Canon Ontology Specification (COS-1)

A Reusable House Specification for Rendering a Canon as a Structured Object

Abstract

This document provides a reusable house specification for rendering a canon as a structured object. COS-1 is not a theory paper specification, not a source-review format, and not a replacement for local ontology documents such as Argument Ontology Specifications. Its purpose is to define what must be made explicit when a body of related theory-objects, papers, field proposals, methodological specifications, and formal frameworks has grown large enough to require representation as a governed whole. A canon, in this specification, is not merely a collection of texts. It is a bounded, revisable, internally related architecture of objects, relations, statuses, entry points, and drift rules. COS-1 therefore requires that a canon be rendered in a way that makes recoverable its identity, purpose, object registry, relation structure, layering, interfaces, public and internal boundaries, stability conditions, unresolved residues, and rules of expansion or revision. The specification below states the purpose of a canon ontology, the distinction between canon and stack, the core writing principles governing a canon specification, the required functions such a document must perform, the default structure through which those functions are usually carried, the evaluation criteria by which a canon rendering should be judged, and the main failure modes a canon ontology must avoid.

1. Purpose

COS-1 is a reusable house specification for a *canon ontology*. It is meant for projects that have grown beyond one paper, one theory-object, or one field proposal, and now require a higher-order artifact capable of rendering the larger architecture as a bounded whole.

A canon ontology exists to do one or more of the following:

- define what belongs to a canon and what does not,
- identify the kinds of objects the canon contains,
- specify how those objects relate to one another,
- clarify which parts of the canon are foundational, derivative, companion, experimental, or frozen,
- state how the canon may be entered, extended, revised, and governed,
- make visible the unresolved tensions and residues of the whole architecture.

A canon ontology is not merely an index, a table of contents, or a bibliography. It is a structural rendering of the canon as an object in its own right.

2. What a Canon Is

For purposes of COS-1, a *canon* is a bounded, revisable, internally related architecture of theory-objects, papers, specifications, field proposals, frameworks, and related artifacts maintained so that local artifacts can remain self-standing while still participating in a larger coherent structure.

A canon is therefore:

- more than a collection,
- more than a stack ordering alone,
- more than a reading list,
- and more than an archive of drafts.

A canon has at minimum:

- a bounded identity,
- an internal object registry,
- relation types among those objects,
- some account of order or layering,
- some distinction between stable and unstable regions,
- and some governance over drift, expansion, and revision.

3. Canon vs Stack

COS-1 distinguishes *canon* from *stack*.

3.1. Canon

A canon is the broader whole. It contains the objects, classes, statuses, interfaces, branches, and governance rules that together define the architecture of a project.

3.2. Stack

A stack is one important relation or ordering structure within a canon. It typically marks dependency, upstream/downstream position, inheritance, or explanatory layering among some subset of the canon's objects.

3.3. Core Distinction

A canon may contain:

- upstream and downstream theory-objects,
- side branches,
- field proposals,
- paper-class specifications,
- formal frameworks,
- source-review specifications,
- frozen papers,
- working objects,
- and internal maintenance documents.

A stack is one structured directional relation among such objects, but it does not exhaust the canon.

Compressed rule. The stack is part of the canon's architecture; it is not identical to the canon.

4. What a Canon Ontology Is

A Canon Ontology Specification is a reader-facing structural artifact whose task is to make the canon recoverable as a governed whole.

At minimum, a successful canon ontology makes recoverable:

- the canon's identity and purpose,
- the kinds of objects it contains,
- the relations among those objects,
- the major layers or regions of the canon,
- the valid entry points into the canon,
- the stability and drift conditions of its parts,
- the residues or unresolved tensions of the canon as a whole,
- and the rules by which the canon may be extended or revised.

COS-1 therefore treats a canon ontology as more than documentation. It is part of the canon's own maintenance and intelligibility.

5. Presentation vs Internal Canon Management

COS-1 is a specification for rendering the canon in intelligible structural form. It is not identical to every internal maintenance system the project may use.

A canon may internally employ:

- ledgers,
- object registries,
- dependency graphs,
- revision logs,
- branch policies,
- source extraction systems,
- or private experimental notes.

Those may be indispensable. But the canon ontology itself should normally translate that internal machinery into a form that makes the canon legible without requiring the reader to inhabit the entire maintenance apparatus.

5.1. Core Principle

A canon ontology should make the canon structurally intelligible in the most direct language adequate to the task, not in the most internally overloaded maintenance language available.

5.2. Default Rule

Unless the canon ontology is itself designed as an internal-only maintenance artifact, it should:

- distinguish public and internal regions of the canon,
- surface only the maintenance detail needed for structural intelligibility,
- avoid collapsing into revision-log prose,
- and avoid becoming a raw database dump.

