The Context Layer Standard

Version 1.0 · August 2026 · Jack Horscraft

What this is

An open specification for writing a business down in a structure that both a person and an AI system can work from.

It is deliberately unimpressive. Plain folders. Plain text. No platform, no schema language, no vendor. The whole thing can be read by someone who has never used an AI tool, and that is a design requirement rather than a concession.

It exists because the same problem keeps appearing. A company buys AI capability and gets generic output, because the model has never been told how that specific business works. The gap is not prompting and it is not tooling. It is that the company's operating knowledge — how it prices, decides, escalates, refuses, and hands over — has never been written anywhere a system can read.

This document specifies what "written down properly" means, precisely enough that two people implementing it separately would produce something recognisably the same.

What this is not

- **Not a data schema.** It does not describe your customer records, your finance system, or your database. It describes the judgment that sits above them.
- **Not an ontology.** No taxonomy to learn, no controlled vocabulary, no graph.
- **Not a RAG configuration.** How you index or retrieve is out of scope and deliberately so.
- **Not a compliance framework.** It will help you meet documentation obligations. It does not claim to satisfy any of them.
- **Not vendor-specific.** Nothing here requires a particular model, tool, or provider. If it did, it would fail its own first principle.
- **Not new.** Anthropic's Skills, Google's Open Knowledge Format, and every serious agent framework have converged on this shape from different directions. This document writes the convergence down in plain English so a business can implement it without a research team.

Terminology

The key words **MUST**, **MUST NOT**, **SHOULD**, **SHOULD NOT**, and **MAY** are used as in RFC 2119.

- **Context root** — the single top-level folder holding the layer.
- **Entry file** — the file at the context root that tells any reader, human or machine, what is here and how to use it.
- **Record** — one file holding one canonical thing.
- **Job** — a working system built on the layer that does something repeatable.
- **Owner** — the named human accountable for a record's accuracy.

Part 1 — Principles

These are the constraints everything else derives from. An implementation that satisfies the letter of Parts 2–7 while breaking one of these is not conformant.

P1 — Plain text, plain structure. The layer MUST be plain text files in nested folders, in a format readable without special software. Markdown is the recommended default. A business MUST be able to open the whole thing in any text editor and read it.

P2 — Human-first. Every record MUST be useful to a person with no AI involved. If a record is only legible to a machine, it is a configuration file, not a context record, and it belongs elsewhere.

P3 — One canonical home. Every fact MUST live in exactly one record. Other records reference it. Restating a fact in two places creates two versions of it, and the second one to drift is the one that gets quoted.

P4 — Portable by construction. The layer MUST NOT depend on any vendor, platform, or model to remain readable and useful. A business that stops working with its implementer, or switches AI provider entirely, keeps everything and loses nothing.

P5 — Boundaries are explicit. What a system may read, may propose, and must never touch MUST be stated in the layer itself, not held as a convention or a verbal understanding.

P6 — Owned by a named person. Every record has one accountable human. A layer with no owners decays into a snapshot of the month it was built.

P7 — Decay is assumed. The specification assumes records go stale and provides for it. Any implementation that has no answer to "how do you know this is still true" is incomplete.

P8 — The structure is the business's own. The layer MUST be organised in the business's own language and around its own concepts. This specification defines the *properties* a structure must hold (Part 2.3), never the names. A structure copied from a template imposes a permanent

translation tax on everyone who files into it, and it is the reason most documentation projects are abandoned within a quarter. The right structure is the one people would have drawn themselves.

Part 2 — Structure

2.1 The context root

An implementation MUST have exactly one context root. It MAY live in the business's existing document store (SharePoint, Google Drive, a Git repository, a synced folder) provided that store preserves plain files and folder hierarchy.

2.2 There is no prescribed folder list

This specification does not tell you what your folders are called, and any version of it that did would be wrong.

A structure works because it mirrors how a particular business actually thinks about itself. A structure imported from a template gets used for a fortnight and abandoned, because every time someone files something they have to translate from their own language into someone else's. The translation cost is small and constant, which is the worst kind, and it is why most documentation projects die quietly.

The other reason is mechanical. A system navigating the layer finds its way using the map in the entry file (Part 3), not by recognising folder names it has seen before. Nothing downstream depends on your folders matching anyone else's. That is precisely what makes the layer portable and what makes this specification implementable by a 4,000-person manufacturer and a sole trader without either of them contorting.

What is normative is that a structure exists, is declared, and holds the seven properties below. What it is called is yours.

2.3 Required properties of a structure

A conformant structure MUST satisfy all seven.

S1 — Declared. The structure MUST be mapped in the entry file: what each area holds, and which area answers which kind of question. An undeclared structure is a folder tree, not a layer.

