The CANONIC Doctrine

CANONIC BOOK SERIES

Dexter Hadley

CANONIC Foundation

Status: **PUBLISHED**

hadleylab.org/BOOKS — Governed Publishing. Every word sourced.

The CANONIC Doctrine
Dexter Hadley

© 2026 CANONIC Foundation. All rights reserved.

Published by CANONIC Foundation.

Contents

0.1 Abstract 31
0.2 Foreword 32

PART I — FOUNDATIONS **34**

1 Chapter 1: The Axiom **36**
1.1 1.1 The Contract 36
1.2 1.2 Clinical Axioms 37
1.3 1.3 Good vs Bad Axioms 37
1.4 1.4 Derivation 38
1.5 1.5 The Axiom as Entry Point 39
1.6 1.6 Axiom Anti-Patterns 39
1.7 1.7 The Three Primitives 40
1.8 1.8 Governance-First Principle 41
1.9 Constraints 42
1.10 1.11 Axiom Versioning 44
1.11 1.12 The Axiom in Production 44
1.12 1.13 Axiom Composition Patterns 45
1.13 1.14 Axiom Testing: The Five-Question Checklist 46

2 Chapter 2: The TRIAD **48**
2.1 2.1 The Three Files 48
2.2 2.2 CANON.md Structure 49
2.3 Axiom 49
2.4 Constraints 49
2.5 Constraints 50
2.6 Axiom 53
2.7 Constraints 53
2.8 2.8 TRIAD Validation Errors and Fixes 55
2.9 2.9 VOCAB.md: The Type System in Practice 57

2.10 2.10 README.md: The Public API . . . 57
2.11 2.11 The TRIAD Completeness Theorem . . . 58
2.12 2.12 TRIAD Anti-Patterns . . . 58

3 Chapter 3: The Inheritance Chain 60

3.1 3.1 Syntax . . . 60
3.2 3.2 Resolution . . . 60
3.3 3.3 Monotonic Accumulation . . . 61
3.4 3.4 Three Forms of Inheritance . . . 61
3.5 3.5 The Root . . . 62
3.6 3.6 Cross-Organization Inheritance . . . 62
3.7 3.7 The Healthcare Inheritance Tree . . . 63
3.8 3.8 Building an Inheritance Chain: Step-by-Step . . . 63
3.9 Axiom . . . 64
3.10 Constraints . . . 64
3.11 Axiom . . . 65
3.12 Constraints . . . 65
3.13 Axiom . . . 66
3.14 Constraints . . . 66
3.15 3.9 Inheritance Errors and Resolution . . . 67
3.16 3.10 Constraint Propagation Mechanics . . . 68
3.17 3.11 Practical Inheritance Patterns . . . 68
3.18 3.12 Debugging Inheritance Problems . . . 70
3.19 3.13 Inheritance Performance and Depth Limits . . . 70
3.20 3.14 Inheritance and the _generated Contract . . . 71
3.21 3.15 Multi-Fleet Inheritance . . . 71
3.22 3.16 Inheritance Anti-Patterns . . . 72

4 Chapter 4: The Eight Dimensions 73

4.1 4.1 The Dimensions . . . 73
4.2 4.2 The Score . . . 74
4.3 4.3 Compliance Tiers . . . 74
4.4 4.4 Clinical Dimension Mapping . . . 74
4.5 4.5 COVERAGE.md . . . 75
4.6 4.6 Kernel Internals . . . 76
4.7 4.7 Dimension-by-Dimension Walkthrough . . . 76
4.8 4.11 Dimension Validation Quick Reference . . . 82

5 Chapter 5: Your First 255 **83**

5.1 5.1 Create the Scope . . . 84
5.2 5.2 Write the TRIAD . . . 84
5.3 Constraints . . . 84
5.4 5.4 Add Community and Practice — ENTERPRISE Tier . . . 86
5.5 5.5 Add LEARNING — AGENT Tier . . . 87
5.6 5.6 Final Pass — 255 . . . 88
5.7 Alignment with GETTING_STARTED.md . . . 88
5.8 5.7 The Gradient Rule . . . 89
5.9 5.8 Common Build Errors and How to Fix Them . . . 89
5.10 5.9 The Build Session: Timing and Workflow . . . 91
5.11 5.10 Validating a Real Clinical Deployment . . . 91
5.12 Axiom . . . 92
5.13 Constraints . . . 92
5.14 5.11 After 255: Maintaining Compliance . . . 93
5.15 5.12 The Score Trajectory . . . 94
5.16 5.13 Cross-Scope Validation . . . 95
5.17 5.14 Post-255 Checklist . . . 95
5.18 5.15 The Clinical Economics of 255 . . . 96

6 Chapter 6: Building a Scope **97**

6.1 6.1 Scope Anatomy . . . 97
6.2 6.2 Naming Convention . . . 97
6.3 6.3 Child Scopes . . . 98
6.4 6.4 Scope vs Leaf . . . 98
6.5 6.5 Full Scope Build: mkdir to 255 . . . 98
6.6 Axiom . . . 99
6.7 Constraints . . . 99
6.8 magic heal: Scaffolding a Scope . . . 104
6.9 Bloat Constraints . . . 104
6.10 6.6 Scope Organization Best Practices . . . 105
6.11 6.7 Scope Deletion and Archival . . . 105
6.12 6.8 Scope Discovery . . . 106
6.13 6.9 Scope Migration . . . 106
6.14 6.10 Scope Templates . . . 107
6.15 6.11 Scope Metrics . . . 107

6.16 6.12 Scope Lifecycle States . . . 107
6.17 6.13 Clinical Vignette: Scope Design for a Multi-Specialty Hospital . . . 108
6.18 6.14 Scope Sizing Guidelines . . . 109
6.19 6.15 Governance Proof: The Scope as Governance Atom . . . 110
6.20 6.16 Scope Migration and Refactoring . . . 111

