The CANONIC Doctrine

Contents

The Dev Manual

How to BUILD in CANONIC. DRY. MATH. FIXED. PURE.

Devs speak programming. This book speaks theirs.

FRONT MATTER

Half-Title

THE CANONIC DOCTRINE

Title Page

THE CANONIC DOCTRINE *The MAGIC Implementation Standard*

CANONIC Series | 1st Edition | 2026

Copyright

Dedication

For every developer who was told to "just ship it." This manual is your compiler.

Epigraph

> "255, or it doesn't deploy."
>
> — CANONIC [G-3]

Foreword

This is a governed document.

Every chapter compiles against the axiom that produced it. Every claim traces to a governance source, a paper, or a blog post. The act of reading this manual is an act of verification. Run `magic validate` on any scope referenced here. The score is the proof [P-7].

This is not a manual about governance. This is a manual about building. Governance IS compilation. Your CANON.md is your grammar. Your VOCAB.md is your type system. Your README.md is your header file. The validator is the compiler. 255 is the target. Everything else is syntax [B-10].

If you are a developer, an agent, an operator, or an architect who builds governed systems, this manual is written for you. It explains how to build in CANONIC — in programming, not prose.

Do X. Run Y. The result is Z.

TABLE OF CONTENTS

PART I — FOUNDATIONS

Chapter 1: The Axiom

Every governed scope begins with one sentence. That sentence is the axiom.

1.1 The Contract

The axiom is a declaration. Not a description. Not a mission statement. Not aspirational prose. It is the single assertion from which every governance decision in the scope derives. Think of it as the type signature of your entire governance scope — if the scope's behavior is consistent with the axiom, the scope type-checks. If not, it fails [B-5].

```
## Axiom

**WALLET stores COIN for every USER. Every event signed.**
```

If you can derive governance decisions from the axiom, the axiom is correct. If decisions contradict the axiom, either the decision is wrong or the axiom needs revision. The axiom is the root of a derivation tree — not a suggestion, not guidance, but the mathematical foundation from which everything in the scope follows [B-5].

1.2 Clinical Axioms

For health IT architects building governed clinical AI, the axiom is the clinical contract. It declares what the scope does, who it serves, and what guarantee it makes. Every clinical AI deployment in CANONIC begins with a clinical axiom:

```
## Axiom

**MammoChat serves breast imaging with governed BI-RADS INTEL. Every recommendation cited.**
```

This axiom declares three binding obligations: (1) the scope serves breast imaging, not general medicine — domain boundary enforced; (2) the evidence is BI-RADS INTEL, not general clinical evidence — evidence standard declared; (3) every recommendation is cited — provenance guarantee. If MammoChat generates an uncited recommendation, it violates its axiom. The violation is a compilation error.

```
## Axiom

**OncoChat serves oncology with governed NCCN INTEL. Every guideline versioned. Every interacti
```

This axiom adds an operational guarantee: every interaction audited. This constraint propagates to every component of the OncoChat scope — the TALK engine must log interactions, the LEDGER must record them, the CHAIN must hash-link them. The axiom drives the architecture.

1.3 Good vs Bad Axioms

Axiom	Analysis	Verdict
"WALLET stores COIN for every USER. Every event signed."	Declarative, specific, testable	GOOD

Axiom	Analysis	Verdict
“PAPERS publishes governed content. Every publication evidenced.”	Declarative, specific, testable	GOOD
“MammoChat serves breast imaging with governed BI-RADS INTEL. Every recommendation cited.”	Domain-bounded, evidence-specified, provenance-guaranteed	GOOD
“We aim to be compliant.”	Aspirational, vague, untestable	BAD
“This scope is a scope.”	Circular, zero information content	BAD
“Our AI helps patients.”	Vague, no boundary, no evidence standard, untestable	BAD

The pattern is clear: good axioms are declarative (state what IS, not what you hope), specific (name the domain, the evidence, the guarantee), and testable (the validator can check compliance). Bad axioms are aspirational, vague, or circular. If your axiom contains the words “aim,” “strive,” “best effort,” or “endeavor” — rewrite it. Those are mission statements, not axioms [B-5].

1.4 Derivation

Every constraint in CANON.md derives from the axiom. Every file in the scope derives from the constraints. Every child scope inherits the axiom’s obligations. The derivation chain is the compilation chain [P-7].

```
Axiom → Constraints → Files → Children → Validation → Score
```

A score of 255 means every derivation holds. A score below 255 means at least one derivation failed. The bitmask identifies which [G-2].

For clinical AI, the derivation chain has specific implications. Consider the MammoChat axiom: “Every recommendation cited.” This axiom derives the following constraints:

```
MUST:     Cite evidence source for every recommendation
MUST:     Include BI-RADS category reference for every screening assessment
MUST:     Version evidence base with effective date
MUST NOT: Generate recommendations without INTEL backing
MUST NOT: Reference evidence outside the governed INTEL layer
```

Each constraint derives from the axiom. Each file in the scope (INTEL.md, COVERAGE.md, LEARNING.md) satisfies one or more constraints. Each child scope (a specific hospital’s MammoChat deployment) inherits these constraints. The derivation is traceable from the axiom to every file in the scope tree. If a constraint cannot be derived from the axiom, the constraint is either wrong or the axiom is incomplete.

1.5 The Axiom as Entry Point

In compiler theory, every program has an entry point — `main()`. In CANONIC, every scope has an entry point — the axiom. The validator reads the axiom first. Everything else compiles against it [P-7].

When `magic validate` runs against a scope, the first thing it reads is the axiom in CANON.md. The axiom establishes the evaluation context. The constraints are evaluated against the axiom. The files are evaluated against the constraints. The score is computed from the file evaluation. The entry point IS the axiom.

For clinical informatics engineers integrating CANONIC into an EHR environment, this means that every governed AI module starts with a single sentence. That sentence defines the clinical contract. The contract drives the architecture. The architecture compiles to a score. The score is the proof. Start with the axiom. Everything else follows.

1.6 Axiom Anti-Patterns

In practice, developers make predictable mistakes with axioms. Here are the anti-patterns and their corrections:

The empty axiom: `**Governed scope.**` — This says nothing. It has zero information content. Every CANONIC scope is governed by definition. The axiom must declare *what* is governed, *how* it is governed, and *what guarantee* it makes.

The kitchen-sink axiom: A paragraph describing everything the scope does, every technology it uses, and every stakeholder it serves. Axioms are one sentence. If you need more than one sentence, you need constraints, not a longer axiom.

The copy-paste axiom: Copying the parent scope's axiom verbatim. The child scope should specialize the parent's axiom, not duplicate it. MammoChat's axiom specializes the healthcare CHAT axiom — it adds breast imaging and BI-RADS. It does not repeat the parent's general clinical CHAT declaration.

The aspirational axiom: `**We strive to provide the best clinical decision support.**` — "Strive" is untestable. "Best" is undefined. Replace with a declarative, testable statement: `**MammoChat serves breast imaging with governed BI-RADS INTEL. Every recommendation cited.**`

Write the axiom. Derive the constraints. Build the scope. Validate to 255.

Chapter 2: The TRIAD

Three files. One truth. Minimum viable governance.

2.1 The Three Files

File	Purpose	Compiler Equivalent	Clinical Parallel
`CANON.md`	Declaration — what you believe	Grammar / entry point	Clinical protocol
`VOCAB.md`	Language — what your words mean	Type system	Medical terminology

File	Purpose	Compiler Equivalent	Clinical Parallel
`README.md`	Interface — what you offer the world	Header file / public API	Service description

The TRIAD is the minimum viable governance declaration. Without it, the scope score is 0. With it, you have the foundation for a governed scope that the validator can evaluate, that other scopes can inherit from, and that auditors can examine [B-5].

2.2 CANON.md Structure

````
# {SCOPE} - CANON

inherits: {parent/path}
version: {YYYY-MM}

---

## Axiom

**{One sentence. The contract.}**

---

## Constraints

\```
MUST:     {binding obligation}
MUST NOT: {binding prohibition}
\```

---

*CANON | {SCOPE} | {DOMAIN}*
````

Every governed Markdown file follows this structure: header, `inherits:`, separator, content, separator, footer [G-1].

For clinical informatics engineers, the CANON.md structure maps directly to clinical protocol governance. Consider a MammoChat deployment at a specific hospital:

````
# MAMMOCHAT-UCF - CANON

inherits: hadleylab-canonic/MAGIC/SERVICES/TALK

---

## Axiom
````

~~~~
**MammoChat serves UCF College of Medicine breast imaging with governed BI-RADS INTEL. Every re

---

## Constraints

\```
MUST:     Cite BI-RADS category for every screening assessment
MUST:     Version evidence base with ACR publication date
MUST:     Log every clinical interaction to LEDGER
MUST:     Maintain PHI boundary - no patient identifiers in governance metadata
MUST NOT: Generate uncited clinical recommendations
MUST NOT: Process PHI outside the local deployment perimeter
MUST NOT: Override radiologist clinical judgment
\```

---

*CANON | MAMMOCHAT-UCF | HEALTHCARE*
~~~~

The constraints are not suggestions. They are binding obligations that `magic validate` will check. "MUST NOT: Process PHI outside the local deployment perimeter" is a HIPAA §164.312 technical safeguard expressed as a governance constraint. If the deployment violates it, validation fails.

2.3 VOCAB.md Structure

~~~~
# VOCAB

inherits: {parent/VOCAB.md}

| Term | Definition |
|------|------------|
| {TERM} | {Definition. Precise. No stubs.} |

---

*VOCAB | {SCOPE}*
~~~~

An undefined term is a type error. A stub definition (`"--"` or `"Governed term."`) is a gap, not a closure. Every SCREAMING_CASE term in a scope must resolve to a definition [G-1].

In clinical informatics, vocabulary precision is not optional — it is patient safety. A VOCAB.md for a clinical AI scope must define every clinical term the scope uses, because ambiguity in clinical terminology produces clinical errors:

~~~~
# VOCAB

inherits: hadleylab-canonic/MAGIC/VOCAB.md
~~~~

```
| Term | Definition |
|------|-----------|
| BI-RADS | Breast Imaging Reporting and Data System. ACR standardized classification (0-6) fo
| SCREENING | Population-level breast imaging for asymptomatic patients. Distinct from DIAGNOS
| DIAGNOSTIC | Targeted breast imaging for symptomatic patients or abnormal screening findings
| TRIAGE | Classification of screening findings into follow-up categories based on BI-RADS ass
| PHI | Protected Health Information as defined by HIPAA §160.103. |
| INTEL | Governed knowledge unit with provenance - source, date, evidence grade, citation. |
| LEDGER | Append-only audit trail. Every governed event recorded. |

---

*VOCAB | MAMMOCHAT-UCF*
```

If the MammoChat scope uses the term "SCREENING" without a VOCAB.md definition, the validator flags it — just as a compiler flags an undefined variable. The clinical vocabulary IS the type system. Undefined terms produce type errors. Ambiguous terms produce runtime bugs. In clinical AI, runtime bugs are patient safety events.

2.4 README.md Structure

```
# {SCOPE} - {Title}

inherits: {parent/path}

---

{Scope description. What it does. What it offers. How to use it.}

---

*{SCOPE} | README | {DOMAIN}*
```

The README is the public interface. External consumers read it. Internal consumers inherit from it. In a clinical informatics context, the README is the service description that the hospital's clinical informatics committee reviews when evaluating an AI deployment [B-5].

2.5 The Recursive Property

The TRIAD appears at every level of the governance tree. Organization has TRIAD. Team has TRIAD. Project has TRIAD. Service has TRIAD. The validator does not distinguish scopes by size — only by compliance [B-5].

```
canonic-canonic/             TRIAD
  MAGIC/                     TRIAD
    SERVICES/                TRIAD
      TALK/                  TRIAD
hadleylab-canonic/           TRIAD
  MAGIC/                     TRIAD
    SERVICES/                TRIAD
  MAMMOCHAT/             TRIAD
    MAMMOCHAT-UCF/       TRIAD
```

Same three files, every scale. A 5,000-employee health system and a single-physician AI deployment use the same TRIAD. The clinical informatics committee reviewing a hospital-wide AI governance proposal evaluates the same file structure that a solo developer creates for a proof-of-concept. The TRIAD scales because it is minimal — three files, not thirty.

2.6 The TRIAD in EHR Integration

For health IT architects integrating CANONIC with existing EHR systems (Epic, Cerner, MEDITECH), the TRIAD maps to the integration contract. The CANON.md declares what the integration does and what constraints it obeys. The VOCAB.md defines the terminology that the integration uses — FHIR resource types, HL7 message types, clinical terminology standards. The README.md describes the integration interface — what endpoints it exposes, what data flows it supports, what clinical workflows it enables.

When the EHR vendor asks "what does your AI governance framework require?" — the answer is three files. When the hospital's IT security team asks "what are the constraints on this AI deployment?" — the answer is CANON.md. When the clinical informatics committee asks "what terms does this system use and what do they mean?" — the answer is VOCAB.md.

Three files. One truth. Minimum viable governance. Everything else builds on this foundation.

Chapter 3: The Inheritance Chain

`inherits:` is a binding obligation. Not metadata. The compiler resolves and enforces it [B-6].

3.1 Syntax

```
inherits: hadleylab-canonic/DEXTER/PAPERS
```

This declares: "I accept all governance constraints from `hadleylab-canonic/DEXTER/PAPERS`." It is not a reference. It is not a dependency. It is a binding obligation — the scope declares that it satisfies every constraint in its parent's governance chain, from the parent all the way to the root.

3.2 Resolution

When `magic validate` runs:

1. Read the `inherits:` line.
2. Walk the chain upward to root.
3. Collect all constraints at every level.
4. Verify the target scope satisfies every collected constraint.
5. Broken link or violated constraint = compilation error [B-6].

The resolution is deterministic. The same `inherits:` chain always produces the same constraint set. The validator does not use heuristics, does not make judgments, and does not apply discretion. It resolves the chain, collects the constraints, and checks them. Pass or fail. 255 or less.

For clinical informatics engineers, this resolution model has a critical implication: when your MammoChat deployment inherits from `hadleylab-canonic/MAGIC/SERVICES/TALK`, every constraint in the TALK service's governance chain — from TALK to SERVICES to MAGIC to hadleylab-canonic to canonic-canonic — is collected and enforced on your deployment. If the root declares "MUST: Validate to 255 before production deployment," your deployment inherits that obligation. No exceptions. No overrides.

3.3 Monotonic Accumulation

A child scope CAN add constraints beyond its parent. A child scope CANNOT remove or weaken a parent's constraints. This is the Liskov Substitution Principle applied to governance: a subtype extends but never violates the parent contract [B-6].

The parent's governance score is the floor. The child's score = floor + whatever the child adds.

In a hospital network deploying CANONIC, monotonic accumulation means that the network's root governance (HIPAA compliance, PHI boundary enforcement, LEDGER recording) cannot be weakened by any hospital in the network. A hospital can ADD constraints — state-specific regulations, site-specific PHI handling policies, local IRB requirements. A hospital cannot REMOVE constraints. The network's governance floor is architecturally enforced.

This is why hospital CISOs find the inheritance model compelling: the governance floor is not a policy that humans must remember to follow. It is an architectural constraint that the validator enforces automatically. A department cannot deploy a clinical AI that violates the hospital's governance floor, because the validator will reject it. The inheritance chain IS the access control.

3.4 Three Forms of Inheritance

Form	Syntax	Scope	Example
Current	`.`	Same scope	`inherits: .` — scope-local files
Relative	`SCOPE/PATH`	Relative to fleet root	`inherits: MAGIC/SERVICES/TALK`
Cross-fleet	`fleet/SCOPE/PATH`	Cross-organization	`inherits: canonic-canonic/MAGIC`

Only LANGUAGE, MAGIC, and GOV paths may be hardcoded. Everything else is compiled [G-1].

