recording consent laws, which vary by jurisdiction. The pack stores consent status explicitly.

Off Mode

The AI participates in the conversation but does not capture anything into a knowledge pack. Useful for informal conversations or when participants decline note-taking.

4. PACK.yaml Fields

```
notes:
  mode: active           # active | passive | off
  participants: [Alice, Bob] # All meeting participants
  disclosed: true        # Was AI note-taking disclosed?
  consent: null          # Required for mode: passive
                        #   obtained | declined | pending
```

Field Semantics

Field	Required	Description
mode	Yes	Note-taking mode for this pack
participants	Yes	All participants in the meeting/conversation
disclosed	Yes	Whether AI note-taking was disclosed to all participants
consent	Only for passive	Explicit consent status for full transcript recording

5. Disclosure vs Consent

Disclosure (for active mode): Informational. "My AI assistant is taking notes." This is professional courtesy, analogous to saying "Sarah is joining to take notes." The AI should be able to check this field and remind the user: "Note-taking disclosure is pending for this meeting."

Consent (for passive mode): Legal. "May I record this conversation?" Required in two-party consent jurisdictions. The AI must not activate passive mode without `consent: obtained`.

Consent Workflow

- Meeting starts with `consent: pending`
- User discloses AI note-taking to participants
- User updates `disclosed: true`
- If passive mode needed: user obtains and records consent
- If consent declined: system falls back to `mode: active` OR `mode: off`

6. Internal-Only Shorthand

For conversations where Tim is the only human participant (solo work sessions, Tim + AI only):

```
notes:
  mode: active
  participants: [Tim]
  disclosed: true          # Always true for internal
  consent: null            # Not applicable
```

No disclosure or consent needed. The AI takes notes freely.

7. Three-LLM Architecture Integration

The notes system integrates with the three-LLM conversation architecture:

LLM	Role in note-taking
Voice LLM	Fast conversation partner. Does not take notes directly.
Recording LLM	The note-taker. Receives transcript stream, extracts structured claims. Checks <code>notes.mode</code> to determine behavior.
Intelligence LLM	On-demand analyst. Results feed back to recording LLM for capture.

The recording LLM checks `notes.mode` before activating:

- `active` → Extract claims, action items, decisions
- `passive` → Full transcript + extraction (requires `consent: obtained`)
- `off` → Do not activate

8. Real-Time Consistency Checking

During `mode: active` or `mode: passive`, the recording LLM cross-references spoken claims against loaded knowledge packs. When a discrepancy is detected:

1. Recording LLM identifies the contradiction
2. Alert sent to visual channel (Canvas, phone notification — never spoken aloud)
3. Alert is non-accusatory: "Pack says X. Spoken claim was Y. Worth clarifying?"
4. User decides whether to address it in the conversation

Alert configuration in `PACK.yaml`:

```
notes:
  alert_threshold: 0.70      # Only alert on claims with confidence > threshold
  alert_channel: canvas     # canvas | notification | both
```

Low-confidence claims (`{0.40}`) are already uncertain — no need to alert. High-confidence claims (`{0.95}`) contradicted in conversation are a real signal.

Self-Driving Presentation Playback — Knowledge Pack Companion Spec

Date: 2026-05-09 **Status:** Experimental — schema may evolve before v0.9 **Editor:** Timothy Kompanchenko

1. Purpose

PLAYBACK is the spec layer for **self-driving voice presentations** of Knowledge Packs. A presentation is a structured, multi-phase narration where an assistant walks the listener through a pack's content — claims, evidence, red flags, decisions — at a chosen depth and tier.

This companion specifies:

- **PlaybackPlan** — the structured plan a renderer follows to deliver a presentation.
- **AudienceProfile** — the listener-specific context that shapes plan composition.
- **Discovery dialogue** — how the renderer establishes the audience profile at session start.
- **Phase advancement contract** — separating "what to play next" (data-driven) from "when to advance" (app-controlled).
- **Form variants** — short / medium / long with element-selection rules.
- **Pack-play vs library-play** — same contract, different planners.

Out of scope: visual choreography (auto-scroll, image reveal, view switching). PLAYBACK v1 is voice-only; visual extensions are a v2 concern.

2. Why a separate companion

VOICE.md governs **voice views** — pre-rendered prose delivered as a single artifact. PLAYBACK governs **dynamic narration** — a runtime composition that selects, sequences, and delivers content from a pack's claims, views, and extensions. Voice views are nouns (files); playback is a verb (process).

The relationship: PLAYBACK sits one layer above VOICE. A renderer consults the pack's voice views, claims, AI brief, and audience context, and produces a phase-by-phase narration tuned to the listener at session time.

3. AudienceProfile

AudienceProfile is the listener-specific context, structured as a small object passed once at the start of a presentation session. It governs which phases are included, how dense each is, and what register and emphasis the narration adopts.

```
audience_profile:
  familiarity: fresh           # fresh | some_context | expert
  duration_tier: medium       # short | medium | long
  purpose: orientation        # see "purpose" enum below
  domain_lens: buyer         # see "domain_lens" enum below
  register: curatorial        # plain | curatorial | technical | investor (per
VOICE.md §4.2)
  inferred: false            # true if the profile was derived from defaults rather
                             than dialogue
```

Field	Required	Type	Description
familiarity	Yes	enum	Listener's prior subject-matter exposure. Drives phase depth.
duration_tier	Yes	enum	Time budget. Drives phase count and word budget.
purpose	Yes	enum	Why the listener is here. Drives phase ordering and emphasis.

Field	Required	Type	Description
domain_lens	No	enum	Listener's role-frame within the pack's domain. Refines emphasis (e.g., a buyer hears value claims first; compliance hears risk first).
register	No	enum	Voice register from VOICE.md §4.2. Defaults to curatorial for art/heritage packs, plain otherwise.
inferred	Yes	boolean	true when the profile defaulted; false when explicitly captured via discovery dialogue.

3.1 purpose enum

The `purpose` field names why the listener is here. The original six values (`orientation`, `decision`, `risk_review`, `press`, `research`, `demo`) are joined by six use-case-named values added 2026-05-19:

Value	Listener context
<code>orientation</code>	New to the subject; wants the lay of the land.
<code>decision</code>	Has to decide something concrete (buy / consign / approve / reject).
<code>risk_review</code>	Reviewing for risk specifically — adverse media, sanctions, exposure.
<code>press</code>	Journalist or writer assembling a story.
<code>research</code>	Academic or analytic study; deeper than orientation, lighter than decision.
<code>demo</code>	Showing the system off; not a real research session.
<code>pre_meeting</code>	Brief before a meeting with the subject. Wants interests and obvious risks, fast.
<code>hiring</code>	HR vetting context. Wants credentials verified, achievements substantiated, adverse media checked.
<code>partnership</code>	Evaluating a potential partner or counterpart. Wants culture fit, track record, integrity surface.
<code>competitive_intel</code>	Studying a competitor. Wants activity, traction, relationships, market position.
<code>sales_prep</code>	Preparing a sales engagement. Wants decision-maker map, interests, hooks.
<code>appreciation</code>	Admiring or tracking the subject. Wants achievements, cultural resonance, integrity surface light.

3.2 domain_lens enum

The `domain_lens` field names the listener's role-frame. Original six values (`buyer`, `consigner`, `compliance`, `journalist`, `collector`, `general`) are joined by six use-case-named values added 2026-05-19:

Value	Role-frame
<code>buyer</code>	Listening as a potential buyer / acquirer.
<code>consigner</code>	Listening as a potential consigner / seller.
<code>compliance</code>	Listening as a compliance officer / auditor.
<code>journalist</code>	Listening as press / journalist / fact-checker.
<code>collector</code>	Listening as a private collector with no immediate transaction.
<code>general</code>	No specific role; orienting.
<code>hr_reviewer</code>	Listening as a hiring manager / HR / reference-check.
<code>partner_evaluator</code>	Listening as a potential business partner.
<code>competitor_analyst</code>	Listening as an analyst studying this subject as a competitor.
<code>admirer</code>	Listening to admire or learn about the subject for its own sake.
<code>meeting_host</code>	Listening shortly before a meeting with the subject.
<code>salesperson</code>	Listening to prepare a sales engagement.

3.3 Extensibility

Both enums are **extensible** — KP:1 v0.9 may add further values as new use cases emerge in production. Renderers **MUST** handle unknown enum values gracefully:

- Unknown `purpose` → treat as `orientation`.
- Unknown `domain_lens` → treat as `general`.

This prevents an older renderer from crashing on a newer pack's profile, and it lets new values ship via spec updates without renderer re-releases. Producers **SHOULD** use the registered values from the tables above whenever possible; **SHOULD NOT** invent ad-hoc values when an existing one fits.

`AudienceProfile.register` composes with — but does not replace — VOICE's producer-side `register`. Producer-side `register` controls how voice views are *written*; `AudienceProfile.register` is the listener-

side selection for the *current session*. Where they conflict, the session selection wins for that session's narration; the underlying voice view file is unchanged.

4. PlaybackPlan

A PlaybackPlan is the deterministic output of a planner that takes (pack, audience_profile, duration_tier) as input. It is the contract between the planner and the playback runtime.

```

playback_plan:
  pack_id: marquet-le-passeur
  audience_profile:
    familiarity: fresh
    duration_tier: medium
    purpose: orientation
    register: curatorial
    inferred: false
  duration_tier: medium
  phases:
    - phase_id: ph_001_verdict
      narrative_hint: "Open with the headline verdict."
      source_refs:
        - extensions.ai_brief.headline
      view_target: null # optional view to surface during this phase (visual
layer, v2)
      transition_intent: opening
      word_budget: 60
      min_tier: short # which duration tiers include this phase
    - phase_id: ph_002_red_flag
      narrative_hint: "Surface the highest-confidence concern."
      source_refs:
        - extensions.ai_brief.redFlags[0]
      transition_intent: emphasis
      word_budget: 80
      min_tier: short
    - phase_id: ph_003_claim_lead
      narrative_hint: "State the strongest claim with confidence."
      source_refs:
        - claims.C001
        - evidence.E007
      transition_intent: building
      word_budget: 120
      min_tier: medium
      # ... more phases ...
  resume_policy:
    on_interrupt: pause # pause | continue | restart_phase
    on_resume: restate_phase_lead # restate_phase_lead | continue_seamlessly
  handoff_policy:
    on_question: bookmark_then_handle # always interrupt for questions
    on_long_silence: pause # pause if listener is silent past inactivity_seconds
    inactivity_seconds: 12

```

Field	Required	Description
pack_id	Yes	Pack identity from PACK.yaml name. For library-play, pack_id is the library identifier.
audience_profile	Yes	The profile this plan was composed against. Stored for audit.
duration_tier	Yes	The selected tier; controls which phases are included via min_tier.
phases[]	Yes	Ordered phase list. Each phase has phase_id, narrative_hint, source_refs[], optional view_target, transition_intent, word_budget, min_tier.
phases[].source_refs[]	Yes	References into pack content the renderer reads from. See "Source-ref grammar" below for the normative path syntax.
phases[].view_target	No	Optional view name the renderer surfaces during the phase (visual layer extension).
phases[].transition_intent	Yes	Hint for prosody between phases: opening, emphasis, building, qualifying, closing.

Field	Required	Description
<code>phases[].word_budget</code>	Yes	Approximate word count budget for the phase narration.
<code>phases[].min_tier</code>	Yes	Smallest <code>duration_tier</code> that includes this phase. Lets one plan describe short/medium/long with a single ordered list.
<code>resume_policy</code>	Yes	Contract for what the runtime does when narration is paused (interruption or barge-in).
<code>handoff_policy</code>	Yes	Contract for question-handling and inactivity.

4.1 Source-ref grammar (normative)

Each entry in `source_refs[]` is a string with one of the following prefixes. Validators **MUST** reject plans whose `source_refs[]` entries do not match this grammar.

Prefix	Meaning	Example
<code>claims.<ID></code>	A claim ID from <code>claims.md</code> (matches the claim ID grammar in CORE.md §5).	<code>claims.C001</code>
<code>evidence.<ID></code>	An evidence ID from <code>evidence.md</code> .	<code>evidence.E007</code>
<code>history.<ID></code>	A superseded or retracted claim from <code>history.md</code> . Used when the playback plan needs to narrate why a claim changed.	<code>history.C001</code>
<code>views.<name></code>	A view declared in <code>PACK.yaml</code> 's <code>views</code> array (name field).	<code>views.briefing</code>
<code>extensions.<path></code>	A dotted path into the manifest's <code>extensions</code> block. Path segments are object keys; bracketed indices select array members.	<code>extensions.ai_brief.headline</code> , <code>extensions.ai_brief.redFlags[0]</code>

The grammar is regular: `^(claims|evidence|history|views)\. [^.] +$` for the first four forms; `^extensions\. [A-Za-z_][A-Za-z0-9_]* (\. [A-Za-z_][A-Za-z0-9_]* | \[ \d+ \] ) *$` for the extensions form. Producers **MUST** emit references that resolve to existing content in the pack at plan-generation time. Consumers **SHOULD** treat unresolvable references as soft errors (skip the phase or substitute a fallback) rather than fatal validation failures, since pack content can drift between plan generation and playback.

4.2 Form variants

A single `PlaybackPlan` can describe all three duration tiers via `min_tier` filtering on phases:

Tier	Target duration	Phase count	Includes
short	≤ 90 seconds	3-4	verdict + headline + 1 red flag + 1 next step
medium	~3 minutes	5-7	1. 2 primary claims + best evidence + uncertainty + closing
long	~8 minutes	10-14	1. counterpoints + Q&A pause invitations

The planner emits **ONE** plan that covers all three; the runtime filters by `min_tier ≤ duration_tier` to select the active phase set. This lets a listener say "give me the longer version" mid-session without re-planning — the runtime promotes to the next tier and includes additional phases.

5. Discovery dialogue

When `AudienceProfile.inferred` would be `true`, the renderer **SHOULD** attempt a brief dialogue at session start to capture the profile explicitly. The default dialogue is three short questions:

- **Q1** — "Before I walk this, are you hearing it fresh, or do you already know the subject?" Sets `familiarity`.
- **Q2** — "Do you want the quick read, the three-minute version, or the deep dive?" Sets `duration_tier`.
- **Q3** — (only if context is unclear) "And are you listening as a buyer, a consigner, compliance, press, or just orienting?" Sets `purpose` + optional `domain_lens`.

5.1 Defaults

When the listener says "just play it" without dialogue, the renderer SHOULD default to `{familiarity: fresh, duration_tier: medium, purpose: orientation}` and `set` `inferred: true`.

When the listener says "give me the short version" or similar, the renderer SHOULD skip discovery, `set` `duration_tier: short`, and `proceed`.

When the listener is a returning user (the renderer has prior-session context), the dialogue MAY be replaced with: "short, three-minute, or continue where we left off?"

6. Phase advancement — renderer-owned, not model-decided

The runtime that advances phases is part of the **renderer** (the application that consumes a `PlaybackPlan` and drives the voice pipeline), not a tool the language model can call. This is normative for renderers, since it is the contract that makes a `PlaybackPlan` interpretable consistently across renderer implementations:

Rule (P1). A KP:1-conformant renderer **MUST NOT** expose a phase-advancement tool to the language model. Phase advancement is decided by the renderer based on `PlaybackPlan` ordering, the listener's interruption signals, and the resume / handoff policies. A renderer that does expose such a tool **MUST** document the resulting non-conformance and **SHOULD NOT** advertise itself as KP:1 PLAYBACK-conformant.

Rationale: if the model could advance phases, it could also produce inconsistent narration sequences (skipping ahead, looping, going back) that would conflict with the deterministic plan. Keeping advancement in the renderer preserves the plan's value as a contract — a plan emitted by one composer renders identically across conformant renderers.

The model MAY have a tool that **captures the audience profile** at session start (one tool call). The model MAY have tools that **navigate to a view** or **read a section** as part of the visual layer (out of scope for v1). The phase-advancement tool is the one explicitly forbidden.

7. Pack-play vs library-play

The same `PlaybackPlan` schema serves both:

- **Pack-play** — narration of a single pack. The planner reads from `claims.md`, `views/voice/*.md`, `extensions.ai_brief`, etc.
- **Library-play** — narration of a library snapshot ("3 packs need attention; here are the highest-priority red flags"). The planner reads from a library-aggregator output that selects items across packs, and the `pack_id` field identifies the library rather than a single pack.

Library-play is **triage**, not concatenation. The planner **SHOULD NOT** produce a synthetic "mega-pack" by joining pack claims — that produces a shapeless narration. Instead, library-play selects the most decision-relevant items across packs and narrates them with cross-pack framing.

8. Concurrency

A presentation session typically uses a real-time voice pipeline (e.g., a streaming TTS / dialogue system) plus the application's playback controller.

Rule (P2). Only the real-time voice pipeline's actor MAY send the protocol's "create response" / "cancel response" / "create item" messages. Only the playback controller (or equivalent) MAY decide which phase is current. Stale calls — those originating from a previous phase epoch — **MUST** verify the current playback epoch before mutating playback state.

This rule prevents two well-known races: (a) advancing a phase mid-response causes the runtime to commit to two narrations at once; (b) a stale interrupt handler from a previous phase cancels the current phase's response.

The contract is enforceable at the application layer; PLAYBACK does not specify a wire-level invariant.

9. Status and stability

This entire companion is **experimental** for v0.8.5-preview. Producers MAY emit `PlaybackPlan` artifacts and consumers SHOULD treat them as advisory.

Commitments retained from v0.8.4-preview

The following are committed for the v0.8.4-preview tag and will not change without a deprecation cycle:

- The top-level `PlaybackPlan` shape (`pack_id`, `audience_profile`, `duration_tier`, `phases[]`, `resume_policy`, `handoff_policy`).
- `AudienceProfile` field set (`familiarity`, `duration_tier`, `purpose`, `domain_lens`, `register`, `inferred`) and the enum values for the four required fields.
- The four primary `register` values (`plain` / `curatorial` / `technical` / `investor`) and their tie to [VOICE.md §4.2](#).
- Rule P1 (the language model MUST NOT have a phase-advancement tool) and Rule P2 (only the real-time voice pipeline actor sends protocol messages; only the playback controller decides current phase). These are the architectural invariants the format depends on.
- The `min_tier` mechanism (one `PlaybackPlan` describes all three tiers via filtering).

Mutable before v0.9

The schema may evolve based on observed behavior:

- The phase `source_refs[]` vocabulary is likely to expand once real packs ship presentations.
- `transition_intent` enum values may consolidate or expand based on prosody experiments.
- `handoff_policy.inactivity_seconds` defaults will be tuned empirically.
- The `register` field's interaction with [VOICE.md §4.2](#) may need clarification once both fields ship in real packs.

Producers MAY emit `PlaybackPlan` artifacts in v0.8.5-preview packs as advisory metadata; consumers SHOULD treat them as informational and tolerant of schema drift.

The status will graduate from experimental to active after at least three real packs ship with playback plans and at least two distinct renderers have demonstrated successful end-to-end narration against the plans.

10. Related

- [VOICE.md §4.2](#) — the four primary voice registers (`plain`, `curatorial`, `technical`, `investor`). `AudienceProfile.register` is the session-time selector across this axis.
- [MULTILINGUAL.md §3.3](#) — locale-specific sub-registers (`Halychyna`, `Quebec`, `MSA`, etc.). `sub_register` is informational per [MULTILINGUAL.md §3.3](#) (not a normative enum); the runtime MAY honor it if set on the source voice view, but MUST not depend on its presence and SHOULD not treat its absence as an error.
- [EXTENSIONS.md §2.1 ai_brief](#) — `ai_brief.headline` / `redFlags` / `beCarefulAbout` / `next` are common phase `source_refs`.
- [COMPOSITION.md](#) — multi-pack composition for meeting packs. A meeting-pack `PlaybackPlan` may aggregate phases across the composed packs in the order `COMPOSITION` declares.

Reconciliation — Knowledge Pack Companion Spec

Date: 2026-05-09 **Status:** Stub — full design deferred to v0.9 / v1.0 **Editor:** Timothy Kompanchenko

Purpose

When two or more Knowledge Packs make claims about the same entity or topic and those claims drift apart over time — different confidence values, different evidence emphasis, contradictory positions — the ecosystem needs a **reconciliation protocol**. This companion will specify how packs surface drift, how reconciliation is triggered, and how the resulting consolidated view is encoded.

Status

This document is an **explicit placeholder**. The full reconciliation design is deferred to KP:1 v0.9 or v1.0, contingent on observing measurable drift across at least three real packs. Premature design before observed drift would be wishful architecture; the spec records the intent while honestly admitting the design is not yet ready.

The full protocol will become part of the public normative specification after at least three real-world drift instances have been observed and the design is informed by those instances. Until then, the public contract is to annotate disagreements and surface them for review.

Related, but distinct

- `CONSISTENCY.md` — current cross-pack consistency patrol. Detects contradictions and staleness across packs in a workspace; surfaces drift as a signal but does not consolidate.
- `LIFECYCLE.md` — pack-internal claim lifecycle (active / superseded / disputed / retracted). Operates within a single pack.

Reconciliation differs from both: it is the **cross-pack consolidation protocol** for genuine multi-source drift, applied after `CONSISTENCY` surfaces a sustained disagreement that `LIFECYCLE`-internal mechanisms cannot resolve.

What to do today (v0.8.5-preview)

Until the full protocol is specified, agents encountering cross-pack contradictions **SHOULD** produce a *consistency report* rather than a *reconciled pack*. The output is an annotated comparison, surfaced to a human reviewer, that uses the existing primitives:

- `⚡packB#claim-id` — name the cross-pack reference at the point of disagreement.
- `⊗!` / `⊗~` from `AUTHORING.md` §4 — apply *within* a single annotated pack to characterize the disagreement (use `⊗!` only when one side is demonstrably wrong; default to `⊗~` for genuine productive tension).
- **Do NOT silently merge.** Do not produce a third "reconciled" pack that erases the disagreement. The format's value is preserving epistemic tension, not flattening it.

