PMS-LOGIC – Model Specification

A PMS-conform application layer for logic: pre-moral foundations, logical limits, and post-moral effects (responsibility without “ought”)

Version: PMS-LOGIC_1.0 · Spec basis: `pms-logic.yaml`
Author: T. Zöller · Formalisation assistance: ChatGPT (GPT-5.2 Thinking)
Language: EN · Status: Model spec (aligned with `schema_meta.status = "draft"`)
Depends on: `PMS.yaml` (`schema_version = "PMS_1.1"`)

---

1. Purpose and scope of this specification

This document specifies the *PMS-LOGIC* layer in a concise, technical form. It is based on the YAML file `pms-logic.yaml` (with `schema_version = "PMS-LOGIC_1.0"`) and renders its structure, constraints, and semantics explicit for human readers and computational systems.

PMS-LOGIC is an **application profile** (overlay) of the Praxeological Meta-Structure (PMS; Δ–Ψ). It does **not** redefine PMS operators, dependencies, layers, or derived axes. Instead, it formalizes a **logic lens** that is: **pre-moral in justification** (no norms, values, duties, or prescriptions introduced), **logical in ordering** (consistency without closure-authority), and **post-moral in effect** (responsibility becomes morally readable as a field-effect under asymmetry and irreversibility, without generating “ought”).

The specification covers, in particular:

- the **schema_meta** block (identity, status, authorship, explicit dependency on PMS_1.1);
- the **validity gate** enforcing X (Distance), reversibility, and D (dignity-in-practice);
- the **paper lens** definition (tripartition: pre-moral / logical / post-moral) and non-operator constructs;
- **reduced signatures** for the three levels and key threshold markers (non-closure, moral readability, effective capacity);
- **failure-mode constraints** (apparent norms reconstructed as structural constraints; non-prescriptive);
- the **drift catalogue** (structural failure modes: dogmatism, authority capture, pseudo-ought injection, etc.);
- **scale matrices** (projection onto PMS derived axes; structural only);
- **counter-readings** (misreadings and structural responses);
- the **example suite schema** (scene-bound, operator-readable vignettes).

> **Core idea**
> PMS-LOGIC treats logic as an indispensable ordering tool whose structural boundary is **non-closure** (Λ carried within □). Morality is **not** an operator but a readability field of responsibility that emerges **post hoc** under Ω and Θ. The profile reconstructs “norm-like” statements as **failure-mode constraints** (self-binding and restraint), not as obligations or prescriptions.

2. High-level structure of the YAML model

2.1 Top-level keys

Key	Description	Role in the model
`schema_version`	Version identifier (`"PMS-LOGIC_1.0"`).	Compatibility and citation
`schema_meta`	Model identity, authors, status, and explicit dependency on PMS_1.1; paper source metadata.	Meta-information / inheritance statement
`validity_gate`	Entry conditions for PMS-application: X, reversibility, D.	Application firewall
`operator_reference`	Fixed operator set Δ–Ψ and dependency hygiene notes.	Non-redefinition guarantee
`overlays`	Optional amplification overlays (non-operators). Empty in PMS-LOGIC.	Explicit statement of "no new operators"
`paper_lens`	Tripartition (pre-moral / logical / post-moral), key claims, theses, and non-operator constructs.	Semantic lens declaration
`reduced_signatures`	Reduced signatures for the three levels and key threshold markers (non-closure, moral readability, effective capacity).	Threshold definitions
`failure_mode_constraints`	Apparent "norms" reconstructed as structural constraints (non-prescriptive).	Constraint layer (no "ought")
`drift_catalogue`	Structural failure modes (operator distortions): dogmatism, authority capture, pseudo-ought injection, etc.	Drift awareness
`scale_matrices`	Structural projections (e.g., onto derived axes A, C, R, E, D).	Comparability / interpretive compression
`counter_readings`	Common misreadings and structural responses (without ethics injection, metaphysics, or weaponization).	Misuse resistance
`glossary`	Paper glossary (non-operator definitions; operator meanings remain canonical in PMS.yaml).	Terminological hygiene
`example_suite_schema`	Schema for repo vignettes (scene-bound, operator-readable, non-instructional).	Implementation discipline

Conceptual separation
PMS defines the operator grammar (Δ–Ψ). PMS-LOGIC defines how that grammar is used to keep logic *indispensable* while preventing its elevation to closure-authority, and how responsibility becomes readable without translating it into "ought."

3. Validity gate and scope constraints

3.1 Application gate

- **X (Distance):** stop-capability must remain active; no fusion into verdict, authority, or closure claims.
- **Reversibility:** all readings are scene-bound and revisable; no global person labels.
- **D (Dignity-in-practice):** no shaming/ranking; structure-only critique of configurations.