3.5 The Root

The root of the entire system is `canonic-canonic`. It defines: fundamental primitives (INTEL, CHAT, COIN), service composition, tier algebra, validation framework (MAGIC). Every organization inheriting from `canonic-canonic` accepts these foundational constraints [B-6].

```
canonic-canonic (root - LUCA)
    hadleylab-canonic (organization)
         DEXTER (author scope)
              PAPERS (domain)
                   governance-as-compilation (leaf scope)
```

```
          MAGIC (governance engine)
                SERVICES (service tree)
                      TALK (conversation service)
                            MAMMOCHAT (product)
```

In evolutionary biology, the root is the Last Universal Common Ancestor (LUCA). In CANONIC, `canonic-canonic` is the LUCA — the ancestor from which every governed scope descends. The parallel is structural, not metaphorical. The inheritance chain proves descent. The constraint propagation proves homology. The 255-bit score proves fitness [P-3].

3.6 Cross-Organization Inheritance

Organizations inherit from the root. The inheritance chain crosses GitHub org boundaries:

```
canonic-canonic                (org: canonic-canonic)
    hadleylab-canonic         (org: hadleylab-canonic)
    adventhealth-canonic      (org: adventhealth-canonic)
    canonic-python            (org: canonic-python)
```

Each org is a monophyletic clade — a complete branch descended from a single ancestor. The `inherits:` chain proves descent from the common ancestor [P-3].

For health IT architects deploying CANONIC across a multi-hospital network, cross-organization inheritance is the federation model. Hospital A and Hospital B both inherit from the network root. The network root inherits from `canonic-canonic`. The governance chain crosses organizational boundaries while maintaining constraint propagation. Hospital A's MammoChat deployment inherits the network's HIPAA constraints AND Hospital A's site-specific constraints AND the TALK service's clinical constraints AND the root's foundational constraints. The entire chain is resolved, collected, and enforced at validation time.

3.7 The Healthcare Inheritance Tree

A realistic healthcare deployment creates an inheritance tree that looks like this:

```
canonic-canonic                         (root)
    hadleylab-canonic                  (organization)
          MAGIC/SERVICES/TALK          (TALK service)
                MAMMOCHAT              (product)
                      MAMMOCHAT-UCF    (hospital deployment)
                      MAMMOCHAT-ADVENT (hospital deployment)
```

Each level adds constraints: - **canonic-canonic**: Foundational primitives, validation framework - **hadleylab-canonic**: Organization-specific governance - **TALK**: Conversation service constraints (systemPrompt, disclaimer, PHI boundary) - **MAMMOCHAT**: Product constraints (BI-RADS evidence, imaging INTEL) - **MAMMOCHAT-UCF**: Site constraints (UCF IRB requirements, Florida state regulations)

The constraint set at each leaf is the union of all ancestor constraints. The leaf cannot weaken any ancestor's constraint. The tree IS the governance architecture. Build the tree correctly, and the governance follows.

Chapter 4: The Eight Dimensions

Eight questions. Binary answers. One score.

4.1 The Dimensions

Bit	Code	Dimension	Evidence	Question
0	D	Declarative	CANON.md exists	What do you believe?
1	E	Evidential	VOCAB.md exists	Can you prove it?
2	T	Transparent	ROADMAP.md exists	Where are you going?
3	R	Reproducible	{SCOPE}.md exists	Who are you?
4	O	Operational	COVERAGE.md exists	How do you work?
5	S	Structural	inherits: + Axiom + MUST/SHOULD	What shape are you?
6	L	Linguistic	LEARNING.md exists	What have you learned?
7	LANG	Language	LANGUAGE inherited	How do you express?

Each dimension is binary: present or absent. The score is a bitmask: `f(D,E,T,R,O,S,L,LANG)` [G-2].

4.2 The Score

`score = SUM(d_i * 2^i), i=0..7, where d_i in {0,1}`

Range: [0, 255]. 255 means all eight dimensions satisfied. 0 means none [P-1].

The score is deterministic. The same governance files always produce the same score. There is no human judgment in the scoring. There is no discretion. There is no “well, it’s close enough.” The files exist or they don’t. The structures are present or they’re not. The score is a mathematical fact, not an assessment.

4.3 Compliance Tiers

Tier	Composition	Score	Healthcare Minimum
COMMUNITY	D + E + S	35	Research prototype
BUSINESS	COMMUNITY + R	43	Internal pilot
ENTERPRISE	BUSINESS + T + O	63	Clinical deployment
AGENT	ENTERPRISE + L	127	Learning clinical AI
FULL (MAGIC)	AGENT + LANG	255	Production clinical AI

Tiers are cumulative. You cannot skip dimensions. BUSINESS requires COMMUNITY. AGENT requires ENTERPRISE [G-2].

For clinical informatics engineers, the tier system maps to clinical AI deployment readiness. A scope at COMMUNITY tier has declared its purpose and defined its terms — but it has no roadmap, no reproducibility guarantee, and no operational coverage assessment. It is a research prototype. It is not ready for clinical deployment.

A scope at ENTERPRISE tier has everything BUSINESS has, plus transparency (ROADMAP.md — where is this scope going?) and operational coverage (COVERAGE.md — how does this scope actually work?). This is the minimum tier for clinical deployment — the hospital's compliance committee can review the COVERAGE assessment, and the roadmap shows the governance trajectory.

A scope at AGENT tier has LEARNING.md — accumulated intelligence from governance events. The scope learns from its operation. This is where clinical AI starts to provide continuous quality improvement signals — not just serving clinical queries, but learning from governance events and improving governance over time.

A scope at 255 (FULL) has language governance — the scope expresses itself in a controlled vocabulary that inherits from the LANGUAGE standard. This is production-grade clinical AI — fully governed, fully documented, fully auditable, fully compliant.

4.4 Clinical Dimension Mapping

Each dimension maps to specific healthcare compliance requirements:

Dimension	HIPAA	FDA Part 11	Joint Commission	HITRUST
D (Declarative)	Purpose limitation	Record declaration	Service definition	Risk scope
E (Evidential)	PHI evidence	ALCOA evidence	Quality evidence	Security evidence
T (Transparent)	Processing transparency	Change control	Quality improvement plan	Monitoring plan
R (Reproducible)	Reproducible controls	Validation protocol	Reproducible quality	Reproducible security
O (Operational)	Operational safeguards	Operational validation	Operational compliance	Operational controls
S (Structural)	Structural integrity	System structure	Organizational structure	Framework structure
L (Linguistic)	Pattern detection	Change detection	Quality learning	Continuous monitoring
LANG (Language)	Controlled vocabulary	Legibility	Quality vocabulary	Security vocabulary

When a clinical informatics engineer builds a scope to 255, the scope simultaneously satisfies governance requirements across every major healthcare compliance standard. The dimensions are not healthcare-specific — they are universal governance dimensions. But in healthcare, each dimension maps to specific regulatory requirements. The mapping is not approximate. It is structural.

4.5 COVERAGE.md

COVERAGE.md answers the eight questions explicitly. One question per dimension. PASS or FAIL per question. The validator cross-references COVERAGE.md against actual file presence [G-2].

```
# COVERAGE

inherits: .

| # | Dimension | Question | Answer | Status |
|---|-----------|----------|--------|--------|
| 1 | D | What do you believe? | MammoChat serves breast imaging with governed BI-RADS INTEL |
| 2 | E | Can you prove it? | VOCAB.md: 47 clinical terms defined, zero stubs | PASS |
| 3 | T | Where are you going? | ROADMAP.md: Q1 expand evidence base, Q2 add diagnostic mode |
| 4 | R | Who are you? | MAMMOCHAT-UCF.md: scope description, evidence chain, citations | PASS
| 5 | O | How do you work? | This file - operational coverage assessment | PASS |
| 6 | S | What shape are you? | inherits: hadleylab-canonic/MAGIC/SERVICES/TALK, axiom present
| 7 | L | What have you learned? | LEARNING.md: 12 governance events logged, 3 patterns captur
| 8 | LANG | How do you express? | LANGUAGE: inherits canonic-canonic/LANGUAGE | PASS |

Score: 255/255
```

COVERAGE.md is not documentation. It is a self-assessment that the validator verifies. If COVERAGE.md claims PASS for dimension D but CANON.md does not exist, the validator catches the discrepancy. COVERAGE.md is the scope's claim about its own compliance. The validator is the auditor that checks the claim.

4.6 Kernel Internals

The bit weights, hex values, and tier boundary calculations are kernel internals. They are implemented in the C binary (`magic.c`) and are not published in governance prose [G-3]. The tier names, dimension names, and composition formulas are public. The scoring algorithm is public. The bit-weight assignments are PRIVATE.

For developers: do not attempt to reverse-engineer the bit weights from the tier scores. The kernel is the kernel. The validator is the compiler. You do not need to know the internal register allocation strategy of `gcc` to write C programs. You do not need to know the bit weights of `magic.c` to build governed scopes. Write the files. Run `magic validate`. The score is the result.

PART II — BUILDING

Chapter 5: Your First 255

Zero to FULL in one session. This chapter walks through building a governed clinical AI scope from scratch — starting with an empty directory and ending with a fully validated 255-bit scope that satisfies healthcare compliance requirements by architecture [B-4].

5.1 Create the Scope

We will build a governed scope for a clinical TALK agent — a MedChat deployment at a community hospital. The scope will inherit from the TALK service governance and add site-specific clinical constraints.

```
mkdir -p MEDCHAT-COMMUNITY
cd MEDCHAT-COMMUNITY
```

5.2 Write the TRIAD

Start with the three foundational files. Each file follows the governed Markdown structure: header, `inherits:`, separator, content, separator, footer.

CANON.md:

```
# MEDCHAT-COMMUNITY - CANON

inherits: hadleylab-canonic/MAGIC/SERVICES/TALK

---

## Axiom

**MedChat serves Community Hospital clinical staff with governed medical INTEL. Every recommenc

---

## Constraints

\```
MUST:     Cite evidence source for every clinical recommendation
MUST:     Log every interaction to LEDGER with timestamp and actor
MUST:     Maintain PHI boundary - no patient identifiers in governance metadata
MUST:     Version evidence base with publication dates
MUST NOT: Generate uncited clinical recommendations
MUST NOT: Process PHI outside the local deployment perimeter
MUST NOT: Override clinician clinical judgment
\```

---

*CANON | MEDCHAT-COMMUNITY | HEALTHCARE*
```

Note the constraints: each derives from the axiom. "Every recommendation cited" becomes `MUST: Cite evidence source`. "Every interaction audited" becomes `MUST: Log every interaction to LEDGER`. The axiom drives the constraints. The constraints drive the architecture.

VOCAB.md:

```
# VOCAB
```

```
inherits: hadleylab-canonic/MAGIC/VOCAB.md

| Term | Definition |
|------|-----------|
| MEDCHAT | General-purpose clinical TALK agent serving medical questions across specialties |
| INTEL | Governed knowledge unit with provenance - source, date, evidence grade, citation |
| PHI | Protected Health Information as defined by HIPAA §160.103 |
| LEDGER | Append-only audit trail - every governed event recorded with timestamp and actor |
| EVIDENCE-BASE | Collection of governed INTEL units backing clinical recommendations |

---

*VOCAB | MEDCHAT-COMMUNITY*
```

Every SCREAMING_CASE term the scope uses must be defined. No stubs. No circular definitions. If you use the term PHI in your CANON.md constraints, define PHI in VOCAB.md.

README.md:

```
# MEDCHAT-COMMUNITY - Clinical Decision Support

inherits: hadleylab-canonic/MAGIC/SERVICES/TALK

---

MedChat deployment for Community Hospital clinical staff. Provides governed clinical decision s

---

*MEDCHAT-COMMUNITY | README | HEALTHCARE*
```

5.3 First Validation — COMMUNITY Tier

```
git add CANON.md VOCAB.md README.md
git commit -m "GOV: bootstrap MEDCHAT-COMMUNITY - TRIAD"
magic validate
```

Score: ~35. COIN minted: ~35. Tier: COMMUNITY [B-4].

The scope exists. It has declared its purpose (D), defined its terms (E), and established its structure (S). Three dimensions satisfied. This is the minimum viable governance — the scope is on the map. It is not yet ready for clinical deployment. But it exists, and its existence is governed.

5.4 Add Reproducibility and Operations — ENTERPRISE Tier

Now add the files that take the scope from COMMUNITY to ENTERPRISE — the minimum tier for clinical deployment in a hospital setting.

MEDCHAT-COMMUNITY.md (the spec — R dimension):

```
# MEDCHAT-COMMUNITY

inherits: .

---

## Scope Intelligence

| Dimension | Value |
|-----------|-------|
| Subject | Clinical decision support for Community Hospital |
| Audience | Hospitalists, NPs, PAs, nurses, pharmacists |
| Evidence | UpToDate, DynaMed, specialty society guidelines |
| Status | Initial deployment |

---

*MEDCHAT-COMMUNITY | SPEC | HEALTHCARE*
```

COVERAGE.md (operational coverage — O dimension):

```
# COVERAGE

inherits: .

| # | Dimension | Question | Answer | Status |
|---|-----------|----------|--------|--------|
| 1 | D | What do you believe? | MedChat serves Community Hospital with governed INTEL | PASS
| 2 | E | Can you prove it? | VOCAB.md: 5 terms defined, zero stubs | PASS |
| 3 | T | Where are you going? | ROADMAP.md pending | FAIL |
| 4 | R | Who are you? | MEDCHAT-COMMUNITY.md: scope description | PASS |
| 5 | O | How do you work? | This file | PASS |
| 6 | S | What shape are you? | inherits: TALK service, axiom present | PASS |
| 7 | L | What have you learned? | LEARNING.md pending | FAIL |
| 8 | LANG | How do you express? | LANGUAGE pending | FAIL |

Score: pending validation
```

ROADMAP.md (transparency — T dimension):

```
# MEDCHAT-COMMUNITY - ROADMAP

inherits: .

---

## Done
- GOV: TRIAD created. COMMUNITY tier achieved.

## Now
```

```
- Build to ENTERPRISE tier. Add COVERAGE, SPEC, ROADMAP.

## Next
- Add LEARNING.md. Achieve AGENT tier.
- Deploy to clinical pilot. Collect governance events.

---

*MEDCHAT-COMMUNITY | ROADMAP | HEALTHCARE*
```

```
git add MEDCHAT-COMMUNITY.md COVERAGE.md ROADMAP.md
git commit -m "GOV: MEDCHAT-COMMUNITY - COVERAGE + SPEC + ROADMAP"
magic validate
```

Score climbs. The scope now has reproducibility (R), operations (O), and transparency (T). Tier: ENTERPRISE. This is the minimum for clinical deployment — the compliance committee can review the COVERAGE assessment, the spec describes what the scope does, and the roadmap shows where it is going.

5.5 Add LEARNING — AGENT Tier

Create LEARNING.md (pattern capture — L dimension):

```
# LEARNING

inherits: .

---

Evidence lane for MEDCHAT-COMMUNITY.

## Patterns

| Date | Signal | Pattern | Source |
|------|--------|---------|--------|
| 2026-02-27 | GOV_FIRST | Governance files created before clinical deployment. | Step 0 |

---

*LEARNING | MEDCHAT-COMMUNITY | HEALTHCARE*
```

```
git add LEARNING.md
git commit -m "GOV: MEDCHAT-COMMUNITY - LEARNING"
magic validate
```

Score climbs to AGENT tier. The scope now learns from its operation — governance events are captured as LEARNING patterns. This is where clinical quality improvement begins.