S2 — Derived from the business. Area names MUST come from the language the business already uses for itself. If the team says "jobs" rather than "projects", or "matters", "sites", "accounts", "runs", "cases" — that is the correct name. Renaming a business's own concepts to match a standard is a defect, not compliance.

S3 — Separable. Each area MUST have a stateable scope such that any given record has one obvious home. If two areas both plausibly hold the same record, they are one area with an unresolved name, and P3 will break there first.

S4 — Shallow. Working depth SHOULD NOT exceed three levels below the context root. Depth is where structures go to die: past three levels, nobody files correctly and nobody browses.

S5 — Archive separated. Superseded material MUST be held separately from working material, by whatever name (`archive/`, `retired/`, `old/`). See 2.5.

S6 — Consistently named. Naming MUST follow one convention applied uniformly. Which convention is the business's choice. Recommended: lowercase, hyphen-separated, descriptive of content rather than date or author — `pricing-awkward-jobs.md` rather than `Pricing v3 FINAL (JH).md`. Recommended, not required; a business with an established convention that people actually follow SHOULD keep it, because an inconsistently-applied better convention is worse than a consistently-applied adequate one.

S7 — Complete for the questions asked. Every recurring question the business asks itself MUST have an area it would obviously live in. Gaps are legitimate and expected — an area that exists and is empty is an honest statement that nothing is written there yet, and it surfaces as a finding rather than hiding as an absence.

2.4 How to derive the structure

The structure is discovered, not chosen. The method, which is the same one used in a Context Assessment:

1. **List the questions.** Over a fortnight, collect the questions that get asked of the linchpin — in person, over Slack, by email. Aim for forty or more. Do not tidy them.
2. **Cluster them.** Group by what kind of thing would answer them, not by department. "How do we price this?" and "do we take this client?" cluster together as judgment; "how do we onboard?" and "how do we invoice?" cluster as procedure.
3. **Name the clusters in their words.** Use whatever the business calls that cluster when talking normally. If there is no existing word, the cluster is probably two clusters.
4. **Test with a stranger.** Give someone unfamiliar the area names and three real questions. If they can say which area each belongs in without help, the structure holds. If they hesitate, S3 has failed somewhere and the boundary needs naming.
5. **Stop.** Six to ten top-level areas is typical. Fewer than four usually means the clusters are too coarse to file into. More than twelve means departments have been recreated, which reproduces the silos rather than crossing them.

2.5 The archive

Superseded records MUST be moved to the archive area, and that area MUST be treated as immutable and out of working scope. Records are never edited there and never read as current. A system operating on the layer MUST NOT treat archived material as authoritative.

Deletion is not the alternative. The reasoning in a superseded decision is often the most valuable thing in the layer, and the question "why did we stop doing it that way" has no other answer.

2.6 Worked examples — illustrative only

None of the following is normative. They are three real shapes, showing the same seven properties expressed in three businesses' own language.

A 60-person surveying practice

```
context/
├── CONTEXT.md
├── the-practice/     what we do, our standards, how we price
├── judgment/         the calls only a partner makes, and why
├── how-we-work/      procedures, with the judgment marked
├── clients/          per-client history and preferences
├── instructions/     recurring job types and their gotchas
├── whats-allowed/    permissions, classification, data rules
├── systems/          the jobs running on this layer
└── superseded/
```

A specialist manufacturer, 200 staff

```
context/
├── CONTEXT.md
├── products/
├── process/          line procedures, with judgment calls marked
├── why-we-do-it-this-way/   decision records — their name, kept
├── people-and-roles/
├── customers-and-suppliers/
├── safety-and-limits/ permissions, classification, what never leaves site
├── automations/
└── archive/
```

A solo consultant

```
context/
├── CONTEXT.md
├── business/         offers, pricing, positioning, voice
├── decisions/
├── clients/
├── how-i-work/       procedures and standards
├── boundaries/
├── jobs/
└── archive/
```

Note what is constant across all three: an entry file, a home for judgment separate from procedure, a place for relationships, an explicit boundaries area, a home for the working systems, and a separated archive. Note what is not constant: any of the names.

A general-purpose starting point, for a business with no strong existing language of its own — a default to depart from rather than a target to hit: `business/`, `decisions/`, `procedures/`, `people/`, `relationships/`, `boundaries/`, `jobs/`, `archive/`. If three of these get renamed during implementation, the implementation is going well.

Part 3 — The entry file

One file at the context root is the single most important file in the implementation. It is what any reader — a new starter, a consultant, or a model — reads first to know where to look and what is permitted.

An entry file MUST exist at the root. Its name SHOULD be `CONTEXT.md`; it MAY differ where a platform or an existing tool dictates otherwise. This is the one filename the specification has an opinion about, and only because a reader arriving cold needs somewhere obvious to start.