Worked example

Pack A (`market-snapshot.kpack`) asserts:

```
- [C001] Global widget market is approximately $2B in Q1 2025.
  {0.85|r|E001|2025-03-31|investigated}
```

Pack B (`market-snapshot-v2.kpack`) asserts:

- [C001] Global widget market is approximately \$3B in Q4 2025.
{0.85|r|E007|2025-12-31|investigated}

A v0.8.5-preview reconciliation report (a new pack widget-market-reconciliation.kpack) would look like:

- [C001] Pack A reports widget market at ~\$2B (Q1 2025); Pack B reports ~\$3B (Q4 2025).
{0.95|o|E001,E002|2026-05-09} Direct observation that the two source packs assert different values. ↔market-snapshot#C001, ↔market-snapshot-v2#C001
- [C002] The figures concern different quarters and do not establish a contradiction.
{0.78|i|E001,E002|2026-05-09|investigated|judgment} ↔C001
- [C003] For a Q4 question, use the Q4 report with its stated methodology and retain the Q1 report as dated context.
{0.65|i|E001,E002|2026-05-09|investigated|judgment} ←C002
A human reviewer must confirm source quality and methodology before adopting either figure.

The bullets use CORE claim syntax (a full report pack also needs the manifest, header/frontmatter, section and local E001/E002 records), reference the source packs via ↔, and explicitly defer the final canonical answer. This is the v0.8.5-preview answer for cross-pack disagreement: **annotate, surface, defer to human** — do not auto-flatten into consensus.

The full reconciliation protocol (when it ships in v0.9 / v1.0) will define how reports like this become canonical reconciliations and what semantic constraints govern the consolidated graph.

KP:1 — Rationale and Positioning

Status: Draft (informative — not normative for KP:1 conformance) **Date:** 2026-05-09

This companion holds the parts of the KP:1 specification that are **rationale, philosophy, and positioning** rather than implementable contract. Sections that previously lived in SPEC.md as numbered chapters (§15 Design Principles, §17 Style Systems, §20 Relationship to Existing Standards) are collected here so that SPEC.md and CORE.md can serve their implementer-facing purpose without interleaving rationale with normative prose.

This document is **informative only**. Where it appears to specify behavior that is not also specified in CORE.md, SPEC.md, or a topic-authoritative companion, the rationale here does not constrain implementations. It records the reasoning behind the format's choices.

1. Design Principles

The 25 principles below informed the format's choices. They are reference material for understanding why KP:1 is shaped the way it is. They are not themselves a conformance contract — the conformance contract lives in CORE.md, the JSON Schema, and the PEG grammar.

1. **The Voyager Principle** — Zero dependencies beyond "a device that can process text." No internet, no registry, no parser. The files are the knowledge.
2. **Inspectability over readability** — Optimized for AI consumption. Humans CAN inspect (it's plain text), but format favors density over reading comfort.
3. **Density-optimized plain text** — Maximum knowledge per token. Compressed metadata {0.95|i|E001|2026-03-01}, symbolic relations →⇐, minimal prose.
4. **Contradictions are knowledge** — Disagreements between claims are surfaced, not resolved. Resolution is context-dependent. (This is *epistemic* pluralism, not a metaphysical claim: the format records that observers hold conflicting beliefs, not that a contradiction is true of the world — paraconsistent at the level of the pack while leaving the world bivalent.)
5. **Append-only** — Claims are superseded (⌘), not edited. The past is preserved.
6. **Freeform with conventions** — Predicates, entity types, and sections are freeform. Vocabulary overlays are optional.
7. **Claims file is self-contained** — An LLM reading only claims.md can reason about the domain. Evidence and entities are supporting.
8. **Git-native** — Files, directories, diffs, PRs. No special infrastructure required.
9. **Any LLM, any system** — Portable by default. No platform lock-in. From a 7B local model to a frontier model.
10. **Progressive loading** — Hub always loaded, detail on demand, evidence on citation. Never dump everything into context.
11. **Navigable** — Every pack knows what's adjacent. Root index (KNOWLEDGE.yaml) for discovery, hub sub_packs for structure, inline ⇆ for depth.
12. **Constitutional core** — The claim syntax (- [ID] headline {metadata} prose), confidence range (0-1), evidence references (E + id), relation symbols (→⇐~⇐→), and Rosetta Header (<!-- KP:N) are frozen across ALL spec versions. Everything else may evolve.
13. **Don't over-atomize** — Claims need enough context to prevent semantically valid but context-wrong extractions. Not every domain benefits from decomposition. A claim should be the smallest unit that is independently falsifiable — no smaller (Hu et al., Decomposition Dilemmas, NAACL 2025).
14. **Hallucination tolerance, not elimination** — Hallucination is mathematically inevitable even for optimal estimators (Liu et al. 2025). Knowledge Packs make hallucination detectable and traceable through confidence, evidence, and contradictions. They do not make it impossible.
15. **Constrain before voting** — Logit-level constrained decoding (1x cost) is the primary stability intervention. Multi-run voting (3-5x cost) is a fallback. The deterministic contract pipeline: constrain → canonicalize → validate → escalate.

16. **Three surfaces, one truth** — Claims.md is the reasoning surface (AI-optimized). Display views are the visual surface (human-reading-optimized). Voice views are the auditory surface (human-listening-optimized). All represent the same knowledge. Views are derived from claims, never authoritative. If they disagree, claims win.
17. **Renderer, not compositor** — When the AI presents knowledge to humans, it should render pre-built views, not compose content on the fly. This is faster, more consistent, and separates the reasoning task (understanding) from the presentation task (showing).
18. **Composition over duplication** — Meeting and composite packs reference standing packs rather than duplicating their knowledge. A meeting pack is a lens — an agenda-shaped view over existing packs with meeting-specific additions. Knowledge lives once, in its canonical pack. Every piece of knowledge has exactly one canonical home — the standing pack where it is authored and maintained. Other packs may reference via `see_also` links or render content via composition. A pitch pack or meeting pack that independently re-defines a concept owned by a standing pack is a duplication violation. Test: if the standing pack's claim changes, do all other appearances automatically reflect it? If not, the knowledge is duplicated.
19. **Archive, never delete** — Packs are archived, not deleted. Archived packs remain searchable, loadable, and in git history. Archival is the default lifecycle outcome for ephemeral packs; deletion requires explicit, deliberate action.
20. **One view, one question** — Each display view answers one question a human would ask. The overview is a router (what, why, where-to-go), not a summary of everything. Split on meaning, never on arbitrary line counts. A view exceeding 80 lines almost always needs splitting; a view under 10 lines almost always needs merging. (Shneiderman: overview first, then details on demand.)
21. **Reconcile before archive** — Ephemeral packs can contain knowledge that does not exist elsewhere. Before archiving, review which findings need standing access and which accounts remain in the retained event record. Preserve attribution, scope and references to the original material. Flag unresolved placement decisions; archival must not silently discard knowledge. See [LIFECYCLE.md §4](#).
22. **Knowledge is the source (pack-as-master)** — Knowledge packs are the canonical source of truth for their epistemic contents. Agent instruction files, operations (planning, tasks), and presentations (slides, documents) reference knowledge packs but are not knowledge themselves. Databases, caches, and search indices that store or project pack data are derived representations — operational infrastructure that serves packs, not the other way around. When a derived representation disagrees with the pack, the pack is correct. Any index can be dropped and rebuilt from packs. Packs can be stored in any medium (files, databases, caches) without changing their authority. Only knowledge is a knowledge pack. (See [STORAGE.md](#).)
23. **Locale first, then surface** — Multilingual views are organized by locale, then by surface type: `views/uk/voice/briefing.md`, not `views/voice/uk/briefing.md`. Locale is the higher-order grouping.
24. **Keep context that changes meaning** — Claims state what is believed and why; views present that knowledge with the qualifications needed to understand it. Names, roles, dates, methods and decision context belong with a claim when they affect its attribution, scope, interpretation or reasons for change. Fuller source narratives belong in `evidence.md`. Remove incidental process narration only when the assertion's meaning remains intact. Neither a person's request nor a tool's involvement alone establishes a conclusion, but an attributed request, decision or method can itself be knowledge worth retaining.
25. **Short-term memory, long-term memory** — Standing packs organize lasting knowledge by subject. Ephemeral packs bring material together for a particular event or task. These are organizational analogies, not a model's internal memory. Consolidation makes useful findings available to later work while preserving their source context and references to the retained event record. Event-specific roles, experiences and decisions can remain valuable after the event; archival placement does not make them non-knowledge. (See [LIFECYCLE.md §1.1](#).)

Independence. The ownership and capture-resistance commitments these principles imply — zero-dependency readability, freedom to fork, no required registry, optional-not-mandatory

cryptography — are stated as a companion: see [INDEPENDENCE.md](#). Their throughline is *subtraction, not addition*: the format stays ownable by requiring less, because every dependency removed is a chokepoint that cannot be captured.

2. Style Systems

A style system is an external, versioned specification that tells a renderer how to visually present view content. **Style systems are NOT part of the Knowledge Pack** — they are referenced by packs and defined externally. This section records the rationale for keeping them external and a recommended shape for producers building their own.

Why External

- **Separation of concerns** — Knowledge is stable; presentation evolves independently
- **Reusability** — One style system serves many packs across a product
- **Voyager Principle preserved** — Packs work without a style system (markdown is self-rendering)
- **Brand consistency** — All packs in a product share visual identity

PACK.yaml Reference

```
style: acme-professional      # Style system identifier
```

A null or omitted `style` field means the views render as plain markdown — which is always valid.

Style System Contents

A style system defines:

Component	Purpose	Example
Semantic markers	What > [!metric] etc. mean visually	Green badge with large number
Table formatting	Column alignment, header styles	Zebra-striped, right-aligned numbers
Typography	Heading hierarchy, emphasis	H1 for title, H2 for sections, bold for key terms
Color semantics	What colors mean in context	Green = confirmed, amber = tentative, red = disputed
Layout patterns	Structural arrangements	Card grid, sidebar + main, timeline
Confidence rendering	How to display confidence levels	Color-coded bars, Sherman Kent labels

The Rendering Stack

```
claims.md → views/*.md → style system → rendered output
(knowledge)  (content)   (presentation) (platform-specific)
```

Each layer can exist independently:

- Claims without views = AI can reason, can't display pre-built content
- Views without style = renders as clean markdown (always works)
- Views with style = renders with visual richness and consistency

Style System Format (Recommended)

Style systems are defined as YAML files:

```
# STYLE.yaml
name: acme-professional
version: 2026.03.21
description: "Professional style system – data-rich, confidence-aware, clean typography"

markers:
  metric:
    render: "highlighted-badge"
    description: "Key metric – prominent display with large value"
  highlight:
    render: "callout-primary"
```

```
description: "Important insight – visually distinct callout"
risk:
  render: "callout-warning"
  description: "Risk indicator – amber/red callout"
status:
  render: "status-pill"
  description: "Current state – compact status indicator"
source:
  render: "attribution-subtle"
  description: "Source attribution – small, muted text"

confidence:
  display: "bar"                                # bar | label | badge | none
  colors:
    high: "#22c55e"                            # 0.85+
    medium: "#f59e0b"                          # 0.50-0.84
    low: "#ef4444"                             # <0.50

tables:
  header: "bold"
  alignment: "auto"                            # auto-detect from content
  numbers: "right-aligned"

typography:
  h1: "Pack title – large, bold"
  h2: "Section – medium, with subtle divider"
  h3: "Subsection – small caps or bold"
```

Platform-Specific Renderers

The style system is platform-neutral. Renderers interpret it for specific targets:

Renderer	Output
Canvas (OpenAI)	Rendered markdown with formatting hints
HTML	CSS-styled web page
PDF	Formatted document via reportlab/Typst
Terminal	ANSI-colored markdown
Slack	Slack-formatted blocks

The renderer is NOT part of the Knowledge Pack spec — it's part of the consuming platform. The style system bridges the gap between platform-neutral views and platform-specific rendering.

3. Cognitive Perception Layer

The `display` block in `PACK.yaml` and the per-view `hint` field help people recognize a pack, understand its purpose and find relevant material. The normative spec for these fields lives in [SPEC.md §18](#) and [SPEC.md §3](#) (manifest); this section explains how to author them. Its time budgets are design targets, not measured performance or extra conformance requirements.

The Primary Constraint: Cognitive Load

Make introductory display fields easy to understand without changing their meaning. A person encountering the pack may:

- Have never seen it before
- Be deciding whether to open it
- Be unfamiliar with the domain
- Need a clear route to the detailed record

Use plain language and explain unfamiliar terms in short introductions. Keep attribution, uncertainty and scope when they affect meaning. Do not remove a necessary technical detail merely to make a sentence shorter.

Full display and voice views serve their declared audience. A specialist view may need terminology and detail that a brief introduction does not. Both should remain faithful to the record and keep a route to supporting claims and sources.

An accessible view helps its audience understand rich knowledge. It is not a replacement for the full record. Selecting relevant material should preserve the qualifications needed for the task.

Perception Stages

A useful introductory view supports four common steps. The timings below are authoring targets for a brief introduction, not universal measurements of how people read.

1. **Recognize** (< 1 second) — What is this? Can I identify it in a list, on a tab, in a breadcrumb?
2. **Comprehend** (< 3 seconds) — What is it about? What does it address?
3. **Engage** (< 5 seconds) — Why should I care? What makes me want to read further?
4. **Navigate** (< 10 seconds) — Which section should I open? What will I find there?

Renderers can use the SPEC's display metadata to support these steps. The appropriate detail and time depend on the audience and task.

Why each display field exists

short_title (Recognition)

A human scanning a list of 20 packs, or glancing at a tab bar, or hearing a voice assistant name a pack — they need to recognize THIS pack in under one second. The full title ("Utility-Scale Solar Energy Market Analysis and Cost Trajectories") takes too long. The abbreviation ("SEM") requires insider knowledge. The short title occupies the sweet spot: long enough to be unambiguous, short enough to be instant.

The name a human would use to refer to this pack in conversation. 2-4 words. Must be unambiguous without the full title.

Good	Bad	Why
"Solar Market"	"Utility-Scale Solar Energy Market Analysis and Cost Trajectories"	Too long — the full title belongs in the H1, not here
"Wind Outlook"	"Wind Energy Assessment"	"Assessment" is vague; "Outlook" implies forward-looking analysis
"Battery Storage"	"battery-storage-economics-2026-03"	File names are not titles

abbreviation (Tight Contexts)

Some display contexts have almost no space — a mobile breadcrumb, a badge next to a notification, a voice assistant referencing a pack mid-sentence. The abbreviation is the absolute minimum identifier. It sacrifices clarity for brevity, which is why it exists alongside `short_title`, not instead of it. 2-5 characters. Must be pronounceable or a recognized acronym. Used in badges, breadcrumbs, tabs.

Good	Bad	Why
"SEM"	"USEMPV"	Not pronounceable
"Wind"	"WE"	Ambiguous without context

tagline (Comprehension)

After recognizing a pack by title, the next question is "what is this about?" The tagline answers that in a single breath. It's the bridge between the title (which names the thing) and the content (which explains it in full). Without a tagline, the human must open the pack and read before they can even assess relevance. The tagline prevents that wasted effort — it's a promise of what's inside, delivered before the reader commits.

One sentence that tells a stranger what the subject IS. Not what the pack contains — what the subject is about. Should be readable aloud in under 5 seconds. Natural language, not a label.

Good	Bad	Why
"Utility-scale solar market trends and cost analysis"	"Executive summary with key metrics and status"	That describes the pack format, not the subject
"Supply chain risks for global solar deployment"	"Solar risk overview pack"	Reads like a filename, not a sentence
"Grid storage economics for renewable integration"	"Battery analysis pack"	The good version answers WHAT and WHY

hook (Engagement)

Comprehension without engagement is a dead end. A person can understand what a pack is about and still not care. The hook exists because human attention is not given — it's earned. A tagline tells you what the plate is; the hook is the aroma that makes you want to eat. It's the most emotionally or intellectually compelling sentence in the entire pack, surfaced to the cover because burying it on page 3 means nobody gets there.

The single most compelling sentence in the entire pack. A provocation, a surprising insight, or a promise that creates the desire to read further. If you had to convince someone to open this pack with one sentence, this is it.

Good	Bad	Why
"The real competitor is the status quo of neglect, not other platforms."	"This pack covers risks, timeline, and team."	That's a table of contents, not a hook
"We're raising 1.5M euro to capture the 18-month window before incumbents wake up."	"Fundraising information for Q2 2026."	No urgency, no insight, no reason to care

If you can't find a hook, the pack may not have a clear thesis yet. That's a content problem, not a display problem.

hint per view (Navigation)

The cover shows sections, but a title alone ("Risks", "Timeline") is a label — it tells you the category, not the value. A human deciding where to click needs to know what they'll GET from opening that section, not what category it belongs to. The hint transforms a menu of labels into a menu of promises. Without it, navigation is a guessing game — the reader has to open sections speculatively. With it, they open with intent.

One sentence per section that helps a human decide: should I open this? Describes the VALUE of the section, not its structure. Max ~15 words.

Good	Bad	Why
"Three risks that could block the Sep 2026 pilot"	"Risk register with 3 rows"	Structure is not value
"6 months built, 5 months to launch"	"Timeline table"	The good version tells you the insight
"What to say when Scott asks about AI safety"	"Technical Q&A section"	The good version tells you what you'll GET

The Stranger Test

After authoring all display fields, read them in sequence:

[short_title] — [tagline]. [hook]. Sections: [hint 1], [hint 2], [hint 3].

If a stranger reading this sequence can answer "what is this, why should I care, and where should I start?" — the fields pass. If not, rewrite.

What these fields are NOT

- They are not marketing copy (that belongs in a website, not a spec).
- They are not aesthetic choices (fonts, colors, layout are renderer decisions).
- They are not summaries of the content (that's what views are for).

They are **cognitive handles** — the minimum information a human brain needs to recognize, locate, engage with, and navigate this pack across any display context.

4. Relationship to Existing Standards

KP:1 sits in a small ecosystem of AI-facing content formats. The table below records how KP:1 relates to each, where the boundary is, and what each format is for.

Standard	Relationship
Agent instruction files (e.g., AGENTS.md)	Orthogonal — agent instruction files configure behavior, Knowledge Packs provide facts
Agent Skills (SKILL.md)	Complementary — Skills declare <code>requires_knowledge</code> for pack dependencies
MCP Resources	Knowledge Packs can be served as MCP resources at runtime
llms.txt	Discovery — llms.txt could point to available Knowledge Packs
Entity-Claim-Evidence (ECS) systems	Implementation pattern — ECS is an operational index; Knowledge Packs are the canonical source. See STORAGE.md
schema.org / JSON-LD knowledge graphs	Adjacent — schema.org/JSON-LD is the dominant production approach for structured knowledge on the web. KP:1 differs in three ways: (1) plain-text density (<code>{0.95 i E001 2026-03-01}</code> is materially more token-efficient than the equivalent JSON-LD object); (2) first-class uncertainty and contradiction primitives that JSON-LD vocabularies do not standardize; (3) the three-surface architecture (claims / display / voice) is intrinsic, not a rendering layer over a single semantic graph. Both approaches can compose: a KP:1 pack can map to JSON-LD via MAPPING.md .
JSON-LD / RDF	Inspiration — semantic structure, but optimized for neural nets, not symbol processors
Nanopublications	Spiritual ancestor — assertion + provenance, but in markdown, not RDF

For the field-by-field translation analysis between KP:1 and RDF / JSON-LD / PROV-O / Nanopublications, see [MAPPING.md](#).

This file is companion to [SPEC.md](#) and [CORE.md](#). Where [SPEC.md](#) or [CORE.md](#) specify form, [RATIONALE.md](#) explains why. Where the two appear to conflict, [SPEC.md](#) and [CORE.md](#) are authoritative.

KP:1 Mapping to RDF/JSON-LD/PROV

Status: Draft (informative — not normative for KP:1 conformance) **Date:** 2026-04-04 **Audience:** Semantic Web practitioners evaluating KP:1 interoperability

This is not advocacy. It is an honest field-by-field translation analysis. For the *why KP:1 exists* argument, see [positioning/kp-why-not-existing-standards.md](#).

1. Purpose

KP:1 has its own claim syntax, evidence model, and relation types. This document shows what happens when you try to translate between KP:1 and three standards: RDF/JSON-LD, PROV-O, and Nanopublications. For each KP:1 concept, it identifies whether the translation is clean (round-trips without loss), lossy (information degrades), or impossible (no target equivalent).

The goal is to let someone who knows RDF answer: *Can I use my existing toolchain? What do I lose? What do I gain?*

2. Methodology

Every KP:1 concept is mapped to its closest target representation. Each mapping is graded:

- **Clean** — round-trips without loss. The target representation preserves all semantics and can reconstruct the KP:1 original.
- **Lossy** — a reasonable representation exists, but information degrades. The target captures the broad intent but loses precision, nuance, or structure.
- **None** — no standard equivalent. Representing this requires inventing custom vocabulary.

"Custom vocabulary" is not a failure — it's how RDF works. But it means your existing SPARQL queries, reasoners, and validators won't understand the concept without extension.

3. RDF/JSON-LD Mapping

3.1 Pack → Named Graph

A Knowledge Pack maps naturally to an **RDF named graph** (or a **TriG dataset**). The pack URI serves as the graph name; pack metadata becomes triples in a metadata graph.