7 Chapter 7: Building a Service 112

7.1 7.1 The Service Constraint . . . 112
7.2 7.2 The 14 Services . . . 112
7.3 7.3 Service Directory Structure . . . 114
7.4 7.4 Instance vs Service . . . 114
7.5 7.5 Building a Service: Step-by-Step . . . 114
7.6 Constraints . . . 115
7.7 7.6 Service Composition Patterns . . . 116
7.8 7.7 Service Governance Constraints . . . 117
7.9 7.8 Service HTTP Routes . . . 117
7.10 7.11 Service Anti-Patterns . . . 119
7.11 7.12 Clinical Service Example: TALK . . . 119
7.12 7.13 Clinical Vignette: Building the CONTRIBUTE Service for Clinical Trials . . . 120
7.13 Constraints . . . 120
7.14 7.14 Service Lifecycle and Deprecation . . . 121
7.15 7.15 Service Metrics . . . 122
7.16 7.16 Governance Proof: Service Composition . . . 122
7.17 7.17 Service Contract Completeness Checklist . . . 123
7.18 7.18 Service Dependency Graph . . . 124
7.19 7.19 Service Templates for Rapid Bootstrap . . . 124
7.20 7.20 Clinical Vignette: Service Composition Prevents Governance Gap . . . 125
7.21 7.21 The Service as Governance Primitive . . . 126

8 Chapter 8: Building a Product 127

8.1 8.1 SHOP.md . . . 127
8.2 8.2 Pricing by Tier . . . 127
8.3 8.3 COST_BASIS . . . 128
8.4 8.4 Checkout Flow . . . 128
8.5 8.5 Building a Product: Step-by-Step . . . 128
8.6 8.6 Product Types . . . 130
8.7 8.7 Product Validation Rules . . . 130

8.8 8.8 Multi-Scope Products . . . 130
8.9 8.9 Product Economics . . . 130
8.10 8.10 Product Discovery . . . 131
8.11 8.11 Clinical Product Examples . . . 131
8.12 8.12 Product Lifecycle . . . 131
8.13 8.13 Clinical Vignette: Product Launch at a Regional Health System . . . 132
8.14 8.14 Product Bundling and Composition . . . 134
8.15 8.16 Product Versioning . . . 135
8.16 8.17 Governance Proof: The Product Chain . . . 135
8.17 8.18 Product Composition Patterns . . . 136
8.18 8.19 Product Quality Metrics . . . 137
8.19 8.20 Product Licensing and Reuse Rights . . . 138
8.20 8.21 Clinical Product Example: Nursing Governance Suite . . . 139

9 Chapter 9: Composition and Federation 140

9.1 9.1 Composition . . . 140
9.2 9.2 Federation Architecture . . . 140
9.3 9.3 Multi-Hospital Federation . . . 141
9.4 9.4 Scale Evidence . . . 141
9.5 9.5 ORG/USER Topology . . . 142
9.6 9.6 The GALAXY . . . 142
9.7 9.7 Federation Protocol . . . 142
9.8 9.8 Composition Depth . . . 143
9.9 9.9 Building a Federation: Step-by-Step . . . 143
9.10 Constraints . . . 144
9.11 9.10 Federation vs Centralization . . . 145
9.12 9.11 Clinical Vignette: Federation During a Joint Commission Survey . . . 145
9.13 9.12 Federation Economics . . . 146
9.14 9.13 Federation Conflict Resolution . . . 147
9.15 9.14 Federation Monitoring . . . 147
9.16 9.15 Governance Proof: Federation Correctness . . . 148
9.17 9.16 Federation Onboarding Protocol . . . 148
9.18 9.17 Federation Governance Tiers . . . 149
9.19 9.18 Federation Data Sovereignty . . . 150
9.20 9.19 Clinical Vignette: Federation Enables Multi-Site Clinical Trial Governance . . . 150
9.21 FEDERATION: Operational . . . 151

9.22 The WITNESS Protocol . . . 152

10 Chapter 10: LEARNING 153

10.1 10.1 Axiom . . . 153
10.2 10.2 The IDF Generalization . . . 153
10.3 10.3 LEARNING.md Pattern Table . . . 154
10.4 10.4 Record Shape . . . 155
10.5 10.5 Storage Architecture . . . 155
10.6 10.6 Epoch Rotation . . . 156
10.7 Epoch Rotation: Operational . . . 157
10.8 10.7 INTEL Sources and Cross-Scope Propagation . . . 157
10.9 10.8 The LEARNING Question and MAGIC Closure . . . 158
10.10 10.9 CLI Operations . . . 158
10.11 10.10 Error Handling and Failure Modes . . . 159
10.12 10.11 Clinical Vignette: MammoChat NCCN Guideline Update . . . 160
10.13 10.12 Clinical Vignette: Tamoxifen-to-Aromatase Inhibitor Protocol Shift . . . 161
10.14 10.13 Network Map and VITAE Integration . . . 162
10.15 10.14 The Onboarding Pipeline . . . 163
10.16 10.15 Closure . . . 163

11 Chapter 11: TALK 164

11.1 11.1 Axiom . . . 164
11.2 11.2 The Composition . . . 164
11.3 11.3 Building a Clinical TALK Agent . . . 165
11.4 11.4 Channel Governance . . . 165
11.5 11.5 Contextual Agents . . . 166
11.6 11.6 Persona Resolution . . . 166
11.7 11.7 The Disclaimer Architecture . . . 167
11.8 11.8 Per-User Dashboard and Session Governance . . . 167
11.9 11.9 CLI Operations and Build Pipeline . . . 168
11.10 11.10 The TALK Data Flow . . . 168
11.11 11.11 Cross-User Messaging . . . 169
11.12 11.12 COIN Minting per Session . . . 170
11.13 11.13 Error Handling and Failure Modes . . . 170
11.14 11.14 Clinical Vignette: MammoChat BI-RADS 4A Triage . . . 171
11.15 11.15 Clinical Vignette: OncoChat Pembrolizumab Eligibility . . . 172
11.16 11.16 View Architecture . . . 172