6. Core Writing Principles

6.1. Recoverability Principle

A canon ontology must make the architecture of the canon recoverable. A competent reader should be able to say what the canon is, what it contains, how it is organized, and where its unresolved tensions remain.

6.2. Boundedness Principle

A canon ontology must define the canon's boundary. It should identify what belongs, what does not, and what remains merely adjacent, provisional, or external.

6.3. Typed-Object Principle

The canon should not be rendered as a flat list of texts. Its objects should be typed where possible: theory paper, theory-object, field proposal, house specification, formal framework, source-review specification, companion object, and so on.

6.4. Relation Principle

A canon ontology must specify relations among objects rather than merely naming them side by side.

6.5. Layer Principle

A canon ontology should make visible the larger layering or regional structure of the canon: upstream, middle-layer, downstream, methodological, formal, public-facing, internal, or equivalent.

6.6. Entry Principle

A canon ontology should show how the canon may be entered. Not every reader or user approaches the architecture from the same point.

6.7. Drift Principle

A canon ontology must distinguish stable from unstable regions and specify what kinds of change count as local revision, interface change, or canon-level change.

6.8. Residue Principle

A canon ontology must leave visible what the canon as a whole still does not settle.

6.9. Local Independence Principle

A canon ontology should preserve the principle that local artifacts may remain self-standing even when they participate in the canon.

Compressed house rule. Local independence, global composability.

7. Core Standard

A canon ontology counts as successful only if it makes all of the following clear:

1. what the canon is,

2. why the canon exists,
3. what kinds of objects belong to it,
4. how those objects relate,
5. what the major layers or regions are,
6. how the canon can be entered,
7. what parts are stable or unstable,
8. what remains unresolved,
9. and how the canon may be revised or expanded.

A strong canon ontology does not merely document a project. It renders the project's larger architecture inspectable, criticizable, and governable.

8. Required Functions of a Canon Ontology

Every canon ontology written under COS-1 must perform the following functions, whether through distinct sections or an equivalent readable structure.

8.1. Identity Function

The document must state what the canon is.

Minimum requirement. A competent reader can state the canon's identity in one to three sentences.

8.2. Purpose Function

The document must state why the canon exists.

Minimum requirement. A competent reader can state what problem or architectural need the canon is meant to answer.

8.3. Boundary Function

The document must identify what belongs to the canon and what does not.

Minimum requirement. The reader can tell the difference between canonical, adjacent, experimental, and external objects.

8.4. Registry Function

The document must specify the kinds of objects the canon contains.

Minimum requirement. The canon's contents are typed rather than presented as one flat undifferentiated mass.

8.5. Relation Function

The document must define the relations by which canonical objects are connected.

Minimum requirement. At least the major relation types are named and used consistently.

8.6. Layer Function

The document must identify the major layers, regions, or structural zones of the canon.

Minimum requirement. A competent reader can describe the canon's larger architecture.

8.7. Entry Function

The document must identify valid entry points into the canon.

Minimum requirement. The reader can see how to approach the canon without reading everything first.

8.8. Stability Function

The document must distinguish stable, frozen, working, and experimental parts of the canon.

Minimum requirement. A competent reader can tell what is fixed, what is revisable, and what is volatile.

8.9. Residue Function

The document must state what the canon still does not settle.

Minimum requirement. The canon's unresolved tensions are visible rather than hidden.

8.10. Governance Function

The document must specify how the canon may be revised, extended, or reorganized.

Minimum requirement. There is some account of what counts as local revision, interface change, and canon-level change.

9. Default Document Structure

The following is the default house packaging format for a Canon Ontology Specification. The functions below must be performed, but section names may be merged or renamed so long as recoverability is preserved.

0. Metadata

Include as useful:

- canon name,
- canon version,
- ontology version,
- status,
- freeze level,
- maintainer or author,
- primary domains,
- scope note.

1. Canon Purpose Statement

Required.

State:

- why the canon exists,
- what it is meant to make possible,
- why a canon-level artifact is needed.

2. Canon Identity

Required.

State:

- the canon's compact thesis or orienting identity,
- what kind of canon it is,
- its minimal and maximal intended scope.

3. Canon Kernel

Required.

This section should identify:

- the canon's most load-bearing commitments,
- the concepts or theses without which the canon would lose identity,
- and the distinction between kernel and extension where useful.

4. Canon Object Registry

Required.

This section should identify:

- the major object classes inside the canon,
- representative instances where useful,
- and any special status objects such as field proposals or house specs.

5. Relation Ontology

Required.

This section should define the main relation types used across the canon, such as:

- upstream of,
- downstream of,
- companion to,
- inherits from,
- exports to,
- constrains,
- refines,
- exemplifies,
- subfield of,
- framework within,
- paper-class specification for.