KP:1 field	RDF representation	Grade
name	Graph URI (e.g., <kp:solar-energy-market>)	Clean
version (CalVer)	dct:issued with xsd:date	Lossy — CalVer semantics (snapshot date, not SemVer) are lost
domain	dct:subject with a SKOS concept or custom hierarchy	Lossy — KP's hierarchical path (energy/market-analysis) has no standard encoding
author	dct:creator	Clean
kind	Custom predicate (kp:kind)	None — no RDF equivalent for claim/definition/policy/mixed
confidence.scale	Custom predicate (kp:confidenceScale)	None
freshness	dct:modified	Clean
license	dct:license	Clean
sensitivity	Custom predicate (kp:sensitivity)	None
visibility	Custom predicate (kp:visibility)	None
channels	Custom predicate (kp:channel)	None
provenance.role	Custom predicate (kp:authorRole)	None — PROV-O has prov:hadRole but no vocabulary for manufacturer/independent/etc.
tier	Custom predicate (kp:tier)	None
display	Custom predicates	None — cognitive perception layer has no RDF analogue

Verdict: Pack-level metadata is roughly 40% clean (standard Dublin Core fields), 60% custom vocabulary. The pack *structure* (directory of files) has no RDF equivalent — RDF models graphs, not file systems.

3.2 Claim → RDF Statement

A KP:1 claim maps to an **RDF-star reified triple** (RDF 1.2) or a **named graph per claim** (RDF 1.1). The headline becomes the object of an assertion triple; metadata attaches to the reifier.

Using RDF 1.2 reifiers:

```
PREFIX kp:    <https://kp.dev/ontology/>
PREFIX prov:  <http://www.w3.org/ns/prov#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>

:C001 rdf:reifies << :solar-energy-market kp:claims "Cost decline is structural, not cyclical -
driven by manufacturing scale" >> .
:C001 kp:confidence      "0.95"^^xsd:float .
:C001 kp:claimType       kp:inferred .
:C001 kp:evidence         :E001, :E002 .
:C001 dct:date            "2026-03-01"^^xsd:date .
:C001 kp:depth            kp:exhaustive .
:C001 kp:nature           kp:judgment .
```

This captures the structured metadata but loses:

- **The headline as an independent assertion.** In KP:1, the headline is a self-contained proposition ("Cost decline is structural"). In RDF, it becomes a string literal — the triple is *about* the pack having that claim, not an assertion of the claim's content as a first-class statement.
- **Claim ID semantics.** KP:1 IDs (C001) are pack-local, sequential, versionable (C001-v2). RDF URIs are global. The local-ID convention is lost.
- **Context prose.** The free-text context ("10/10 analyses converged. Learning curve held for 40 years.") has no structured home. It becomes `rdfs:comment` or a custom annotation — searchable but not semantic.

3.3 Confidence

KP:1	RDF	Grade
Confidence value (0.0–1.0 float)	Custom <code>kp:confidence</code> with <code>xsd:float</code>	Lossy — no standard confidence predicate exists
Confidence scale (e.g., <code>sherman_kent</code>)	Custom <code>kp:confidenceScale</code> on pack graph	None — RDF has no confidence scale concept
<code>normalize: true</code> declaration	Implicit (values are floats)	Lossy — the explicit "these are normalized" signal is lost

RDF has no standard vocabulary for epistemic confidence. The W3C Uncertainty Ontology (URW3-XG) defined `urw3:weight` but never advanced beyond incubator status. In practice, everyone uses custom predicates. KP:1's confidence model is *richer* than anything standard in RDF — the combination of numeric value + named scale + normalization flag + investigation depth has no off-the-shelf target.

3.4 Claim Metadata (Type, Depth, Nature)

KP:1 concept	Closest RDF	Grade
Claim type (o/i/c/i)	PROV-O derivation types (partial)	Lossy — see §4
Investigation depth (assumed/investigated/exhaustive)	No standard equivalent	None
Claim nature (judgment/prediction/meta)	No standard equivalent	None

Claim type partially maps to provenance concepts — observed $\approx$ primary source, reported $\approx$ secondary source, computed $\approx$ algorithmic derivation, inferred $\approx$ reasoning. But PROV-O models *how entities were derived*, not *how claims were acquired*. The epistemological distinction (what kind of knowledge is this?) has no standard RDF encoding.

Investigation depth is KP:1's novel two-axis model: confidence × depth. A claim at 0.50/exhaustive means "we looked hard and it's genuinely uncertain." A claim at 0.90/assumed means "we haven't checked but it feels right." No RDF vocabulary captures this distinction.

Claim nature distinguishes factual assertions from judgments, predictions, and meta-claims. RDF has no equivalent — all triples are equally assertional.

3.5 Relations

KP:1 relation	Closest RDF/ontology	Grade
→ supports	CiTO cito:supports	Lossy — CiTO captures citation-level support but not KP:1's epistemic "this belief strengthens that belief"
⊗ contradicts	CiTO cito:disagreesWith	Lossy — CiTO has disagrees/disputes/refutes intensity levels but no qualifier for <i>why</i>
⊗! contradicts:error	No standard equivalent	None — CiTO's cito:refutes is closest but doesn't encode "one is wrong, prioritize resolution"
⊗~ contradicts:tension	No standard equivalent	None — "both informative, preserve as productive disagreement" has no standard encoding
← requires	Custom or dct:requires	Lossy — Dublin Core requires is for resource dependencies
~ refines	skos:narrower (partial)	Lossy — SKOS narrower is taxonomic, not epistemic refinement
⊘ supersedes	prov:wasRevisionOf	Clean — good semantic match
↔ see_also	rdfs:seeAlso	Clean

The best standard vocabulary for KP:1 relations is **CiTO** (Citation Typing Ontology, <http://purl.org/spar/cito/>), not AIF. CiTO provides cito:supports, cito:agreesWith, cito:disagreesWith, cito:disputes, cito:critiques, and cito:refutes — a better fit than AIF's abstract argument nodes.

But contradiction qualifiers remain the hardest mapping. CiTO distinguishes intensity (disagrees < disputes < refutes) but not the *nature* of the conflict. KP:1's three-way distinction — unqualified (⊗), error (⊗!: one is wrong), tension (⊗~: both informative) — encodes why the contradiction matters, not how strongly someone disagrees. No standard vocabulary captures this.

An export would use CiTO for the base relations and custom kp: predicates for the qualifiers. Consumers with CiTO would understand the support/attack structure; the error/tension distinction would degrade to opaque custom properties.

3.6 Evidence

KP:1 field	RDF	Grade
Evidence ID (E001)	URI for a prov:Entity	Clean
Evidence type (open source vocabulary, e.g. document, interview)	Profile-specific source classification	Lossy
Source (free text)	dct:source or prov:wasAttributedTo	Clean
Captured date	prov:generatedAtTime	Clean
Evidence prose	rdfs:comment	Lossy — loses structured role
Claim → Evidence link	prov:wasDerivedFrom or prov:hadPrimarySource	Clean

Evidence entries map well to PROV-O entities. The main loss is evidence *type* — the open evidence-source vocabulary needs a profile-specific mapping. The four-way observed/reported/computed/inferred classification belongs to claims, not evidence records.

3.7 Example: Solar C001 as JSON-LD

```
{
  "@context": {
    "kp": "https://kp.dev/ontology/",
    "kpd": "https://kp.dev/data/",
    "prov": "http://www.w3.org/ns/prov#",
    "cito": "http://purl.org/spar/cito/",
    "dct": "http://purl.org/dc/terms/",
    "xsd": "http://www.w3.org/2001/XMLSchema#"
  }
}
```

```
{
  "@id": "kpd:solar-energy-market/C001",
  "@type": "kp:Claim",
  "kp:headline": "Cost decline is structural, not cyclical – driven by manufacturing scale",
  "kp:confidence": { "@value": "0.95", "@type": "xsd:float" },
  "kp:claimType": { "@id": "kp:inferred" },
  "kp:depth": { "@id": "kp:exhaustive" },
  "kp:nature": { "@id": "kp:judgment" },
  "dct:date": { "@value": "2026-03-01", "@type": "xsd:date" },
  "prov:wasDerivedFrom": [
    { "@id": "kpd:solar-energy-market/E001" },
    { "@id": "kpd:solar-energy-market/E002" }
  ],
  "cito:supports": { "@id": "kpd:solar-energy-market/C002" },
  "kp:contradictsTension": { "@id": "kpd:solar-energy-market/C003" },
  "kp:context": "10/10 analyses converged. Learning curve (22% cost reduction per doubling) has held for 40 years."
}
```

Note on JSON-LD version: This example uses JSON-LD 1.1, which cannot represent RDF 1.2 triple terms or reifiers natively. The Turtle examples in §3.2 use RDF 1.2 reifier syntax (`rdf:reifies`); a full JSON-LD representation of that pattern would require JSON-LD 1.2 (in development). The example above models the claim as a typed node with custom predicates, which is valid JSON-LD 1.1 but does not capture the reification semantics.

What's preserved: claim identity, headline, confidence value, evidence links, date, support relation (via CiTO), context prose.

What's lost: confidence scale semantics (Sherman Kent), the Rosetta header (self-describing format), frontmatter (pack identity in claims.md), the dense/verbose encoding distinction, positional field semantics, claim ID locality (C001 is now a global URI), contradiction qualifier semantics (tension degrades to a custom predicate), and the fact that this is one claim in a file designed for sequential AI consumption.

What's gained: global identity (URI), machine-queryable via SPARQL, composable with any RDF graph, processable by standard RDF tooling, standard citation semantics via CiTO.

4. PROV-O Mapping

4.1 Evidence Provenance Chain

KP:1 evidence entries map to PROV-O entities with generation activities:

```
:E001 a prov:Entity ;
  prov:wasGeneratedBy :analysis_activity ;
  prov:generatedAtTime "2026-03-01"^^xsd:date ;
  prov:wasAttributedTo :jane_chen ;
  dct:source "Internal analysis – 10 independent assessments across multiple AI models" .

:analysis_activity a prov:Activity ;
  prov:used :irena_report, :bnef_data .
```

This is a good fit. PROV-O was designed for exactly this — tracking where data came from.

4.2 Claim Derivation

KP:1 claims that reference evidence map to PROV-O derivation:

```
:C001 a prov:Entity ;
  prov:wasDerivedFrom :E001, :E002 ;
  prov:wasAttributedTo :jane_chen ;
  prov:generatedAtTime "2026-03-01"^^xsd:date .
```

The derivation is clean. But PROV-O's derivation subtypes don't align with KP:1's claim types:

KP:1 claim type	PROV-O closest	Fit
observed (primary source)	prov:hadPrimarySource	Good
reported (secondary source)	prov:wasQuotedFrom	Partial — quoting is narrower than reporting
computed (algorithmic)	prov:qualifiedDerivation with prov:hadActivity	Lossy — requires the qualified derivation pattern to link the computation Activity
inferred (reasoned)	prov:wasDerivedFrom (generic)	Lossy — no standard "inferred via reasoning" subtype

4.3 Supersession → prov:wasRevisionOf

KP:1's `o` (supersedes) maps cleanly:

```
:C001-v2 prov:wasRevisionOf :C001 .
:C001 prov:wasInvalidatedBy :supersession_activity .
```

This is one of the best mappings in this document. PROV-O's revision and invalidation model closely matches KP:1's supersession semantics, including the audit trail (history.md ≈ invalidated entities).

4.4 What PROV-O Adds That KP:1 Lacks

- **Activity modeling.** PROV-O can represent the *process* that produced a claim (who ran what analysis, when, using what tools). KP:1 evidence records the *result* but not the process chain.
- **Qualified patterns.** PROV-O's qualified counterparts (prov:qualifiedDerivation, prov:qualifiedAssociation, prov:qualifiedAttribution, etc.) allow attaching arbitrary metadata to any provenance relationship — timestamps, roles, confidence, methods. This is where most of PROV-O's real expressive power lives, and it's more flexible than KP:1's fixed-field model. A prov:Derivation node can carry prov:hadActivity, prov:hadUsage, and custom annotations that KP:1 evidence entries cannot express.
- **Agent roles.** prov:hadRole on qualified associations can model reviewer, editor, approver roles per claim. KP:1 has provenance.role at the pack level only. The PAV ontology (pav:authoredBy, pav:curatedBy, pav:createdBy) provides finer author-role distinctions that plug into PROV-O.
- **Delegation chains.** prov:actedOnBehalfOf models organizational authority. KP:1 has no delegation concept.
- **Plans.** prov:hadPlan on qualified associations can link activities to their governing methodology. KP:1's ontology field is a freeform string, not a structured plan reference.

4.5 What KP:1 Has That PROV-O Cannot Represent

- **Confidence values.** PROV-O has no uncertainty model.
- **Investigation depth.** No PROV-O equivalent.
- **Claim nature.** No distinction between factual/judgment/prediction/meta.
- **Typed contradiction.** PROV-O models derivation, not disagreement. Two entities can both exist in a PROV graph, but there's no way to say "these two conflict, and the conflict is informative."
- **Views.** PROV-O models provenance, not presentation.

5. Nanopublication Mapping

5.1 Structural Alignment

Nanopublications have three named graphs: assertion, provenance, publication info. KP:1 claims have a parallel structure:

Nanopub graph	KP:1 equivalent	Alignment
Assertion (the claim itself)	Claim headline + context prose	Good — both are atomic assertions
Provenance (how the assertion was derived)	Evidence entries + claim type + depth	Partial — KP:1 evidence is richer but less formally structured
Publication Info (who published, when, signature)	PACK.yaml provenance + version + freshness	Good — both track authorship and timing

A single KP:1 claim maps to one nanopublication. A KP:1 pack maps to a set of nanopublications sharing publication info.

5.2 Where the Analogy Holds

- **Atomic assertions.** Both formats center on individual, identifiable claims rather than documents. A nanopub assertion graph $\approx$ a KP:1 claim headline.
- **Provenance per claim.** Both attach provenance to individual assertions, not to documents. KP:1's per-claim evidence references mirror nanopub's per-assertion provenance graph.
- **Immutability + versioning.** Nanopubs use trusty URIs (content-addressed, immutable). KP:1 uses CalVer + supersession. Different mechanisms, same intent: you don't edit published claims, you supersede them.
- **Author attribution.** Both record who made the assertion and when.

5.3 Example: Solar C001 as Nanopublication

```
@prefix np:    <http://www.nanopub.org/nschema#> .
@prefix kp:    <https://kp.dev/ontology/> .
@prefix kpd:   <https://kp.dev/data/> .
@prefix prov:  <http://www.w3.org/ns/prov#> .
@prefix cito:  <http://purl.org/spar/cito/> .
@prefix dct:   <http://purl.org/dc/terms/> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .

:C001-nanopub {
  :C001-nanopub a np:Nanopublication ;
  np:hasAssertion      :C001-assertion ;
  np:hasProvenance     :C001-provenance ;
  np:hasPublicationInfo :C001-pubinfo .
}

:C001-assertion {
  kpd:solar-energy-market kp:claims
  "Cost decline is structural, not cyclical – driven by manufacturing scale" .
}

:C001-provenance {
  :C001-assertion prov:wasDerivedFrom kpd:solar-energy-market/E001 ,
                                     kpd:solar-energy-market/E002 .
  :C001-assertion prov:generatedAtTime "2026-03-01"^^xsd:date .
}

:C001-pubinfo {
  :C001-nanopub prov:wasAttributedTo <https://orcid.org/example/jane-chen> .
  :C001-nanopub dct:created "2026-03-18"^^xsd:date .
  :C001-nanopub dct:license <https://creativecommons.org/licenses/by/4.0/> .
}
```

What's preserved: assertion content, evidence links, timestamp, authorship, license.

What's lost: confidence (0.95), claim type (inferred), investigation depth (exhaustive), claim nature (judgment), context prose, relations ($\rightarrow$ C002, $\otimes$ ~C003), section membership, the Rosetta header, and the fact that this is one claim in a curated pack of 8. These would require custom annotations on the assertion or provenance graphs with no standard vocabulary.

5.4 Where It Breaks

Confidence model. Nanopubs have no standard confidence annotation. Some life-science nanopubs use custom predicates for evidence strength, but there's no shared vocabulary. KP:1's confidence + depth + nature model has no nanopub equivalent.

Relations between claims. Nanopubs are self-contained units. Relations between nanopubs (A supports B, A contradicts C) require external association graphs or conventions like `np: introduces` and custom association predicates. KP:1's inline relation syntax ($\rightarrow$ C002, $\otimes$ ~C003) is first-class, not an afterthought.

Contradiction preservation. Nanopubs in the same network can make conflicting assertions, but the conflict is implicit — there's no typed contradiction relation. KP:1's $\otimes/\otimes!/ \otimes\sim$ distinction has no nanopub analogue.

Views. Nanopubs have no presentation layer. KP:1's views (display, voice) are a separate concern with no nanopub equivalent.

Pack as container. A nanopub is a standalone unit. KP:1 packs are *curated collections* with shared metadata, confidence scales, and cross-claim relations. The container semantics (sections, frontmatter, Rosetta header) have no nanopub equivalent — the closest analogue would be a nanopub index or collection, which is not yet standardized.

Authoring format. Nanopubs are RDF (serialized as TriG). KP:1 is Markdown. This is not a mapping issue but a practical one: KP:1 is designed for human authoring and LLM consumption; nanopubs are designed for machine interchange.

6. Translation Loss Summary

KP:1 Concept	RDF/JSON-LD	PROV-O	Nanopub
Pack (PACK.yaml)	Lossy (named graph + Dublin Core, ~40% custom)	N/A	Lossy (publication info graph, no container)
Claim (ID + headline + metadata)	Lossy (reified triple, headline becomes literal)	Lossy (entity, no claim semantics)	Lossy (assertion graph, no confidence)
Confidence (0–1 float)	None (custom predicate)	None	None
Confidence scale (sherman_kent, etc.)	None (custom predicate)	None	None
Claim type (o/r/c/i)	Lossy (PROV derivation subtypes, partial)	Lossy (partial via derivation)	None
Investigation depth	None	None	None
Claim nature (judgment/prediction/meta)	None	None	None
Relation: supports ($\rightarrow$)	Lossy (CiTO cito:supports)	None	None (external association)
Relation: contradicts ($\otimes$)	Lossy (CiTO cito:disagreesWith)	None	None
Contradiction qualifiers ($\otimes!$ / $\otimes\sim$)	None	None	None
Relation: requires ($\leftarrow$)	Lossy (dct:requires)	None	None
Relation: refines ($\sim$)	Lossy (skos:narrower)	None	None
Relation: supersedes ($\supset$)	Clean (prov:wasRevisionOf)	Clean	Lossy (trusty URI versioning)
Relation: see_also ($\leftrightarrow$)	Clean (rdfs:seeAlso)	None	None
Evidence entry	Clean (prov:Entity)	Clean	Clean (provenance graph)
Evidence type	Lossy (PROV derivation subtypes)	Lossy	None
Evidence source/captured	Clean (dct:source + prov:generatedAtTime)	Clean	Clean
Rosetta header	None	None	None
Frontmatter (pipe-delimited)	None (encoding-level, no semantic target)	None	None
Views (display/voice)	None	None	None
Sections (## headings)	None (organizational, no RDF equivalent)	None	None

Score: 4 clean, 8 lossy, 9 none out of 21 concepts mapped across three standards (counting the best grade achieved across any target).

Restated: ~19% of KP:1 concepts translate cleanly to existing standards. ~38% translate with degradation. ~43% require custom vocabulary.

7. Implications

Export: KP → RDF

What you'd build: A serializer that emits one typed `kp:Claim` node per claim, with CiTO predicates for support/contradiction relations, custom `kp:` namespace predicates for confidence, depth, nature, claim type, and contradiction qualifiers. Evidence maps to PROV-O entities. Pack metadata maps to Dublin Core + custom predicates in a metadata graph. Relations use a mix of standard (`prov:wasRevisionOf`, `rdfs:seeAlso`, `cito:supports`, `cito:disagreesWith`) and custom (`kp:contradictsTension`, `kp:contradictsError`) predicates.

What you'd lose: The self-describing Rosetta header. The compact encoding (dense/verbose distinction). The file-system structure (pack as directory). The three-surface rendering model (views). The confidence scale semantics. Claim ID locality. The authoring-optimized format.

What you'd gain: SPARQL queryability. Composability with any RDF graph. Standard tooling (validators, visualizers, reasoners — though reasoners won't understand the custom predicates). Global identity via URIs. The ability to merge claims from different packs into a single graph.

Import: RDF → KP

What you'd gain: Confidence, investigation depth, and claim nature as first-class fields (not annotations). Typed contradiction preservation. A self-describing, LLM-optimized format. A curated container (pack) with lifecycle, versioning, and views. Zero-dependency consumption.

What you'd invent: Confidence values (RDF triples don't carry uncertainty). Investigation depth (no source signal). Claim nature classification. Contradiction qualifiers. Section organization. Views. Essentially, the import would require a human or AI to perform epistemic annotation — deciding how confident each triple is, how thoroughly it's been investigated, and whether conflicting triples represent errors or productive tensions.

Interoperability Strategy: Bridge vs Native

A **bridge** approach — export KP:1 to RDF for query and interchange, with documented loss — is practical today. The KP ontology (`kp:`) would be small (~20 classes/properties) and could be published as OWL. Consumers without the ontology would still see standard PROV-O provenance and Dublin Core metadata; the KP-specific semantics (confidence, depth, contradiction types) would degrade to opaque custom predicates.

A **native** approach — treating KP:1 as the authoritative format and building tooling around it — is the current trajectory. RDF becomes an export target, not the source of truth.

The two are not mutually exclusive. The most pragmatic path: author in KP:1 (human-friendly, LLM-optimized), export to RDF/JSON-LD (machine interchange, SPARQL query), accept the documented loss, and don't pretend the round-trip is lossless.

This document follows the same honesty standard as the positioning piece it extends. If a mapping is wrong, imprecise, or unfair to a cited standard, that is a bug.

Independence — Knowledge Pack Companion Spec

Status: Draft companion (normative in intent, documentary in force — see "What makes these guarantees binding" below). **Date:** 2026-06-13 **Editor:** Timothy Kompanchenko

Purpose

KP:1 exists so that knowledge can be **owned**, not rented. A person (or organization) should be able to hold their knowledge as files, hand those files to someone else in fifty or five hundred years, and have them remain readable and usable — without paying anyone to maintain them, without a blessed tool, without a registry, and without permission from whoever happens to steward the format at the time.