11.17 11.17 Community Learning Dashboard . . . 173
11.18 11.18 Closure . . . 173

12 Chapter 12: SHOP 175

12.1 12.1 Axiom . . . 175
12.2 12.2 Discovery Architecture . . . 175
12.3 12.3 SHOP.md Cards . . . 176
12.4 12.4 Pricing by Tier . . . 177
12.5 12.5 The Composable Frontmatter Pattern . . . 177
12.6 12.10 The Catalog Data Structure . . . 180
12.7 12.11 Bag Review Pattern . . . 181
12.8 12.12 Error Handling and Failure Modes . . . 182
12.9 12.13 Clinical Vignette: Hospital Network Procurement . . . 182
12.10 12.14 Clinical Vignette: Score Regression and Drift Gate . . . 183
12.11 12.15 Per-Page Wallet Constraint . . . 184
12.12 12.16 Product Type Taxonomy . . . 184
12.13 12.17 Closure . . . 185

13 Chapter 13: LEDGER 186

13.1 13.1 Properties . . . 186
13.2 13.2 Event Types . . . 186
13.3 13.3 Event Shape . . . 187
13.4 13.4 The LEDGER as Healthcare Audit Trail . . . 187
13.5 13.5 Querying the LEDGER . . . 188
13.6 13.6 CLI Operations . . . 189
13.7 13.7 Storage Architecture . . . 190
13.8 13.8 The CLOSE Event and Monthly Reconciliation . . . 190
13.9 13.9 The TRANSFER Event and Fee Structure . . . 191
13.10 13.10 The DEBIT:DRIFT Event . . . 192
13.11 13.11 Error Handling and Failure Modes . . . 193
13.12 13.12 Clinical Vignette: HIPAA Audit Response . . . 194
13.13 13.13 Clinical Vignette: Governance Velocity Dashboard . . . 194
13.14 13.14 The SETTLE Event and Fiat Exit . . . 195
13.15 13.15 LEDGER Capacity Planning . . . 195
13.16 13.16 Closure . . . 196

14 Chapter 14: WALLET 197

14.1 14.1 Axiom . . . 197

14.2 14.2 Append-Only Event Chain . . . 197
14.3 14.3 The 7 Invariants . . . 198
14.4 14.4 Dual-Write Architecture . . . 198
14.5 14.5 Monthly CLOSE . . . 199
14.6 14.6 The WALLET in Healthcare Governance . . . 199
14.7 14.7 WALLET CLI Operations . . . 199
14.8 14.8 WALLET Error Handling . . . 200
14.9 14.9 WALLET Data Structure . . . 200
14.10 14.10 WALLET Capacity . . . 201
14.11 14.11 Clinical Vignette: Governance Portfolio . . . 201
14.12 14.12 WALLET and Regulatory Reporting . . . 201
14.13 14.13 Clinical Vignette: WALLET Forensics After a Governance Incident . . . 202
14.14 14.14 WALLET Cross-Reference Verification . . . 203
14.15 14.15 WALLET Privacy Architecture . . . 203
14.16 14.16 WALLET Lifecycle States . . . 204
14.17 14.17 WALLET Performance Characteristics . . . 204
14.18 14.18 Governance Proof: The WALLET as Financial Instrument . . . 205
14.19 14.19 WALLET Migration Between Organizations . . . 205
14.20 14.20 WALLET Consolidation for Multi-Role Identities . . . 206
14.21 14.21 WALLET Delegation . . . 207

15 Chapter 15: VAULT 208

15.1 15.1 Axiom . . . 208
15.2 15.2 The Dual Projection Model . . . 208
15.3 15.3 VAULT Structure . . . 209
15.4 15.4 Auth Architecture . . . 209
15.5 15.5 VAULT CLI . . . 210
15.6 15.6 VAULT and Clinical Data Governance . . . 210
15.7 15.7 VAULT Data Flow Architecture . . . 210
15.8 15.8 VAULT Encryption at Rest . . . 211
15.9 Ed25519: Enforced . . . 212
15.10 15.9 VAULT Key Management . . . 212
15.11 15.10 VAULT Dashboard Views . . . 213
15.12 15.11 VAULT Integration with MONITORING . . . 215
15.13 VAULT . . . 215
15.14 15.13 VAULT vs Traditional Data Warehouses . . . 217

15.15 15.14 VAULT Audit Capabilities . . . 218
15.16 15.15 VAULT Error Handling . . . 219
15.17 15.16 VAULT Governance Proof . . . 219

16 Chapter 16: API 221

16.1 16.1 Axiom . . . 221
16.2 16.2 Endpoint Architecture . . . 221
16.3 16.3 Authentication and Rate Limiting . . . 222
16.4 16.4 Request/Response Governance . . . 222
16.5 16.5 Error Handling . . . 223
16.6 16.6 Clinical Integration Patterns . . . 223
16.7 16.7 API Route Governance via HTTP.md . . . 223
16.8 Routes . . . 223
16.9 16.8 Request Lifecycle . . . 224
16.10 16.9 API Versioning . . . 225
16.11 16.10 API Payload Schemas . . . 226
16.12 16.11 API Monitoring and Metrics . . . 227
16.13 16.12 API and the Workers Runtime . . . 227
16.14 16.13 Clinical API Example: EHR Integration . . . 228
16.15 16.14 Clinical Vignette: API Governance Prevents Unaudited Access . . . 229
16.16 16.15 API Idempotency and Retry Safety . . . 230
16.17 16.16 API Governance Proof . . . 230
16.18 16.17 API Webhook Integration . . . 231
16.19 16.18 API Gateway Topology . . . 232

17 Chapter 17: CHAIN 233

17.1 17.1 Axiom . . . 233
17.2 17.2 The Hash Chain . . . 233
17.3 17.3 Verification Algorithm . . . 234
17.4 17.4 Fork Detection . . . 234
17.5 17.5 CHAIN and Healthcare Compliance . . . 235
17.6 17.6 CHAIN CLI Operations . . . 235
17.7 17.7 The Genesis Event . . . 236
17.8 17.8 Chain Compaction . . . 237
17.9 17.9 Dual-Write Architecture . . . 238
17.10 17.10 Chain Branching for Federation . . . 239
17.11 17.11 CHAIN Error Handling . . . 240