5.6 Final Pass — 255

Close the vocabulary. Fix any structural gaps. Add the LANGUAGE inheritance. Ensure every SCREAMING_CASE term in every file resolves to a VOCAB.md definition:

```
# Fix COVERAGE.md - update FAIL to PASS for dimensions now satisfied
# Ensure all cross-references are valid
# Verify inherits: chains resolve
git add -A
git commit -m "GOV: MEDCHAT-COMMUNITY - close to 255"
magic validate
```

Score: 255. COIN minted: total 255. Tier: FULL [B-4].

The scope is fully governed. Every dimension satisfied. Every file present. Every term defined. Every constraint traceable to the axiom. The scope is ready for clinical deployment — not because someone signed off on it, but because the validator confirmed it. 255 is the proof.

5.7 The Gradient Rule

```
gradient = to_bits - from_bits
if gradient > 0: MINT:WORK(amount=gradient)
if gradient < 0: DEBIT:DRIFT(amount=abs(gradient))
if gradient = 0: no COIN (neutral drift)
```

Only improvement mints. Staying at 255 mints zero — there is nothing to improve. Going backward costs COIN through DEBIT:DRIFT. The gradient rule ensures that governance investment is economically visible and governance decay is economically penalized [P-8].

For the clinical informatics team: the total COIN minted for building this scope from 0 to 255 is exactly 255 COIN. That COIN is on the LEDGER. When the hospital's compliance committee asks "how much governance work has been done on MedChat?" — the answer is on the LEDGER, in COIN, auditable by anyone with access.

Chapter 6: Building a Scope

Anatomy, naming, child scopes.

6.1 Scope Anatomy

A scope is a directory with a CANON.md file. That is the only requirement. Everything else accumulates [G-1].

```
MY-SCOPE/
  CANON.md          ← D dimension (required)
  VOCAB.md          ← E dimension
  README.md         ← S dimension (with inherits: + axiom)
  {MY-SCOPE}.md     ← R dimension (the spec)
  COVERAGE.md       ← O dimension
  ROADMAP.md        ← T dimension
  LEARNING.md       ← L dimension
  SHOP.md           ← economic projection
  INTEL.md          ← scope intelligence
```

6.2 Naming Convention

GOV (`~/CANONIC/`) uses SCREAMING_CASE `.md` files. RUNTIME (`~/.canonic/`) uses lowercase [G-2].

Context	Convention	Example
SCOPE directory	SCREAMING_CASE	`SERVICES/LEARNING/`
LEAF content	lowercase-kebab	`code-evolution-theory.md`
EXTERNAL (GitHub slug)	lowercase	`canonic-python`
SERVICE directory	SINGULAR	`SERVICES/{SINGULAR}/`
INSTANCE directory	PLURAL	`{USER}/{INSTANCES}/`

Never mix singular and plural. SERVICE = SINGULAR. INSTANCE = PLURAL [G-1].

6.3 Child Scopes

A child scope inherits from its parent. Create a subdirectory with CANON.md:

```
PARENT/
  CANON.md
  CHILD/
    CANON.md     ← inherits: PARENT
```

The child's score starts at the parent's floor and accumulates upward [B-6].

6.4 Scope vs Leaf

A SCOPE has a CANON.md. A LEAF does not. Scopes are governance containers. Leaves are content [G-2].

```
DEXTER/            ← SCOPE (has CANON.md)
  BLOGS/           ← SCOPE (has CANON.md)
    2026-02-18-what-is-magic.md    ← LEAF (no CANON.md)
    2026-02-23-your-first-255.md   ← LEAF
```

Chapter 7: Building a Service

SERVICE = SINGULAR. 14 services. One composition.

7.1 The Service Constraint

Services project GOV into runtime products. Every service MUST compose the INTEL primitive. CHAT and COIN are optional [G-4].

```
Primitive → Service
INTEL     → LEARNING
CHAT      → TALK
COIN      → SHOP
```

Primitives are files. Services are directories [G-4].

7.2 The 14 Services

#	Service	Primitives	Role
1	LEARNING	INTEL	Governed discovery, IDF generalization
2	TALK	CHAT + INTEL	Contextual conversation agents
3	SHOP	COIN + INTEL	Public economic projection
4	LEDGER	COIN	Append-only economic truth
5	WALLET	COIN	Per-USER economic identity
6	VAULT	COIN + INTEL	Private economic aggregate
7	API	COIN	HTTP COIN operations
8	CHAIN	COIN	Cryptographic integrity, hash-linked events
9	MINT	COIN	Gradient minting, RUNNER tasks
10	IDENTITY	COIN	Ed25519 keys, KYC anchors
11	CONTRIBUTE	COIN + INTEL	External WORK, bronze/gold curation
12	NOTIFIER	CHAT + INTEL	Event notification, inbox delivery
13	MONITORING	INTEL	Runtime metrics, governance scoring
14	DEPLOY	COIN + INTEL	Governed artifact delivery, rollback

7.3 Service Directory Structure

```
SERVICES/
  LEARNING/
    CANON.md        ← SERVICE axiom
    LEARNING.md     ← SERVICE spec
    VOCAB.md
    README.md
    COVERAGE.md
    ...
```

Each service is a governed scope. No cross-scope leakage. Routes driven from governed indices, never hardcoded [G-4].

7.4 Instance vs Service

Service directories define schemas. Instance directories hold content. Instances live at USER scope, not nested in SERVICES/ [G-3].

```
SERVICES/WALLET/          ← schema (SINGULAR)
{USER}/WALLETS/           ← instances (PLURAL)
```

Chapter 8: Building a Product

SHOP.md. Cards. COST_BASIS.

8.1 SHOP.md

Every product is a governed scope compiled to 255, listed in SHOP.md with a Card [B-7].

```
## Card

| Field | Value |
|-------|-------|
| title | {Product Name} |
| type | {BOOK, PAPER, SERVICE, content} |
| price | {N} COIN |
| status | AVAILABLE |
| synopsis | {1-2 sentence description} |
| route | /{path/to/scope}/ |
```

8.2 Pricing by Tier

Tier	COIN Range	Audience
COMMUNITY	0-35	Everyone — free/near-free
ENTERPRISE	63	Business buyers
AGENT	127	Developers, governors
FULL	255	General public, flagship

8.3 COST_BASIS

The cost basis of a product is the total COIN minted by governance work producing that product [B-7]:

```
cost_basis(product) = SUM(MINT:WORK.amount)
  WHERE work_ref matches product scope
```

A book with 20 chapters, each 0-to-255: `20 * 255 = 5,100 COIN` cost basis.

Cost basis is derivable from the LEDGER by any user. Transparency is architectural [P-8].

8.4 Checkout Flow

1. Reader selects product. Price displayed in COIN.
2. System checks reader's WALLET balance.
3. SPEND event: reader debited, author credited. Both hash-chained.
4. Product access granted [B-7].

Chapter 9: Composition and Federation

Bilateral. Galaxy. User scopes. Multi-hospital governance at scale.

9.1 Composition

CANONIC composes vertically (inheritance) and horizontally (federation). Vertical composition is parent → child: constraints accumulate monotonically. Horizontal composition is org → org: governance is verified through hashes without sharing private data [B-13].

For health IT architects designing multi-hospital AI governance, these two composition axes map to the two primary integration patterns:

Vertical composition: A hospital deploys MammoChat. MammoChat inherits from the TALK service. The TALK service inherits from MAGIC. MAGIC inherits from canonic-canonic. The constraints accumulate from root to leaf. This is the deployment architecture — how governance flows from the standard to the clinical deployment.

Horizontal composition: Five hospitals in a health network each deploy MammoChat independently. Each hospital's MammoChat inherits from the same TALK service. Each hospital adds site-specific constraints. The hospitals share governance metadata (scores, hashes, tier status) without sharing clinical data (PHI, patient records). This is the federation architecture — how governance coordinates across organizational boundaries without violating data privacy.

9.2 Federation Architecture

Federation is privacy-preserving distributed governance across multiple organizations. In healthcare, federation is not optional — it is a HIPAA requirement. PHI stays local. Only governance metadata crosses organizational boundaries [B-13].

Stays Local (HIPAA-protected)	Gets Shared (Governance metadata)
Raw patient data (PHI)	Compliance scores (0-255)
Clinical records	Validation hashes (CHAIN)
API credentials	Tier status (COMMUNITY → FULL)
Internal patterns	Aggregate LEARNING signals
Employee records	COIN events (anonymized)
Business contracts	GALAXY topology

The federation boundary is architecturally enforced. The governance metadata that crosses organizational boundaries contains no PHI, no patient identifiers, no clinical data. It contains only governance artifacts: scores, hashes, tier statuses, and COIN events. A health network's CISO can verify this boundary by auditing the governance metadata format — it is defined in the CANON.md constraints of the federation scope.

9.3 Multi-Hospital Federation

For a five-hospital health network deploying CANONIC, the federation architecture looks like this:

```
canonic-canonic                  (root)
    network-canonic             (network ORG)
         hospital-a-canonic     (hospital A ORG)
              mammochat-a      (MammoChat at Hospital A)
         hospital-b-canonic     (hospital B ORG)
              mammochat-b      (MammoChat at Hospital B)
```

```
hospital-c-canonic      (hospital C ORG)
    mammochat-c        (MammoChat at Hospital C)
network-galaxy          (GALAXY frontend)
```

Each hospital's MammoChat is governed independently — validated against its own governance files, producing its own COIN, recording its own LEDGER. The federation layer aggregates governance metadata: the network's GALAXY shows all five MammoChat deployments, their scores, their tier statuses, and their COIN trajectories. The network CISO sees the aggregate governance posture. The hospital-level CISOs see their site-specific governance details. The federation preserves both the aggregate view and the local privacy.

9.4 Scale Evidence

The federation model is not theoretical. It is deployed. 1 developer. 19 GitHub organizations. 185+ repositories. All validate to 255 [B-13].

The scale proves that federation works at the organizational level — 19 independent organizations with independent governance, coordinated through shared governance metadata, validated by the same standard. For a health network with five hospitals, the federation model has been proven at nearly 4x the required scale.

9.5 ORG/USER Topology

ORG is the container. USER is the repo. The mapping to GitHub is direct: `github.com/{org}/{user}` [G-3].

```
canonic-canonic/              ORG (root)
  canonic-canonic.github.io   USER (platform frontend)
hadleylab-canonic/            ORG (proof org)
  hadleylab-canonic.github.io USER (proof frontend)
adventhealth-canonic/         ORG (hospital system)
  adventhealth-canonic.github.io USER (hospital frontend)
```

For clinical informatics engineers, the ORG/USER topology means that each hospital system in the federation is a separate GitHub organization with its own repositories, its own governance files, and its own validation pipeline. The governance metadata is shared through the federation layer. The clinical data never leaves the hospital's GitHub organization.

9.6 The GALAXY

The GALAXY renders the federated topology as an interactive visualization — the governance equivalent of a network operations center. Every ORG, every PRINCIPAL, every SERVICE, every SCOPE is visible as a node. Compliance rings show 8-dimension status using the DESIGN token system. INTEL flow pulses through edges where INTEL.md exists [G-5].

For a hospital board viewing the GALAXY, the visualization provides an immediate, intuitive understanding of the institution's AI governance posture — which deployments are at 255, which are still climbing, which have DEBIT:DRIFT events, and how the governance network connects across departments and sites.

For a health IT architect, the GALAXY is the debugging tool for governance architecture — when a deployment fails validation, the GALAXY shows its position in the inheritance tree, its parent constraints, and its specific dimension gaps. The governance architecture is visible, not just describable.

PART III — THE SERVICES

Chapter 10: LEARNING

INTEL primitive as service. Governed discovery. IDF generalization [G-11].

10.1 Axiom

LEARNING is INTEL applied. Every discovery governed. Every gradient evidenced [G-11].

10.2 The IDF Generalization

LEARNING generalizes the Invention Disclosure Form pattern beyond patents to all governed scopes. Every discovery — code pattern, compliance insight, architectural decision — is a governed LEARNING record [G-11].

10.3 LEARNING.md Pattern Table

```
| Date | Signal | Pattern | Source |
|------|---------|---------|--------|
| 2026-02-26 | EVOLUTION | Full rebuild. | Plan file |
| 2026-02-26 | NEW_CONSTRAINT | IP compliance. | CANON.md |
```

Signals: `EVOLUTION`, `NEW_CONSTRAINT`, `NEW_SCOPE`, `GOV_FIRST`, `EXTINCTION`.

10.4 Record Shape

Field	Content
Pattern	What was discovered
Date	When
Priority	Source evidence
Dimensions	8-dim validation mapping to 255
Assertions	Structured claims
Evidence	Provenance chain
References	Cross-scope links
Gradient	What changed (the delta)

10.5 Storage Architecture

Flat service layout. CAS fanout with git-style 2-char prefix buckets (`hash[:2]/hash[2:]`). Manifest sharding — one shard per signal type, thin index. Incremental discovery — checkpoint per repo, scan only new commits [G-11].

Hard limits: No single manifest file with 100K+ entries. No single flat CAS directory with 100K+ files.

10.6 Epoch Rotation

LEARNING.md rotates at epoch boundaries. Frozen epochs archive as `LEARNING-{EPOCH}.md`. After rotation, delete the archive file — the rotation event in active LEARNING.md is the record [G-3].

10.7 INTEL Sources

LEARNING ingests LEDGER streams: TALK, CONTRIBUTE, EMAIL, PROVISION are all INTEL sources [G-11].

Chapter 11: TALK

CHAT + INTEL composed. Governed conversation. The service that powers MammoChat, OncoChat, MedChat, LawChat, FinChat, and every clinical AI agent in the CANONIC ecosystem [G-12].

11.1 Axiom

TALK is CHAT + INTEL composed. Industry determines the voice. INTEL provides the knowledge [G-12].

This axiom establishes two binding obligations: (1) TALK must wire INTEL — the agent never speaks without governed knowledge backing its response; (2) the industry determines the voice — the agent speaks in the professional vocabulary of its domain, not in generic AI language.

11.2 The Composition

```
TALK = CHAT + INTEL
```

TALK must wire INTEL — never speak without knowledge. Industry determines the voice — never generic. This composition is the architectural formula for every clinical AI agent in the CANONIC ecosystem [G-12].

For clinical informatics engineers, this composition means that building a clinical TALK agent requires two things: a governed INTEL layer (the clinical evidence base) and a CHAT configuration (the conversation engine with clinical voice settings). The INTEL layer provides what the agent knows. The CHAT configuration determines how the agent speaks. The TALK service composes both into a governed conversation.

11.3 Building a Clinical TALK Agent

The pipeline from governed INTEL to clinical conversation follows a deterministic compilation path:

```
INTEL.md → compile → CANON.json { systemPrompt } → talk.js → clinical agent
```

Step 1: Wire the INTEL. The agent's knowledge comes from INTEL.md — the scope intelligence file that defines the evidence sources, the domain boundaries, and the cross-scope connections. For MammoChat, the INTEL.md references BI-RADS classifications, ACR guidelines, and breast imaging evidence. The INTEL is not injected at runtime. It is compiled into the agent's systemPrompt at build time.

Step 2: Compile the systemPrompt. The build pipeline reads the INTEL.md, extracts the scope intelligence, and compiles it into a systemPrompt that is embedded in the CANON.json output. The systemPrompt tells the agent what it knows, what it does not know, and what constraints it must obey. The systemPrompt is a governed artifact — derived from the INTEL, constrained by the CANON, and compiled by the build pipeline.

Step 3: Configure the CHAT. The TALK service's CHAT configuration sets the conversation parameters — the persona (clinical, legal, financial), the disclaimer (always displayed), the session management (every turn logged to LEDGER), and the PHI boundary (no patient identifiers in the conversation metadata).

Step 4: Deploy. The compiled agent serves clinical conversations through the TALK frontend — a governed conversation interface that displays the disclaimer, logs every interaction, and enforces the CANON constraints.