This companion states the independence guarantees that the format already provides, and commits the project to preserving them. It does not introduce new mechanisms; it names the ones that exist and makes the commitment explicit so that future stewardship cannot quietly erode it.

What makes these guarantees binding

A sentence in a specification document is **not** itself a lock — a future editor or foundation that controls this repository can revise any prose. The real, irrevocable guarantees live one layer beneath this document:

- the **CC-BY-4.0 license** on every published version of the specification (a perpetual, irrevocable grant — anyone may use, adapt, and fork the text of any released version, forever); and
- the **distributed copies** of packs and of the spec that already exist outside any single party's control.

This document is therefore a **commitment and a Schelling point**, not the lock itself. Its force is that it states plainly what the license and the format design already make true, so that a departure from these guarantees is visible as a departure.

The guarantees

1. **Zero-dependency readability (the directory is sufficient).** A conforming pack **MUST** be fully parseable and usable from the directory contents and the Rosetta header alone — no network, no registry, no server, no blessed runtime, no living maintainer, and no resolution of any external URL. The Rosetta header ([CORE.md §4](#)) carries the minimum parse hint inline; the optional `spec_uri` is an onboarding convenience and **MUST NOT** be fetched as a condition of validation ([CORE.md §3](#)). Any conforming reader — from a small local model running on consumer hardware to a frontier model — must be able to read a pack with nothing else.
2. **Freedom to fork.** The specification text is CC-BY-4.0; anyone may fork it. The KP:1 trademark exists only to prevent a fork from impersonating the canonical specification — it is **defensive only and MUST NEVER be used to hinder a clean fork**. A fork is welcome to exist under any other name (see [GOVERNANCE.md](#)). Forking the content is a right, not a transgression; only the name is protected.
3. **No required registry or central service — ever.** Conformance **MUST NOT** depend on any central index, discovery service, certification authority, reconciliation service, or other coordination point. Composition and cross-pack reference are peer-to-peer (by relative reference), never via a mandated hub.
4. **Cryptography is optional for use.** Signing and integrity metadata are available ([ARCHIVE.md](#)) and recommended for sealed distribution, but **reading and using a pack MUST NEVER require keys, signatures, or verification tooling**. A plain, unsigned directory is a valid pack.
5. **Accrued knowledge stays portable.** A system that accumulates knowledge or memory on a person's behalf **SHOULD** let that person export the accrued state as a portable KP pack at any

time. The format's purpose is defeated if the knowledge a person generates can be read back only through the tool that produced it.

A principle: capture-resistance is subtraction, not addition

No artifact with a name and a maintainer can be made *impossible* to capture. What can be made **pointless** to capture is one that requires nothing: capture the name, the registry, or the stewarding organization, and every existing pack keeps working, in everyone's hands, forkable forever. The captured thing is then an empty shell.

It follows that the durable defense of independence is **fewer dependencies and fewer authorities**, not more governance. Every structure added to "protect" the format — a registry, a certification program, a mandatory anything — is itself a new thing that can be captured. The guarantees above are deliberately framed as removals (no required service, no required tool, no required key, no required permission), because removals cannot be turned into chokepoints.

Honest limits

- This is **capture-resistant, not capture-proof**. The goal is to make capture worthless, not impossible.
- The format is independent; **intelligence is not free**. Running a powerful model over a pack costs real compute, which may concentrate in whoever pays for it. The guarantee the format can make is the **floor**: a local-model path over your own packs must always remain a fully functional way to read and reason, even when a more capable hosted option exists. Independence means the floor is never removed — not that the ceiling is free. That floor itself rests on the continued availability of open-weight models and capable consumer hardware; it relocates dependence to a diffuse, competitive base rather than eliminating it.
- Stewardship of the specification should be funded **independently of any single implementer**, with no funder holding authority over normative decisions, so that the standard never bends to the commercial interest of one of its consumers.

This file is a companion to [SPEC.md](#), [CORE.md](#), and [GOVERNANCE.md](#). Where it appears to specify parser behavior, the normative contract lives in [CORE.md](#), the JSON Schema, and the PEG grammar; where it appears to specify governance, [GOVERNANCE.md](#) is authoritative.

KP:1 Specification

Status: Released Editor's Draft — v0.8.5-preview (2026-10-13) **Editor:** Timothy Kompanchenko

The normative contract for Knowledge Pack version 1.

Three normative lanes

KP:1 carries normative weight across three documents, each with a distinct role. A conformant implementation reads the lanes relevant to the surfaces it implements: a claim-file parser starts with CORE and its grammar; support for additional kinds, composition, locales or archives requires the relevant companions.

CORE.md — implementer surface. The minimum required to be KP:1-conformant. CORE.md is intentionally narrow: just enough to build interoperable parsers and validators. If you are implementing KP:1, start here; CORE.md and its normative artifacts define the generic claim-bearing directory surface. Additional surfaces require their authoritative companions.

SPEC.md — full normative spec, rationale, and ecosystem. The comprehensive document. SPEC.md predates CORE.md and covers the same ground plus rationale, ecosystem context, and historical decisions. Where CORE.md and SPEC.md appear to disagree on the implementable surface, CORE.md (the narrower, more recent extraction) is authoritative; SPEC.md provides the surrounding "why."

Companion documents — topic-authoritative for their domains. Each companion (VOICE, MULTILINGUAL, ARCHIVE, COMPOSITION, DEFINITIONS, etc.) owns its declared format surface. For matters within its domain, the companion is the normative authority — not merely an "extension" of CORE/SPEC. A producer or consumer that needs to handle locales correctly must read MULTILINGUAL.md; CORE.md alone is not sufficient.

AUTHORING, MAPPING, RATIONALE and STORAGE are informative. EXTENSIONS is a producer-profile catalog: its profile requirements are conditional on claimed profile support, not universal KP:1 requirements. Experimental and stub documents retain their stated limits. COMPATIBILITY.md records authority, current reference capabilities and draft behavior and support boundaries.

Companions do not weaken CORE: where CORE makes a claim, that claim holds. But for topics CORE explicitly punts to a companion (per CORE's *see* [COMPANION] cross-references), the companion carries the full normative weight on that topic.

Document	Topic
AUTHORING.md	Producer-side decision rubrics — how to make sound choices when authoring
VOICE.md	Voice surfaces for rendering claims as prose
COMPOSITION.md	Multi-pack composition and meetings
LIFECYCLE.md	Pack lifecycle management
MULTILINGUAL.md	Multilingual support
ORGANIZATION.md	Pack organization and categories
CONSISTENCY.md	Cross-pack consistency patrol
CONVENTIONS.md	Naming conventions
STORAGE.md	Storage formats and serialization
BUNDLE.md	Bundling and distribution
ARCHIVE.md	ZIP archive, integrity chain, signatures
DEFINITIONS.md	Definition and policy document kinds
EXTENSIONS.md	Producer-defined extensions. * blocks catalogue
NOTES.md	AI note-taking integration
PLAYBACK.md	Self-driving presentation playback (experimental)
RECONCILIATION.md	Cross-pack reconciliation protocol (stub — design deferred to v0.9 / v1.0)
RATIONALE.md	Rationale and positioning (informative — design principles, style systems, relationship to other standards)

AI agents: see [AGENTS.md](#) at the repo root for a task-indexed reading map (six canonical tasks: parse / author / reconcile / translate / compose / playback). It points you at the minimum required reading per task, so you don't have to load all ~9,800 lines of spec prose across the 22 spec/*.md files into your context window.

Interoperability

MAPPING.md maps KP:1 concepts to RDF/JSON-LD, PROV-O, and Nanopublications with clean/lossy/impossible grades for each field.

Validation

Formal grammar and test fixtures live in `../conformance/`.

Knowledge Pack Specification — Changelog

v0.8.5-preview — 2026-10-13

Coherent directory contract, preservation boundaries and reference coverage.

- **Evidence / identity:** CORE §10 and AR-19 register `markdown-backslash-v1`; SC-13 rejects duplicate evidence IDs. Shared reference parsing segments before decoding, handles blockquote/list metadata and validates required fields/dates. Unknown encodings preserve their body with a warning. Contract tests cover indentation, line endings, source-authored escapes and record-shaped source excerpts.
- **History and claims:** active/history IDs share one pack-local namespace for SC-02 through SC-05. Historical bullets preserve metadata and relations. Verbose field order is unrestricted, with required/optional cardinality checked after parsing. Fixtures exercise two generations, retraction and an empty current state after the last withdrawal. Strict history parsing keeps literal record boundaries rather than repairing blank lines or missing final newlines. The compiler accepts all required verbose-field orders. Empty manifests return diagnostics; missing declared fixtures fail.
- **Kinds and carriers:** the CORE file matrix agrees with DEFINITIONS for definition/policy/mixed packs. Canonical generic filenames remain required; alternate mappings require a profile. Definition/policy document-schema validation and full compiler history remain outside reference coverage.
- **Composition:** `selection` joins the published composition schema. Purpose remains free text; purpose, collection and scope are nonempty and required. The selection fixture exercises live member references. An optional manifest declaration names only the canonical composition file, which must exist. Preparation date is not a historical snapshot; collection membership and pinned-revision resolution belong to supporting profiles.
- **Seals and archives:** payload v2 binds identity and its quoted `seal_version: "2"` selector; legacy v1 remains explicitly described. Hash exclusions use exact components at any depth plus AppleDouble prefixes. Fixed hash vectors cover ordinary dotfiles, root-only receipt exclusion, NFC filenames and raw bytes. Hash-policy changes can affect historical receipts and are distinct from signing-payload version.
- **Storage and sharing:** semantic JSON fidelity is distinct from sealed-byte fidelity. Integer representation alone does not preserve bytes. Reading/clipboard bundles are projections, not complete lossless archives.
- **Profiles:** the entity/case/event catalog expands; operational per-claim associations have no generic file carrier. Case pins require unambiguous profile resolution; a newer sibling is not proven supersession. Original multilingual sources must survive the legacy transcript cardinality limit; locale paths must resolve unambiguously.
- **Purpose:** README and positioning explain the AI-first, lasting knowledge-system intention in plain language: retain context, relationships, uncertainty and changes for reasoning and continuation. Application operations remain distinct from the file format.
- **Context and change:** authoring/consistency/lifecycle guidance preserves material attribution, roles, dates, methods and decision context. Event placement is not a judgment that knowledge has no lasting value. Support revisions preserve complete issued records. The acquisition example distinguishes a bounded no-match search from unknown coverage; compiler pointers and leans are presentation preferences, not adopted assessments or permission to act.
- **Examples and maturity:** authoring/examples distinguish source reports, dated facts and adopted assessments; preserve attribution; use valid split IDs and registered relations. [COMPATIBILITY.md](#) records authority, capabilities and support boundaries. The current-belief compiler's deterministic preference is not truth or evidence independence. Public citable metadata, reference version strings and the searchable specification PDF identify this released preview.
- **Verification:** the fixture/example suite grows from the released 23 cases to 29 (10 valid, 14 invalid, 5 examples). Twenty-two reference contract tests exercise the changes above, and 50 graph compiler tests exercise its bounded projection. Registered evidence fields are closed, selection member references are nonempty, optional space after the evidence blockquote marker is

accepted and structured composition claims pass strict PEG parsing. A suite pass is not whole-format or application certification.

Compatibility is not claimed monotonic: selection scope, history uniqueness, evidence fields and archive hash rules can change validation or verification results. Preserve accepted historical bytes and receipts; target a pinned spec revision and declared profiles. This is a **released editor's draft**, archived under a version DOI, with preview compatibility and implementation limits. The [decision record](#) records editor acceptance through [issue #52](#) and [PR #53](#).

v0.8.4-preview — 2026-06-13

Independence made explicit: a capture-resistance companion, the offline-validation invariant exercised, and capture-resistance non-goals.

Adds `spec/INDEPENDENCE.md` (NEW companion) — it states, and commits the project to, the ownership and capture-resistance guarantees the format already provides: zero-dependency readability (the directory + Rosetta header alone is sufficient; `spec_uri` is never fetched), freedom to fork (CC-BY-4.0; the KP:1 mark is defensive only and never used against clean forks), no required registry or central service, optional-not-mandatory cryptography, and portability of accrued knowledge. It is documentary in force — the irrevocable lock is the CC-BY-4.0 license on each published version plus distributed copies — and normative in intent. Throughline: capture-resistance is *subtraction, not addition*.

- **maximal.kpack** now declares an intentionally-unresolvable `spec_uri` (`https://example.invalid/kp`), exercising the `CORE.md` §3 invariant that validation **MUST NOT** fetch or depend on `spec_uri` (offline-first). The fixture also closes a coverage gap — the "maximal" pack previously omitted the optional `spec_uri/spec_version` discovery fields.
- **positioning/kp-intent-and-principles.md** records the capture-resistance Non-Goals (no required registry/index/central service; no model-specific embeddings inside a pack; no read-time cryptography requirement; no external-identity/ledger anchoring; no gating the core read/own/local-model path) and adds an "Independence by design" operating principle.
- **spec/CORE.md** Appendix C lists the new companion; **spec/RATIONALE.md** clarifies that contradiction-preservation is epistemic pluralism (paraconsistent at the pack level), not a metaphysical claim, and points to the companion; **GOVERNANCE.md** links its trademark/fork rationale to the independence guarantees.

Monotonic: no breaking changes and no new conformance requirement; existing packs continue to validate and the suite result is unchanged (23/23 default and `--strict`).

v0.8.3-preview — 2026-06-10

The grammar becomes executable: PEG-driven strict validation, a packaged CLI, `kpack new`, and anchor-aware link checking.

This patch preview machine-executes the normative PEG grammar for the first time (`conformance/run.py --strict`), packages the reference CLI as an installable console script with `argparse` end-to-end, implements `kpack new` (scaffold a validated pack from the starter), verifies documentation anchors against GitHub heading slugs, and sharpens the README's first-screen storefront.

Monotonic: no breaking changes; the default validation path is unchanged and existing v0.8.2-preview packs continue to validate. AR-17 records two document-structure tolerances the shipped corpus already used.

Added

- **--strict (PEG layer)** — the runner parses each pack's `claims.md` through `grammar/kp-claims.peg` itself, loaded at runtime via a mechanical Ford-PEG → parsimonious translation (documented in `run.py`; the shipped grammar stays implementation-neutral). Strict errors append after the regex/semantic layer; composition packs are exempt (narrative `claims.md` by design); the runner exits loudly if `parsimonious` is absent rather than degrading silently. `requirements.txt` pins

`parsimonious>=0.11`, and CI runs the strict suite in both jobs (Python 3.11–3.14). Making strict the default is deferred to a later release.

- **AR-17** (CORE Appendix B, `grammar/README.md`) — blank lines are permitted between structural elements, and a section MAY open with a blockquote description mirroring the document heading. Both surfaced by the grammar's first machine execution; both already used throughout the fixtures and examples. CORE §5's sections prose updated to match.
- **Packaging** — minimal `pyproject.toml`; `pip install -e .` from a checkout puts `kpack` on `PATH` (console script). The CLI implementation moved to `reference/kpack_cli.py` with `argparse` end-to-end: unknown flags and subcommands fail loudly with exit 2 (the previous dispatcher silently dropped unrecognized `lint` flags). `kpack lint` forwards `--strict`. Editable installs are the supported layout; a plain site-packages install refuses `lint` with a message naming that.
- **kpack new <name> [--from hello-world]** — copies the starter, rewrites the pack identity in lockstep (PACK.yaml name/version and the `claims.md` frontmatter per SC-07/SC-08, titles, metadata/captured dates), validates the scaffold strict before reporting success, and warns if starter identity ever survives the rename.
- **Anchor-aware link checking** — `scripts/check_links.py` verifies `#fragments` against the target document's real anchors: GitHub heading slugs (underscores preserved, fence-character tracking, -N dedup) and explicit HTML anchors. Same-file fragment links, titled links, absolute paths, and percent-encoded targets are all covered.
- **README storefront** — a prose-vs-KP:1 before/after panel under "What is KP:1?", the metadata-block legend beside the relation legend, an "If you are an AI agent" routing subsection, and MAPPING grade counts (4 clean / 8 lossy / 9 none of 21 concepts) in Interoperability.

Changed

- The `compat` (3.11/3.12/3.13) CI jobs are now required status checks on `main` alongside `ci` — the published Python floor is enforced on every merge, not merely reported.
- README Status section resequenced: published artifacts first, editor's- draft caveats second (same sentences).
- ACKNOWLEDGMENTS: AI drafting assistance is attributed at the vendor/ product level, and a line notes the format was drafted in workflows of exactly the kind it addresses.

Fixed

- `dangling-relation-target.kpack` carried an accidental second violation — a mid-line relation followed by prose, illegal per AR-04, which the permissive regex layer never noticed. The relation now ends its line, so the fixture fails for exactly its declared reason (SC-05).
- Three stale documentation anchors, surfaced by the first anchor-aware link check and verified against GitHub's rendered HTML: `ARCHIVE.md` → CORE §4 (Rosetta header), `EXTENSIONS.md` → CORE Appendix B (AR-09), and `EXTENSIONS.md` §3.3 → `STORAGE.md` §3.2 (where `KPclaim` is actually defined; the cited CORE section never defined it).

v0.8.2-preview — 2026-06-10

Conformance hardening, the first runnable `kpack` subcommand, and a cross-document consistency pass.

This patch preview makes JSON Schema `format` assertions actually enforce, validates `composition.yaml` against its published schema, ships `kpack lint` as the first implemented reference-CLI subcommand, adds a copy-this starter example, opens the repository's feedback funnel, and lands a structured consistency pass across the spec documents and formal artifacts. Monotonic: no breaking changes; existing v0.8.1-preview packs continue to validate unchanged.

Added

- **Format-assertion enforcement** — `requirements.txt` now pins `jsonschema[format-nongpl]`, so `date`, `date-time`, and `uri` formats in `PACK.yaml` / `signatures.yaml` / `composition.yaml` validation are enforced rather than silently skipped (bare `jsonschema` ignores formats it has no validator for). The runner warns on `stderr` when the optional validators are absent.

- **composition.yaml schema validation** — the runner validates composition files against `kp-composition.schema.json` whenever the file is present; CORE AR-14 updated to record the behavior.
- **Two negative fixtures** — `bad-format-fields.kpack` and `bad-composition-schema.kpack`; the suite grows to 23/23 (18 fixtures + 5 examples).
- **kpack lint** — first implemented subcommand of the reference CLI; delegates to `conformance/run.py --pack` with identical exit codes. The runner gains `--json` and `--no-color` flags plus `NO_COLOR`/terminal auto-detection.
- **examples/hello-world.kpack** — the minimal copy-this starter, validated as part of the conformance suite so it can never drift.
- **Repository gates** — CI now runs full-tree and commit-message hygiene scans, the relative-link check, and a Python 3.11–3.13 compatibility matrix alongside the primary 3.14 job; `workflow_dispatch` enabled.
- **Community surfaces** — six issue forms matching CONTRIBUTING's feedback lanes, `SECURITY.md` with private vulnerability reporting enabled, and repository badges.

Changed

- **README opening** — the KP:1 self-quote leads as a three-line hook; the title, anti-flattening pitch, badge row, and status block sit in the first screen; the Goethe and Barenboim epigraphs open "What is KP:1?". New "When to reach for it — and when not" scoping box and a relation-symbol legend under the Quick Start.
- **CORE §6.1 / AR-04** — relations-placement language now states the exact normative shape (relations are separated from the closing `}` by a space and form a comma-separated list ending the line); the conformance README documents that the regex runner is deliberately more permissive than the PEG at line level — implement to the PEG.
- **VOICE §4 and MULTILINGUAL §3 subsections numbered** (4.1–4.3 and 3.1–3.3), making the long-standing §4.1 / §3.3 cross-references and anchor slugs resolve.

Fixed

- **PEG grammar** — relations on the metadata line now parse (the prior `MetadataTrailingProse` swallowed them as prose, contradicting CORE, AR-04, and the shipped fixtures); `EvidenceFieldName` gains `reliability/credibility` (defined in CORE §10); SC-12 added to the grammar's semantic-constraint inventory.
- **AUTHORING.md worked examples** — seven nature-bearing examples placed the nature keyword in position 5 (depth's slot); all now use the empty-depth form `{...|date||nature}` per CORE §6.1.
- **entities.md deprecation record** unified to v0.7.7 (2026-04-28) across CORE, SPEC, and EXTENSIONS — three documents previously carried three different versions.
- **Cross-references** — dangling pre-publication pointers in `CONVENTIONS.md` (`i18n.md`, `namespaces.md`, `privacy.md`, a workspace schema path, and a stale self-name); `COMPOSITION.md`'s voice-architecture pointer (SPEC §15 never described it); `LIFECYCLE.md`'s stale M-prefixed claim IDs in the reconciliation example; `reference/kpack`'s ARCHIVE section citations; the nonexistent `AudienceProfile.persona` field name in `VOICE.md` (`→ .register`); the `domain_lens` base-set count (six, not five); four feature-matrix cells in `examples/INDEX.md`; SPEC's relation-verb table gains the missing `see_also` row and the provenance `role` comment gains `co-author`; `history.md` aligned to OPTIONAL (CORE wins).
- **ARCHIVE.md** — merge-parent integrity posture stated plainly (`parent.merge_parents` is covered by neither the signing payload nor the content hash and is unauthenticated lineage metadata); §4/§5 merge-parent wording reconciled; the `kp-signatures.schema.json` artifact is now linked from §4; AR-14 banner names the shipped schema.
- **scripts/check_links.py** — a character-class bug (`[^http]`) meant any relative link whose target begins with `h`, `t`, or `p` was never checked; the checker now also exits non-zero on breakage so CI can gate on it.
- **Documented suite counts** corrected to match the runner across all surfaces (19/19 `→` 23/23 as fixtures and examples landed).