17.12 17.12 CHAIN Performance . . . 241
17.13 17.13 Clinical Vignette: Chain Integrity During FDA Audit . . . 241
17.14 17.14 Chain Migration and Key Rotation . . . 243
17.15 17.15 Chain Anchoring to External Systems . . . 244
17.16 17.16 Governance Proof: Chain Integrity . . . 245

18 Chapter 18: MINT 246

18.1 18.1 Axiom . . . 246
18.2 18.2 The Gradient Rule . . . 246
18.3 18.3 Maximum Mint Per Scope . . . 247
18.4 18.4 Supply Ceiling . . . 247
18.5 18.5 RUNNER Tasks . . . 248
18.6 18.6 MINT and the Healthcare Governance Economy . . . 248
18.7 18.7 MINT CLI Operations . . . 249
18.8 18.8 MINT Data Structures . . . 250
18.9 18.9 DEBIT: The Negative Gradient . . . 251
18.10 18.10 Epoch-Based Supply Analysis . . . 252
18.11 18.11 MINT Error Handling and Failure Modes . . . 252
18.12 18.12 MINT Integration with WALLET . . . 253
18.13 18.13 Clinical Vignette: FinChat Compliance Mint . . . 253
18.14 18.14 MINT Batch Operations . . . 254
18.15 18.15 MINT and Multi-Author Scopes . . . 255
18.16 18.16 Closure . . . 255

19 Chapter 19: IDENTITY 256

19.1 19.1 Axiom . . . 256
19.2 19.2 The KYC Pipeline . . . 256
19.3 19.3 Onboarding Flow . . . 257
19.4 19.4 The SOP-009/010/011 Pipeline . . . 258
19.5 19.5 IDENTITY and Clinical AI Governance . . . 258
19.6 19.6 IDENTITY Data Model . . . 258
19.7 Axiom . . . 259
19.8 Readers . . . 259
19.9 Writers . . . 259
19.10 19.8 Ed25519 Key Architecture . . . 260
19.11 19.9 Multi-Identity and Role Separation . . . 260
19.12 19.10 IDENTITY Revocation . . . 261

19.13 19.11 IDENTITY and Federation . . . 261
19.14 19.12 Clinical Vignette: Identity Verification Prevents Unauthorized AI Governance . . . 261
19.15 19.13 Identity Lifecycle and Credential Monitoring . . . 262
19.16 19.14 IDENTITY Metrics . . . 263
19.17 19.15 Governance Proof: The Identity Chain . . . 264
19.18 19.16 IDENTITY Federation Across Organizations . . . 264
19.19 19.17 IDENTITY Audit Reports . . . 265
19.20 19.18 IDENTITY and Role-Based Access Control . . . 266
19.21 19.19 Clinical Vignette: IDENTITY Prevents Credential Fraud . . . 267

20 Chapter 20: CONTRIBUTE 269

20.1 20.1 Axiom . . . 269
20.2 20.2 The Contribution Pipeline . . . 269
20.3 20.3 Contribution Tiers . . . 270
20.4 20.4 The RESPONSES Directory . . . 270
20.5 20.5 CONTRIBUTE and Clinical Community Building . . . 270
20.6 20.6 CONTRIBUTE CLI Operations . . . 271
20.7 20.7 Contribution Review Workflow . . . 272
20.8 Review . . . 272
20.9 Decision . . . 273

21 BI-RADS FAQ Expansion 274

21.1 20.12 Clinical Vignette: Cross-Institutional Contribution Network . . . 275
21.2 20.13 Contribution Quality Metrics . . . 276
21.3 20.14 Contribution Conflict Resolution . . . 277
21.4 20.15 Contribution Expiration and Refresh . . . 278
21.5 20.16 Governance Proof: The Contribution Chain . . . 278
21.6 20.17 Contribution Templates by Clinical Domain . . . 279
21.7 20.18 Contribution Impact Metrics . . . 280
21.8 20.19 Contribution Retraction . . . 281

22 Chapter 21: NOTIFIER 282

22.1 21.1 Axiom . . . 282
22.2 21.2 Route Architecture . . . 282
22.3 21.3 The Inbox Model . . . 283
22.4 21.4 API Routes . . . 283
22.5 21.5 Record Shape . . . 283
22.6 21.6 LEDGER Projection . . . 284

22.7 21.7 Clinical Alert Patterns . . . 284
22.8 21.8 NOTIFIER Configuration via CANON.md . . . 285
22.9 Notify . . . 285
22.10 21.9 Inbox Architecture . . . 286
22.11 21.10 NOTIFIER CLI . . . 287
22.12 21.11 Notification Deduplication . . . 288
22.13 21.12 NOTIFIER and HIPAA Communication Requirements . . . 288
22.14 21.13 Clinical Vignette: NOTIFIER Prevents Stale Guideline Deployment . . . 288
22.15 21.14 Notification Routing Patterns . . . 290
22.16 21.15 Notification Templates . . . 291
22.17 21.16 Notification Analytics . . . 291
22.18 21.17 Governance Proof: The Notification Chain . . . 292
22.19 21.18 NOTIFIER Channel Architecture . . . 293
22.20 21.19 NOTIFIER Quiet Hours and On-Call Rotation . . . 293
22.21 21.20 NOTIFIER and Cross-Scope Notification Aggregation . . . 294