11.4 Channel Governance

Every TALK channel is governed by a CANON.md scope. Each channel has: - A `systemPrompt` derived from INTEL.md — the agent's knowledge boundary - A disclaimer (always displayed) — the governance notice that the agent is AI, not a clinician - A session ledger (every conversation turn recorded) — the audit trail - A PHI boundary — the architecturally enforced line between clinical data and governance metadata [G-12]

The channel governance is not optional. A TALK agent without a governed systemPrompt is an ungoverned chatbot. A TALK agent without a disclaimer is a liability risk. A TALK agent without a session ledger is a HIPAA audit failure. The governance is the architecture. Remove the governance, and the agent is not a TALK agent. It is an ungoverned AI chatbot — and in clinical healthcare, an ungoverned AI chatbot is dangerous.

11.5 Contextual Agents

Each scope with TALK enabled produces a contextual agent. The agent answers questions from that scope's INTEL, governed by that scope's axiom, in the voice of that scope's industry [G-12].

This means that EVERY governed scope in the CANONIC ecosystem can have a TALK agent — a conversational interface that answers questions about that scope's content, governed by that scope's axiom, in the voice appropriate to that scope's audience. A paper scope produces a paper agent that discusses the paper's findings. A service scope produces a service agent that explains the service's architecture. A book scope produces a book agent that navigates the book's content.

For clinical AI, contextual agents mean that each clinical evidence domain can have its own governed conversational interface:

Scope	Agent	Voice	INTEL Source
MammoChat	Breast imaging agent	Clinical-radiological	BI-RADS, ACR
OncoChat	Oncology agent	Clinical-oncological	NCCN, drug DBs
MedChat	General clinical agent	Clinical-general	UpToDate, DynaMed
LawChat	Legal research agent	Legal-formal	Case law, statutes
FinChat	Financial agent	Financial-regulatory	CMS, CPT, ICD-10

11.6 Persona Resolution

The CANON.md Persona table determines the agent's voice. The persona is not cosmetic — it is governance. A clinical agent must speak in clinical language. A legal agent must speak in legal language. The persona ensures that the agent's communication style matches the professional expectations of its audience:

Field	MammoChat Value	OncoChat Value	LawChat Value
tone	clinical	clinical	formal-legal
audience	radiologists, technologists	oncologists, pharmacists	attorneys, compliance
voice	clinical-third	clinical-third	legal-third
warmth	clinical-neutral	clinical-neutral	formal-neutral

The persona resolution is compiled from the CANON.md persona fields into the systemPrompt. The build pipeline does not guess the persona. It reads it from governance files. The persona IS governed.

11.7 The Disclaimer Architecture

Every TALK agent displays a disclaimer — a governance notice that the agent is AI-generated content, not a clinical diagnosis, not legal advice, not financial guidance. The disclaimer is not optional. It is a CANON.md constraint that the TALK service enforces [G-12].

For clinical TALK agents, the disclaimer is a clinical safety and legal liability requirement:

```
This is AI-generated clinical decision support. It is not a clinical diagnosis
or treatment recommendation. All clinical decisions must be made by qualified
healthcare professionals based on their independent clinical judgment and the
individual patient's clinical circumstances. Every recommendation cites its
evidence source for independent verification.
```

The disclaimer is displayed on every conversation turn. It cannot be hidden, minimized, or dismissed by the user. The TALK service enforces this display as a governance constraint, not a UI preference. In clinical AI, the disclaimer is the first line of liability protection — and the governance framework ensures it is never omitted.

11.8 Per-User Dashboard and Session Governance

Every USER principal gets a dashboard at `/TALK/{USER}/`. The dashboard shows the user's conversation history, active sessions, and COIN activity. Cross-user messages are delivered via governed inbox [G-12].

Session governance means that every conversation between a clinician and a TALK agent is a governed session — opened, recorded, and closed on the LEDGER. The session record includes: who initiated the session (IDENTITY), when the session started (timestamp), how many turns the session contained, what INTEL was cited, and when the session ended. The session record does NOT include PHI — patient identifiers, clinical data, and protected information stay in the clinical system. The session record contains only governance metadata.

For hospital compliance teams, session governance provides the audit trail that HIPAA §164.312(b) requires — a record of every AI-assisted clinical decision support interaction, timestamped, attributed, and preserved on the LEDGER. The audit trail is not reconstructed from server logs. It is produced by the governance architecture as a byproduct of normal operation.

Chapter 12: SHOP

COIN + INTEL. Public economic projection [G-4].

12.1 Axiom

SHOP projects governed products into the marketplace. SHOP.md is the public interface. Cards are the units [B-7].

12.2 SHOP.md Cards

```
## Card

| Field | Value |
|-------|-------|
| title | CANONIC-DOCTRINE |
| type | BOOK |
| price | 255 COIN |
| status | AVAILABLE |
| synopsis | The dev manual. |
| route | /BOOKS/CANONIC-DOCTRINE/ |
```

12.3 Listing Requirements

1. Scope compiled to 255.
2. SHOP.md created with Card.
3. Committed (governance work — may mint COIN).
4. Attestations received as readers purchase [B-7].

12.4 What Can Be a Product

Books, papers, blog posts, services, APIs, templates. Requirement: compiled to 255 [B-7].

Chapter 13: LEDGER

COIN. Append-only economic truth. The audit trail that IS the compliance [P-8].

13.1 Properties

The LEDGER is an immutable, append-only log of all governed activity. It does not track transactions like a bank. It tracks provenance: who did what, when, with what evidence, under what governance, why it mattered. The LEDGER is the single source of truth for the entire CANONIC economy — every COIN minted, every COIN spent, every drift event, every reconciliation [B-3].

For clinical informatics engineers, the LEDGER is the architectural answer to HIPAA §164.312(b) — "implement hardware, software, and procedural mechanisms that record and examine activity in information systems that contain or use electronic protected health information." The LEDGER IS those mechanisms. Every governed clinical AI interaction is a LEDGER event. Every event is timestamped, attributed, hash-linked, and append-only. The audit trail is not a separate system that records what the clinical AI does. It is the same system — governance and audit are architecturally unified.

13.2 Event Types

Event	Direction	Description	Healthcare Example
MINT:WORK	Credit	Gradient-based minting from governance work	Compliance officer advances MammoChat to ENTERPRISE tier
MINT:SIGNUP	Credit	500 COIN new-user bonus	New clinical informatics engineer onboards
MINT:PYRAMID	Credit	500 COIN referral bonus	Department refers colleague to governance program
DEBIT:DRIFT	Debit	Regression penalty	Unvalidated model update degrades OncoChat score
TRANSFER	Both	COIN movement (5% fee)	Hospital purchases INTEL layer from another institution
SPEND	Both	Product purchase	Clinician accesses governed clinical content
SETTLE	Debit	Fiat exit	Organization converts COIN to monetary value
CLOSE	Neutral	Monthly reconciliation	End-of-month governance accounting

Eight exhaustive events. No ninth event exists. The economy is closed by enumeration — there is no LEDGER event type that is not in this list. This enumeration is critical for auditability: when a HIPAA auditor asks "what types of events does this system record?" — the answer is these eight, and no others [P-8].

13.3 Event Shape

```
{
  "id": "a1b2c3d4e5f6",
  "prev": "z9y8x7w6v5u4",
  "ts": "2026-02-26T14:30:00Z",
  "event": "MINT:WORK",
  "user": "dexter",
  "amount": 92,
  "delta": 92,
  "work_ref": "hadleylab-canonic/DEXTER/BOOKS/CANONIC-DOCTRINE",
  "signature": "ed25519hex..."
}
```

Each event hash-linked to its predecessor via the `prev` field. Ed25519 signed after cutoff. The hash linkage means that if any event in the chain is modified — even a single character — every subsequent hash breaks. Tampering is detectable. The LEDGER is append-only by design and tamper-evident by cryptography [P-8].

13.4 The LEDGER as Healthcare Audit Trail

For hospital compliance teams, the LEDGER serves simultaneously as:

HIPAA audit trail: Every clinical AI interaction recorded with actor, timestamp, governance context, and outcome. The LEDGER satisfies §164.312(b) by architecture.

FDA 21 CFR Part 11 electronic records: Every LEDGER event is attributable (actor identified), legible (JSON format), contemporaneous (timestamped at event time), original (append-only, no modifications), and accurate (hash-verified). The LEDGER satisfies ALCOA by design.

SOX internal controls: Every financial governance event (FinChat interactions, coding decisions, claims processing) recorded with full provenance. The LEDGER provides the internal control documentation that financial auditors require.

Joint Commission quality records: Every governance event that contributes to clinical quality improvement is on the LEDGER. The quality improvement trail is complete, timestamped, and auditable.

One LEDGER. Four compliance standards satisfied simultaneously. The audit trail is not four separate systems. It is one governed event log that maps to every standard through the compliance matrix.

13.5 Querying the LEDGER

The LEDGER is queryable. Clinical informatics teams can extract governance analytics from the LEDGER for compliance reporting, quality improvement, and operational analysis:

```
# All MINT:WORK events for MammoChat in Q1 2026
SELECT * FROM ledger
  WHERE event = 'MINT:WORK'
  AND work_ref LIKE '%MAMMOCHAT%'
  AND ts BETWEEN '2026-01-01' AND '2026-03-31'
```

```
# Total DEBIT:DRIFT across all clinical AI scopes
SELECT SUM(amount) FROM ledger
  WHERE event = 'DEBIT:DRIFT'
  AND work_ref LIKE '%SERVICES/TALK%'

# Governance velocity for the institution
SELECT
  SUM(CASE WHEN event = 'MINT:WORK' THEN amount ELSE 0 END) as minted,
  SUM(CASE WHEN event = 'DEBIT:DRIFT' THEN amount ELSE 0 END) as drifted,
  minted - drifted as velocity
FROM ledger
  WHERE ts >= '2026-01-01'
```

These queries produce the governance metrics that hospital boards, compliance committees, and regulatory auditors need — governance velocity, drift rates, compliance trends, and economic return on governance investment. The metrics are derived from governed events, not from surveys or self-assessments.

Chapter 14: WALLET

COIN. Per-USER economic identity [P-8].

14.1 Append-Only Event Chain

No mutable balance field. Balance = SUM(credits) - SUM(debits). Hash-linked events [P-8].

```
balance(user) = SUM(credits) - SUM(debits) FROM timeline(user)
```

14.2 The 7 Invariants

#	Invariant	Enforcement
1	Append-only	No in-place edits
2	Chain-verifiable	Each event hash includes predecessor
3	Dual-write	USER timeline + ORG timeline + LEDGER
4	Derived balance	Computed, not stored
5	Ed25519-signed	After cutoff, every event signed
6	Monthly CLOSE	Mandatory reconciliation
7	Supply ceiling	Pre-MINT check against ceiling

14.3 Monthly CLOSE

```
1. For each USER: derive balance, compare to snapshot, flag MISMATCH.
2. Walk LEDGER chain: verify each event has corresponding WALLET event.
3. Append CLOSE event to each USER TIMELINE.
4. Report: total in circulation, unreconciled count, mismatches.
```

Chapter 15: VAULT

COIN + INTEL. Private economic aggregate [G-4].

15.1 Projection Files

SHOP.md = public (filesystem-discoverable). VAULT.md = private (auth-gated) [G-3].

15.2 VAULT Operations

VAULT aggregates COIN events and USER economic identity. Auth-gated access. The VAULT is the private counterpart to the public SHOP [G-4].

Chapter 16: API

HTTP COIN operations. Authentication [G-4].

16.1 Scope

API exposes COIN operations via HTTP. Authentication required. Every API call is a governed event — MINT, SPEND, TRANSFER, SETTLE — recorded in the LEDGER [G-4].

16.2 Constraints

Routes driven from governed indices. No hardcoded endpoints. Every response traces to a governance scope. Rate limiting derived from tier [G-4].

Chapter 17: CHAIN

Cryptographic integrity. Hash-linked events [G-4].

17.1 The Hash Chain

Every LEDGER event links to its predecessor via hash. The chain is verifiable: walk backward from any event to genesis. Integrity is structural, not policy [P-8].

17.2 Verification

```
verify(event) = hash(event.prev + event.data) == event.id
```

If any event in the chain is modified, every subsequent hash breaks. Tampering is detectable in O(n) [P-8].

Chapter 18: MINT

Gradient minting. RUNNER tasks [G-4].

18.1 The Gradient Rule

```
gradient = to_bits - from_bits
```

Positive gradient mints. Negative gradient debits. Zero gradient is neutral drift [P-8].

18.2 Maximum Mint Per Scope

```
max_mint(scope) = 255 - 0 = 255 COIN
```

A scope can mint at most 255 COIN over its lifetime. After reaching 255, all subsequent changes are neutral [P-1].

18.3 Supply Ceiling

```
SUPPLY_CEILING = unique_scopes * 255
```

No override. No quantitative easing. The only way to expand supply is to create new governance scopes — to do new work [P-8].

18.4 RUNNER Tasks

MINT operates via RUNNER tasks: discrete governance operations (validate, heal, commit) that produce gradients [G-4].

Chapter 19: IDENTITY

Ed25519 keys. KYC anchors. Onboarding [G-4].

19.1 Key Pairs

Every USER has an Ed25519 key pair. The public key is the on-chain identity. The private key signs LEDGER events [G-4].

19.2 KYC Anchors

IDENTITY anchors the USER to a real-world identity via KYC. The anchor is governance-verified, not self-declared. VITAE.md is the identity evidence gate [G-6].

19.3 Onboarding

```
1. Create USER scope.
2. Generate Ed25519 key pair.
3. Submit VITAE.md (identity evidence).
4. MINT:SIGNUP - 500 COIN credited.
5. magic validate → score.
```

Starting balance: 500 COIN. First governance work mints additional COIN [B-4].

Chapter 20: CONTRIBUTE

External WORK. Bronze/gold curation [G-4].

20.1 Scope

CONTRIBUTE governs external work — contributions from outside the core organization. Each contribution is curated: bronze (accepted) or gold (featured) [G-4].

20.2 Constraints

Every CONTRIBUTE action is COIN. Bronze contributions mint at standard rate. Gold contributions mint at elevated rate. All contributions are LEDGER events with full provenance [G-4].

Chapter 21: NOTIFIER

Event notification. Inbox delivery. Cross-user messaging [G-8].

21.1 Axiom

NOTIFIER delivers governed events to governed recipients. Every notification is a LEDGER event [G-8].

21.2 The Inbox Model

NOTIFIER stores messages in a KV-backed inbox keyed by recipient principal. Messages are immutable once delivered. Read-state is tracked per-message via acknowledgment [G-8].

21.3 Routes

```
POST /talk/send       → deliver message (auth required, rate: 10/hr)
GET  /talk/inbox      → read inbox for scope (auth required, rate: 100/hr)
POST /talk/ack        → acknowledge (mark read) messages
```

21.4 Record Shape

Field	Type	Content
id	string	Unique message identifier
ts	ISO-8601	Delivery timestamp
from	string	Sender principal
to	string	Recipient principal
message	string	Content body
read	boolean	Acknowledgment state

21.5 LEDGER Projection

Every send mints a LEDGER event: `type=NOTIFIER`, `key={sender}→{receiver}`, `work_ref=notifier:{id}`. Notification delivery IS economic work [G-8].

Chapter 22: MONITORING

Runtime metrics. Governance scoring. Real-time visibility [G-9].

22.1 Axiom

MONITORING is continuous governance scoring. Real-time visibility, not snapshots [G-9].

22.2 The /metrics Endpoint

```
GET /api/v1/metrics → Prometheus text exposition format
```

In-memory counters (no external dependencies):

Metric	Labels	Type
`canonic_api_requests_total`	endpoint, method, status	counter
`canonic_api_request_duration_seconds`	endpoint	summary (p50/p95/p99)
`canonic_auth_total`	result (ok/fail)	counter

22.3 Extended Health Check

```
GET /api/v1/health → {"status", "port", "checks": {"ledger_head", "vault_dir", "wallet_valid"}
```

Health checks verify live dependency state — not stale configuration [G-9].