Compatibility

- **Monotonic relaxation of nothing; no breaking changes.** All v0.8.1-preview packs validate unchanged. Format-assertion enforcement only rejects values that were already invalid per the schemas (malformed dates and URIs); all five reference examples pass unchanged.

v0.8.1-preview — 2026-05-10

Voice-view register metadata promoted into the conformance schema.

This patch preview makes the `register` / `sub_register` voice-view metadata surface explicit in `PACK.yaml` validation, matching the companion-spec language introduced in v0.8.0-preview. It is a monotonic schema addition: existing v0.8.0-preview packs continue to validate unchanged.

Added

- **`views[].register`** (`conformance/grammar/kp-pack.schema.json`) — optional voice-view diction/emphasis profile. Accepted values are the four VOICE.md primary registers: `plain`, `curatorial`, `technical`, and `investor`.
- **`views[].sub_register`** (`conformance/grammar/kp-pack.schema.json`) — optional free-form locale-specific refinement such as `uk-halychyna` or `fr-quebec`. This remains informational, not a closed normative enum.
- **Maximal conformance fixture coverage** — `conformance/fixtures/valid/maximal.kpack/PACK.yaml` now exercises `register: investor` and `sub_register: uk-halychyna` on a voice view.
- **`scripts/check_links.py`** — a small local-link checker for Markdown files, intentionally separate from the conformance runner.

Changed

- Current release/status markers are refreshed from v0.8.0-preview to v0.8.1-preview across the README, CITATION metadata, current CORE/SPEC headers, conformance fixture docs, and the reference `kpack` stub.
- `CONTRIBUTING.md` now names the current preview tag as v0.8.1-preview.

Compatibility

- **No breaking schema change.** Both new fields are optional and live inside existing view declarations. Packs valid under v0.8.0-preview remain valid under v0.8.1-preview.
- The `register` enum is intentionally limited to the four primary VOICE.md registers; `sub_register` stays free-form so locale-specific practice can evolve without forcing a schema bump for every dialectal distinction.

Notes

- The Zenodo concept DOI remains the generic citation target for the current KP:1 preview line. The v0.8.0-preview snapshot DOI remains available as the previous citable snapshot.

v0.8.0-preview — 2026-05-09

Three-lane normative architecture, EXTENSIONS validation, spec self-discovery, AI-first packaging (AGENTS.md + AUTHORING.md), and companion-doc readiness for v0.8.0.

The v0.8.0 preview consolidates a deliberate architectural pass on the spec's normative structure (the three lanes), an honest validation of every active extension block, two new spec-level fields that let a sealed `.kpack` archive be self-explaining to a cold receiver, several companion-doc additions (RECONCILIATION stub, VOICE `register` axis, MULTILINGUAL `en-GB` row + sub-registers), and the AI-first packaging that the format's framing requires (root-level `AGENTS.md` task-routing file plus `spec/AUTHORING.md` producer-side decision rubrics). No core schema is broken; pre-v0.8.0 packs continue to validate.

Added

- **Three normative lanes** (`spec/README.md`) — explicit declaration that KP:1 carries normative weight across CORE.md (implementer surface), SPEC.md (full spec + rationale + ecosystem), and topic-authoritative companions. Companions are NOT mere extensions of CORE/SPEC: for matters within their domain, the companion is the normative authority. CORE.md and SPEC.md gain Lane: lines in their metadata blocks pointing at the README overview.
- **AGENTS.md** (NEW, repo root) — AI-first routing file. Provides a task→reading map for the four canonical tasks (parse, author, reconcile, translate) plus composition and playback. Tells a cold AI agent exactly which files to load and which to skip per task, so the agent doesn't have to ingest 9,000+ lines of normative prose before doing useful work. Includes a MUST-NOT list distilling the most common failure modes (translating `claims.md`, omitting Rosetta header, defaulting to `⊗~` for every contradiction, emitting 1.0 confidence everywhere). Closes the central gap all four external reviewers flagged.
- **spec/AUTHORING.md** (NEW companion) — producer-side decision rubrics. Provides explicit decision trees for the closed vocabularies CORE.md defines but does not adjudicate: claim type (`o/i/c/i`), claim nature (factual / judgment / prediction / meta), contradiction qualifier (`⊗ / ⊗!` / `⊗~`), confidence calibration within Sherman Kent bands, depth (assumed / investigated / exhaustive), supersede vs edit vs split, granularity, and content routing (claims / evidence / views). Each rubric carries worked examples, anti-patterns, and hostile-reading sanity checks (Agreeable Pack, Atomic Dust, Confidence Hallucination, View Laundering). The format's vocabulary without rubrics was the single largest authoring gap; AUTHORING.md closes it.
- **spec_uri and spec_version** (PACK.yaml manifest root, optional) — discovery pointer that lets a previously-unfamiliar consumer (especially a cold receiver of a sealed `.kpack` archive) fetch the full spec from the URL the pack itself names. Complements the inline Rosetta header (the minimum-viable parse hint at the top of `claims.md`). New "Spec Discovery" subsection in CORE.md §3 and "Format discoverability" subsection in ARCHIVE.md §2 codify the cold-receiver protocol: consumers SHOULD read PACK.yaml first and follow `spec_uri` (when declared). Default published location: <https://github.com/tymofiy/kp>.
- **extensions.translations** (EXTENSIONS.md §2.8, NEW block) — evidentiary multilingual transcripts for packs whose canonical claims are English (per MULTILINGUAL §2 P1) but whose underlying evidence is in another language (handwritten witness statements, foreign-language press, recorded interviews). The English claim remains the single normative assertion; `translations` carries original-language transcripts as audit trail without making them co-canonical. Inaugural producer: the reference handwritten-inventory intake pipeline.
- **VOICE.md register axis** — recommended metadata header field on voice views with four spec values: `plain`, `curatorial`, `technical`, `investor`. Composes orthogonally with `pace` (speed vs diction). Distinct from app-side `AudienceProfile.persona` (consumer concept).
- **en-GB language tag** (MULTILINGUAL.md §3.2 table) — mechanical sibling of `en-US` distinguished by spelling, idiom, and a small set of dialect-specific terms. Other `en-*` tags reserved.
- **Register Sub-distinctions** (MULTILINGUAL.md §3.3, NEW subsection) — informational `sub_register` voice-view metadata field for locale-specific sub-registers (Western Ukrainian / Halychyna, Quebec French, peninsular Spanish, MSA vs spoken Arabic). Refines within the four VOICE.md primary registers rather than replacing them. Reviewer-responsibility note: a pack declaring `sub_register` MUST be reviewed by a native speaker of that sub-register.
- **RECONCILIATION.md** (NEW companion stub) — explicit placeholder for the cross-pack consolidation protocol. Full design deferred to v0.9 / v1.0 contingent on observing measurable drift across at least three real packs. Distinguishes from CONSISTENCY (single-workspace patrol) and LIFECYCLE (single-pack claim lifecycle).
- **kpack CLI status disclaimer** (SPEC.md §13) — explicit note that the `kpack` commands described in §13 (`kpack lint`, `kpack render`, `kpack reconcile`, `kpack translate`, etc.) describe **planned reference tooling**, not commands that ship in this repository today. The only command that exists today is `python3 conformance/run.py`. Closes the "fictional CLI" critique.
- **examples/art-acquisition-decision.kpack/** (NEW reference example) — anonymized buyer-side decision-support pack. Demonstrates the full repertoire of contradiction qualifiers (`⊗`, `⊗!`, `⊗~`),

supersession (⊃), all four claim types (o/i/c/i), multiple confidence calibrations, judgment / prediction / meta natures, and three audience-specific views (overview, counsel, voice briefing in curatorial register). Walks the decision rubrics from `spec/AUTHORING.md` end-to-end on a realistic decision-support scenario.

- **examples/auction-house-consignment-review.kpack/** (NEW reference example) — anonymized consigner-side decision pack. An auction house declines a sculpture consignment over a foundry-mark attribution dispute, with conditions for reconsideration. Demonstrates the decline path (judgment-shaped recommendation against acquisition rather than for it), cross-pack ↔ references, evidence-diversity over consignor-supplied + house-specialist material, and four audience-specific views including a formal-register consignor decline letter. Together with `art-acquisition-decision.kpack`, the two examples cover both sides of high-stakes asset-decision support.
- **spec/LIFECYCLE.md §6 Claim Supersession Cascade** (NEW section) — normative rule for what happens to dependent claims (linked via →supports / ←requires / ~refines) when a parent claim is superseded with ⊃. The cascade is reviewed by the editor, not auto-invalidated; the rationale and a worked example are inline. Cross-pack effects are explicitly deferred to `RECONCILIATION.md`.
- **spec/RATIONALE.md** (NEW companion, informative) — collected rationale, philosophy, and positioning. Houses the 25 numbered Design Principles (formerly `SPEC.md §15`), the Style Systems specification and rationale (formerly `SPEC.md §17`), and the Relationship to Existing Standards table (formerly `SPEC.md §20`). Bifurcation lets `SPEC.md` focus on the implementer-facing surface without interleaving rationale prose. Marked explicitly "informative — not normative for KP:1 conformance."

Bifurcated

- **spec/SPEC.md §15 / §17 / §18 / §20** — sections moved to `RATIONALE.md` and replaced with stubs that link forward.
 - §15 Design Principles → `RATIONALE.md §1`. All 25 numbered principles.
 - §17 Style Systems → `RATIONALE.md §2`. Style-system rationale, schema, renderer pipeline. `SPEC.md §17` retains only the brief `PACK.yaml` `style` field reference.
 - §18 Cognitive Perception Layer → split. `SPEC.md §18` retains the **normative** display block field list, the `hint` field, and the fallback hierarchy (the contract a renderer needs). The **rationale** (perception stages with timed budgets, Why-each-field-exists boxes, Stranger Test, Good/Bad authoring tables) moves to `RATIONALE.md §3`.
 - §20 Relationship to Existing Standards → `RATIONALE.md §4`. Comparison to `AGENTS.md` / `llms.txt` / `SKILL.md` / `MCP` / `RDF` / `Nanopublications`.
- **Cross-reference sweep.** All "(Principle N)" / "(SPEC.md §15, Principle N)" references in the rest of the spec corpus updated to point at `RATIONALE.md §1` instead. Touched: `COMPOSITION.md`, `CONSISTENCY.md`, `DEFINITIONS.md`, `LIFECYCLE.md`, `SPEC.md`, `STORAGE.md`.
- **CORE.md / SPEC.md companion lists refreshed.** `CORE.md` Appendix C and `SPEC.md §19` now enumerate all 22 `spec/*.md` files (was missing `AUTHORING`, `RATIONALE`, `EXTENSIONS`, `ARCHIVE`, `PLAYBACK`, `RECONCILIATION`, `MAPPING`). `CORE.md` Scope (§1) likewise.
- **CORE.md / SPEC.md status headers refreshed** to KP:1 Public Draft – 2026-05 (v0.8.0-preview), date 2026-05-09. The 2026-04 / 2026-03-29 labels were stale relative to `README.md` and `CITATION.cff`.
- **CORE.md entities.md directory entry** updated from `OPTIONAL` to `DEPRECATED` (since v0.7.4) with forward-pointer to `extensions.entities`.

Tooling

- **reference/kpack** (NEW contract-pointer stub) — closes the "fictional CLI" critique with a concrete pointer rather than just a `SPEC.md` disclaimer. Running `./reference/kpack <subcommand>` prints the spec section that defines that subcommand's contract (e.g., `kpack reconcile → spec/RECONCILIATION.md`, `kpack lint → spec/SPEC.md §13`, `kpack play → spec/PLAYBACK.md`). The stub does not implement any subcommand; it points at the contract. The only fully-implemented validator in this repo remains `python3 conformance/run.py`. The stub covers 19 subcommands.
- **python3 conformance/run.py --pack PATH** (NEW runner flag) — validate a single pack outside the bundled fixture/example set. Closes the "drop into examples/ to validate" workaround that didn't

actually validate copied packs (the EXAMPLE_ORDER list was hardcoded). Existing `python3 conformance/run.py` (full suite) behavior is unchanged.

- **Cross-pack reference parsing in the runner.** Extended `_REL_DENSE` regex to match `⊕pack_name#section_ref` form per CORE §9 and AR-16. Previously, cross-pack `⊕` references were silently dropped from the relation graph; SC-05's `#`-exemption was a no-op because no `#`-bearing target ever entered the graph. The auction-house example pack's cross-pack references are now visible to the runner.

Semantic Constraints

- **SC-12 (NEW): When nature is prediction, confidence MUST be ≤ 0.95 .** Predictions about future states have irreducible uncertainty; the 0.99+ band is reserved for trivially-falsifiable claims per `AUTHORING.md` §5. The constraint catches the "future-date confidence inflation" pattern (a prediction at 0.99 on a 100-year horizon) as a hostile-reading failure.

Editorial polish (final pre-tag)

The following polish items landed:

- **Runner-vs-CORE relation-on-metadata-line divergence:** runner accepts relations after `}` on the metadata line; CORE §6.1 had previously said "relation symbols on that line are treated as prose, not parsed as relations." Resolution: relax CORE §6.1 + AR-04 to *allow* relations after `}` on the metadata line (matching what the runner already does and what `AUTHORING` / `LIFECYCLE` examples already show). The "greedy prose" framing was over-stated; relations are matched first, remaining trailing text is prose.
- **Cross-pack `⊕` parsing in the runner:** see Tooling above.
- **`AUTHORING.md` examples compact-form note:** a compact-form note now appears at the top of §1 explaining that worked examples drop the `-` list marker and the line break for readability; real claim syntax is two-line per CORE §6.1.
- **`AUTHORING.md` content-routing example bold-wrapped blockquote evidence:** the example evidence block now uses CORE-correct `> **type:** ...` syntax matching CORE §10's blockquote requirement.
- **`SPEC.md` Quick Start bold-wrapped blockquote evidence:** same fix in the most-copied example in the spec.
- **`SPEC.md` §9 supersession example:** ``supersedes: C011`` replaced with ``relations: supersedes C011`` matching CORE verbose relation syntax. The dense `⊗C011` continuation-line form is also documented as the alternative.
- **`art-acquisition-decision.kpack` C005 reclassification:** the catalog-raisonné dating claim was annotated `⊠|judgment`; "judgment" was wrong because the claim asserts what the catalog says (reported source content, not analyst evaluation). Annotation removed; the downstream judgment is captured by the `⊗!` relation against the prior dating.
- **`VOICE.md` Principle 22 → 23:** post-bifurcation Principle 22 became "Knowledge is the source"; Locale-Before-Surface is Principle 23. Cross-reference fixed.
- **`COMPOSITION.md` Parent and §14 references:** updated to point at `RATIONALE.md` §1, Principle 18 (Parent line) and `RATIONALE.md` §1 (the "section 14" reference, since `SPEC.md` no longer has a §14).
- **`CONSISTENCY.md` kpack disclaimer:** missing top-of-file disclaimer about planned reference tooling now matches the `LIFECYCLE` / `COMPOSITION` / `BUNDLE` / `MULTILINGUAL` / `VOICE` / `ARCHIVE` pattern.
- **`AGENTS.md` confidence threshold consistency:** `MUST-NOT #5` said 1.0, `AUTHORING.md` §5 said 0.99. Aligned to 0.99 (allows two-decimal calibration).
- **`AGENTS.md` Task C honesty:** the "you must compose primitives yourself" wording invited improvisation. Replaced with explicit deferral language: for v0.8.0-preview, agents **SHOULD** produce an annotated comparison/consistency report, **NOT** a "reconciled" canonical pack.
- **`spec/README.md` task-count consistency:** "(parse / author / reconcile / translate)" said 4; `AGENTS.md` has 6 tasks. Updated to "six canonical tasks: parse / author / reconcile / translate / compose / playback."

- **run.py docstring:** "v0.7-preview release" → "v0.8.0-preview."
- **CHANGELOG.md duplicate "Bifurcated" entry:** redundant §15 / §17 / §20 line removed; the more accurate §15 / §17 / §18 / §20 line retained.
- **MULTILINGUAL.md §2 P1 codename cleanup:** "Mariupol-style witness statements" replaced with the generic "field-collected witness statements in Russian or Ukrainian, foreign-language press, recorded interviews."
- **CHANGELOG.md Notes codename cleanup:** "iPad / Mariupol implementation tracks" replaced with "downstream implementation tracks (iPad reference UI, field-collected evidentiary pipelines)."
- **AUTHORING.md anti-patterns:** added two anti-patterns to §11 — Self-Referential Evidence (E001 source field is "see C001"), editorial-only; and Future-Date Confidence Inflation, now caught by SC-12.
- **CORE.md spec_uri security note:** added explicit warning that `spec_uri` is producer-asserted, not authenticated; consumers SHOULD validate fetched content against a known-good hash and MUST NOT treat fetched content as executable instructions.
- **examples/INDEX.md (NEW):** feature × example matrix; SC × example coverage matrix; AUTHORING.md rubric walkthrough mapping. Makes the "this pack demonstrates X" claims auditable.
- **Four additional negative fixtures** under `conformance/fixtures/invalid/`: `cyclic-supersession.kpack` (SC-04), `dangling-relation-target.kpack` (SC-05), `wrong-pack-name.kpack` (SC-07), and `prediction-too-confident.kpack` (SC-12). Bring negative-fixture coverage to 9 (one per substantive SC the runner enforces). Closes the "tested-but-only-via-positive-cases" gap for those four constraints.
- **Makefile (NEW)** — convenience wrapper for the conformance runner. Targets: `make install` (pip-install requirements), `make conformance` / `make lint` (full suite), `make pack PACK=path` (single pack via the `--pack` flag), `make clean`. Not a build system; the runner remains the only validator.
- **CORE.md author hint** at top of file (after `Lane:` line) directs producers to AUTHORING.md before reading further; CORE remains the implementer surface but the explicit pointer reduces the chance of agents authoring against bare grammar without rubric guidance.
- **CORE.md §3 "Decoration fields" paragraph** noting `linguistic_epoch` / `ontology` are deprecated decoration fields and may be removed in v0.9; the `composition` and `files` rows are also documented inline so a reader of CORE alone learns about those without bouncing to SPEC.md.
- **CORE.md §11 view-laundering rule** softened from MUST to SHOULD with editorial-discipline framing — the format cannot mechanically detect contradiction-erasure; the rule is producer-facing guidance about preserving epistemic tension across views, not a runner-enforceable constraint. The hostile-reading anti-pattern is preserved verbatim in AUTHORING.md.
- **PLAYBACK.md §9 "Stable for v0.8.0-preview"** subsection now lists the parts of the PlaybackPlan schema that will not change without a deprecation cycle (top-level shape, AudienceProfile, four primary registers, P1 and P2 architectural rules, the `min_tier` mechanism). The rest of §9 continues to enumerate what may evolve before v0.9.
- **RECONCILIATION.md "What to do today"** section adds a worked v0.8.0-preview reconciliation report example demonstrating how to surface cross-pack disagreement using existing primitives (`⊗packB#claim-id`, `single-pack ⊗!` / `⊗~`) without auto-flattening into a "reconciled" canonical pack. Makes the deferral concrete instead of abstract.
- **RATIONALE.md §4 schema.org / JSON-LD knowledge graphs row** added to the Relationship to Existing Standards table. Explicitly addresses the "why not just use schema.org?" question with three concrete differences (token density, first-class uncertainty/contradiction primitives, three-surface architecture) and notes the two approaches compose via [MAPPING.md](#).
- **AGENTS.md Task E secondary** column now references [LIFECYCLE.md §6](#) (supersession cascade) — relevant when a composed standing pack supersedes a claim the meeting pack referenced.

Conformance after these additions: 19/19 (was 13/13 at v0.8.0-preview ship; new examples added to the auto-validated suite, then four additional negative fixtures for SC-04 / SC-05 / SC-07 / SC-12).

Polish (final pre-tag audit)

A last consistency pass before tagging surfaced and resolved the following nits. None substantive; all cosmetic-class.

- **CHANGELOG.md broken relative links** — RATIONALE.md and MAPPING.md links had incorrectly pointed through a redundant `spec/` prefix, resolving to `spec/spec/...` from inside the `spec/` directory. Stripped the redundant prefix; LIFECYCLE.md, MAPPING.md, RATIONALE.md links from CHANGELOG now resolve to siblings. Also fixed a typo: §17 Style Systems was mislabeled §15 Style Systems in the bifurcation bullet.
- **spec/CORE.md SC table ordering** — SC-12 was sorted between SC-04 and SC-05 by mistake; moved to the end of the table after SC-11 so the table reads in numerical order.
- **examples/README.md install command path** — `pip install -r ../conformance/requirements.txt` referenced a nonexistent file; corrected to `../requirements.txt` (the actual location at the repo root).
- **examples/INDEX.md --pack example path** — `python3 conformance/run.py --pack solar-energy-market.kpack` would fail from the repo root because the pack lives at `examples/solar-energy-market.kpack`; corrected the example.
- **examples/INDEX.md + examples/README.md solar-energy-market description drift** — the pack was described as exercising "all four claim types," "all relation types," and "multiple views (display + voice)." It actually has only `o/r/i` (no `c`), three relation symbols (`→` / `⊗` / `←`), and a single overview view. The matrix and prose now describe the actual surface honestly; the larger packs remain the place to look for the full feature set.
- **conformance/fixtures/README.md SC-07 terminology** — the wrong-pack-name fixture's prose called the `pack:` field a "Rosetta header" field; per CORE.md §3 the Rosetta header is the `<!-- KP:1 ... -->` comment, while `pack:` lives in the YAML frontmatter that follows. Tightened to "frontmatter `pack:` field" with an inline parenthetical clarification of the distinction.
- **conformance/fixtures/README.md PEG vs regex framing** — said the runner "parses each `claims.md` against `grammar/kp-claims.peg`," contradicting `conformance/README.md`'s honest disclosure that the runner uses equivalent regex patterns and that the PEG file is the normative reference. Aligned the wording.
- **conformance/run.py --pack with no path** — silently fell through to the full-suite branch and exited green. Now prints `usage: run.py --pack PATH` and exits 2.
- **CHANGELOG.md count fix** — two references to "all 19 `spec/.md` files" corrected to "all 22 `spec/.md` files" (matches the actual `spec/` listing and the ~9,400 lines of spec prose across the 22 `spec/*.md` files count in AGENTS.md).
- **CI bump** — `actions/setup-node@v6.3.0` → `@v6.4.0` (only floating Action that wasn't on its latest tag at audit time). `actions/checkout@v6.0.2` and `actions/setup-python@v6.2.0` confirmed already at latest. `pyyaml>=6.0.3` and `jsonschema>=4.26` confirmed at latest PyPI versions; `markdownlint-cli@0.48.0` confirmed at latest npm version.