23 Chapter 22: MONITORING 296

23.1 22.1 Axiom . . . 296
23.2 22.2 The /metrics Endpoint . . . 296
23.3 22.3 Extended Health Check . . . 297
23.4 22.4 Governance Dashboard Metrics . . . 298
23.5 22.5 Alerting Integration . . . 298
23.6 22.6 Constraints . . . 298
23.7 22.7 MONITORING CLI Operations . . . 299
23.8 22.8 Metric Cardinality and Governance . . . 300
23.9 22.9 Drift Detection Pipeline . . . 300
23.10 22.10 Grafana Dashboard Templates . . . 301
23.11 22.11 MONITORING and Service-Level Objectives . . . 302
23.12 22.12 Clinical MONITORING Example . . . 303
23.13 22.13 Clinical Vignette: MONITORING Catches Silent Model Drift . . . 303
23.14 22.14 MONITORING Data Retention . . . 305
23.15 22.15 MONITORING Cross-Fleet Comparison . . . 305
23.16 22.16 MONITORING and Incident Response . . . 306
23.17 22.17 Governance Proof: Observability Completeness . . . 307
23.18 22.18 MONITORING and Regulatory Audit Readiness . . . 308
23.19 22.19 Custom Dashboard Compilation . . . 309

24 Chapter 23: DEPLOY 310
24.1 23.1 Axiom . . . 310
24.2 23.2 The Full Pipeline . . . 310
24.3 23.3 Deploy Gates . . . 311
24.4 23.4 DESIGN-First Deploy Order . . . 311
24.5 23.5 Rollback . . . 312
24.6 23.6 Containerization . . . 312
24.7 23.7 Cloudflare Workers Deployment . . . 313
24.8 23.8 CI/CD Integration . . . 313
24.9 23.9 DEPLOY and Clinical AI Operations . . . 314
24.10 23.10 Deploy Artifact Registry . . . 314
24.11 23.11 Deploy Environments . . . 315
24.12 23.12 Deploy Freeze . . . 316
24.13 23.13 Multi-Site Deploy Orchestration . . . 317
24.14 23.14 Clinical Vignette: DEPLOY Prevents Ungoverned Model Update . . . 317
24.15 23.15 Blue-Green Deployment Pattern . . . 319
24.16 23.16 Deployment Approval Workflows . . . 320
24.17 23.17 Deployment Metrics . . . 321
24.18 23.18 Governance Proof: Deployment Soundness . . . 322
24.19 23.19 Deploy and Federation Boundaries . . . 323
24.20 Deploy: March 2026 . . . 323

25 Chapter 24: Cross-Axiomatic Validation 325
25.1 24.1 The Compilation Chain . . . 325
25.2 24.2 Evidence Bridges . . . 326
25.3 24.3 Upward Compilation . . . 326
25.4 24.4 Cross-Scope Reference Resolution . . . 326
25.5 24.5 The Cross-Axiom Consistency Check . . . 327
25.6 24.6 Evidence Bridge Types . . . 328
25.7 24.7 The Cross-Axiom Matrix . . . 328
25.8 24.8 Circular Reference Detection . . . 329
25.9 24.9 Fleet-Wide Compilation . . . 330
25.10 24.10 Cross-Axiomatic Validation in CI . . . 330
25.11 24.11 Clinical Cross-Validation Pattern . . . 331
25.12 24.12 Validation Depth Control . . . 331
25.13 24.13 Error Catalog . . . 332

25.14 24.14 Cross-Validation and the LEDGER . . . 332
25.15 24.16 Transitive Reference Healing . . . 334
25.16 24.17 Visualization of Cross-References . . . 334
25.17 24.18 Cross-Validation Reporting . . . 335
25.18 24.19 Cross-Validation Anti-Patterns . . . 336
25.19 24.20 Cross-Validation Performance Optimization . . . 337
25.20 24.21 Cross-Validation Metrics . . . 338
25.21 24.22 Cross-Validation in Multi-Organization Fleets . . . 338
25.22 24.23 Cross-Validation Summary . . . 339

26 Chapter 25: Contextual Agents 340

26.1 25.1 The Agent Pipeline . . . 340
26.2 25.2 The FHIR INTEL TALK COIN Pipeline . . . 341
26.3 25.3 systemPrompt Compilation . . . 341
26.4 25.4 Persona Resolution . . . 342
26.5 25.5 Frontmatter Wiring . . . 342
26.6 25.7 Agent Context Window Management . . . 345
26.7 25.8 Multi-Agent Routing . . . 345
26.8 25.9 Agent Testing and Validation . . . 346
26.9 25.10 Agent Lifecycle . . . 347
26.10 25.11 Agent Constraints Enforcement . . . 348
26.11 25.12 Cross-Vertical Agent Patterns . . . 348
26.12 25.14 Agent Versioning . . . 349
26.13 25.15 The Complete Agent Architecture . . . 350
26.14 25.16 Agent Error Handling . . . 351
26.15 25.17 Agent Deployment Checklist . . . 352
26.16 25.18 Agent Discovery . . . 352
26.17 25.19 Agent and the LEARNING Loop . . . 353

27 Chapter 26: The _generated Contract 354

27.1 26.1 The Rule . . . 354
27.2 26.2 Compiled Outputs . . . 354
27.3 26.3 Provenance . . . 355
27.4 26.4 The _generated Directory Structure . . . 355
27.5 26.5 CANON.json Schema . . . 356
27.6 26.6 The Build Pipeline . . . 357
27.7 26.7 Never Hand-Edit Rules . . . 357

27.8 26.8 Fixing the Contract, Not the Output . . . 358
27.9 26.9 Compiler Error Handling . . . 359
27.10 26.10 Incremental Compilation . . . 359
27.11 26.11 The Provenance Chain . . . 360
27.12 26.12 Clinical Significance . . . 360
27.13 26.13 The _generated Lifecycle . . . 361
27.14 26.14 Build Reproducibility . . . 362
27.15 26.15 Multi-Repository _generated . . . 362
27.16 26.16 Error Recovery Procedures . . . 362
27.17 26.17 _generated and Version Control . . . 363
27.18 26.18 _generated Diffing . . . 364
27.19 26.19 Understanding Compilation Stages . . . 364
27.20 26.20 Debugging _generated Issues . . . 365
27.21 26.21 _generated and Testing . . . 366
27.22 26.22 _generated File Sizes . . . 366
27.23 26.23 The Golden Rule . . . 367
27.24 26.24 _generated Artifacts Per Scope Type . . . 367
27.25 26.25 _generated and Caching . . . 368
27.26 26.26 The Contract in One Sentence . . . 369
27.27 26.27 _generated and Disaster Recovery . . . 369