22.4 Constraints

MUST NOT block service on metrics collection failure. Metrics are observability, not a gate. If metrics collection fails, the service continues — the failure itself becomes a metric [G-9].

Chapter 23: DEPLOY

Governed artifact delivery. Build validates, deploy ships, rollback recovers [G-10].

23.1 Axiom

DEPLOY is governed artifact delivery. Build before deploy. Never ship unvalidated artifacts [G-10].

23.2 The Pipeline

```
build → magic validate (255) → deploy (DESIGN first, then fleet)
```

DESIGN theme deploys first because fleet sites use `remote_theme`. Dependency order is architectural, not configurable [G-10].

23.3 Gates

Gate	Condition	Block
Build	build must pass	BLOCK_DEPLOY
Freeze	FROZEN state active	BLOCK_DEPLOY (unless override)
PRIVATE	PRIVATE scope in public fleet	BLOCK_DEPLOY

23.4 Rollback

```
rollback <site> [commit]      # Default: HEAD~1
```

Resets fleet site to previous commit. Uses `--force-with-lease` (safe force push). Prompts for confirmation [G-10].

23.5 Containerization

```
FROM python:3.11-slim
COPY bin/ VAULT/ LEDGER/ CONFIG/ SERVICES/
EXPOSE 8255
USER nobody
HEALTHCHECK CMD python3 -c "urllib.request.urlopen('http://localhost:8255/api/v1/health')"
```

Non-root. No secrets in image layers. Health check validates live endpoint [G-10].

PART IV — INTEL COMPILATION

Chapter 24: Cross-Axiomatic Validation

Axiom chains. Upward compilation. Evidence bridges.

24.1 The Compilation Chain

Every chapter's axiom compiles against its parent's axiom, which compiles against the book's axiom, which compiles against the BOOKS axiom, which compiles against the root:

```
Chapter axiom
  → Book axiom
    → DEXTER/BOOKS axiom
      → DEXTER axiom
        → hadleylab-canonic root
          → canonic-canonic root
```

A claim in Chapter 4 that contradicts the book's axiom fails validation — the same way a function that violates its module's type contract fails compilation [P-7].

24.2 Evidence Bridges

Cross-scope evidence flows through INTEL.md files. A claim in one scope can reference evidence from another scope via `references:` in frontmatter [G-3].

```
references:
  paper: PAPERS/governance-as-compilation
  blog: DEXTER/BLOGS/2026-02-18-what-is-magic
```

The validator resolves references and verifies the cited scope exists and compiles [P-7].

24.3 Upward Compilation

Validation walks upward. A child scope that compiles at 255 against a parent at 255 contributes to fleet-wide compilation. The fleet is a compilation unit [P-7].

Chapter 25: Contextual Agents

INTEL.md → CANON.json → systemPrompt → talk.js. The pipeline that turns governed knowledge into clinical AI agents.

25.1 The Agent Pipeline

Every clinical AI agent in the CANONIC ecosystem is produced by the same compilation pipeline. Governed INTEL goes in. A contextual agent comes out. The pipeline is deterministic — the same INTEL always produces the same agent:

```
INTEL.md (scope knowledge)
  → LEARNING.md (patterns)
    → Compiler (magic compile)
      → CANON.json {
          systemPrompt,
          breadcrumbs,
          brand,
          welcome,
          disclaimer
        }
        → talk.js (per-scope CHAT + INTEL agent)
```

Each scope with INTEL and TALK produces a contextual agent. MammoChat is produced by this pipeline from breast imaging INTEL. OncoChat is produced by this pipeline from oncology INTEL. MedChat is produced by this pipeline from general clinical INTEL. The pipeline is the same. The INTEL differs. The agent differs. The governance is identical [G-12][G-13].

25.2 The FHIR → INTEL → TALK → COIN Pipeline

For clinical informatics engineers integrating CANONIC with EHR systems, the agent pipeline extends to include FHIR resource composition — the bridge between clinical data systems and governed AI agents:

```
FHIR Resource (HL7 FHIR R4)
  → INTEL.md (governed knowledge unit)
    → systemPrompt (compiled agent knowledge)
      → TALK agent (clinical conversation)
        → COIN event (governance economics)
          → LEDGER (audit trail)
```

Step 1: FHIR → INTEL. Clinical data from EHR systems (Epic, Cerner, MEDITECH) is represented as HL7 FHIR resources — Patient, Observation, DiagnosticReport, MedicationRequest. These FHIR resources are composed into governed INTEL units. A Patient's screening history becomes INTEL. A DiagnosticReport's findings become INTEL. The FHIR data enters the governance framework through the INTEL layer.

Step 2: INTEL → systemPrompt. The governed INTEL units are compiled into the agent's systemPrompt by `magic compile`. The systemPrompt contains the agent's knowledge — what clinical evidence it can cite, what guidelines it references, what contraindications it knows about, what disclaimer it must display.

Step 3: systemPrompt → TALK agent. The compiled systemPrompt drives the TALK agent's behavior. When a clinician queries the agent, the agent answers from its systemPrompt — citing governed INTEL, speaking in the clinical voice defined by the persona, and enforcing the constraints defined in CANON.md.

Step 4: TALK → COIN. Every clinical conversation is a governed event. The event mints COIN. The COIN is on the LEDGER. The clinical governance labor is economically visible.

Step 5: COIN → LEDGER. The LEDGER records the entire pipeline — from the FHIR resource composition through the INTEL governance through the clinical conversation through the COIN event. The audit trail is complete. The provenance is transparent. The compliance is architectural.

This pipeline is the core healthcare architecture pattern. Every clinical AI deployment in CANONIC follows this pipeline. The variation is in Step 1 (what FHIR resources? what clinical domain?) and Step 2 (what evidence base? what guidelines?). Everything else is shared infrastructure.

25.3 systemPrompt Compilation

The `systemPrompt` is compiled from INTEL.md — not hand-written. The compilation reads the scope's INTEL (evidence sources, knowledge units, cross-scope connections), the scope's CANON (axiom, constraints, persona), and the scope's LEARNING (accumulated patterns) and produces a systemPrompt that:

- Declares the agent's identity (from the axiom)
- Defines the agent's knowledge boundary (from INTEL sources)
- Sets the agent's constraints (from CANON MUST/MUST NOT)
- Configures the agent's voice (from the persona table)
- Includes the disclaimer (always, non-optional)

The systemPrompt is a compiled artifact. Do not hand-edit it. If the agent says something wrong, fix the INTEL or the CANON — not the systemPrompt. The systemPrompt is output, not source [G-13].

25.4 Persona Resolution

Persona resolution determines how the agent speaks. For clinical agents, persona resolution ensures that a breast imaging agent speaks like a radiologist, not like a general-purpose chatbot:

Scope Type	Tone	Audience	Voice	Warmth
BOOK	narrative	readers	second-person	personal
PAPER	academic	researchers	third-person	formal
SERVICE (clinical)	clinical	clinicians	clinical-third	clinical-neutral
SERVICE (legal)	formal	attorneys	legal-third	formal-neutral
SERVICE (financial)	precise	finance team	business-third	professional

For SERVICE type scopes, persona resolution is industry-specific — determined by the scope's position in the governance tree and the domain constraints inherited from the parent. A MammoChat agent inherits clinical persona from the healthcare governance tree. A LawChat agent inherits legal persona from the legal governance tree [G-12].

25.5 Frontmatter Wiring

Each chapter or scope enables TALK via frontmatter:

```
---
talk: inline
---
```

The agent answers questions about that chapter's content, governed by that chapter's axiom, in the voice of that chapter's persona. Every chapter in this book, every chapter in CANONIC CANON, every paper, every blog post — each can have a contextual agent that serves as a governed conversational interface to its content [G-12].

For clinical informatics engineers, frontmatter wiring means that clinical evidence documents — treatment protocols, clinical practice guidelines, drug interaction databases — can each have a governed conversational agent that lets clinicians query the document through natural language, with every response sourced to the document's content and governed by the document's CANON.

Chapter 26: The _generated Contract

Compiler outputs. Provenance. Fix the contract, not the output.

26.1 The Rule

If a file is `_generated`, it was produced by the compiler. Do not hand-edit it. If the output is wrong, fix the compiler or the contract (CANON.md) — not the output [G-3].

26.2 Compiled Outputs

Source	Output	Location
CANON.md	CANON.json	`~/.canonic/`
GOV tree	scopes.json	`~/.canonic/_data/`
INTEL.md	systemPrompt	CANON.json
SHOP.md	product cards	CANON.json
DESIGN.md	DESIGN.css	Theme repo

26.3 Provenance

Every `_generated` file traces to its source governance file. The compilation pipeline is auditable: source → compiler → output [G-7].

Chapter 27: Scope Intelligence

Corpus. Sources. Cross-scope connections. Backpropagation.

27.1 INTEL.md

INTEL.md is the scope's intelligence file. It contains:

```
## Scope Intelligence

| Dimension | Value |
|-----------|-------|
| Subject | {what this scope knows} |
| Audience | {who consumes it} |
| Sources | {evidence corpus} |
| Status | {current state} |

## Evidence Chain

| Layer | Source | Count | Status |
|-------|--------|-------|--------|
| 1 | Governance sources | N | INDEXED |
| 2 | Papers | N | INDEXED |
| 3 | Blogs | N | INDEXED |

## Cross-Scope Connections

| Service | Role |
|---------|------|
| TALK | {how TALK wires to this scope} |
| COIN | {economic shadow} |
| LEDGER | {audit trail} |
```

27.2 Backpropagation

When a child scope's INTEL changes, the change propagates upward through the inheritance chain. Parent INTEL.md files may need updating. This is governance backpropagation — error signals flow upward [P-1].

27.3 Corpus Management

The evidence corpus is indexed in INTEL.md. Sources are cited as [B-XX] (blogs), [P-XX] (papers), [G-XX] (governance sources). Every claim traces to a citation [G-3].

PART V — DESIGN

Chapter 28: The Naming Convention

SCOPE/LEAF/EXTERNAL. GOV/RUNTIME.

28.1 The Two Worlds

World	Location	Convention	Files	Authors
GOV	`~/CANONIC/`	SCREAMING_CASE	`.md`	Human
RUNTIME	`~/.canonic/`	lowercase	any	Machine

GOV is the source. RUNTIME is the output. GOV drives RUNTIME. RUNTIME never writes GOV [G-2].

28.2 GOV Naming

Type	Convention	Example	Rule
SCOPE	SCREAMING_CASE	`SERVICES/LEARNING/`	Has CANON.md
LEAF	lowercase-kebab	`code-evolution-theory.md`	No CANON.md
EXTERNAL	lowercase	`canonic-python`	GitHub slug

28.3 SERVICE = SINGULAR

```
SERVICES/LEARNING/     ← SINGULAR (schema)
SERVICES/TALK/         ← SINGULAR (schema)
{USER}/WALLETS/        ← PLURAL (instances)
{USER}/LEDGERS/        ← PLURAL (instances)
```

Never mix singular and plural [G-1].

28.4 RUNTIME Naming

RUNTIME artifacts use lowercase. CANON.json, scopes.json, fleet.json. No SCREAMING_CASE in `~/.canonic/` [G-2].

Chapter 29: DESIGN Tokens

CSS token discipline. Sass partials.

29.1 Token Discipline

All visual values must use tokens. No magic numbers. No hardcoded colors [G-9].

Category	Token Pattern	Example
Spacing	`--space-*`	`--space-md`
Color	`--fg`, `--dim`, `--accent`, `--tx-*`, `--status-*`	`--accent`
Font size	`--font-*`	`--font-md`
Border radius	`--radius-*`	`--radius-sm`
Z-index	`--z-*`	`--z-modal`
Shadow	`--shadow-*`	`--shadow-card`
Transition	`--transition-*`	`--transition-fast`

Only `_TOKENS.scss` and `_THEMES.scss` may contain literal values [G-9].

29.2 The 23 Sass Partials

Ordered layers 0-19 [G-10]:

```
_TOKENS, _RESET, _LAYOUT, _GRID, _COMPONENTS, _DECK,
_UTILITIES, _ANIMATION, _RESPONSIVE, _THEMES, _TALK,
_CHAT, _POST, _GALAXY, _MOCK, _PRODUCTS, _TIERS,
_FOUNDATION, _SHOP, _FLEET, _NAV, _AUTH, _LATEX
```

29.3 DESIGN.css

One stylesheet for all surfaces. DESIGN.css is universal. No forking per surface. No hardcoded content in renderers [G-9].

```
DESIGN = CANON.md → CANON.json → HTML/Swift/Kotlin
DESIGN.css = universal renderer
```

29.4 Breakpoints

Three breakpoints. No more [G-9]:

Breakpoint	Width
Mobile	480px
Tablet	640px
Desktop	768px

All grid components collapse to `1fr` at 640px. WCAG AA text contrast required (4.5:1 minimum) [G-9].

Chapter 30: The CHAT Layer

_CHAT.scss. CUSTOM layout. Accent governance.

30.1 CUSTOM Layout

Chat surfaces use the CUSTOM layout. Styled exclusively by `_CHAT.scss` — no external CSS [G-9].

30.2 _CHAT.scss

```
// _CHAT.scss - the only file that styles chat surfaces
.chat-container { ... }
.chat-message { ... }
.chat-input { ... }
```

All CHAT components inherit tokens from `_TOKENS.scss`. Accent colors resolve from the scope's DESIGN.md [G-9][G-10].

30.3 Seven Layouts

Layout	Purpose
`default.html`	Standard page
`DECK.html`	Presentation deck
`CUSTOM.html`	Chat / interactive
`post.html`	Blog post
`paper.html`	Research paper
`book.html`	Book chapter

All share `HEAD.html` + `SCRIPTS.html` [G-10].

Chapter 31: The GALAXY Visualization

scopes.json. Shapes. Compliance ring.

31.1 Data Source

GALAXY renders from `scopes.json`, compiled by `build-scopes-json` from the GOV tree [G-5].

31.2 Visual Encoding

Element	Shape	Description
ORG	Icon + brand mark	Stars — gravitational anchors
PRINCIPAL	Icon + compliance ring	Flagship — governors
SERVICE	Icon + service glyph	Functional units
SCOPE	Dot/circle	Governance containers
VERTICAL	Icon + industry glyph	Knowledge domains
USER	Pill/text badge	Observers/affiliates

31.3 Compliance Ring

8 arc segments = 8 MAGIC dimensions. Filled = present. Gap = missing. 255 = complete ring, full glow [G-5].

31.4 Tier Glow

Tier	Score	Color	Glow
MAGIC	255	`#00ff88`	20px
AGENT	127+	`#2997ff`	12px
ENTERPRISE	63+	`#bf5af2`	8px
BUSINESS	43+	`#ff9f0a`	4px
COMMUNITY	35+	`#fbbf24`	2px
NONE	<35	`#ff453a`	0

31.5 INTEL Flow

Edges where INTEL.md exists pulse with green particles. Edges without INTEL are static and dim. Missing INTEL = missing LANG dimension = blocked at AGENT tier [G-5].

31.6 Brand Marks

CANONIC = (U+2229). HADLEYLAB = (U+2632) [G-5].

PART VI — ECONOMICS

Chapter 32: COIN and the WALLET

Event types. Supply ceiling. Conservation.

32.1 COIN = WORK

COIN is not cryptocurrency. COIN is a receipt — cryptographically signed, timestamped, attributed, permanently ledgered. WORK = COIN = PROOF [B-3].