Changed

- **EXTENSIONS.md** — clarified producer-profile maturity as of v0.8.0: `ai_brief` and `research` were active; `entities`, `relations`, `intent`, `playbook` and `workspace` were experimental. Documented the maturity-policy exception for the inaugural `extensions.translations` producer. Public examples and implementation references use generic labels.
- **Companion-doc Parent: lines** — dropped stale `> **Parent:** SPEC.md v0.X` lines from MULTILINGUAL, CONVENTIONS, BUNDLE, ORGANIZATION, CONSISTENCY, NOTES, STORAGE in the master-plan release; the post-master-plan cleanup commit extended this to ARCHIVE / VOICE / LIFECYCLE (no anchor → drop entire line) and softened version pins on COMPOSITION / DEFINITIONS / EXTENSIONS while preserving section anchors. CORE.md Derived from: SPEC.md v0.7 was also drift-stripped to Derived from: SPEC.md. Several (v0.4) / (v0.6) section header pins removed from SPEC.md.
- **spec/CORE.md** — added cross-references to AUTHORIZING.md from §1 (scope) and to SPEC.md §10 from §2 (validation.yaml schema). AR-10 row gained worked positive/negative tokenization examples (`~C002` is the relation, `~refines` and `~the manuscript` are prose).
- **spec/EXTENSIONS.md** — `extensions.intent` enum framing softened to clarify it is the reference implementation's vocabulary (producers MAY define alternative values), not a closed normative enum embedded in an explicitly informative document.

- **spec/SPEC.md** — Quick Start `claims.md` example gained the previously-omitted Rosetta header, so a fresh agent copying the example produces a pack that actually passes conformance. The cleanup commit also fixed pedagogical examples in `COMPOSITION.md` §3 (ASCII relation glyphs `-> * <- ~ / <-> →` KP symbols `→ ⊗ ← ~ ⊙ ⇔`; `[M001]-[M005] → [C001]-[C005]`) and `LIFECYCLE.md` (`[M001]-[M008] → [C001]-[C008]`; `EM003/EM004 → E003/E004`) so the worked examples actually validate against CORE.
- **spec/MULTILINGUAL.md** — section ordering corrected (Evidentiary multilingual exception is §12; Decision Log is §13; the master-plan ship had them reversed). Decision Log D1a entry updated correspondingly. The §12 + D1a inaugural-producer narrative replaced specific project codenames with the generic witness statements collected in the field framing.
- **spec/MAPPING.md** — Status: Reference document `→` Status: Draft (informative – not normative for KP:1 conformance) to harmonize sibling vocabulary while preserving the informative-vs-normative distinction.
- **spec/ORGANIZATION.md** — directory listing in §1 refreshed to include all 22 spec files (CORE, README, ARCHIVE, DEFINITIONS, EXTENSIONS, MAPPING, PLAYBACK, RECONCILIATION, STORAGE were missing from the v0.7-era listing).
- **SPEC.md title** — v0.7 `→` v0.8.0.
- **README.md** — status banner v0.7-preview `→` v0.8.0-preview; conformance fixture count corrected (15 test fixtures plus 4 reference examples, totaling 19/19 validated); `pip install -r conformance/requirements.txt` bootstrap line added; citation example updated to v0.8.0-preview while preserving the v0.7-preview Zenodo DOI as historical reference.
- **CITATION.cff** — version 0.7-preview `→` 0.8.0-preview; date-released 2026-04-06 `→` 2026-05-09. v0.7-preview Zenodo DOI tagged (historical) pending v0.8.0-preview Zenodo deposit (queued as WS-TAG).

Compatibility

- **No core schema break.** `spec_uri` and `spec_version` are optional manifest fields. `extensions.translations` lives in the existing `extensions` lane (added in v0.7.4). The `VOICE register` field is recommended, not required. Pre-v0.8.0 packs continue to validate against the v0.8.0-preview conformance runner.
- **Companions reaffirmed normative.** The "Three normative lanes" framing does not weaken CORE: where CORE makes a claim, that claim still holds. But for topics CORE explicitly punts to a companion (per CORE's see `[COMPANION]` cross-references), the companion now carries explicit full normative weight on that topic, codifying what was implicit.

Notes

- A v0.8.0-preview Zenodo deposit will follow this entry (WS-TAG: tag, deposit, GitHub release).
- The v0.8.0 *final* (non-preview) release will follow once the deferred RECONCILIATION design is observed-and-grounded (≥ 3 real drift instances) and the remaining downstream implementation tracks (iPad reference UI, field-collected evidentiary pipelines) have shipped against the spec.

v0.7.7.1 — 2026-05-09

Schema-only conformance relaxations.

Schema-side fixes for two classes of conformance failures surfaced in the 2026-04-30 ecosystem audit. No core spec or companion-document changes; pure schema bumps.

Changed

- **display.short_title maxLength 30 `→` 60** — Real canonical names of organisations and people overflow 30 chars (e.g., "LVMH Moët Hennessy Louis Vuitton", "Ministry of Culture and Strategic Communications of Ukraine"). The field's intent (constrained display contexts) is preserved — UI ellipsis at smaller widths is the rendering layer's job, not the schema's. Description updated to reflect the wider envelope and the rendering-layer responsibility.
- **role enum gained "co-author"** — Authorship role for packs co-authored by humans and AI (substrate authoring with parallel drafting sessions). Existing roles unchanged.

Compatibility

- **Monotonic relaxation, no breaking changes.** Pre-v0.7.7.1 packs continue to validate (anything that fit the 30-char `short_title` cap still fits the 60-char cap; the new `role` value is additive). Producers MAY now emit `short_title` up to 60 chars and `role: "co-author"`.

v0.7.7 — 2026-04-28

Extensions catalogue + `entities.md` deprecation.

The KP:1 manifest root remains closed; the `extensions` lane (added in v0.7.4) is the sanctioned compatibility surface for producer-defined metadata. This release catalogues the producer-defined extension blocks that the reference implementation uses today and retires the never-fully-realized `entities.md` companion file in favour of the typed `extensions.entities` block.

Added

- **EXTENSIONS.md** (NEW companion document) — informative catalogue of producer-defined extension blocks under `extensions.*` in active use across the reference implementation. Documents `extensions.ai_brief` (the v0.7.4 example), `extensions.entities` (typed entity graph), `extensions.relations` (typed edges between entities, mirroring the `relation_types` vocabulary in DEFINITIONS.md §3), `extensions.intent` + `extensions.intent_schema_version`
 - `extensions.intent_derivation` (decision-frame layer), `extensions.playbook` (runtime trace of playbook execution), `extensions.workspace` (Decision Workspace bookkeeping), and `extensions.research` (free-text research metadata). Cross-cutting sections cover canonical entity ID minting (`ent_<type>_<6-hex>`), the recommended `external_ids` vocabulary, and the per-claim `extensions.entity_refs` / `extensions.relation_refs` annotation. Migration notes cover `entities.md` → `extensions.entities` and free-text `anchor_subject` → typed entities.
- **README.md companion table entry** for EXTENSIONS.md.

Changed

- **DEFINITIONS.md §3** — `relation_types` example block prose extended with a pointer to the `extensions.relations` consumer pattern. The vocabulary itself is unchanged; the addition documents that producers now write relation instances under `extensions.relations` in `PACK.yaml` rather than in a separate file.
- **SPEC.md §7 (entities.md)** — section retitled "Deprecated: entities.md" with a deprecation banner pointing to `extensions.entities` as the successor. The full prior content is preserved for legacy reference.

Deprecated

- **entities.md** — used in only five packs out of ~770 and never landed as a practical primitive. Producers MUST NOT emit `entities.md` in new packs. Equivalent and richer information now lives under `extensions.entities` (typed entity records with stable canonical entity IDs and external-ID vocabulary) and `extensions.relations` (typed edges with optional temporal and attribute data per DEFINITIONS.md §3). A one-shot migration script (`scripts/migrate-entities-md-to-extensions.mjs`, shipped with the reference producer implementation) rewrites legacy packs into the new shape.

Compatibility

- **No core schema change.** The manifest root is unchanged. Producers and consumers continue to round-trip pre-v0.7.7 packs without modification.
- **Consumers MUST ignore unknown extension content** per the extensions lane contract (CORE.md §1.1, SPEC.md §3.2). Pre-existing producers and consumers that do not understand the new blocks continue to function; the new blocks are additive.
- **Legacy entities.md packs continue to validate** against the v0.7.7 conformance runner. The deprecation is a producer-side guidance, not a validation removal.

v0.7.6 — 2026-04-19

Documented optional Admiralty grading fields on evidence.

Added

- **reliability and credibility evidence fields** — Documented in CORE.md §10 as optional evidence fields with normative values from the NATO Admiralty Code. `reliability` is the source-tier letter (A–F); `credibility` is the per-fact information credibility digit (1–6). Together they form the standard intelligence-tradecraft A1–F6 grade. Both were previously accepted as open-vocabulary AR-07 fields and ignored by validators; this change makes the values normative without breaking any existing pack (both remain optional, parsers MUST tolerate absence).

Notes

No breaking changes. Packs produced before this entry remain valid. Packs that already used `reliability` informally (e.g. `**reliability:** moderate`) MAY continue to do so but SHOULD migrate to the single-letter A–F form for renderer interoperability.

v0.7.5 — 2026-04-18

Signatures schema alignment + composition-pack file requirements.

Added

- **pack_id in signatures.yaml** — Stable pack identity across versions, emitted by current sealers and used by relays. Documented in ARCHIVE.md §4 with a prose MAY/SHOULD policy for backwards compatibility (schema accepts absence; new archives SHOULD include it). `pack_name` and `pack_version` also documented as optional self-describing receipts.
- **parent.merge_parents** — Optional array inside the `parent` block for branch-merge lineage. Each entry has the same `version/pack_hash` shape as `parent` itself. Documents a latent capability declared in the sealer implementation so the refreshed schema does not reject valid merge-lineage archives.
- **Composition-pack file requirements** — New subsection in SPEC.md §2 listing the file-presence rules for composition packs: `composition.yaml` REQUIRED, `claims.md` REQUIRED (intentionally minimal, about the composition context only), `evidence.md` OPTIONAL. Cross-references `spec/COMPOSITION.md` for semantics.
- **Signatures schema validation in the conformance runner** — `conformance/run.py` now validates any `signatures.yaml` it finds against `kp-signatures.schema.json`, with `jsonschema.FormatChecker` enabled so declared formats (`date-time`) are actually enforced. Empty-file and YAML parse errors are surfaced with friendly messages. Previously the runner was silent on signatures.
- **Fixture: composition.kpack** — Valid-pack fixture demonstrating the composition-pack minimum shape (`composition.yaml` present, no `evidence.md`, prose-only `claims.md`).
- **Fixture: signatures.yaml on maximal.kpack** — Exercises the signatures schema with a populated `parent` block and the optional `pack_name/pack_version` fields.

Changed

- **kp-signatures.schema.json** — Regenerated to match ARCHIVE.md §4 (v0.7.3+ shape). Required fields are now `algorithm`, `pack_hash`, `files`, `sealed_at`, `sealed_by`. Optional: `pack_id`, `pack_name`, `pack_version`, `parent` (with `version` + `pack_hash` + optional `merge_parents`), `signature` (with `method` + `value` + `key_id`). Previous pre-v0.7.3 fields (`signed_at`, `signing_key`, `signature` as a flat string) removed. Schema root closed with `additionalProperties: false`, mirroring the manifest-schema pattern. The `files` map now enforces "MUST NOT include signatures.yaml itself" via `propertyNames`, not prose alone.
- **conformance/run.py** — `validate_pack()` detects composition packs (those with `composition.yaml`) and skips the `evidence.md` presence check for them. Evidence-ID extraction now tolerates a missing `evidence.md` (empty set rather than exception). `signatures.yaml` is validated against the new schema when present, with `FormatChecker` enabled so `format: date-time` on `sealed_at` is actually enforced.

Fixed

- **Schema-vs-impl drift** — The prior `kp-signatures.schema.json` required fields (`signed_at`, `signing_key`, `signature` as a flat string) that neither the reference sealer nor the reference relay has produced or consumed since v0.7.3. Any current sealed archive would have failed schema validation. Aligned.
- **Composition-pack validation hole** — Packs with `composition.yaml` and no `evidence.md` (valid per `COMPOSITION.md` design) were being rejected by the conformance runner for the missing-file reason. No longer.

v0.7.4 — 2026-04-16

Manifest extension lane — standardizes where experiments belong without widening the core schema.

Added

- **extensions** (`PACK.yaml`, optional object) — Explicit lane for experimental or implementation-specific manifest metadata. Root-level unknown fields remain invalid; experiments now belong under `extensions`.

Changed

- **CORE.md** — Documents the manifest root as closed and defines the `extensions` object as the sanctioned compatibility lane.
- **SPEC.md** — Adds example manifest usage and guidance that experimental fields such as `ai_brief` belong under `extensions`, not at the manifest root.
- **Schema** — Adds `extensions` as an allowed top-level object while preserving `additionalProperties: false` at the manifest root.

v0.7.3 — 2026-04-12

Archive format — sealed, hashed, versioned single-file transport for Knowledge Packs.

Added

- **Archive companion spec** (`ARCHIVE.md`) — Defines the ZIP-based archive format for pack transport between systems. Covers content hashing (SHA-256, deterministic, container-independent), `signatures.yaml` schema, version chains with parent hash references, sealing protocol, and verification. Resolves AR-14 (`signatures.yaml` schema deferred to Phase C2).
- **signatures.yaml schema** — Formalized in `ARCHIVE.md` §4 with normative field types. Fields: `algorithm`, `pack_hash`, `files` (per-file hashes, lowercase hex), `sealed_at`, `sealed_by`, `parent` (version chain), optional `signature` (digital signing). Required in sealed archives.
- **Version chain model** — Each sealed pack version references its parent's `pack_hash`, forming a verifiable integrity chain. Branching acknowledged; merge-parent semantics deferred to a future version.
- **Seal & transport CLI commands** — `kpack seal`, `kpack verify`, `kpack verify --chain`, `kpack extract`, `kpack info`. (`kpack archive` remains reserved for lifecycle archival per `LIFECYCLE.md`.)
- **Security considerations** (§10) — ZipSlip/path traversal protection, symlink rejection, duplicate entry rejection, OS metadata exclusion, line ending advisory, Unicode NFC normalization.
- **Conformance levels** — Sealed archive (with `signatures.yaml`, integrity chain) vs. export archive (without, convenience only).
- **Signing payload** — Signature binds `algorithm` + `pack_hash` + `sealed_at` + `sealed_by` + `parent.version` + `parent.pack_hash`, preventing metadata tampering.

Changed

- **CORE.md** AR-14 updated — `signatures.yaml` schema now resolved by `ARCHIVE.md`. `composition.yaml` remains deferred.
- **CORE.md** scope paragraph updated — `ARCHIVE.md` added to the list of companion documents.
- **CORE.md** Appendix C updated — `ARCHIVE.md` added to companion specifications table.

- **README.md** — ARCHIVE.md added to companion documents table.
- **SPEC.md** — `signatures.yaml` file role updated to reference ARCHIVE.md.
- **Hash computation** (§3) — Mandates NFC Unicode normalization for paths, lowercase hex digests, bare relative paths (no `./` prefix), and root-only `signatures.yaml` exclusion. Adds canonicalization notes for line endings and OS metadata.
- **Signing methods** — Clarified that HMAC-SHA256 provides tamper detection only, not non-repudiation. Ed25519 recommended as default. RSA uses PSS padding.
- **ZIP safety** — Added mandatory safety requirements (§2): path traversal rejection, symlink rejection, duplicate entry rejection, OS metadata stripping.
- **ZIP64** — Changed from SHOULD to MUST for archives exceeding 4 GB.
- **CLI examples** — Updated to show `-o` output paths, avoiding the `.kpack` file-vs-directory coexistence issue.
- **Visibility table** — Corrected from `kpack archive` to `kpack seal`.

v0.7.2 — 2026-04-12

Schema tightening and new metadata fields — informed by an open-standards review.

Added

- **evidence_basis** (PACK.yaml, optional string) — Narrative description of the evidence foundation underpinning a pack. Informational, not machine-actionable.
- **tags** (PACK.yaml, optional array) — Topical tags for discovery and classification. Enforced as unique, strict kebab-case strings (`^[a-z0-9]+(-[a-z0-9]+)*$`; no trailing hyphens).
- **Voice view fields** on view declarations — `voice` (boolean), `duration` (string, pattern `~N seconds` or `~N minutes`), `pace` (enum: `brisk`, `measured`, `deliberate`). Formalizes the PACK.yaml ↔ VOICE.md contract.
- **Voice view conditional** — When `voice: true`, `duration` and `pace` are REQUIRED on that view entry. Prevents incomplete voice metadata.

Changed

- **vocabulary additionalProperties** loosened from `string` to `oneOf [string, array<string>]`. Array branch requires `minItems: 1` and `uniqueItems: true`. Backward-compatible: existing string-valued entries remain valid.
- **pace** constrained to enum matching VOICE.md (`brisk`, `measured`, `deliberate`). Previously unconstrained string.
- **duration** constrained to pattern `^~\d+\s+(seconds?|minutes?)$`. Previously unconstrained string.
- **CORE.md** updated to document `evidence_basis`, `tags`, and `voice view` fields in the Core Optional Fields table.
- **VOICE.md** examples updated to include `display_as` on voice view declarations (required by schema but previously omitted in examples).
- **Maximal conformance fixture** updated to exercise all new fields.

v0.5 — 2026-03-24

Epistemic refinements — informed by Barenboim analysis of knowledge vs. information.

Added

- **Investigation depth** (claim field, position 5) — `assumed` / `investigated` / `exhaustive`. Distinguishes uninvestigated assumptions from informed uncertainty. A 0.50 at `exhaustive` depth is more valuable knowledge than 0.90 at `assumed`.
- **Claim nature** (claim field, position 6) — `judgment` / `prediction` / `meta`. Omitted = factual assertion. Enables agents to reason differently about interpretive conclusions, forward-looking claims, and claims about the state of knowledge itself.
- **Contradiction qualifiers** — `⊗!` (error: one is wrong) and `⊗~` (tension: both informative, preserve). Bare `⊗` remains the unqualified default.

- **Verbose alternative syntax** — Named-field format (`confidence: 0.95 | type: inferred | ...`) accepted alongside dense positional notation. Both valid KP:1. Tooling must parse both.
- **history.md format** — Defined structure for superseded and retracted claims with metadata and context.
- **Evidence reliability guidance** — Evidence prose should describe source reliability, perspective, and relationship to the claim.

Changed

- **claims.md is active-only** — Resolves the v0.4 contradiction between "active claims ONLY" and "append-only lifecycle." Superseded/retracted claims move to history.md. Disputed claims stay active (the dispute IS the current state).
- **Split rule reframed** — Hub packs provide the interpretive framework (thesis, tensions, confidence landscape), not just the most important facts.
- **Consistency checking** updated for contradiction qualifiers: `⊗~` (tension) skipped by patrol, `⊗!` (error) prioritized, stale disputes (>90 days) flagged.

v0.4 — 2026-03-22

Three-surface architecture and ecosystem expansion.

Added

- **Three-surface architecture** — Voice views join reasoning (claims.md) and display (views/) as a third surface optimized for spoken delivery
- **Multilingual support** — Locale subdirectories (`views/{locale}/`), translation workflow, drift detection, status tracking. Companion spec: `MULTILINGUAL.md`
- **Pack lifecycle management** — Three lifecycle types (permanent, seasonal, ephemeral), intelligent archival with claim reconciliation, visibility controls. Companion spec: `LIFECYCLE.md`
- **Meeting pack composition** — Composition over standing packs via `composition.yaml`, pre-load/on-demand references, agenda-driven structure. Companion spec: `COMPOSITION.md`
- **Bundle/export format** — Single-file format for sharing outside IDE (full bundle + clipboard variant). KP:1 marker. Companion spec: `BUNDLE.md`
- **Notes metadata** — AI note-taking mode (active/passive/off), disclosure tracking, consent for passive mode. Companion spec: `NOTES.md`
- **Cross-pack consistency** — Automated patrol for contradictions, staleness, broken references. Real-time alerting during conversations. Companion spec: `CONSISTENCY.md`
- **Linguistic conventions** — 30-rule American English convention table, Merriam-Webster as spelling authority, 7-level fallback hierarchy. Companion spec: `CONVENTIONS.md`
- **Pack organization** — Nested categories, two-zone model (clean packs vs legacy), migration strategy. Companion spec: `ORGANIZATION.md`
- **7 new design principles** (§14, #16-22): three surfaces one truth, composition over duplication, archive never delete, reconcile before archive, knowledge is the source, locale first then surface, renderer not compositor

Changed

- Two-surface architecture → **three-surface architecture** throughout
- `PACK.yaml`: added `visibility`, `lifecycle`, `locales`, `notes` optional field blocks
- File structure: added `composition.yaml`, voice view directory, locale subdirectories
- Tooling: added extended CLI commands (bundle, archive, reconcile, translate, patrol, new)
- Spec now lives in `spec/` directory with companion documents, not as a single file in `packs/`

v0.3 — 2026-03-21

Views layer and two-surface architecture.