6. Layer and Order Map

Required.

This section should identify:

- the canon's layers or structural regions,
- stack relations where relevant,
- and any other organizing principles such as branches or domains.

7. Entry Points

Required.

This section should specify valid ways of entering the canon, for example:

- introductory entry points,
- formal entry points,
- institutional critique entry points,
- field-proposal entry points,
- methodological entry points.

8. Stability and Drift Map

Required.

This section should identify:

- frozen objects,
- stable objects,
- working objects,
- experimental objects,
- and the kinds of drift they are allowed to undergo.

9. Canon Residue Statement

Required.

This section should identify:

- unresolved tensions,
- open burdens,
- missing branches,
- unfinished interfaces,
- and larger questions the canon does not yet settle.

10. Expansion and Revision Rules

Required.

This section should specify:

- how new objects enter the canon,
- what counts as local revision,
- what counts as interface change,
- what counts as canon-level change,
- and who or what governs such changes if relevant.

11. Public vs Internal Canon

Recommended.

This section should distinguish:

- the public canon,
- the internal canon,
- what belongs to each,
- and what should not be treated as public-facing canonical material.

12. Conclusion

Required.

This section should include:

- the canon's most important structural result,
- its strongest identity statement,
- and the main implication of having rendered the canon explicitly.

Appendices

Optional but recommended.

Useful appendices include:

- object tables,
- relation tables,
- layer maps,
- branch maps,

- frozen object index,
- experimental object index,
- reading-path suggestions.

10. Canon Object Classes

COS-1 recommends that a canon ontology distinguish at least some of the following object classes where relevant:

- **Theory-object:** a structured conceptual object that may admit multiple renderings.
- **Theory paper:** a reader-facing prose rendering of a theory-object.
- **Field proposal:** an object proposing a discipline, field, or area of inquiry.
- **House specification:** a specification governing a reusable paper class, artifact class, or methodological form.
- **Formal framework:** a formal or quasi-formal architecture used within the canon.
- **Source review object:** a literature-facing synthesis artifact or source-cluster review.
- **Companion object:** an object whose primary role is to clarify or support another.
- **Bridge object:** an object that links otherwise separate domains or regions of the canon.
- **Meta-object:** an object that governs canon structure itself.

These classes may be refined further, but the canon ontology should not leave all objects untyped.

11. Relation Vocabulary

COS-1 recommends a canonical relation vocabulary. Not every canon needs every relation, but relation-types should be made explicit.

- **upstream of / downstream of**
- **inherits from**
- **exports to**
- **companion to**
- **constrains**
- **depends on**
- **refines**

- **clarifies**
- **exemplifies**
- **subfield of**
- **framework within**
- **paper-class specification for**
- **adjacent to**
- **in tension with**

Rule. A canon ontology should prefer a small, stable relation vocabulary over a large and improvised one.

12. Presentation Norms by Section

12.1. Purpose Statement

The purpose section must not merely say that the canon exists because there are many documents. It must state what the canon-level rendering makes possible.

12.2. Identity Section

The identity section must not collapse into a list of local theses. It should state what the canon as a whole is for.

12.3. Kernel Section

The kernel section must distinguish what is load-bearing from what is optional, peripheral, or still experimental.

12.4. Registry Section

The registry section must not be a raw archive dump. It should make object classes visible.

12.5. Relation Section

The relation section must not rely on metaphor alone. Relation-types should be stated plainly.

12.6. Layer Section

The layer section must help the reader understand the larger architecture rather than merely restating relation pairs one by one.

12.7. Entry Section

The entry section must be practical. It should help real users enter the canon without total prior commitment.

12.8. Drift Section

The stability and drift section must make change conditions visible. A canon ontology without drift governance is structurally weak.

12.9. Residue Section

The residue section must not be perfunctory. A canon that pretends to have no unresolved tensions is not being rendered seriously.

13. Evaluation Criteria

A canon ontology should be judged along the following axes.

A. Identity Clarity

Can a competent reader say what the canon is and why it exists?

B. Boundary Discipline

Can the reader tell what belongs to the canon and what does not?

C. Typed Registry Quality

Are the canon's objects rendered as intelligible classes rather than as a flat list?

D. Relation Clarity

Are the relations among objects visible and usable?

E. Layer Legibility

Does the canon's larger architecture become clear?

F. Entry Usability

Can different readers enter the canon from valid points without total immersion?

G. Stability Discipline

Are frozen, stable, working, and experimental objects distinguished clearly?

H. Residue Management

Does the canon leave unresolved tensions visible?

I. Governance Quality

Can the reader tell how the canon changes?

J. Local Independence Preservation

Does the canon preserve the principle that local artifacts can remain self-standing?