32.2 Eight Circulation Events

#	Event	Direction	Mechanism
1	MINT:WORK	+	Governance gradient
2	MINT:SIGNUP	+	New user (500 COIN)
3	MINT:PYRAMID	+	Referral (500 COIN)
4	DEBIT:DRIFT	−	Score regression
5	TRANSFER	±	Movement (5% fee)
6	SPEND	±	Product purchase
7	SETTLE	−	Fiat exit
8	CLOSE	0	Monthly reconciliation

No ninth event. Economy closed by enumeration [P-8].

32.3 Supply Ceiling

```
SUPPLY_CEILING = unique_scopes * 255
```

32.4 Conservation Equation

```
Total(t) = Treasury(t) + Circulation(t) + Archived(t) - Burned(t)
```

Verifiable from LEDGER chain at any time [P-8].

Chapter 33: Gradient Minting

Delta rule. Positive/negative gradients.

33.1 The Rule

```
gradient = to_bits - from_bits
if gradient > 0: MINT:WORK(amount=gradient)
if gradient < 0: DEBIT:DRIFT(amount=abs(gradient))
if gradient = 0: no COIN (neutral drift)
```

33.2 Worked Example

Step	Action	From	To	Gradient	COIN
1	Bootstrap TRIAD	0	35	+35	MINT 35
2	Add COVERAGE + SPEC	35	127	+92	MINT 92
3	Add LEARNING + ROADMAP	127	224	+97	MINT 97

Step	Action	From	To	Gradient	COIN
4	Close to 255	224	255	+31	MINT 31
5	Neutral edit	255	255	0	—
6	Delete LEARNING.md	255	127	−128	DEBIT 128

Total minted: 255. Maximum for any scope [B-4][P-8].

33.3 DEBIT:DRIFT Asymmetry

Earning 128 COIN requires adding a major dimension (hard work). Losing 128 requires deleting one file (trivial destruction). Intentional design — destruction is easy but expensive [P-8].

Chapter 34: The SHOP

Checkout. Pricing. Attestation.

34.1 Product Listing Workflow

```
1. Compile scope to 255.
2. Create SHOP.md with Card.
3. Commit.
4. Receive attestations as readers purchase (SPEND events credit WALLET).
```

34.2 Fiat On-Ramp

Stripe integration converts dollars to COIN via MINT event. Author receives COIN, not dollars. Fiat is the on-ramp; COIN is the economy [B-7].

34.3 Economic Loop

```
Write governance → mint COIN →
List in SHOP → readers buy with COIN →
Author WALLET grows → readers become governors →
They mint COIN → list their products → SHOP grows
```

Chapter 35: COST_BASIS and Pricing

Formula. Examples. Constraints.

35.1 The Formula

```
cost_basis(product) = SUM(MINT:WORK.amount)
  WHERE work_ref matches product scope
```

35.2 Examples

Product	Scopes	Work	Cost Basis
Blog post	1 scope	$0 \to 255$	255 COIN
Book (20 chapters)	20 scopes	20×255	5,100 COIN
Service + 5 sub-scopes	6 scopes	6×255	1,530 COIN

35.3 Pricing Tiers

Tier	Price	Rationale
COMMUNITY	Free	Governance that excludes people isn't governance [B-3]
BUSINESS	$100/year	Builders who earn COIN deserve enterprise status
ENTERPRISE	Contract	Regulated operations need custom compliance
FOUNDATION	Free	Nonprofits at enterprise scale shouldn't pay [B-3]

PART VII — THE CLOSURE

Chapter 36: Governance as Type System

VOCAB = types. CANON = contracts. 255 = compiles.

36.1 The Isomorphism

Governance IS compilation. Not metaphorically. Structurally. Five components map one-to-one [P-7]:

Compilation	Governance
Source code	Structured Markdown (.md)
Grammar	CANON.md constraints
Type system	VOCAB.md (defined terms)
Entry point (`main()`)	Axiom
Header files (`.h`)	README.md (public interface)
Compiler	MAGIC validator
Target (machine code)	255-bit score
Linker	`inherits:` chain
Type error	Vocabulary violation
Compilation error	Missing dimension

36.2 VOCAB as Type System

An undefined term is a type error. Every SCREAMING_CASE term must resolve to a definition in VOCAB.md or an ancestor's VOCAB.md. The validator enforces vocabulary closure — three rules: VOCABULARY CLOSURE, INHERITANCE INTEGRITY, COVERAGE ALIGNMENT [P-7].

36.3 The Compilation Target

```
loss(scope) = 255 - score(scope)
```

The gradient is a vector across 8 binary dimensions. Each missing dimension contributes `2^i` to the loss. Fixing the highest-weighted missing dimension yields the steepest descent toward 255. This is literal gradient descent on a discrete landscape [P-7].

36.4 The Six Theorems

The CANONIC-PAPER [P-7] proves 6 theorems establishing the governance-compilation isomorphism: 1. Compiler Correspondence — governance maps isomorphically to compilation. 2. Validation Decidability — scope validity decidable in O(n). 3. Monotonic Accumulation — child cannot weaken parent. 4. Score Decomposition — 255 = sum of binary dimensions. 5. Gradient Convergence — `heal()` converges to 255. 6. Economic Coupling — compilation produces economic output.

Chapter 37: Governance as Compiler

Validation = type-checking. Scopes = modules.

37.1 The Compiler Pipeline

```
git commit → magic validate → 255-bit score → MINT:WORK → LEDGER → SHIP
```

Five phases [P-7]: 1. **Parse**: Read governance files, resolve `inherits:` 2. **Compile**: Compute score against target (255) 3. **Mint**: gradient $> 0 \rightarrow$ MINT:WORK; gradient $< 0 \rightarrow$ DEBIT:DRIFT 4. **Ledger**: Immutable record (build log) 5. **Ship**: Compiles = ships. Does not compile = does not ship. No waivers.

37.2 Error Taxonomy

Error Type	Compiler Equivalent	Example
Missing file	Syntax error	No CANON.md
Undefined term	Type error	SCREAMING_CASE term not in VOCAB.md
Broken inheritance	Linker error	`inherits: nonexistent/scope`
Missing dimension	Semantic error	Score < 255
Regression	Regression test failure	DEBIT:DRIFT

37.3 Continuous Governance

Pre-commit hook fires on every change. Feedback in seconds, not months. The audit is obsolete — the compiler is the auditor [P-7].

37.4 The Optimization Model

```
heal(scope) → identify missing dimensions → fix highest-weight first → revalidate
```

`heal()` operates as backpropagation. Forward pass = `validate()`. Loss = `255 - bits`. Backpropagation = `heal()`. Weight update = pattern adjustment [P-1].

Chapter 38: Governance as Version Control

git = LEDGER. Commits = COIN. Tags = certification.

38.1 Git IS the Governance Engine

Every governance event is a git commit. Every commit is validated. Every validation produces a score. Every positive gradient mints COIN. The LEDGER is the commit history with economic metadata [P-7].

38.2 Certification = Git Tags

`magic-tag` certifies a scope at 255. The tag is immutable, signed, auditable. Registered in TAGS.md — append-only [G-6].

```
magic-tag v1.0.0
```

Requirements: - `magic validate` → 255 - VITAE.md exists (identity gate) - Tag signed (no unsigned tags in production) - TAGS.md updated [G-6]

38.3 The Molecular Clock

Commits accumulate at a constant rate (mutation rate). The molecular clock is a chronometer for evolutionary distance between scopes [P-2].

```
Pattern accumulation rate  constant (linear in time)
```

Chapter 39: The LEARNING Closure

Accumulated intelligence subsumes every paradigm.

39.1 The Eighth Dimension

LEARNING (L) is the seventh bit. It represents accumulated intelligence — patterns, discoveries, corrections, epoch transitions. No programming language achieves this alone. Languages express. LEARNING accumulates [G-11].

39.2 What Languages Achieve

Every programming paradigm covers some governance dimensions:

Paradigm	Dimensions Covered	What's Missing
OOP (Java, C++, Python)	D, S (scopes, encapsulation)	L, LANG, T
Functional (Haskell, OCaml)	O, S (immutability, purity)	L, LANG, T
Type Systems (TypeScript, Rust)	E, LANG (types = vocab)	L, T, O
Concurrent (Go, Erlang)	O, T (message passing)	L, LANG
Logic (Prolog, Datalog)	D, E (axioms, derivation)	L, LANG, T
Smart Contracts (Solidity)	D, E, O (contracts, ledger)	L
Proof Assistants (Coq, Lean)	D, E, S (axioms, theorems)	L, T

Every paradigm has gaps. LEARNING closes them all [G-22].

39.3 The LEARNING Dimension

LEARNING is not syntax. It is not a feature. It is the accumulated intelligence of a governed scope — what it discovered, what it corrected, what it learned from its own evolution. LEARNING.md is the evidence [G-11].

No language provides LEARNING natively. CANONIC provides it as a governance dimension. Therefore, CANONIC governance subsumes every programming paradigm: it provides the eight dimensions that no single paradigm achieves alone [G-22].

Chapter 40: Why Every Language Needed Governance

OOP, FP, logic, formal — and what they missed.

40.1 The Gap

Programming languages solve expressiveness. They do not solve governance. A Haskell program can be pure, immutable, and provably correct — and still have zero governance. No axiom. No vocabulary closure. No compliance score. No COIN. No LEARNING [G-22].

40.2 The Three FOUNDATION Axioms

Axiom	Compiler Equivalent	Function
TRIAD	Syntax	Every scope declares its grammar

Axiom	Compiler Equivalent	Function
INHERITANCE	Scope resolution	Children inherit, no loopholes
INTROSPECTION	Type system	The system validates itself

These three axioms map governance to compiler theory constructs — the same constructs that every programming language implements. The difference: programming languages apply them to code. CANONIC applies them to governance. The abstraction level is one higher [P-7][G-22].

40.3 What They Missed

Language Family	What It Nailed	What It Missed
C	Systems, bare metal, FFI	Governance, types (weak), LEARNING
Python	Expressiveness, libraries	Governance, types (gradual)
Haskell	Purity, proofs	Governance, COIN, LEARNING
Rust	Safety, ownership	Governance, LEARNING
Solidity	Contracts, ledger	Governance above the chain
SQL	Data, queries	Governance, LEARNING

Every language covers part of the landscape. None covers governance. CANONIC is the layer that does [G-22].

Chapter 48: The Closure — CANONIC Against All of Programming

One chapter. The last chapter. The proof.

48.1 The Argument

1. **C is the kernel.** The CANONIC kernel (`magic.c`) is a C binary — 35KB, O(1) compliance checking via bitwise AND. C compiles to a shared library (`.so`/`.dylib`) exposing the C ABI [G-22].

2. **All languages neofunctionalize C through FFI.** In biology, neofunctionalization occurs when a gene duplicates and one copy acquires a new function. In CANONIC: the spec is invariant, the syntax is variant. Python wraps C via `ctypes`. Swift wraps C via `@_silgen_name`. TypeScript wraps C via `ffi-napi`. The wrapper never replaces the kernel [G-22].

3. **Four canonical runtimes.** C (bare metal), Python (server/AI), Swift (Apple/mobile), TypeScript (web/browser). Each passes 10/10 compliance tests. Each scores 255/255 coverage [G-22].

Runtime	FFI	CLI	Tests	Score
C	truth (native)	`magic.c`	10/10	255
Python	`ctypes`	`magic.py`	10/10	255
Swift	`@_silgen_name`	`magic.swift`	10/10	255

Runtime	FFI	CLI	Tests	Score
TypeScript	`ffi-napi`	`magic.ts`	10/10	255

4. **Eight-org language clade.** The phylogenetic tree: Python, TypeScript, Rust, Go, Swift, Kotlin, SQL, WASM — each a niche in the adaptive radiation. 19 total languages in the test spec. 4 closed. 15 TODO. Each follows the same pattern: FFI bridge → 10-test compliance → 255/255 coverage → LEDGER chain [P-3][G-22].

5. **Homology vs analogy.** TRIAD is homologous — inherited from common ancestor (canonic-canonic root). `validate()` across Python/Go/Rust is analogous — convergent under 255-bit selection pressure, different syntax, same function. If it calls the C kernel via FFI, it is homologous (shared truth). If it merely looks similar, it is analogous (convergent) [P-3].

6. **Every paradigm family maps to governance dimensions.** 20 families. 100+ languages. Every one mapped. Every one CLOSED. Detailed tables in Appendix E.

Family	Languages	CANONIC Mapping	Covered	Missing
Imperative & OOP	Java, C++, Python, Swift, Kotlin, Ruby, C#	scopes, inheritance, encapsulation	D, S	L
Functional	Haskell, OCaml, Elixir, Clojure, F#, Elm	immutability, composition, purity	O, S	L
Type Systems	TypeScript, Rust, Idris, Agda	VOCAB = types, CANON = contracts	E, LANG	L
Concurrent & Actor	Go, Erlang/OTP, Akka, Rust channels	LEDGER events, message passing	O, T	L
Logic & Constraint	Prolog, Datalog, MiniKanren, Mercury	axioms, derivation, resolution	D, E	L
Reactive & Dataflow	Rx, Flink, LabVIEW, Lucid	LEDGER streams, event sourcing	T, O	L
Concatenative	Forth, Factor, PostScript, Joy	stack composition, terseness	S	L
Array & Numeric	APL, J, K, NumPy, MATLAB, Julia, R	vectorized operations, data	R, O	L
Metaprogramming	Lisp, Racket, Zig comptime, Rust macros	code-as-data, compilation	D, R	L
Smart Contracts	Solidity, Vyper, Move, Clarity	CANON = contract, LEDGER = chain	D, E, O	L
Proof Assistants	Coq, Lean, Isabelle, TLA+, Alloy	axioms → theorems, CANON → 255	D, E, S	L

Family	Languages	CANONIC Mapping	Covered	Missing
Probabilistic	Stan, PyMC, Church, Turing.jl	uncertainty quantification	E, O	L
GPU & Parallel	CUDA, OpenCL, SYCL, Triton, Halide	parallel execution, throughput	R, O	L
DSLs & Config	Terraform, Puppet, Ansible, Dhall, Nix	infrastructure as code	D, R	L
Markup & Styling	HTML, CSS, Sass, YAML, TOML, JSON	structure, presentation	S, LANG	L
Query & Data	SQL, GraphQL, Datalog, SPARQL, Cypher	data access, relations	E, R	L
Shell & Scripting	Bash, Zsh, PowerShell, Fish, Make	automation, orchestration	T, R	L
Visual & Low-Code	Scratch, Blockly, Node-RED, Retool	accessibility, rapid prototyping	D, S	L
Systems & Bare Metal	C, Assembly, Zig, Rust	the kernel itself	ALL*	L†
LEARNING	**none alone**	**accumulated intelligence**	**L**	—

*Systems languages map to ALL dimensions at the implementation level — they ARE the kernel. †Even systems languages lack LEARNING at the governance level. Code does not learn. Governance does.

7. **LEARNING closes the loop.** Every family in the table above is missing the L dimension. LEARNING — accumulated intelligence — is the dimension no programming language achieves alone. A Haskell program does not learn from its own evolution. A Rust binary does not record its patterns. A SQL query does not know what it discovered yesterday. CANONIC governance provides LEARNING as a first-class dimension. Therefore:

CANONIC governance subsumes every programming paradigm.

The proof is not abstract. It is architectural: - C is the kernel (all languages bind to it via FFI) [G-22] - The spec is invariant (10 compliance tests, 255/255 score) [G-22] - The syntax is variant (each language's idioms) [P-3] - Every paradigm maps to dimensions (table above) [G-22] - Every paradigm is missing L [G-22] - LEARNING closes L - Therefore CANONIC governance CLOSES against all of programming

Q.E.D.