28 Chapter 27: Scope Intelligence 370

28.1 27.1 INTEL.md . . . 370
28.2 27.2 Backpropagation . . . 371
28.3 27.3 Corpus Management . . . 371
28.4 27.4 INTEL.md Anatomy . . . 371
28.5 Scope Intelligence . . . 371
28.6 Evidence Chain . . . 372
28.7 Cross-Scope Connections . . . 372
28.8 27.6 Evidence Layers . . . 373
28.9 27.7 Cross-Scope Connection Patterns . . . 374
28.10 27.8 Backpropagation in Practice . . . 375
28.11 27.9 INTEL Freshness . . . 375
28.12 27.10 INTEL Compilation for Agents . . . 376
28.13 27.11 INTEL.md for Non-Clinical Scopes . . . 376
28.14 27.12 Building INTEL.md from Scratch . . . 377

28.15 Scope Intelligence . . . 377
28.16 Evidence Chain . . . 378
28.17 Cross-Scope Connections . . . 378

29 Step 3: Validate 379

30 Step 4: Build 380

30.1 27.15 INTEL Templates . . . 381
30.2 27.16 Cross-Scope Intelligence Aggregation . . . 381
30.3 27.17 INTEL and Regulatory Compliance . . . 382
30.4 27.18 INTEL.md Migration Patterns . . . 382
30.5 27.19 INTEL Versioning and History . . . 383
30.6 27.20 INTEL and Agent Knowledge Boundaries . . . 384
30.7 27.21 INTEL Completeness Checklist . . . 384
30.8 27.22 INTEL.md as Single Source of Truth . . . 385

31 Chapter 28: The Naming Convention 386

31.1 28.1 The Two Worlds . . . 386
31.2 28.2 GOV Naming . . . 386
31.3 28.3 SERVICE = SINGULAR . . . 387
31.4 28.4 RUNTIME Naming . . . 387
31.5 28.5 File Naming Conventions . . . 387
31.6 28.6 Directory Naming Rules . . . 388
31.7 28.7 The Inheritance Path Convention . . . 389
31.8 28.8 The GOV_ROOT and RUNTIME_ROOT . . . 390
31.9 28.9 Naming Anti-Patterns . . . 390
31.10 28.10 JavaScript and CSS Naming . . . 391
31.11 28.11 The Naming Contract . . . 392
31.12 28.13 Naming and Scope Discovery . . . 393
31.13 28.14 Naming Migration . . . 394
31.14 28.15 Naming in Frontmatter . . . 394
31.15 28.16 Naming in URLs . . . 395
31.16 28.17 Naming Across Languages . . . 396
31.17 28.18 Naming Validation Errors . . . 396
31.18 28.19 Naming Conventions Quick Reference . . . 397
31.19 28.20 Why Naming Matters . . . 398
31.20 28.21 Naming Conventions and Tooling . . . 398
31.21 28.22 Naming and Git . . . 399

31.22 28.23 Naming as Documentation . . . 400
31.23 28.24 Naming Enforcement Automation . . . 400

32 Chapter 29: DESIGN Tokens 401

32.1 29.1 Token Discipline . . . 401
32.2 29.2 The 23 Sass Partials . . . 402
32.3 29.3 DESIGN.css . . . 402
32.4 29.4 Breakpoints . . . 402
32.5 29.5 Visual Design Governance . . . 402
32.6 29.6 The Token Sync Pipeline . . . 403
32.7 29.7 Component Library Mirror . . . 404
32.8 29.8 MCP Integration . . . 404
32.9 29.9 First Composition — MammoChat Marketing Surface . . . 405
32.10 29.10 Token Categories Deep Dive . . . 406
32.11 29.11 Token Validation . . . 409
32.12 29.12 Theme Switching . . . 409
32.13 29.13 Token Export for Native Platforms . . . 410
32.14 29.15 Token Conflict Resolution . . . 411
32.15 29.16 WCAG Compliance Automation . . . 412
32.16 29.17 Token Documentation . . . 412
32.17 29.18 Token Performance Impact . . . 413
32.18 29.19 Token Governance Summary . . . 414
32.19 29.20 Emergency Token Override . . . 414
32.20 29.21 Tokens and the Build Pipeline . . . 415

33 Chapter 30: The CHAT Layer 416

33.1 30.1 CUSTOM Layout . . . 416
33.2 30.2 _CHAT.scss . . . 416
33.3 30.3 Seven Layouts . . . 416
33.4 30.4 _CHAT.scss Anatomy . . . 417
33.5 30.5 Accent Governance . . . 421
33.6 30.6 The CUSTOM Layout . . . 422
33.7 30.7 Message Rendering . . . 423
33.8 30.8 Chat Component Architecture . . . 424
33.9 30.9 Routing Indicators . . . 424
33.10 30.10 Accessibility Requirements . . . 425
33.11 30.11 Clinical Chat Patterns . . . 425

33.12 30.12 Multi-Layout Comparison . . . 426
33.13 30.14 Chat Event Ledgering . . . 428
33.14 30.15 Chat Performance Budget . . . 428
33.15 30.16 Chat Testing . . . 429
33.16 30.17 Chat Internationalization . . . 429
33.17 30.18 Chat Error States . . . 430
33.18 30.19 Chat Layer Summary . . . 431
33.19 30.20 CHAT and the Seven Layouts — Wiring Detail . . . 432
33.20 30.21 CHAT Layer Constraints . . . 433