14. Failure Modes a Good Canon Ontology Must Avoid

1. Archive Dump

The canon ontology becomes a list of files rather than a structured rendering.

2. Flatness

All objects are treated as though they were the same kind of thing.

3. Stack Reductionism

The canon is reduced to one directional dependency chain even though it contains branches, specs, and companion objects.

4. Boundary Blur

The document never makes clear what counts as canonical versus adjacent or experimental.

5. Relation Vagueness

Objects are said to be "connected" without specifying how.

6. No Drift Governance

The canon is rendered as static when it is actually changing.

7. Totalization

The canon is presented as though it already settles everything relevant to its domains.

8. Canon-Local Collapse

Local artifacts become unreadable except as fragments of the larger whole.

9. Maintenance Internalism

The canon ontology reads like a private database export rather than a structural document.

10. No Entry Paths

The canon is rendered as all-or-nothing rather than navigable.

15. Optional Formal Layer

A canon ontology may include a formal or quasi-formal layer. If it does, it should specify at least:

- object classes,
- relation classes,
- layer classes,
- stability states,
- revision states,
- and any inclusion/exclusion predicates.

Rule. Formalization must clarify canon structure, not merely imitate database schema for prestige.

Good uses.

- typed object tables,
- relation matrices,
- layer maps,
- status tables,
- entry-path schemas,
- branch and freeze maps.

16. Reusable COS-1 Template

Title

Metadata

- canon name,
- canon version,
- status,

- freeze level,
- domains,
- maintainer.

1. Canon Purpose Statement

- why the canon exists,
- what it makes possible,
- why a canon-level rendering is needed.

2. Canon Identity

- compact canon thesis,
- canon type,
- minimal scope,
- maximal scope.

3. Canon Kernel

- load-bearing commitments,
- kernel objects,
- optional vs non-optional core.

4. Canon Object Registry

- object classes,
- representative instances,
- special object classes.

5. Relation Ontology

- relation vocabulary,
- relation constraints,
- key examples.

6. Layer and Order Map

- major layers,
- stack relations,
- branches,
- cross-layer links.

7. Entry Points

- introductory entry points,
- advanced entry points,
- formal entry points,
- branch-specific entry points.

8. Stability and Drift Map

- frozen objects,
- stable objects,
- working objects,
- experimental objects,
- drift rules.

9. Canon Residue Statement

- unresolved tensions,
- missing branches,
- open burdens,
- underdefined interfaces.

10. Expansion and Revision Rules

- how new objects enter,
- local revision,
- interface change,
- canon-level change.

11. Public vs Internal Canon

- public canon,
- internal canon,
- relation between them.

12. Conclusion

- most important structural result,
- strongest identity statement,
- next-step implication.

Appendices

- object tables,
- relation tables,
- layer maps,
- frozen object index,
- reading paths.

17. Recoverability Test

A canon ontology satisfies COS-1 only if a competent reader can answer the following questions from the text itself:

1. What is this canon?
2. Why does this canon exist?
3. What kinds of objects does it contain?
4. What are the major relation types among those objects?
5. What is the larger layer or stack structure?
6. How can the canon be entered?
7. What is frozen, stable, working, or experimental?
8. What remains unresolved in the canon as a whole?
9. How does the canon change?
10. How does the canon preserve local independence while maintaining global composability?

A document that cannot answer these questions may still be useful as project notes, but it has not yet rendered the canon as a structured object.

18. Acceptance Test

A canon ontology satisfies COS-1 only if all of the following are true:

1. the canon has a clearly stated identity,
2. the canon has a clearly stated purpose,
3. the canon boundary is visible,
4. the canon's object classes are typed,
5. the relation vocabulary is explicit,
6. the layer or order structure is recoverable,
7. valid entry points are provided,
8. stability and drift conditions are stated,
9. canon-level residues are visible,
10. revision and expansion rules are specified,
11. the canon does not collapse local artifacts into unreadable fragments,
12. the canon reads as a governed architecture rather than a file dump.

19. Recommended House Additions

For canons especially vulnerable to sprawl, hidden drift, or internalism, the following additions are recommended:

- an *Object Type Table*,
- a *Relation Vocabulary Table*,
- a *Layer Map*,
- a *Stability Matrix*,
- an *Entry Path Table*,
- and a brief *Canon Residue Ledger*.

These are aids to canon intelligibility, not replacements for local ontology documents or theory papers.

20. Final Statement

A canon is not merely a collection of documents. It is a bounded, revisable, internally related architecture of objects, relations, statuses, and rules. A good Canon Ontology Specification identifies that architecture clearly enough that the canon becomes inspectable, navigable, extensible, and criticizable without sacrificing the local self-standing character of its parts. Under COS-1, that is the core task: to render the canon as a structured whole while preserving the principle of local independence and global composability.