48.2 The Neofunctionalization Tree

```
C (kernel - LUCA)
```

```
  Python (ctypes) - server, AI, ancestral
  Swift (@_silgen_name) - Apple, mobile
  TypeScript (ffi-napi) - web, browser
  Rust (native) - safety, systems [TODO]
  Go (cgo) - concurrent, server [TODO]
  Zig (native) - systems, modern [TODO]
  Kotlin (JNI) - Android, JVM [TODO]
  Ruby (ffi gem) - scripting [TODO]
  Lua (luajit) - embedded [TODO]
  Julia (ccall) - scientific [TODO]
  Java (JNI) - enterprise [TODO]
  Elixir (NIF) - concurrent, fault-tolerant [TODO]
  Haskell (GHC FFI) - functional, pure [TODO]
  R (.Call) - statistics [TODO]
  C++ (native) - systems, OOP [TODO]
  WASM (emscripten) - universal binary [TODO]
  SQL (direct) - data [TODO]
  Shell (direct) - automation [TODO]
  Assembly (truth) - bare metal [TODO]
```

4 CLOSED. 15 TODO. The closure horizon: 19 languages, each neofunctionalizing C, each passing the 10-test compliance standard, each scoring 255/255 [G-22].

48.3 The Closure Matrix

The master matrix: 20 paradigm families × 8 governance dimensions. See Appendix E for the full table.

48.4 The Phylogenetic Proof

The 16-ORG phylogenetic tree [P-3]:

```
((canonic-canonic,canonic-foundation)GOV,
 (hadleylab-canonic,adventhealth-canonic)PROOF,
 (canonic-magic,canonic-apple)PLATFORM,
 (canonic-python,canonic-typescript,canonic-rust,canonic-go,
  canonic-swift,canonic-kotlin,canonic-sql,canonic-wasm)LANG,
 canonic-industries)ROOT;
```

Five clades. One LUCA. All surviving branches converge on 255 regardless of path. The tree is ultrametric: every tip equidistant from the fitness optimum [P-3].

48.5 LEARNING

Governance IS compilation. LEARNING IS the dimension that compilation alone cannot provide. Code compiles. Governance compiles AND learns. The LEARNING closure:

```
Evolution (P-1) → Mathematics (P-2) → Topology (P-3)
  → Compilation (P-7) → Economics (P-8) → LEARNING
```

Each layer inherits from the one below. The `inherits:` chain is the phylogenetic tree. The code is the evidence. LEARNING accumulates the intelligence. The loop closes.

PART VIII — TOOLCHAIN

Chapter 41: Overview

20+ tools. One pipeline. One direction.

41.1 The Transaction

```
.md → compile → .json → build → site → validate → 255
```

GOV compiles to RUNTIME. That is the only transaction. Governance drives code, never the reverse [G-7].

41.2 The 13 Core Tools

Tool	Transaction
`magic`	.md to score (0-255)
`magic-heal`	.md to settled .md (5-stage)
`build`	GOV tree to generated JSON + Jekyll sites
`build-scopes-json`	GOV tree to scopes.json (galaxy data)
`validate-design`	DESIGN.md 255 Map to theme artifacts (1:1 gate)
`deploy`	Built sites to pushed fleet
`install-hooks`	CANON constraints to git pre-commit enforcement
`magic-tag`	255 state to git tag + TAGS.md entry
`vault`	VAULT to COIN events + economic identity + onboard
`backup`	VAULT + LEDGER to encrypted snapshot (AES-256)
`rollback`	Fleet site to previous commit (force-with-lease)
`load-test`	Concurrent request testing + latency gates
`test-compiler`	Compiler integration test suite

Every tool reads governance, emits runtime. No tool writes governance [G-7].

Chapter 42: magic

C binary. validate/scan/commit/heal/ledger.

42.1 The Kernel

`magic.c` — 35KB C binary. O(1) compliance checking via single bitwise AND. Four verbs [G-22]:

```
magic validate          # Compute score for current scope
magic scan              # Discover all scopes, show scores
magic heal              # Identify missing dimensions
magic ledger            # Show COIN events
```

42.2 validate

```
$ magic validate
SCOPE: hadleylab-canonic/DEXTER/BOOKS/CANONIC-DOCTRINE
SCORE: 255/255
TIER: FULL
```

Reads CANON.md, resolves `inherits:`, checks all 8 dimensions, computes bitmask score [G-7].

42.3 scan

```
$ magic scan
canonic-canonic                   255  FULL
hadleylab-canonic                 255  FULL
hadleylab-canonic/DEXTER          255  FULL
hadleylab-canonic/DEXTER/BOOKS    255  FULL
...
```

Discovers all scopes in the GOV tree. No hardcoded paths. Discovery is structural [G-7].

42.4 heal

```
$ magic heal
MISSING: LEARNING.md (L dimension)
MISSING: ROADMAP.md (T dimension)
ACTION: Create LEARNING.md, ROADMAP.md
```

Identifies missing dimensions. Proposes fixes. Does not auto-fix — governance is human-authored [G-7].

Chapter 43: magic-heal

Python. Five-stage settlement.

43.1 The Five Stages

`magic-heal` is a Python tool that settles governance gaps through five stages:

```
1. SCAN     - identify all scopes below target
2. DIAGNOSE - classify missing dimensions
3. PROPOSE  - generate .md templates
4. SETTLE   - write files (with human approval)
5. VALIDATE - recompute scores
```

43.2 Usage

```
magic-heal hadleylab-canonic/DEXTER/BOOKS/
```

The healer walks the GOV tree, identifies scopes below 255, proposes governance files, settles with approval, and revalidates [G-7].

Chapter 44: build

Pipeline: scopes-json → generate → Jekyll → validate.

44.1 The Build Pipeline

```
build
```

This runs the full pipeline — 15+ stages, strict ordering [G-7]:

```
 0. build-toolchain      → .md contracts to .json runtime
 1. attest-services      → verify service completeness (strict gate)
 2. build-surfaces       → GOV → CANON.json + index.md per fleet site
 3. figures              → Figures tables → _data/*.json
 4. jekyll-exclude       → governance filenames → _config.yml
 5. build-shop-json      → SHOP.md → SHOP.json
 6. _generated markers   → annotate compiled outputs with contract provenance
 7. verify-intel-wiring  → INTEL propagates to all TALK surfaces
 8. validate-vocab       → VOCAB.md inheritance + dedup normalization
 9. validate-hygiene     → constraint dedup cleanliness gate
10. build-scopes-json    → scopes.json (galaxy data, must include magic://GALAXY)
11. build-galaxy-json    → galaxy.json (ORG/SERVICE/USER graph)
12. stripe sync          → Stripe → VAULT wallet sync
13. econ                 → publish wallets, verify wallet chains
14. magic validate       → 255 or reject
```

44.2 _generated Flag

Files produced by `build` are `_generated`. Do not hand-edit. If the output is wrong, fix the contract (CANON.md) or the compiler — not the output [G-3].

44.3 Deploy Order

DESIGN theme pushed first, then fleet sites pushed after. GitHub Pages fetches `remote_theme` at build time [G-10].

Chapter 45: Validation Errors and Healing

Diagnosis. Common issues.

45.1 Common Errors

Error	Cause	Fix
Score 0	No CANON.md	Create CANON.md with axiom
Score 35	TRIAD only	Add COVERAGE.md, {SCOPE}.md
Linker error	Broken `inherits:`	Fix path to parent scope
Type error	Undefined term	Add term to VOCAB.md
Regression	File deleted	Restore file or accept DEBIT:DRIFT

45.2 Diagnostic Flow

```
magic validate → score < 255?
  → magic heal → list missing dimensions
    → create missing files
      → git commit → magic validate → 255
```

45.3 The Heal Loop

```
while score < 255:
    missing = magic_heal(scope)
    for dim in missing:
        create_file(dim)
    score = magic_validate(scope)
```

Do not loop manually. Use `magic-heal` for automated diagnosis [G-7].

Chapter 46: The Build Pipeline

Full pipeline. CI/CD. _generated.

46.1 CI/CD Integration

Two workflows, strict ordering [G-7]:

magic-validate.yml — reusable validation gate:

```
# Runs magic validate, blocks if score < 255
```

magic-build.yml — main build + deploy pipeline:

```
name: MAGIC Build
on: push (main)
concurrency: { group: "pages", cancel-in-progress: false }
permissions: { contents: write, pages: write, id-token: write }

jobs:
  validate:    uses: ./.github/workflows/magic-validate.yml
  build:       needs: validate
    steps:
```

```
 1. Checkout GOV (submodules with GOV_TOKEN)
 2. Bridge GOV path (symlink $HOME/CANONIC)
 3. Checkout + install RUNTIME toolchain (~/.canonic)
 4. Checkout fleet sites (*.github.io repos)
 5. Checkout DESIGN theme repo
 6. Install deps (OpenSSL, Python3, cryptography)
 7. Compile magic.c kernel (cc -O2, smoke test)
 8. Prepare runtime directories
 9. MAGIC Build (build script - 15+ stages)
10. Compiler integration tests (test-compiler)
11. Freeze check (detect FROZEN interface)
12. PRIVATE leak gate (scan fleet for PRIVATE scopes)
13. Deploy fleet (DESIGN first, then sites - skip if FROZEN)
14. Build evidence (validation status, fleet state)
```

46.2 Pre-Commit Hooks

```
install-hooks
```

This installs git pre-commit hooks that run `magic validate` before every commit. Score < 255 blocks the commit [G-7].

46.3 The Full Pipeline

```
Author .md → git commit → pre-commit hook (magic validate) →
  build → scopes.json + CANON.json → jekyll → site →
    deploy → fleet push → validate → 255
```

Chapter 47: Advanced Tools

validate-design. enforce-magic-ip. magic-tag. vault.

47.1 validate-design

Validates DESIGN.md 255 Map against theme artifacts. 1:1 gate — every token in DESIGN.md must have a corresponding CSS variable [G-7].

```
validate-design
```

47.2 magic-tag

Certifies a scope at 255 with a git tag [G-6]:

```
magic-tag v1.0.0
```

Requirements: score = 255, VITAE.md exists, tag signed, TAGS.md updated.

47.3 vault

Manages VAULT operations: COIN events, USER economic identity, key rotation, onboarding pipeline [G-7].

```
vault onboard {user}                   # Onboard USER to economy
vault balance {user}                   # Current COIN balance
vault close --month 2026-02            # Monthly reconciliation
vault key-status --user {user}         # Ed25519 key age + rotation status
vault keygen --rotate --user {user}    # Rotate key pair, archive old
vault ledger {user}                    # USER LEDGER events
vault settle {user}                    # COIN → fiat settlement
```

47.4 enforce-magic-ip

Scans governance prose for kernel internals (bit weights, hex values, tier boundary scores). Any leak is a PRIVATE violation [G-3].

```
enforce-magic-ip
```

47.5 Production Hardening

Runtime services are hardened with seven layers. Every layer traces to a governance constraint [G-4].

CORS — Origin allowlist. API and Worker both reflect the `Origin` header only if it matches a governed origin. Wildcard (`*`) is forbidden in production. Origins: fleet sites + `api.canonic.org` + `*.canonic.org`.

Rate Limiting — API: in-memory, 60 requests/minute per IP. Worker: KV-backed, per-endpoint limits (chat: 60/hr, auth: 20/hr, email: 10/hr, checkout: 20/hr, omics: 200/hr). Exceeding limit returns 429.

CSP — Content Security Policy via `<meta>` tag in DESIGN HEAD.html. `default-src 'self'`, `connect-src` limited to `api.canonic.org` + `*.canonic.org`, `frame-ancestors 'none'`. Plus `X-Content-Type-Options: nosniff` and strict referrer policy.

Retry + Backoff — Exponential backoff with jitter for all external calls (GitHub OAuth, Stripe API, Resend). 3 attempts, 500ms base, 10s timeout. 5xx and network errors trigger retry. 4xx fails immediately.

Structured Logging — JSON to stderr (API) / `console.log` (Worker). Envelope: `{ts, level, service, endpoint, method, status, latency_ms}`. Every request logged. Every log is audit trail.

Graceful Shutdown — SIGTERM handler sets drain flag → new requests get 503 → in-flight requests complete → `server.shutdown()`. No dropped connections.

Key Rotation — Annual Ed25519 rotation. `vault key-status` warns at 330 days. `vault keygen --rotate` archives old keys as `KEY.pub.{date}`, generates new pair. Emergency rotation available for suspected compromise.

Containerization — `python:3.11-slim`, `pip install cryptography`, copies `bin/ VAULT/ LEDGER/ CONFIG/ SERVICES/`, `EXPOSE 8255`, `USER nobody`, healthcheck via `urllib.request.urlopen`. Non-root. No secrets in layers.

Backup — `backup snapshot` creates GPG AES-256 encrypted tar.gz of VAULT + LEDGER + SERVICES + learning. `backup restore` decrypts. `backup verify` validates LEDGER chain and

WALLET integrity.

BACK MATTER

Appendix A: Governance File Reference

File	Dimension	Purpose	Required
CANON.md	D	Axiom + constraints	YES
VOCAB.md	E	Term definitions	YES (for closure)
README.md	S	Public interface	YES
{SCOPE}.md	R	Scope specification	YES (for BUSINESS)
COVERAGE.md	O	8-question assessment	YES (for ENTERPRISE)
ROADMAP.md	T	Forward milestones	YES (for ENTERPRISE)
LEARNING.md	L	Pattern table	YES (for AGENT)
SHOP.md	—	Economic projection	Optional
INTEL.md	—	Scope intelligence	Optional
VITAE.md	—	Identity evidence	Required for certification
TAGS.md	—	Certification registry	Required for tagging

Appendix B: Tier Algebra

Dimension composition — no kernel internals [G-2].

Tier	Composition	Files Required
COMMUNITY	D + E + S	CANON.md + VOCAB.md + README.md
BUSINESS	COMMUNITY + R	+ {SCOPE}.md
ENTERPRISE	BUSINESS + T + O	+ ROADMAP.md + COVERAGE.md
AGENT	ENTERPRISE + L	+ LEARNING.md
FULL (MAGIC)	AGENT + LANG	+ LANGUAGE inherited

Tiers are cumulative. Monotonic accumulation. No skipping.

Appendix C: CLI Quick Reference

Command	Purpose
`magic validate`	Compute scope score
`magic scan`	Discover all scopes
`magic heal`	Diagnose missing dimensions

Command	Purpose
`magic ledger`	Show COIN events
`magic-heal`	Five-stage settlement
`build`	Full pipeline: JSON + Jekyll + validate
`build-scopes-json`	Generate scopes.json
`validate-design`	DESIGN.md CSS gate
`deploy`	Push fleet sites
`install-hooks`	Git pre-commit hooks
`magic-tag`	Certify with git tag
`vault`	COIN operations
`enforce-magic-ip`	Scan for kernel leaks

Appendix D: Naming Convention Quick Reference

Context	Convention	Example
SCOPE directory	SCREAMING_CASE	`SERVICES/LEARNING/`
LEAF content	lowercase-kebab	`code-evolution-theory.md`
EXTERNAL reference	lowercase	`canonic-python`
SERVICE directory	SINGULAR	`SERVICES/WALLET/`
INSTANCE directory	PLURAL	`{USER}/WALLETS/`
GOV files	`.md`	`CANON.md`
RUNTIME files	lowercase, any ext	`CANON.json`

Appendix E: The Closure Tables

20 paradigm families × 8 governance dimensions. 100+ languages mapped.