34 Chapter 31: The GALAXY Visualization 434

34.1 31.1 Data Source . . . 434
34.2 31.2 Visual Encoding . . . 434
34.3 31.3 Compliance Ring . . . 435
34.4 31.4 Tier Glow . . . 435
34.5 31.5 INTEL Flow . . . 435
34.6 31.6 Brand Marks . . . 435
34.7 31.7 galaxy.json Structure . . . 435
34.8 31.8 The vis-network Configuration . . . 437
34.9 31.9 Node Rendering . . . 438
34.10 31.10 Compliance Ring Rendering . . . 439
34.11 31.11 Edge Animation . . . 440
34.12 31.12 Tooltip Content . . . 441
34.13 31.13 Container-Driven Topology . . . 441
34.14 31.14 GALAXY and the Build Pipeline . . . 442
34.15 31.15 Clinical GALAXY Example . . . 443
34.16 31.16 Interaction Patterns . . . 444
34.17 31.18 GALAXY Keyboard Navigation . . . 445
34.18 31.19 GALAXY Search and Filter . . . 446
34.19 31.20 GALAXY Export . . . 447
34.20 31.21 GALAXY Embedding . . . 447
34.21 32.2 Eight Circulation Events . . . 449
34.22 32.3 Supply Ceiling . . . 452
34.23 32.4 Conservation Equation . . . 452
34.24 32.5 The Seven WALLET Invariants . . . 453
34.25 32.6 Dual-Write Architecture . . . 455

34.26 32.7 Clinical Example: Hospital COIN Economy . . . 455
34.27 32.8 COIN as Institutional Memory . . . 456
34.28 32.9 WALLET Reconciliation . . . 456
34.29 32.10 Governance Proof . . . 457
34.30 32.11 COIN Velocity and Economic Health . . . 457
34.31 32.12 COIN and Multi-Organization Accounting . . . 458
34.32 32.13 COIN Tax and Fee Structure . . . 459
34.33 The Economy: Live . . . 459
34.34 32.14 Clinical Vignette: COIN Resolves Budget Allocation Dispute . . . 460

35 Chapter 33: Gradient Minting 461

35.1 33.1 The Rule . . . 461
35.2 33.2 Worked Example . . . 462
35.3 33.3 DEBIT:DRIFT Asymmetry . . . 463
35.4 33.4 The RUNNER Task Pipeline . . . 463
35.5 33.5 Supply Ceiling Proof . . . 464
35.6 33.6 Multi-Scope Gradient Patterns . . . 464
35.7 33.7 Clinical Gradient Example: EHR Migration . . . 465
35.8 33.8 Gradient Visualization . . . 465
35.9 33.9 Gradient Economics Summary . . . 466
35.10 33.10 Clinical Vignette: Gradient Minting During an FDA Software Audit . . . 466
35.11 33.11 Gradient Composition Across Fleet . . . 467
35.12 33.12 Gradient Anti-Patterns . . . 467
35.13 33.13 Governance Proof: Gradient Convergence . . . 468
35.14 33.14 Gradient Visualization . . . 468
35.15 33.15 Gradient Rate and Team Velocity . . . 469
35.16 33.16 Gradient and the LEARNING Feedback Loop . . . 469
35.17 33.17 Clinical Vignette: Gradient Reveals Governance Bottleneck . . . 470

36 Chapter 34: The SHOP 471

36.1 34.1 Product Listing Workflow . . . 471
36.2 34.2 Fiat On-Ramp . . . 471
36.3 34.3 Economic Loop . . . 472
36.4 34.4 Attestation and Reviews . . . 472
36.5 34.5 Public Projection via SHOP.md . . . 473
36.6 34.6 SHOP Categories and Taxonomy . . . 474
36.7 34.7 Multi-Scope Products . . . 474

36.8 34.10 Clinical Vignette: SHOP Fraud Detection at Scale . . . 475
36.9 34.11 SHOP Search and Filtering . . . 477
36.10 34.12 SHOP Access Control . . . 478
36.11 34.13 SHOP Reporting . . . 478
36.12 34.14 Governance Proof: The Market Chain . . . 479
36.13 34.15 SHOP Product Bundling . . . 480

37 Chapter 35: COST_BASIS and Pricing 482

37.1 35.1 The Formula . . . 482
37.2 35.2 Examples . . . 483
37.3 35.3 Pricing Tiers . . . 483
37.4 35.4 Unit Economics . . . 485
37.5 35.5 Enterprise Volume Pricing . . . 485
37.6 35.6 The Pricing Invariant . . . 485
37.7 35.7 Clinical Pricing Example . . . 486
37.8 35.8 Cost Basis Decomposition . . . 486
37.9 35.9 Dynamic Pricing and Market Signals . . . 487
37.10 35.10 Clinical Vignette: COST_BASIS in a Multi-Hospital Procurement . . . 487
37.11 35.11 COST_BASIS Audit and Verification . . . 488
37.12 35.12 Governance Proof: The Pricing Chain . . . 489
37.13 35.13 COST_BASIS and Depreciation . . . 489
37.14 35.14 Cross-Currency COST_BASIS in Federation . . . 490
37.15 27.18 INTEL.md Migration Patterns . . . 490
37.16 27.19 INTEL Versioning and History . . . 491
37.17 27.20 INTEL and Agent Knowledge Boundaries . . . 492
37.18 27.21 INTEL Completeness Checklist . . . 492
37.19 27.22 INTEL.md as Single Source of Truth . . . 493

38 Chapter 36: Governance as Type System 494

38.1 36.1 The Isomorphism . . . 494
38.2 36.2 VOCAB as Type System . . . 495
38.3 36.3 The Compilation Target . . . 496
38.4 36.4 The Six Theorems . . . 496
38.5 36.5 CANON.md as Type Declaration . . . 497
38.6 36.6 Clinical Type System Example . . . 498
38.7 36.7 Type Inference in Governance . . . 498
38.8 36.8 Type Narrowing and Refinement . . . 499

38.9 36.9 Generic Types and Parameterized Governance . . . 500
38.10 36.10 Type Errors in Governance: A Taxonomy . . . 501
38.11 36.11 The Complete Type-Check Pipeline . . . 502
38.12 36.12 Clinical Vignette: Type Safety Prevents Deployment Error . . . 503
38.13 36.13 Governance Proof: The Type Safety Chain . . . 504
38.14 36.14 Type Erasure and Runtime Behavior . . . 505