3.2 Explicit non-goals

- deriving or asserting norms, values, duties, or “ought” statements;
- turning logical coherence into ultimate justification or authority;
- coercive enforcement, legitimacy attribution, person-ranking, irreversible exposure demands;
- metaphysical closure substitutions (theological, ontological, or totalizing narratives);
- clinical inference, personality typing, or mental health claims.

Key firewall
If logic is used as final authority (closure-authority) or responsibility is translated into “ought,” the application has left PMS-LOGIC’s scope and violates the validity gate.

4. PMS-LOGIC as a structural configuration

PMS-LOGIC is defined by a tripartition of levels. Each level is expressed as a reduced signature (shorthand only; PMS dependencies remain authoritative).

Level	Reduced signature	Meaning (structural)
Pre-moral justification	`Δ + ∇ + □ + Λ + A`	Praxis conditions prior to normativity: difference triggers impulse; frames constrain; non-events structure expectation; stabilized patterns emerge without moral evaluation.
Logical boundary management	`□ + Λ + X + Σ`	Logic as ordering: integration and consistency are pursued under explicit non-closure (Λ) and held back from dogmatic totalization by distance (X).
Post-moral effects	`Ω + Θ + Ψ`	Responsibility becomes attributable under asymmetry (Ω) and irreversibility (Θ) once trajectories bind (Ψ). No norms are derived; effects become morally readable post hoc.

Boundary formula
Pre-moral in justification, post-moral in effect.
Logic remains a tool for ordering within frames and under non-closure; it does not become an authority that converts coherence into obligation.

4.1 Moral readability without "ought"

PMS-LOGIC treats "morality" as a *non-operator readability field*, minimally expressible as: `Ω + Θ + Λ + Ψ`. This indicates that moral language becomes tempting where irreversible consequences are attributable under asymmetry, while non-closure persists. The profile does not generate prescriptions.

4.2 Effective capacity boundary (no capacity → no responsibility)

Responsibility presupposes effective, frame-bound capacity with temporal availability: `□ + Θ`. Where effective capacity is absent, no responsibility arises—even if outcomes appear significant later.

Non-equivalence
"Responsibility" here means *attributability of consequences*, not moral desert. It is distinct from guilt, blame, or prescriptive obligation.

5. Risk, drift, and closure-authority failure modes

PMS-LOGIC treats “being coherent” and “being right” as structural stressors: both can trigger drift toward closure-authority and pseudo-obligation. The model’s discipline is to restore X, preserve Λ, and keep □ explicit.

5.1 Drift catalogue (structural failure modes)

- **Dogmatism (Σ-totalization):** coherence is turned into final truth; Λ is denied rather than carried.
- **Authority capture (Ω-inflation):** power gradients are mistaken for epistemic/moral authority; X is suppressed.
- **Moralism (pseudo-ought injection):** responsibility is retranslated into obligation; Ψ is externalized into enforcement.
- **Nihilism (Λ-suppression):** non-events and omissions are treated as irrelevant; temporal consequence is flattened.
- **Technocratic reduction (□-overconstraint):** frames are treated as exhaustive; unmodeled remainder is dismissed.

> **Structural warning**
> The critical failure is not “lack of logic,” but logic elevated to **authority**: forced closure, hidden normativity, or power laundering through coherence.

5.2 Apparent norms as constraints (non-prescriptive)

PMS-LOGIC reconstructs norm-like statements as *failure-mode constraints* (self-binding and restraint), not obligations. They function to prevent closure substitution, authority drift, and coercion under Ω and Θ.

6. Minimal Application Protocol (MAP)

1. □ **Frame check:** What is the scene, scope, and boundary of the claim? What is explicitly excluded?
2. **Λ Non-closure integrity:** Where does explanation end? What remainder must be carried rather than replaced?
3. **Σ Integration discipline:** What is coordinated, and what is being totalized? Is coherence being treated as authority?
4. **Ω/Θ Asymmetry & irreversibility audit:** Who bears irreversible cost, exposure, or consequence over time?
5. **X Distance enforcement:** Where is the closure drive (or correctness) replacing restraint and reversibility?
6. □**+Θ Effective capacity test:** Is there frame-bound capacity with temporal availability? If not: no responsibility attribution.
7. **Ψ Self-binding test:** Who is actually bound to consequences (self-binding) vs. who is being bound (externalization)?

Invalidity marker
Any translation of responsibility into “ought” (prescriptive demand), or any coercive move that uses logic as closure-authority, is sufficient for PMS-LOGIC scope invalidity.

7. Implementation notes and citation

The authoritative specification is `pms-logic.yaml`, intended to be loaded after `PMS.yaml`. PMS operator meanings and dependencies remain canonical in PMS.yaml.

```
PMS.yaml → pms-logic.yaml
```

Technical reference:
pms-logic.yaml – PMS-LOGIC Application Profile (PMS-LOGIC_1.0)

Base dependency:
PMS.yaml – Praxeological Meta-Structure (PMS_1.1)

License:
Governed by the license declared in the distribution repository.