Added

- Views layer (pre-rendered display content in `views/`)
 - Two-surface architecture (reasoning + display)
 - Style system references (`STYLE.yaml`)
 - Collection semantics (entity cards, analytics views)
 - View generation (`kpack render`), staleness detection
 - Semantic markers (`> [!metric]`, `> [!highlight]`, etc.)
-

v0.2 — 2026-03-19

Density optimization.

Added

- Density-optimized claim format with compressed metadata `{confidence|type|evidence|date}`
 - Voyager Principle — self-describing files with Rosetta Header
 - Inspectability over readability philosophy
 - Symbolic relations (`→⊗←~⊗↔`)
-

v0.1 — 2026-03-10

Initial specification.

Added

- Core file structure (`PACK.yaml`, `claims.md`, `evidence.md`, `entities.md`)
- Confidence model with Sherman Kent scale
- Claim lifecycle (active, superseded, disputed, retracted)
- Pack hierarchy (hub, detail, standalone)
- Navigation (`KNOWLEDGE.yaml`, `sub_packs`, inline cross-references)
- Validation framework (`validation.yaml`)

kp-claims.peg

Source: <https://github.com/tymofiy/kp/blob/v0.8.5-preview/conformance/grammar/kp-claims.peg>

```
// SPDX-License-Identifier: Apache-2.0
// SPDX-FileCopyrightText: 2026 Timothy Kompanchenko

// KP:1 Claims Document Grammar – Parsing Expression Grammar (PEG)
//
// Scope:      claims.md within a .kpack directory
// Version:    KP:1 (spec version 1)
// Notation:   Bryan Ford PEG (implementation-neutral)
//
// This grammar is normative for KP:1 conformance. Implementations MUST accept
// all documents that match this grammar. Implementations MUST reject documents
// that fail to match, subject to the conformance levels defined in the README.
//
// This grammar is machine-executed by `conformance/run.py --strict`, which
// loads this file through a mechanical Ford-PEG → parsimonious translation
// (rule arrow, character classes, bare any-char; see _translate_peg in
// run.py). The translation does not alter the language accepted. Note for
// implementers: `.` (any character) matches newlines – the Rosetta header
// legend spans multiple lines.
//
// Conventions:
// <- : rule definition
// /   : ordered choice (try left first)
// *   : zero or more
// +   : one or more
// ?   : optional
// !   : not-predicate (lookahead, does not consume)
// &   : and-predicate (lookahead, does not consume)
// .   : any character (Unicode codepoint)
// '...': literal string
// [...]: character class
//
// Unicode symbols used in KP claim syntax:
// U+2192 → RIGHTWARDS ARROW      (supports)
// U+2297 ⊗ CIRCLED TIMES         (contradicts)
// U+2190 ← LEFTWARDS ARROW       (requires)
// U+2298 ∘ CIRCLED DIVISION SLASH (supersedes)
// U+2194 ↔ LEFT RIGHT ARROW      (see_also)
// U+007E ~ TILDE                 (refines)
// U+2014 – EM DASH               (separator in Rosetta header)

// =====
// TOP LEVEL
// =====

// Blank lines between structural elements are permitted (AR-17): Markdown
// rendering conventions require them around headings and list items, and
// the entire shipped corpus uses them. A blank line is a bare newline –
// whitespace-only lines are NOT blank lines and remain illegal.

ClaimsDocument <- RosettaHeader NL Frontmatter NL BlankLine*
                  DocumentHeading (BlankLine* Section)+ BlankLine* EOF

// =====
// ROSETTA HEADER (mandatory, frozen across spec versions)
// =====
// Self-describing legend as an HTML comment. Identifies spec version.
// Content between "KP:N" and "-->" is informational – parsers MUST check
// the spec version but MAY ignore the legend text.

RosettaHeader <- '<!-- KP:' SpecVersion SP RosettaBody '-->'
SpecVersion   <- DIGIT+
```

```

RosettaBody      <- (!('-->') .)*

// =====
// FRONTMATTER (mandatory, pipe-delimited compact format)
// =====
// YAML fences with KP-specific compact syntax inside.
// This is NOT arbitrary YAML – it is a fixed two-line format.

Frontmatter      <- '---' NL FrontmatterPack NL FrontmatterConf NL '---'

FrontmatterPack  <- 'pack:' SP PackName SP '|' SP 'v:' SP CalVer SP '|'
                  SP 'domain:' SP Domain

FrontmatterConf  <- 'confidence:' SP ScaleName SP '|' SP 'normalized'

PackName         <- LOWER_ALNUM IDENT_CHAR*
CalVer           <- DIGIT DIGIT DIGIT DIGIT '.' DIGIT DIGIT '.' DIGIT DIGIT
                  ('-rev' DIGIT+)?
Domain           <- DOMAIN_SEG ('/' DOMAIN_SEG)*
DOMAIN_SEG       <- LOWER_ALNUM IDENT_CHAR*
ScaleName        <- LOWER_ALPHA ('_' / LOWER_ALPHA / DIGIT)*

// =====
// DOCUMENT HEADING (mandatory)
// =====
// H1 with optional entity annotation in brackets, followed by blockquote
// description.
//
// Format: # Title [entity_type|alias1,alias2]
//         > One-line description (may span multiple > lines)

DocumentHeading <- '# ' HeadingText EntityAnnotation? NL
                  (BlankLine* Description)?

HeadingText      <- (!('[' / NL) .)+

EntityAnnotation <- '[' EntityType ('|' AliasList)? ']'
EntityType       <- IDENT_CHAR+
AliasList        <- AliasEntry (',' AliasEntry)*
AliasEntry       <- [A-Za-z0-9_-]+

Description      <- BlockquoteLine (BlockquoteLine)*
BlockquoteLine   <- '>' SP? (!NL .)+ NL

// =====
// SECTIONS
// =====
// Claims are grouped under ## headings. Heading text is freeform.
// A section MAY open with a blockquote description before its claims,
// mirroring the document-level Description (AR-17).
// A section MAY contain zero claims: a pack can retain history after its
// final current claim is retracted, without manufacturing an assertion.

Section          <- SectionHeading (BlankLine* Description)? (BlankLine* Claim)*
SectionHeading   <- '##' SP (!NL .)+ NL

// =====
// CLAIMS (core grammar – dense and verbose forms)
// =====
// A claim is a markdown list item with structured metadata.
// Both forms begin identically: list marker, optional bold, claim ID, headline.
// They diverge at the metadata block.
//
// Rule: A single claim uses EITHER dense OR verbose metadata (not a mix).

```

```
// Relations may independently use either form.

Claim          <- DenseClaim / VerboseClaim

// --- Dense Claim ---
// Metadata in curly braces, positional pipe-delimited fields.
// Positions 1-4 required, 5-6 optional. Empty interior slots valid.
//
// Example:
//   - [C001] Cost decline is structural
//     {0.95|i|E001,E002|2026-03-01|exhaustive|judgment} Context prose.
//     →C002, @~C003

DenseClaim      <- ListMarker ClaimID SP Headline NL
                  ContinuationLines_Dense

ListMarker      <- '- '

ClaimID         <- BoldWrap_Open? '[' ClaimIDInner ']' BoldWrap_Close?
BoldWrap_Open   <- '**'
BoldWrap_Close  <- '**'
ClaimIDInner    <- 'C' DIGIT+ VersionSuffix?
VersionSuffix    <- '-v' DIGIT+

Headline        <- (!NL .)+

ContinuationLines_Dense
  <- INDENT DenseMetadata MetadataTrailing? NL
    (INDENT ContextOrRelation_Dense NL)*

DenseMetadata   <- '{' Confidence '|' ClaimType '|' EvidenceRefList '|'
                  ISODate OptDepth? OptNature? '}'

Confidence      <- DIGIT '.' DIGIT+
ClaimType       <- [orci]
EvidenceRefList <- EvidenceRef (',' EvidenceRef)*
EvidenceRef     <- 'E' DIGIT+
ISODate         <- DIGIT DIGIT DIGIT DIGIT '-' DIGIT DIGIT '-' DIGIT DIGIT
OptDepth        <- '|' Depth?
Depth           <- 'assumed' / 'investigated' / 'exhaustive'
OptNature       <- '|' Nature?
Nature          <- 'judgment' / 'prediction' / 'meta'

// Text after the closing `}' on the metadata line. Per CORE §6.1 / AR-04
// and the reference runner, relations may appear here – immediately after
// the brace or trailing the context prose – using the same three-case
// construction as continuation lines (the not-predicate in ProseSegment
// keeps prose from swallowing relation symbols).
MetadataTrailing <- SP ContextOrRelation_Dense

// Continuation lines after the metadata line. Three cases:
// 1. Line starts with relation symbols: →C002, @~C003
// 2. Line has prose then relations:    ...40 years. →C002, @~C003
// 3. Line is pure prose:               ...no relations here.
//
// Key insight: ContextProse must NOT greedily consume relation symbols.
// ProseSegment uses a not-predicate to stop before RelationSymbol + ClaimIDRef,
// which avoids the PEG greedy-match problem.

ContextOrRelation_Dense
  <- &RelationSymbol DenseRelations          // case 1
    / ProseSegment (SP? DenseRelations)?    // case 2 or 3

// Prose that stops before a relation symbol followed by a claim ID ref.
// This is the critical fix: tilde (~) is only a relation symbol when
// immediately followed by C + digits (per AR-10). The not-predicate
// handles this naturally: !(RelationSymbol ClaimIDRef) only matches
// ~C### but not ~5% or ~approximate.
```

```

ProseSegment      <- ProseChar+
ProseChar         <- !NL !(RelationSymbol ClaimIDRef) .

DenseRelations   <- DenseRelExpr (',' SP? DenseRelExpr)*
DenseRelExpr     <- RelationSymbol ClaimIDRef

// Relation symbols — ordered choice matters: multi-char before single-char
RelationSymbol   <- '⊗~' / '⊗!' / '⊗' / '→' / '←' / '⊘' / '⊗' / '⊗' / '⊗'

ClaimIDRef       <- 'C' DIGIT+ VersionSuffix?
                  / PackRef '#' SectionRef           // cross-pack reference

PackRef          <- IDENT_CHAR+
SectionRef       <- (!(',' / NL / SP) .)+

ContextProseLine <- ContextProse
ContextProse     <- (!NL .)+

// --- Verbose Claim ---
// Metadata in backtick-delimited named fields.
//
// Example:
// - **[C030]** Supply chain concentration
//   `confidence: 0.90 | type: inferred | evidence: E001 | since: 2026-03-01`
//   Context prose lines.
//   `relations: requires C020`

VerboseClaim     <- ListMarker ClaimID SP Headline NL
                  ContinuationLines_Verbose

ContinuationLines_Verbose
  <- INDENT VerboseMetadata NL
    (INDENT VerboseContentLine NL)*

VerboseMetadata <- ``` VerboseFieldList ```

// Fields may occur in any order. CORE §6.2 requires each required field
// exactly once and each optional field at most once; checked after parsing.
VerboseFieldList <- VerboseField (SP? '|' SP? VerboseField)*

VerboseField     <- VF_Confidence / VF_Type / VF_Evidence / VF_Since
                  / VF_Depth / VF_Nature

VF_Confidence    <- 'confidence:' SP Confidence
VF_Type          <- 'type:' SP VerboseTypeName
VF_Evidence      <- 'evidence:' SP VerboseEvidenceList
VF_Since         <- 'since:' SP ISODate
VF_Depth         <- 'depth:' SP Depth
VF_Nature        <- 'nature:' SP Nature

VerboseTypeName  <- 'observed' / 'reported' / 'computed' / 'inferred'

VerboseEvidenceList <- EvidenceRef (',' SP? EvidenceRef)*

VerboseContentLine <- VerboseRelations / ContextProseLine

VerboseRelations <- ``` 'relations:' SP VerboseRelList ```
VerboseRelList   <- VerboseRelExpr (',' SP? VerboseRelExpr)*
VerboseRelExpr   <- VerboseRelName SP ClaimIDRef

VerboseRelName   <- 'supports' / 'contradicts:error' / 'contradicts:tension'
                  / 'contradicts' / 'requires' / 'refines' / 'supersedes'
                  / 'see_also'

// =====
// EVIDENCE REFERENCES (for evidence.md structure)

```

```
// =====
// Not part of claims.md parsing, but included for completeness.
// evidence.md entries are validated as a separate pass.

EvidenceEntry  <- '### ' EvidenceRef (SP? ('-' / '--') SP? EvidenceTitle)? NL
                EvidenceBody

EvidenceTitle  <- (!NL .)+

EvidenceBody   <- EvidenceBlockquote / EvidenceList

// Blockquote format (pipe-delimited bold fields)
EvidenceBlockquote
    <- ('>' SP? EvidenceBQLine NL)+
        EvidenceProse?

EvidenceBQLine <- ('**' EvidenceFieldName ':'**' SP EvidenceFieldValue
                  (SP '|' SP)?)+

// List format (dash-prefixed key-value)
EvidenceList   <- (NL? '- ' EvidenceListField NL)+
                  EvidenceProse?

EvidenceListField <- EvidenceFieldName ':' SP EvidenceFieldValue

EvidenceFieldName <- 'type' / 'Type' / 'source' / 'Source'
                    / 'captured' / 'Captured' / 'Prepared' / 'prepared'
                    / 'SHA-256' / 'sha-256' / 'Notes' / 'notes'
                    / 'reliability' / 'Reliability'
                    / 'credibility' / 'Credibility'
                    / 'kp-content-encoding' / 'KP-Content-Encoding'

EvidenceFieldValue <- (!('|' / NL) .)+

EvidenceProse   <- NL (!(NL '### ') (!NL .)+ NL)*

// =====
// PRIMITIVES
// =====

INDENT          <- ' ' SP*           // 2+ spaces
SP              <- ' '              // single space
NL              <- '\r\n' / '\n'     // newline (CRLF or LF)
BlankLine       <- NL               // a line containing nothing (AR-17)
EOF             <- !.

DIGIT           <- [0-9]
LOWER_ALPHA     <- [a-z]
LOWER_ALNUM     <- [a-z0-9]
IDENT_CHAR      <- [a-z0-9-]

// =====
// SEMANTIC CONSTRAINTS (not expressible in PEG – checked post-parse)
// =====
//
// These constraints MUST be validated after parsing succeeds:
//
// SC-01: Confidence MUST be in range [0.0, 1.0]
// SC-02: All ClaimIDInner values MUST be unique across active and historical records within the
// pack
// SC-03: All EvidenceRef values in active and historical claims MUST have corresponding entries
//         in evidence.md
// SC-04: Supersession chains (ø) MUST be acyclic
// SC-05: All ClaimIDRef targets in relations MUST exist (within pack or
//         as valid cross-pack references)
// SC-06: SpecVersion in RosettaHeader MUST equal 1 for KP:1 conformance
```

```
// SC-07: FrontmatterPack.PackName MUST match PACK.yaml name field
// SC-08: FrontmatterPack.CalVer MUST match PACK.yaml version field
// SC-09: FrontmatterPack.Domain MUST match PACK.yaml domain field
// SC-10: FrontmatterConf.ScaleName MUST match PACK.yaml confidence.scale
// SC-11: Verbose type names MUST map to their single-letter equivalents:
//         observed=o, reported=r, computed=c, inferred=i
// SC-12: When Nature is 'prediction', Confidence MUST be  $\leq 0.95$ 
//         (predictions have irreducible uncertainty; see CORE §12)
// SC-13: All evidence IDs MUST be unique within evidence.md
```

kp-pack.schema.json

Source: <https://github.com/tymofiy/kp/blob/v0.8.5-preview/conformance/grammar/kp-pack.schema.json>

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://kp.dev/schemas/kp-pack-v1.json",
  "title": "KP:1 PACK.yaml Schema",
  "description": "JSON Schema for the PACK.yaml manifest file in a KP:1 knowledge pack. This schema is normative for KP:1 conformance.",
  "type": "object",
  "required": ["name", "version", "domain", "author"],
  "additionalProperties": false,
  "properties": {
    "name": {
      "type": "string",
      "pattern": "^[a-z0-9][a-z0-9-]*$",
      "description": "Pack identifier. Kebab-case, unique within scope."
    },
    "version": {
      "type": "string",
      "pattern": "^[\\d]{4}\\.\\.\\d{2}\\.\\.\\d{2}(-rev\\d+)?$",
      "description": "CalVer snapshot date. Format: YYYY.MM.DD with optional -revN suffix."
    },
    "domain": {
      "type": "string",
      "pattern": "^[a-z0-9][a-z0-9-]*(/[a-z0-9][a-z0-9-]*)*$",
      "description": "Hierarchical domain path. Example: energy/market-analysis"
    },
    "author": {
      "type": "string",
      "minLength": 1,
      "description": "Maintainer name."
    },
    "spec_uri": {
      "type": "string",
      "format": "uri",
      "description": "Optional URL pointing to the KP:1 specification this pack conforms to. Enables cold consumers (especially recipients of sealed .kpack archives) to fetch the spec on demand. Default published location: https://github.com/tymofiy/kp."
    },
    "spec_version": {
      "type": "string",
      "description": "Optional version label of the KP:1 spec this pack conforms to. Typically a tag like 'v0.8.1-preview'. Pairs with spec_uri to pin against a specific revision of the spec for archival packs."
    },
    "kind": {
      "type": "string",
      "enum": ["claim", "definition", "policy", "mixed"],
      "default": "claim",
      "description": "Primary content type of the pack."
    },
    "description": {
      "type": "string",
      "description": "Human-readable summary of the pack's contents."
    },
    "confidence": {
      "type": "object",
      "required": ["scale"],
      "additionalProperties": false,
      "properties": {
        "scale": {
          "type": "string",
          "description": "Confidence scale identifier. Built-in: sherman_kent, simple. Custom scales use 'custom'."
        }
      }
    }
  }
}
```

```

    "normalize": {
      "type": "boolean",
      "description": "Whether claims use 0-1 normalized confidence values."
    },
    "labels": {
      "type": "object",
      "description": "Custom scale band definitions. Required when scale is 'custom'.",
      "additionalProperties": {
        "type": "number",
        "minimum": 0,
        "maximum": 1
      }
    }
  },
  "description": "Confidence scale configuration."
},
"freshness": {
  "type": "string",
  "format": "date",
  "description": "ISO date of last substantive review."
},
"license": {
  "type": "string",
  "description": "Reuse terms. Example: CC-BY-4.0"
},
"sensitivity": {
  "type": "string",
  "enum": ["public", "internal", "confidential", "restricted"],
  "description": "Data handling classification."
},
"visibility": {
  "type": "string",
  "enum": ["private", "shared", "public"],
  "description": "Discovery scope."
},
"channels": {
  "type": "array",
  "items": {
    "type": "string",
    "pattern": "^[a-z][a-z0-9]*(/[a-z][a-z0-9]*)?$"
  },
  "uniqueItems": true,
  "description": "Distribution routing. Spec vocabulary: team, org, public. Custom: namespace/name."
},
"linguistic_epoch": {
  "type": "string",
  "pattern": "^[a-z]{2,3}(-[A-Z]{2})?(-\\d{4})?$",
  "description": "BCP 47 tag with optional year suffix. Example: en-US-2026"
},
"ontology": {
  "type": "string",
  "description": "Knowledge framework declaration (freeform)."
},
"tier": {
  "type": "string",
  "enum": ["hub", "detail", "standalone"],
  "description": "Pack hierarchy role."
},
"display": {
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "short_title": {
      "type": "string",
      "maxLength": 60,
      "description": "Short display label for constrained contexts (sidebar rows, chips). Canonical names of orgs/people may exceed 30 chars; UI is responsible for ellipsis at smaller

```

```

widths."
  },
  "abbreviation": {
    "type": "string",
    "minLength": 2,
    "maxLength": 5,
    "description": "Pronounceable or recognized acronym."
  },
  "tagline": {
    "type": "string",
    "description": "One sentence: what this pack IS."
  },
  "hook": {
    "type": "string",
    "description": "Most compelling sentence in the pack."
  }
},
"description": "Cognitive perception layer for display contexts."
},
"provenance": {
  "type": "object",
  "required": ["author", "role"],
  "additionalProperties": false,
  "properties": {
    "author": {
      "type": "string",
      "description": "Provenance author name."
    },
    "role": {
      "type": "string",
      "enum": ["manufacturer", "independent", "regulator", "academic", "individual", "co-
author"],
      "description": "Author's relationship to the subject matter."
    },
    "reviewed_by": {
      "type": ["string", "null"],
      "description": "Name of the independent reviewer, or null if unreviewed."
    },
    "review_date": {
      "type": ["string", "null"],
      "format": "date",
      "description": "ISO date of review, or null."
    },
    "signed": {
      "type": "boolean",
      "default": false,
      "description": "Whether the pack has a cryptographic signature."
    }
  }
},
"description": "Trust context and attribution."
},
"lifecycle": {
  "type": "object",
  "required": ["type"],
  "additionalProperties": false,
  "properties": {
    "type": {
      "type": "string",
      "enum": ["permanent", "seasonal", "ephemeral", "experience", "working", "incident"],
      "description": "Lifecycle classification. permanent/seasonal/ephemeral are the
archival-duration trio (see LIFECYCLE.md §1); experience (interactive runtime/keepsake pack),
working (active in-progress engagement knowledge), and incident (time-anchored findings record)
are reference-implementation-emitted types reconciled into the enum 2026-06-14."
    },
    "archive_after_days": {
      "type": ["integer", "null"],
      "minimum": 1,
      "description": "Auto-archive threshold in days."
    }
  }
}

```