E.1: Imperative & OOP

Language	D	E	T	R	O	S	L	LANG	Notes
Java		·	·				·		Classes = scopes, interfaces = contracts
C++		·	·		·		·		Templates, namespaces, headers = README

Language	D	E	T	R	O	S	L	LANG	Notes
C#		·	·				·		Assemblies = scopes, attributes = metadata
Python		·	·		·		·		Modules = scopes, docstrings = README
Ruby		·	·		·		·		Mixins = inheritance, gems = packages
Swift		·	·				·		Protocols = contracts, modules = scopes
Kotlin		·	·				·		Data classes, sealed hierarchies

Covered: D (classes declare), S (encapsulation), R (identity), LANG (expression) **Missing:** L (no learning), T (no temporal governance), E (no vocabulary closure)

E.2: Functional

Language	D	E	T	R	O	S	L	LANG	Notes
Haskell			·				·		Types = VOCAB, purity = governance
OCaml			·				·		Modules = scopes, functors = composition
Elixir		·					·		Supervisors = governance, OTP = operations
Clojure		·			·		·		Immutability, persistent data, REPL

Language	D	E	T	R	O	S	L	LANG	Notes
F#			·				·		Type providers, computation expressions
Elm			·				·		No runtime errors, model-view-update
Erlang		·					·		OTP patterns, supervision trees

Covered: D, S (composition), O (immutability), LANG, some E (types), some T (temporal) **Missing:** L (no learning)

E.3: Type Systems & Dependent Types

Language	D	E	T	R	O	S	L	LANG	Notes
TypeScript			·		·		·		Types = VOCAB, interfaces = contracts
Rust			·				·		Ownership = governance, lifetimes = T
Idris			·				·		Dependent types, total functions
Agda			·				·		Full dependent types, proof objects
Coq			·				·		Calculus of constructions, extraction
Lean			·				·		Tactic proofs, mathlib

Covered: D, E (types = vocabulary), S, R, O, LANG **Missing:** L (no learning), T (limited temporal)

E.4: Concurrent & Actor

Language	D	E	T	R	O	S	L	LANG	Notes
Go		·					·		Goroutines, channels = LEDGER events
Erlang/OTP		·					·		Actors, supervision, let-it-crash
Akka		·					·		Actor model on JVM
Rust channels							·		Ownership + concurrency = safety

Covered: D, T (temporal/message ordering), O (operations), S, R, LANG **Missing:** L (no learning)

E.5: Logic & Constraint

Language	D	E	T	R	O	S	L	LANG	Notes
Prolog			·	·	·		·		Facts = axioms, queries = validation
Datalog			·	·			·		Stratified, decidable
MiniKanren			·	·	·		·		Relational, embedded
Mercury			·				·		Logic + types + modes

Covered: D (axioms), E (derivation), S (structure) **Missing:** L (no learning), T (no temporal), R (limited identity)

E.6: Reactive & Dataflow

Language	D	E	T	R	O	S	L	LANG	Notes
RxJS/RxJava		·		·			·		Observable streams = LEDGER events
Apache Flink		·					·		Event time, exactly-once
LabVIEW		·					·		Visual dataflow, instrumentation
Lucid		·		·	·		·		Intensional, historical values

Covered: T (streams, temporal), O (operations), S, D **Missing:** L (no learning), E (limited evidence)

E.7: Concatenative & Stack

Language	D	E	T	R	O	S	L	LANG	Notes
Forth		·	·	·	·		·		Stack, words = definitions
Factor		·	·		·		·		Quotations, combinators
PostScript		·	·	·	·		·		Page description, graphics
Joy		·	·	·	·		·		Pure concatenative, quotations

Covered: D (definitions), S (stack composition), LANG **Missing:** L, T, E, O, R

E.8: Array & Numeric

Language	D	E	T	R	O	S	L	LANG	Notes
APL/J/K		·	·	·			·		Array operations, vectorized

Language	D	E	T	R	O	S	L	LANG	Notes
NumPy		·	·				·		ndarray, broadcasting
MATLAB		·	·				·		Matrix operations, toolboxes
Julia		·	·				·		Multiple dispatch, JIT
R		·	·				·		Statistical modeling, CRAN

Covered: D, R (identity), O (operations), S, LANG **Missing:** L, T, E

E.9: Metaprogramming

Language	D	E	T	R	O	S	L	LANG	Notes
Lisp/Scheme		·	·		·		·		Homoiconicity, macros
Racket			·		·		·		Language-oriented, contracts
Zig comptime			·				·		Compile-time evaluation
Rust macros			·				·		Procedural + declarative macros
Template Haskell			·				·		Splice-time code generation

Covered: D (declaration), R (identity), S, LANG, some E, some O **Missing:** L, T

E.10: Smart Contracts

Language	D	E	T	R	O	S	L	LANG	Notes
Solidity							·		Contracts = CANON, blockchain = LEDGER
Vyper							·		Pythonic, security-first
Move							·		Resource types, ownership
Clarity							·		Decidable, no reentrancy

Covered: D, E, T (blockchain time), R, O, S, LANG — nearly full **Missing:** L (the chain records but does not learn)

E.11: Proof Assistants

Language	D	E	T	R	O	S	L	LANG	Notes
Coq			·				·		Calculus of constructions
Lean			·				·		Tactics, mathlib
Isabelle			·				·		Classical HOL
TLA+							·		Temporal logic, model checking
Alloy			·				·		Relational logic, bounded

Covered: D (axioms), E (proofs), S (formal structure), R, O, LANG, some T **Missing:** L (proofs do not learn from their evolution)

E.12: Probabilistic

Language	D	E	T	R	O	S	L	LANG	Notes
Stan			·				·		Bayesian inference, HMC
PyMC			·				·		Probabilistic models in Python
Church			·	·	·		·		Universal probabilistic language
Turing.jl			·				·		Julia-based inference

Covered: D, E (evidence/posterior), R, O, S, LANG **Missing:** L, T

E.13: GPU & Parallel

Language	D	E	T	R	O	S	L	LANG	Notes
CUDA		·	·				·		Thread blocks, kernels
OpenCL		·	·				·		Platform-independent
SYCL		·	·				·		C++ standard parallel
Triton		·	·				·		Python GPU compiler
Halide		·	·				·		Schedule/algorithm separation

Covered: D, R, O (parallel operations), S, LANG **Missing:** L, T, E

E.14: DSLs & Config

Language	D	E	T	R	O	S	L	LANG	Notes
Terraform		·					·		Infrastructure as code, state
Puppet		·	·				·		Declarative config management

Language	D	E	T	R	O	S	L	LANG	Notes
Ansible		·	·				·		Agentless, playbooks
Dhall			·				·		Typed, total, decidable
Nix			·				·		Reproducible, functional
Docker		·	·				·		Container images, layers

Covered: D (declarations), R (reproducible), O (operations), S, LANG **Missing:** L, some E, some T

E.15: Markup & Styling

Language	D	E	T	R	O	S	L	LANG	Notes
HTML		·	·	·	·		·		Semantic structure
CSS	·	·	·	·	·		·		Visual presentation
Sass	·	·	·	·	·		·		CSS with variables, nesting
YAML		·	·	·	·		·		Data serialization
TOML		·	·	·	·		·		Configuration
JSON	·	·	·	·	·		·		Data interchange

Covered: S (structure), LANG, some D **Missing:** L, T, E, R, O

E.16: Query & Data

Language	D	E	T	R	O	S	L	LANG	Notes
SQL			·				·		Relational, ACID
GraphQL		·	·		·		·		Schema = types, resolvers
Datalog			·	·			·		Facts + rules, stratified
SPARQL			·		·		·		Semantic web, triples
Cypher		·	·		·		·		Graph queries, Neo4j

Covered: D, E (evidence/data), R, O, S, LANG **Missing:** L, T

E.17: Shell & Scripting

Language	D	E	T	R	O	S	L	LANG	Notes
Bash		·			·	·	·		Pipelines, process control
Zsh		·			·	·	·		Extended Bash, completions
PowerShell		·					·		Object pipeline, cmdlets
Fish		·			·	·	·		User-friendly, autosuggestions
Make		·			·		·		Dependency rules, targets

Covered: D, T (temporal/sequential), R, LANG **Missing:** L, E, O, S (limited)

E.18: Visual & Low-Code

Language	D	E	T	R	O	S	L	LANG	Notes
Scratch		·	·	·	·		·		Visual blocks, educational
Blockly		·	·	·	·		·		Google visual blocks
Node-RED		·		·			·		Flow-based, IoT
Retool		·	·				·		Internal tools, data-binding

Covered: D, S (visual structure), LANG **Missing:** L, E, T (limited), R (limited)

E.19: Systems & Bare Metal

Language	D	E	T	R	O	S	L	LANG	Notes
C		·	·				·		The kernel. FFI substrate.

Language	D	E	T	R	O	S	L	LANG	Notes
Assembly		·	·				·		Machine truth. No abstraction.
Zig			·				·		Comptime, no hidden control flow
Rust			·				·		Ownership, lifetimes, safety

Covered: D, R, O, S, LANG, some E. The kernel itself. Systems languages implement governance dimensions at the hardware level. **Missing:** L. Even C does not learn. The kernel executes. Governance learns.

E.20: The Master Matrix

All 20 families × all 8 dimensions. Every cell: what it maps, what it lacks.

#	Family	D	E	T	R	O	S	L	LANG	Status
1	Imperative & OOP		·	·				·		CLOSED by L
2	Functional			·				·		CLOSED by L
3	Type Systems			·				·		CLOSED by L
4	Concurrent & Actor		·					·		CLOSED by L
5	Logic & Constraint			·	·	·		·		CLOSED by L
6	Reactive & Dataflow		·		·			·		CLOSED by L
7	Concatenative		·	·	·	·		·		CLOSED by L
8	Array & Numeric		·	·				·		CLOSED by L
9	Metaprogramming			·		·		·		CLOSED by L
10	Smart Contracts							·		CLOSED by L
11	Proof Assistants			·				·		CLOSED by L
12	Probabilistic			·				·		CLOSED by L
13	GPU & Parallel		·	·				·		CLOSED by L

#	Family	D	E	T	R	O	S	L	LANG	Status
14	DSLs & Config		·	·				·		CLOSED by L
15	Markup & Styling		·	·	·	·		·		CLOSED by L
16	Query & Data			·				·		CLOSED by L
17	Shell & Scripting		·			·	·	·		CLOSED by L
18	Visual & Low-Code		·	·	·	·		·		CLOSED by L
19	Systems & Bare Metal		·	·				·		CLOSED by L
20	**LEARNING**	·	·	·	·	·	·		·	**THE CLOSURE**

Every family is CLOSED by L.

LEARNING is the dimension no programming language achieves alone. CANONIC governance provides it. Therefore CANONIC governance closes against all of programming.

20/20 families CLOSED. 100+ languages mapped. Q.E.D.

Appendix F: INTEL Compilation Reference

The INTEL compilation chain:

```
1. INTEL.md           - scope knowledge (human-authored)
2. LEARNING.md        - pattern table (human-authored)
3. magic compile      - compiler reads both
4. CANON.json {       - compiled output
     systemPrompt,
     breadcrumbs,
     brand,
     welcome,
     disclaimer
   }
5. talk.js            - per-scope CHAT + INTEL agent
6. User asks question → agent answers from INTEL
```

Persona Resolution for BOOK Type

Field	Value
tone	narrative
audience	readers
voice	second-person

Field	Value
warmth	personal

Persona Resolution for SERVICE Type

Field	Value
tone	industry-specific
audience	domain users
voice	domain-appropriate
warmth	domain-appropriate

systemPrompt Sources

The `systemPrompt` is compiled from: 1. INTEL.md — scope intelligence 2. CANON.md — axiom + constraints 3. VOCAB.md — defined terms 4. Parent INTEL.md — inherited knowledge

---

Appendix G: References

Papers [P-XX]

Code	Title	Key Contribution
P-1	Code Evolution Theory	255-bit fitness function, drift/selection/inheritance
P-2	The Neutral Theory	Molecular clock, fixation probability, heterozygosity
P-3	Evolutionary Phylogenetics	16-ORG tree, 5 clades, mass extinction, ultrametric
P-4	OPTS-EGO	Four dimensions → eight
P-5	CANONIC Whitepaper	Framework overview
P-6	The $255 Billion Dollar Wound	Ghost labor, ungoverned AI cost
P-7	Governance as Compilation	Structural isomorphism, 6 theorems, build pipeline
P-8	Economics of Governed Work	COIN economy closure, 8 events, conservation
P-9	Content as Proof of Work	WORK = COIN = PROOF

Blogs [B-XX]

Code	Title	Key Contribution
B-1	What Is MAGIC?	Three primitives, 255-bit standard

Code	Title	Key Contribution
B-2	The GALAXY	Visualization, compliance ring
B-3	COIN = WORK	Economics, LEDGER, ghost labor
B-4	Your First 255	Onboarding, gradient minting walkthrough
B-5	Three Files One Truth	TRIAD, axiom authoring
B-6	Inherits: The Trust Chain	Inheritance, monotonic accumulation
B-7	SHOP: Your Work for Sale	Products, pricing, COST_BASIS
B-8	COIN for Humans	Economics narrative
B-9	The 255-Bit Promise	The standard
B-10	The Compiler Insight	Origin story, governance = compilation
B-11	Governance First	Gov first principle
B-12	Three Files	TRIAD narrative
B-13	Federation	Privacy-preserving distributed governance
B-14	Org/User	Topology

Governance Sources [G-XX]

Code	Source	Content
G-1	FOUNDATION/LANGUAGE.md	LANGUAGE spec, TRIAD, naming
G-2	MAGIC/DESIGN.md	Tier algebra, dimensions, naming convention
G-3	MAGIC/CANON.md	MAGIC constraints, primitives, projection
G-4	MAGIC/SERVICES/CANON.md	Services constraints, INTEL mandatory
G-5	MAGIC/GALAXY/CANON.md	Galaxy visual language, shapes, ring
G-6	MAGIC/COMPLIANCE/CERTIFICATION/CANON.md	Certification, git tags, VITAE gate
G-7	MAGIC/TOOLCHAIN/TOOLCHAIN.md	9 tools, pipeline, one direction
G-8	MAGIC/SURFACE/SURFACE.md	Surface/platform spec
G-9	MAGIC/SURFACE/DESIGN/CANON.md	DESIGN tokens, WCAG, breakpoints
G-10	MAGIC/SURFACE/JEKYLL/DESIGN.md	23 Sass partials, 132 artifacts
G-11	MAGIC/SERVICES/LEARNING/CANON.md	LEARNING service, IDF, CAS
G-12	MAGIC/SERVICES/TALK/CANON.md	TALK service, CHAT + INTEL
G-13	MAGIC/TOOLCHAIN/RUNTIME/RUNTIME.md	Runtime, talk.js, fleet.json
G-14–G-21	Service CANON.md + SPEC.md	Per-service governance
G-22	FOUNDATION/PROGRAMMING/	Neofunctionalization, 4 runtimes, FFI

Glossary

→ VOCAB.md

All terms defined in this book's VOCAB.md. Key terms:

Term	Definition
CANONIC	Governance framework. INTEL + CHAT + COIN. 255 bits.
MAGIC	The governance compiler. 8 dimensions.
TRIAD	CANON.md + VOCAB.md + README.md
INTEL	Knowledge primitive. Evidence provenance.
CHAT	Governed conversation primitive.
COIN	Attestation receipt for governed work. WORK = COIN.
CLOSURE	Governance maps onto and subsumes every paradigm.
NEOFUNCTIONALIZATION	All languages neofunctionalize C through FFI.
255	Maximum 8-bit score. Full compliance. Deploys.
LEARNING	Accumulated intelligence. The eighth dimension.

Colophon

THE CANONIC DOCTRINE was produced under MAGIC 255-bit governance.

Every chapter is a governed INTEL unit. Every claim traced to a citation. Every commit minted COIN. The act of writing this manual was itself governed work — validated, scored, ledgered.

```
Scope:     hadleylab-canonic/DEXTER/BOOKS/CANONIC-DOCTRINE
Score:     255/255
Tier:      FULL
Compiler:  magic validate
Pipeline:  .md → .json → site → 255
```

Built with CANONIC. Validated by MAGIC. Every word COIN.