It MUST contain:

1. **What this business is** — two or three sentences. Not marketing copy.
2. **How to navigate** — the declared structure per Part 2.3 S1: every area by its local name, what it holds, and the routing rule for which to read for which kind of question. This is the only place the structure is defined, which is why nothing downstream depends on the names.
3. **Load order** — which records should be read before any work is done (typically boundaries, voice or standards, and current priorities).
4. **What is authoritative** — a statement that records are canonical, the archive is not, and where to go when two records disagree.
5. **Boundaries summary** — a pointer to `boundaries/`, plus the two or three absolute prohibitions stated inline so they cannot be missed.
6. **Owners** — who is accountable for the layer overall, and how to reach them.

7. **Last reviewed** — a date.

It MUST NOT exceed what a person will actually read. Two pages is a working ceiling. An entry file that becomes a document in its own right has failed; it is a map, not a territory.

Part 4 — Records

4.1 Universal requirements

Every record MUST carry frontmatter with:

```
title:          human-readable name
owns:           the single thing this record is canonical for
owner:          named accountable person
updated:        YYYY-MM-DD
review:         interval (e.g. 6-months) or "on-change"
classification: open | internal | restricted
status:         live | superseded
```

The `owns:` line is load-bearing. Writing it forces the author to state what this record is the single source of, which is what makes P3 enforceable rather than aspirational. A record that cannot state what it owns is usually two records.

4.2 Decision records

The highest-value record type, and the one almost every business is missing entirely. A decision record captures a choice the business made and, critically, the reasoning that would make it change.

Required fields:

- **Decision** — one sentence, in the active voice.
- **Date and owner.**
- **Trigger** — what prompted the decision.
- **Options considered** — including the ones rejected, briefly.
- **Reasoning** — why this one won.
- **What would change this** — the conditions under which the decision should be revisited or does not apply.
- **Status** — live or superseded, with a pointer to its replacement.

"What would change this" is mandatory and is the field that makes the record usable. A rule without its boundary conditions gets applied everywhere, including the cases it was never meant to cover. This is the single most common failure in AI implementations built on documented process: the system knows the rule and has no idea when to stop applying it.

4.3 Procedure records

How a repeated piece of work gets done.

Required fields:

- **Trigger** — what starts this.
- **Owner** — who is accountable for the outcome.
- **Steps** — the mechanical sequence.
- **Judgment calls** — the points in the sequence where a human decides, what they are weighing, and what a good and bad call look like.
- **Failure modes** — what goes wrong, how it is spotted.
- **Escalation** — who is told, when.

"Judgment calls" is mandatory. Most documented processes record the mechanical steps and omit the expertise, which is exactly backwards: the steps are the part anyone could work out, and the judgment is the part that lives in one person's head. A procedure record without this field documents the easy half.

4.4 Other record types

Named by function, not by folder — each lives in whichever area the business's own structure assigns it (Part 2).

- **Business records** — offers, pricing logic, positioning, standards, voice. One record per separable thing.
- **People records** — roles, what each role owns and decides. Not personnel files: personal data SHOULD be excluded and MUST be classified `restricted` where present.
- **Relationship records** — per client, supplier, or partner: what was agreed, how they prefer to work, the history that would take an hour to explain.

4.5 Linking

Records SHOULD link to related records using a consistent syntax. A link to a record that does not exist yet is legitimate and useful — it marks a known gap rather than an error.

Part 5 — Boundaries

A boundaries area MUST exist, by whatever local name, and MUST be non-empty. This is the part that gets an implementation past legal, IT, and the person in the room who is quietly worried. It is the one area whose *existence* is required even though its name is not.

5.1 Classification

Every record carries one of:

- **open** — could be published without harm.
- **internal** — fine for staff, not for outside the business.
- **restricted** — limited named access; MUST NOT be included in any system context by default.
- **excluded** — content that MUST NOT enter the layer at all. The exclusion itself is recorded; the content is not.

5.2 Permissions

For each folder, the layer MUST state what an automated system may do:

- **read** — may be used to answer.
- **propose** — may draft changes for human approval.
- **write** — may change without review. This SHOULD be empty in most implementations, and an implementation that grants blanket write access to a system is not conformant with P5's intent.

5.3 Data handling

`boundaries/` MUST state where data may be processed — third-party API, private cloud tenancy, or locally hosted — and any categories of information that must never leave the premises. Where a business has this constraint, the layer's design is unaffected: the same structure runs against a locally hosted model. The specification is deliberately silent on which model, and that silence is the point.

Part 6 — Jobs

A job is a working system built on the layer. The layer without jobs is documentation; jobs without a layer are the failed pilot.