```

    },
    "archive_policy": {
      "type": "string",
      "enum": ["reconcile", "auto", "manual", "keep"],
      "description": "How archiving is triggered. keep = retain indefinitely, never auto-
archive (stronger than manual); pairs with type: experience for keepsake packs."
    },
    "reconciled_at": {
      "type": ["string", "null"],
      "format": "date-time",
      "description": "ISO timestamp of last reconciliation."
    },
    "orphan_claims": {
      "type": "array",
      "items": { "type": "string" },
      "description": "Claim IDs not yet assigned to a consolidated pack."
    }
  },
  "description": "Pack lifecycle management."
},
"dependencies": {
  "type": "array",
  "items": {
    "type": "object",
    "required": ["uri"],
    "additionalProperties": false,
    "properties": {
      "uri": {
        "type": "string",
        "description": "Pack URI. Formats: github.com/owner/pack.kpack or local:../
path.kpack"
      },
      "version": {
        "type": "string",
        "description": "Version constraint. Example: >=2026.01"
      },
      "kind": {
        "type": "string",
        "enum": ["claim", "definition", "policy", "mixed"],
        "description": "Dependency type signal."
      }
    }
  }
},
"description": "Other packs this pack builds on."
},
"conflicts": {
  "type": "array",
  "items": {
    "type": "object",
    "required": ["pack", "note", "basis"],
    "additionalProperties": false,
    "properties": {
      "pack": {
        "type": "string",
        "description": "Conflicting pack name."
      },
      "note": {
        "type": "string",
        "description": "Free text explanation of the conflict."
      },
      "basis": {
        "type": "string",
        "enum": ["methodology", "data_source", "temporal", "interpretation", "scope"],
        "description": "Nature of the conflict."
      }
    },
    "resolution": {
      "type": "string",
      "enum": ["prefer-this", "prefer-other", "flag-for-review"],

```

```

      "description": "Recommended resolution strategy."
    }
  },
  "description": "Known contradictions with other packs."
},
"evidence_basis": {
  "type": "string",
  "description": "Narrative description of the evidence foundation underpinning this pack."
},
"tags": {
  "type": "array",
  "items": {
    "type": "string",
    "pattern": "^[a-z0-9]+(-[a-z0-9]+)*$"
  },
  "uniqueItems": true,
  "description": "Topical tags for discovery and classification. Kebab-case."
},
"vocabulary": {
  "type": "object",
  "properties": {
    "extends": {
      "type": "string",
      "description": "Base vocabulary this pack extends."
    }
  }
},
"additionalProperties": {
  "oneOf": [
    { "type": "string" },
    { "type": "array", "items": { "type": "string" }, "minItems": 1, "uniqueItems": true }
  ]
},
"description": "Predicate conventions. Values may be a single string or an array of term identifiers."
},
"composition": {
  "type": "string",
  "const": "composition.yaml",
  "default": "composition.yaml",
  "description": "Optional declaration of the canonical composition.yaml file, which must exist when declared. Generic readers do not resolve alternate composition paths."
},
"files": {
  "type": "object",
  "additionalProperties": {
    "type": "string"
  }
},
"description": "Application filename mappings. Canonical generic files remain required; alternate carriers need a declared profile (CORE section 2)."
},
"views": {
  "type": "array",
  "items": {
    "type": "object",
    "required": ["name", "file", "purpose", "display_as"],
    "additionalProperties": false,
    "properties": {
      "name": {
        "type": "string",
        "description": "View identifier."
      },
      "file": {
        "type": "string",
        "description": "Path within pack. Example: views/overview.md"
      },
      "purpose": {
        "type": "string",

```

```

    "description": "AI-facing: when to display this view."
  },
  "display_as": {
    "type": "string",
    "description": "Human-facing: UI tab/section label."
  },
  "hint": {
    "type": "string",
    "description": "Human-facing: ~15 words, what you'll GET from this section."
  },
  "voice": {
    "type": "boolean",
    "description": "Whether this view is designed for spoken delivery."
  },
  "duration": {
    "type": "string",
    "pattern": "^\\d+\\s+(seconds?|minutes?)$",
    "description": "Approximate spoken duration. Example: ~90 seconds"
  },
  "pace": {
    "type": "string",
    "enum": ["brisk", "measured", "deliberate"],
    "description": "Delivery pace hint for spoken delivery."
  },
  "register": {
    "type": "string",
    "enum": ["plain", "curatorial", "technical", "investor"],
    "description": "Voice register – diction and emphasis profile. See VOICE.md §4.2."
  },
  "sub_register": {
    "type": "string",
    "description": "Locale-specific register sub-distinction (e.g. uk-halychyna, fr-quebec). Informational free-form per MULTILINGUAL.md §3.3 – not a normative enum."
  },
  "if": {
    "properties": { "voice": { "const": true } },
    "required": ["voice"]
  },
  "then": {
    "required": ["duration", "pace"]
  },
  "description": "Pre-rendered view declarations."
},
"style": {
  "type": ["string", "null"],
  "default": null,
  "description": "External style system reference."
},
"sub_packs": {
  "type": "array",
  "items": {
    "type": "object",
    "required": ["name", "uri"],
    "additionalProperties": false,
    "properties": {
      "name": {
        "type": "string",
        "description": "Sub-pack name."
      },
      "uri": {
        "type": "string",
        "description": "Sub-pack URI."
      }
    },
    "summary": {
      "type": "string",
      "description": "Brief content description."
    }
  }
}

```

```

    },
    "load_when": {
      "type": "string",
      "description": "Natural language hint for when to load."
    }
  },
  "description": "Hub pack: child detail pack declarations."
},
"consolidation": {
  "type": "object",
  "additionalProperties": true,
  "description": "Knowledge consolidation targets and configuration."
},
"locales": {
  "type": "object",
  "additionalProperties": true,
  "description": "Multilingual support configuration."
},
"notes": {
  "type": "object",
  "additionalProperties": true,
  "description": "AI note-taking metadata."
},
"extensions": {
  "type": "object",
  "additionalProperties": true,
  "description": "Extension lane for experimental or implementation-specific manifest
metadata. Consumers MUST ignore unrecognized extension content."
}
},
"allof": [
  {
    "if": {
      "required": ["sensitivity"],
      "properties": { "sensitivity": { "const": "confidential" } }
    },
    "then": {
      "properties": {
        "channels": {
          "not": {
            "contains": { "enum": ["public", "org"] }
          }
        }
      }
    }
  },
  {
    "if": {
      "required": ["sensitivity"],
      "properties": { "sensitivity": { "const": "restricted" } }
    },
    "then": {
      "properties": {
        "channels": {
          "not": {
            "contains": { "enum": ["public", "org"] }
          }
        }
      }
    }
  },
  {
    "if": {
      "required": ["sensitivity"],
      "properties": { "sensitivity": { "const": "internal" } }
    },
    "then": {

```

```

    "properties": {
      "channels": {
        "not": {
          "contains": { "const": "public" }
        }
      }
    },
    {
      "if": {
        "required": ["tier"],
        "properties": { "tier": { "const": "hub" } }
      },
      "then": {
        "required": ["sub_packs"]
      }
    }
  ]
}

```

kp-composition.schema.json

Source: <https://github.com/tymofiy/kp/blob/v0.8.5-preview/conformance/grammar/kp-composition.schema.json>

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://kp.dev/schemas/kp-composition-v1.json",
  "title": "KP:1 composition.yaml Schema",
  "description": "JSON Schema for the composition.yaml file in a KP:1 composite knowledge pack.
  Defines how composite packs reference standing packs rather than duplicating knowledge.",
  "type": "object",
  "required": [
    "type",
    "pre_load"
  ],
  "additionalProperties": false,
  "properties": {
    "type": {
      "type": "string",
      "enum": [
        "meeting-prep",
        "briefing",
        "research",
        "presentation",
        "selection"
      ],
      "description": "Composition type. Determines which type-specific fields are required."
    },
    "meeting": {
      "type": "object",
      "required": [
        "date",
        "duration",
        "participants"
      ],
      "additionalProperties": false,
      "properties": {
        "date": {
          "type": "string",
          "format": "date-time",
          "description": "Meeting start time with timezone. ISO 8601 datetime."
        },
        "duration": {
          "type": "string",
          "pattern": "^\\d+[mh]$",
          "description": "Expected duration. Examples: 30m, 60m, 2h."
        },
        "participants": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "minItems": 1,
          "description": "People in the meeting."
        },
        "location": {
          "type": "string",
          "description": "Meeting location: video-call, in-person, phone, or address."
        },
        "context": {
          "type": "string",
          "description": "Background context for the AI about this meeting."
        }
      },
      "description": "Meeting metadata. Required when type is meeting-prep."
    },
    "agenda": {
```

```

    "type": "array",
    "items": {
      "type": "object",
      "required": [
        "topic"
      ],
      "additionalProperties": false,
      "properties": {
        "topic": {
          "type": "string",
          "description": "Human-readable agenda item title."
        },
        "reference": {
          "type": "string",
          "description": "Pack name this topic draws from. Uses name from KNOWLEDGE.yaml or
PACK.yaml."
        },
        "focus_claims": {
          "type": "array",
          "items": {
            "type": "string",
            "pattern": "^C[0-9]+(-v[0-9]+)?$"
          },
          "description": "Specific claim IDs from the referenced pack to emphasize."
        },
        "talking_points": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "description": "Key points to cover under this topic."
        }
      }
    },
    "description": "Ordered agenda items. Required when type is meeting-prep."
  },
  "briefing": {
    "type": "object",
    "required": [
      "recipient",
      "purpose",
      "date"
    ],
    "additionalProperties": false,
    "properties": {
      "recipient": {
        "type": "string",
        "description": "Person receiving the briefing."
      },
      "purpose": {
        "type": "string",
        "description": "Purpose of the briefing."
      },
      "date": {
        "type": "string",
        "format": "date",
        "description": "Briefing date. ISO 8601 date."
      },
      "format": {
        "type": "string",
        "enum": [
          "voice",
          "document",
          "presentation"
        ],
        "description": "Intended delivery surface."
      },
      "duration": {

```

```

        "type": "string",
        "pattern": "^\\d+[mh]$",
        "description": "Target delivery time. Examples: 15m, 30m, 1h."
    },
    {
        "description": "Briefing metadata. Required when type is briefing."
    },
    "sections": {
        "type": "array",
        "items": {
            "type": "object",
            "required": [
                "heading"
            ],
            "additionalProperties": false,
            "properties": {
                "heading": {
                    "type": "string",
                    "description": "Section heading for the briefing."
                },
                "reference": {
                    "type": "string",
                    "description": "Pack name this section draws from."
                },
                "focus_claims": {
                    "type": "array",
                    "items": {
                        "type": "string",
                        "pattern": "^C[0-9]+(-v[0-9]+)?$"
                    },
                    "description": "Specific claim IDs from the referenced pack to emphasize."
                },
                "narrative_hint": {
                    "type": "string",
                    "description": "Guidance for the voice or document renderer on framing."
                }
            }
        }
    },
    "description": "Ordered briefing sections. Required when type is briefing."
},
"research": {
    "type": "object",
    "required": [
        "question",
        "date"
    ],
    "additionalProperties": false,
    "properties": {
        "question": {
            "type": "string",
            "description": "The research question being investigated."
        },
        "hypothesis": {
            "type": "string",
            "description": "Working hypothesis to test against sources."
        },
        "date": {
            "type": "string",
            "format": "date",
            "description": "Research composition date. ISO 8601 date."
        },
        "depth": {
            "type": "string",
            "enum": [
                "quick",
                "thorough",
                "exhaustive"
            ]
        }
    }
},

```

```

        "description": "Level of analysis expected."
      },
    },
    "description": "Research metadata. Required when type is research."
  },
  "sources": {
    "type": "array",
    "items": {
      "type": "object",
      "required": [
        "pack"
      ],
      "additionalProperties": false,
      "properties": {
        "pack": {
          "type": "string",
          "description": "Pack name used as a research source."
        },
        "relevance": {
          "type": "string",
          "description": "Why this pack matters to the research question."
        },
        "focus_claims": {
          "type": "array",
          "items": {
            "type": "string",
            "pattern": "^C[0-9]+(-v[0-9]+)?$"
          },
          "description": "Specific claim IDs from the source pack to emphasize."
        }
      }
    }
  },
  "description": "Research source packs with relevance annotations. Required when type is research."
},
"presentation": {
  "type": "object",
  "required": [
    "title",
    "date"
  ],
  "additionalProperties": false,
  "properties": {
    "title": {
      "type": "string",
      "description": "Presentation title."
    },
    "audience": {
      "type": "string",
      "description": "Intended audience description."
    },
    "date": {
      "type": "string",
      "format": "date",
      "description": "Presentation date. ISO 8601 date."
    },
    "slide_count": {
      "type": "integer",
      "minimum": 1,
      "description": "Target number of slides."
    },
    "style": {
      "type": "string",
      "description": "Visual style reference for consistency."
    }
  }
},
"description": "Presentation metadata. Required when type is presentation."
},

```

```

"slides": {
  "type": "array",
  "items": {
    "type": "object",
    "required": [
      "title"
    ],
    "additionalProperties": false,
    "properties": {
      "title": {
        "type": "string",
        "description": "Slide title."
      },
      "reference": {
        "type": "string",
        "description": "Pack name this slide draws from."
      },
      "focus_claims": {
        "type": "array",
        "items": {
          "type": "string",
          "pattern": "^C[0-9]+(-v[0-9]+)?$"
        },
        "description": "Specific claim IDs to feature on this slide."
      },
      "visual_hint": {
        "type": "string",
        "description": "Guidance for the presentation renderer on visuals."
      }
    }
  },
  "description": "Ordered slide definitions. Required when type is presentation."
},
"selection": {
  "type": "object",
  "required": [
    "purpose",
    "collection",
    "scope"
  ],
  "additionalProperties": false,
  "properties": {
    "purpose": {
      "type": "string",
      "description": "Why this selection exists. Examples: exhibition, loan, insurance schedule, valuation.",
      "minLength": 1,
      "pattern": "\\S"
    },
    "collection": {
      "type": "string",
      "description": "Name of the single collection the members are drawn from.",
      "minLength": 1,
      "pattern": "\\S"
    },
    "scope": {
      "type": "string",
      "description": "What the selection covers, and by implication what it excludes.",
      "minLength": 1,
      "pattern": "\\S"
    },
    "prepared_for": {
      "type": "string",
      "description": "Party the selection was prepared for, when there is one."
    },
    "date": {
      "type": "string",
      "format": "date",

```

```

        "description": "Date the selection was prepared. ISO 8601 date."
    },
    "description": "Selection metadata. Required when type is selection."
},
"members": {
    "type": "array",
    "items": {
        "type": "object",
        "required": [
            "reference"
        ],
        "additionalProperties": false,
        "properties": {
            "reference": {
                "type": "string",
                "description": "Pack name of a member item. The member remains held by its
collection; a selection asserts no membership.",
                "minLength": 1,
                "pattern": "\\S"
            },
            "focus_claims": {
                "type": "array",
                "items": {
                    "type": "string",
                    "pattern": "^C[0-9]+(-v[0-9]+)?$"
                },
                "description": "Specific claim IDs relevant to this selection."
            },
            "note": {
                "type": "string",
                "description": "Why this member is in the selection, when that is not obvious."
            }
        }
    },
    "description": "Members of the selection. Required when type is selection."
},
"pre_load": {
    "type": "array",
    "items": {
        "type": "string"
    },
    "description": "Pack names to load into context before the session begins."
},
"on_demand": {
    "type": "array",
    "items": {
        "type": "string"
    },
    "description": "Pack names available for loading when topics arise during the session."
},
"allOf": [
    {
        "if": {
            "required": [
                "type"
            ],
            "properties": {
                "type": {
                    "const": "meeting-prep"
                }
            }
        },
        "then": {
            "required": [
                "meeting",
                "agenda"
            ]
        }
    }
]

```

```

    ]
  }
},
{
  "if": {
    "required": [
      "type"
    ],
    "properties": {
      "type": {
        "const": "briefing"
      }
    }
  },
  "then": {
    "required": [
      "briefing",
      "sections"
    ]
  }
},
{
  "if": {
    "required": [
      "type"
    ],
    "properties": {
      "type": {
        "const": "research"
      }
    }
  },
  "then": {
    "required": [
      "research",
      "sources"
    ]
  }
},
{
  "if": {
    "required": [
      "type"
    ],
    "properties": {
      "type": {
        "const": "presentation"
      }
    }
  },
  "then": {
    "required": [
      "presentation",
      "slides"
    ]
  }
},
{
  "if": {
    "required": [
      "type"
    ],
    "properties": {
      "type": {
        "const": "selection"
      }
    }
  }
},

```

```
    "then": {  
      "required": [  
        "selection",  
        "members"  
      ]  
    }  
  }  
]  
}
```

kp-signatures.schema.json

Source: <https://github.com/tymofiy/kp/blob/v0.8.5-preview/conformance/grammar/kp-signatures.schema.json>

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://kp.dev/schemas/kp-signatures-v1.json",
  "title": "KP:1 signatures.yaml Schema",
  "description": "JSON Schema for the signatures.yaml integrity manifest in a KP:1 knowledge pack. Aligned with ARCHIVE.md §4 (v0.7.3+).",
  "type": "object",
  "required": ["algorithm", "pack_hash", "files", "sealed_at", "sealed_by"],
  "additionalProperties": false,
  "properties": {
    "algorithm": {
      "type": "string",
      "enum": ["SHA-256"],
      "description": "Hash algorithm. Currently only SHA-256 (future-proofed for upgrades).",
    },
    "seal_version": {
      "type": "string",
      "enum": ["2"],
      "description": "ARCHIVE.md \u00a74.2 signing-payload version. ABSENT means v1, whose payload does not cover pack_id. Itself a signed field, so it cannot be stripped to select a weaker payload.",
    },
    "pack_id": {
      "type": "string",
      "description": "Stable pack identity across versions. OPTIONAL in the schema for backward compatibility with pre-documentation archives; new archives SHOULD include it.",
    },
    "pack_name": {
      "type": "string",
      "description": "Optional self-describing pack name (equal to PACK.yaml name). Enables receipts and relay indexing without a second file read.",
    },
    "pack_version": {
      "type": "string",
      "description": "Optional self-describing pack version (equal to PACK.yaml version). Enables receipts without a second file read.",
    },
    "pack_hash": {
      "type": "string",
      "pattern": "^[a-f0-9]{64}$",
      "description": "Pack content hash computed per ARCHIVE.md §3 (lowercase hex SHA-256 digest, 64 characters).",
    },
    "files": {
      "type": "object",
      "additionalProperties": {
        "type": "string",
        "pattern": "^[a-f0-9]{64}$"
      },
      "propertyNames": {
        "not": { "const": "signatures.yaml" }
      },
      "minProperties": 1,
      "description": "Map of NFC-normalized relative file paths (with / separator) to their lowercase hex SHA-256 digests. MUST NOT include the root-level signatures.yaml itself (enforced via propertyNames).",
    },
    "sealed_at": {
      "type": "string",
      "format": "date-time",
      "description": "ISO 8601 UTC timestamp of sealing (e.g., 2026-04-12T14:30:00Z).",
    },
    "sealed_by": {
```

```

    "type": "string",
    "description": "Identifier of the sealing system (e.g., capture-client, analysis-service).",
  },
  "parent": {
    "type": "object",
    "required": ["version", "pack_hash"],
    "additionalProperties": false,
    "properties": {
      "version": {
        "type": "string",
        "description": "Parent pack's version string from its PACK.yaml."
      },
      "pack_hash": {
        "type": "string",
        "pattern": "^[a-f0-9]{64}$",
        "description": "Parent pack's pack_hash (lowercase hex)."
      },
      "merge_parents": {
        "type": "array",
        "items": {
          "type": "object",
          "required": ["version", "pack_hash"],
          "additionalProperties": false,
          "properties": {
            "version": {
              "type": "string",
              "description": "Additional parent's version string."
            },
            "pack_hash": {
              "type": "string",
              "pattern": "^[a-f0-9]{64}$",
              "description": "Additional parent's pack_hash (lowercase hex)."
            }
          }
        }
      },
      "description": "Additional parent references for branch-merge lineage. OPTIONAL; absent in single-parent archives."
    }
  },
  "description": "Parent version reference. Absent for v1 of a pack; present for v2 and later."
},
"signature": {
  "type": "object",
  "required": ["method", "value", "key_id"],
  "additionalProperties": false,
  "properties": {
    "method": {
      "type": "string",
      "enum": ["ed25519", "hmac-sha256", "rsa-pss-sha256"],
      "description": "Signing method identifier."
    },
    "value": {
      "type": "string",
      "description": "Base64-encoded (standard, padded) signature over the signing payload defined in ARCHIVE.md §4.2."
    },
    "key_id": {
      "type": "string",
      "description": "Opaque key identifier for locating the verification key."
    }
  },
  "description": "Optional digital signature. When present, all three sub-fields are REQUIRED."
}
}

```

References and source documents

- [spec/CORE.md](#)
- [spec/SPEC.md](#)
- [spec/COMPATIBILITY.md](#)
- [spec/AUTHORING.md](#)
- [spec/VOICE.md](#)
- [spec/COMPOSITION.md](#)
- [spec/LIFECYCLE.md](#)
- [spec/MULTILINGUAL.md](#)
- [spec/ORGANIZATION.md](#)
- [spec/CONSISTENCY.md](#)
- [spec/CONVENTIONS.md](#)
- [spec/STORAGE.md](#)
- [spec/BUNDLE.md](#)
- [spec/ARCHIVE.md](#)
- [spec/DEFINITIONS.md](#)
- [spec/EXTENSIONS.md](#)
- [spec/NOTES.md](#)
- [spec/PLAYBACK.md](#)
- [spec/RECONCILIATION.md](#)
- [spec/RATIONALE.md](#)
- [spec/MAPPING.md](#)
- [spec/INDEPENDENCE.md](#)
- [spec/README.md](#)
- [spec/CHANGELOG.md](#)
- [conformance/grammar/kp-claims.peg](#)
- [conformance/grammar/kp-pack.schema.json](#)
- [conformance/grammar/kp-composition.schema.json](#)
- [conformance/grammar/kp-signatures.schema.json](#)
- [Citation metadata](#)
- [Version archive](#)
- [Acknowledgments](#)