Each job MUST be specified as a record — in whichever area the structure assigns to working systems — with:

- **What it does** — one sentence.
- **What it reads** — the specific records it depends on.
- **Trigger** — manual, scheduled, or event-driven.
- **Human gate** — what a person confirms before anything leaves the business or changes a system of record. A job that acts externally with no gate MUST be documented as such and MUST be explicitly signed off by a named owner.
- **Failure behaviour** — what it does when it does not know. "Says it does not know" is a valid and preferred answer.
- **Owner.**

A conformant implementation SHOULD deliver two or three jobs rather than one or ten. One does not prove the layer generalises. Ten is a delivery programme, and it will not survive its first month of maintenance.

Part 7 — Ownership, decay, and handover

7.1 Freshness

Every record carries `updated` and `review`. A record past its review interval is **stale**. Stale records remain readable and MUST NOT be silently deleted, but:

- Systems operating on the layer SHOULD state staleness when answering from a stale record.
- A conformant implementation MUST have a mechanism — automated or a diarised human pass — that surfaces stale records on a stated cadence.

This is the requirement most implementations skip and the reason most of them are worthless within a year.

7.2 Handover

An implementation is not complete until handed over. The handover MUST be written and MUST name:

- The person who owns the layer.
- How to add a record, in steps, written for someone who was not involved in the build.
- The review cadence and who runs it.
- What to do when two records disagree.
- How to extend a job or commission a new one.

A verbal handover is not a handover. If the only person who can extend the layer is the person who built it, the business has swapped one dependency for another, which is the exact problem the layer exists to solve.

Part 8 — Conformance

Three levels. Each is checkable, and a claim of conformance names its level.

Level 1 — Legible

- Context root exists, with a structure satisfying all seven properties in Part 2.3.
- Entry file present and complete per Part 3, including the declared structure map.
- All records carry required frontmatter.
- **The stranger test passes:** a person unfamiliar with the business can find the answer to a real operational question using only the layer, and can correctly place three new records without being told where they go.

Level 2 — Operable

Everything in Level 1, plus:

- Decision records exist for the recurring judgment calls the business identified as key-person dependent.
- Procedure records exist for its highest-volume repeated work, including the judgment-calls field.
- `boundaries/` complete: classification applied, permissions stated, data handling declared.
- At least two jobs specified and running.

Level 3 — Owned

Everything in Level 2, plus:

- Every record has a named owner and a review interval.
- A staleness mechanism is running and has completed at least one cycle.
- Written handover complete, and a named person other than the implementer has added a record unaided.
- **Provider portability demonstrated:** the layer has been exercised against at least two different model providers with no change to its contents.

Level 3's final criterion is the one that separates this from a documentation exercise. If switching provider requires rewriting the layer, the layer was never portable and P4 was decorative.

Part 9 — Readiness dimensions

For assessing a business that has not implemented the standard, eight dimensions derive directly from the parts above. Each is scored independently.

#	DIMENSION	QUESTION IT ANSWERS	STANDARD REFERENCE
1	Decision capture	Are repeated judgment calls written down with their reasoning?	Part 4.2
2	Procedure documentation	Is the expertise recorded, or only the mechanical steps?	Part 4.3
3	Onboarding	How much of a new starter's first month happens verbally?	Parts 3, 7.2
4	Document reuse	How often is the same artefact rebuilt from scratch?	Part 4
5	Tool sprawl	Is knowledge fragmented across systems nobody can query together?	Part 2
6	Permission clarity	Does anyone know what a system is allowed to touch?	Part 5.2
7	Data boundaries	Is there a stated position on what may leave the premises?	Part 5.3
8	Handover risk	If the linchpin left tomorrow, what leaves with them?	Parts 6, 7

Part 10 — Versioning

This document is versioned. Implementations state the version they conform to.

- **1.0** (August 2026) — first public release.

Breaking changes increment the major version. Clarifications and added examples increment the minor. Implementations conformant to a prior major version remain conformant to that version; nothing here obliges anyone to migrate.

A note on why this is free

Publishing a specification and selling implementation against it is an old and honest arrangement. The spec is more useful to me widely adopted than closely held, and a business that reads this and implements it alone has my genuine blessing — that outcome is better for them than a consultant they didn't need.

What it costs to implement well is time and interrogation. The hard part was never the file format. It is getting what one person knows out of their head and into a form that survives them, and that is a conversation, not a template.

Citing this document

> Horscraft, J. (2026). *The Context Layer Standard*, version 1.0. Available at https://www.thesecondlayer.co.uk/context/standard

Corrections, disagreements and implementation reports are all welcome: jack@thesecondlayer.co.uk