39 Chapter 37: Governance as Compiler 506

39.1 37.1 The Compiler Pipeline . . . 506
39.2 37.2 Error Taxonomy . . . 507
39.3 37.3 Continuous Governance . . . 508
39.4 37.4 The Optimization Model . . . 508
39.5 37.5 Scope as Compilation Unit . . . 509
39.6 37.6 Clinical Compiler Example . . . 510
39.7 37.7 Compiler Passes and Ordering . . . 510
39.8 37.8 Compiler Diagnostics . . . 511
39.9 37.9 Incremental Compilation . . . 511
39.10 37.10 Compiler Extensions . . . 512
39.11 37.11 The Isomorphism Theorem . . . 512
39.12 37.12 Clinical Vignette: Compiler Catches Cross-Scope Regression . . . 513
39.13 37.13 Compiler Optimization: Dead Code Elimination . . . 514
39.14 37.14 Compiler Warnings vs Errors . . . 515
39.15 37.15 Compiler Performance Profiling . . . 516
39.16 37.16 Governance Proof: Compiler Correctness . . . 516
39.17 37.17 Compiler Extensions and Custom Validators . . . 517
39.18 37.18 Clinical Vignette: Compiler Catches Cross-Scope Type Mismatch . . . 518

40 Chapter 38: Governance as Version Control 519

40.1 38.1 Git IS the Governance Engine . . . 519
40.2 38.2 Certification = Git Tags . . . 520
40.3 38.3 The Molecular Clock . . . 521
40.4 38.4 LEDGER as Append-Only Log . . . 521
40.5 38.5 Branching as Governance Experiments . . . 521
40.6 38.6 Git Blame as Governance Attribution . . . 522
40.7 38.7 Clinical VCS Example . . . 522
40.8 38.8 Merge Conflicts as Governance Disputes . . . 523
40.9 38.9 Git Hooks as Governance Enforcement . . . 523

40.10 38.10 Git Bisect as Governance Debugging . . . 524
40.11 38.11 Clinical Vignette: Version Control Prevents Guideline Collision . . . 524
40.12 38.12 Version Control Metrics . . . 525
40.13 38.13 Rebase vs Merge: Governance Implications . . . 526
40.14 38.14 Cherry-Pick as Governance Backport . . . 526
40.15 38.15 Governance Archaeology . . . 527
40.16 38.16 Governance Proof: Version Control Completeness . . . 528
40.17 38.17 Git Submodules as Federation Mechanism . . . 528
40.18 38.18 Git Hooks as Governance Enforcement Points . . . 529
40.19 38.19 Clinical Vignette: Git Bisect Finds Governance Regression . . . 530

41 Chapter 39: The LEARNING Closure 531

41.1 39.1 The Eighth Dimension . . . 531
41.2 39.2 What Languages Achieve . . . 532
41.3 39.3 The LEARNING Dimension . . . 533
41.4 39.4 The LEARNING Loop . . . 533
41.5 39.5 Intelligence Emergence . . . 533
41.6 39.6 Clinical LEARNING Example . . . 534
41.7 39.7 LEARNING as Evolutionary Memory . . . 535
41.8 39.8 LEARNING Epochs and Transitions . . . 535
41.9 39.9 Cross-Scope LEARNING Transfer . . . 536
41.10 39.10 LEARNING and the INTEL-CHAT-COIN Triad . . . 536
41.11 39.11 Clinical Vignette: LEARNING Prevents Repeated Failure . . . 537
41.12 39.12 LEARNING Signal Taxonomy . . . 538
41.13 39.13 LEARNING Quantification . . . 539
41.14 39.14 Governance Proof: LEARNING Closure . . . 539
41.15 39.15 LEARNING Anti-Patterns . . . 540
41.16 39.16 LEARNING and Federated Intelligence . . . 541

42 Chapter 40: Why Every Language Needed Governance 542

42.1 40.1 The Gap . . . 542
42.2 40.2 The Three FOUNDATION Axioms . . . 543
42.3 40.3 What They Missed . . . 543
42.4 40.4 The Abstraction Stack . . . 544
42.5 40.5 Why Governance Cannot Be a Library . . . 545
42.6 40.6 The Proof by Elimination . . . 545
42.7 40.7 The Twenty Paradigm Families . . . 545

42.8 40.8 Clinical Implication: The Governed Hospital . . . 546
42.9 40.9 Language Composability and the Governance Bridge . . . 547
42.10 40.10 The Governance Completeness Argument . . . 548
42.11 40.11 Why No Language Will Ever Reach Level 4 . . . 549
42.12 40.12 The Convergence Thesis . . . 549
42.13 40.13 Clinical Vignette: The Polyglot Radiology Department . . . 550

43 Chapter 41: Overview 552

43.1 41.1 The Transaction . . . 552
43.2 41.2 The 16 Core Tools . . . 552
43.3 41.3 The Invariant . . . 553
43.4 41.4 The Tool Taxonomy . . . 553
43.5 41.5 Tool Composition . . . 555
43.6 41.6 The Clinical Context . . . 555
43.7 41.7 Tool Discovery . . . 556
43.8 41.8 Tool Versioning . . . 557
43.9 41.9 The Toolchain Guarantee . . . 557
43.10 41.10 Tool Dependency Graph . . . 558
43.11 41.11 Clinical Vignette: Toolchain in a Hospital CI Pipeline . . . 558
43.12 41.12 Tool Error Aggregation . . . 559
43.13 41.13 Governance Proof: Toolchain Soundness . . . 559
43.14 41.14 Tool Configuration and Environment . . . 560
43.15 41.15 Tool Portability . . . 560
43.16 41.16 Tool Telemetry and Privacy . . . 561
43.17 41.17 Tool Extension Points . . . 561
43.18 41.18 Toolchain Performance Benchmarks . . . 562
43.19 41.19 Clinical Vignette: Toolchain Saves ER Deployment . . . 562
43.20 41.20 Toolchain Governance Summary . . . 563

44 Chapter 42: magic 564

44.1 42.1 The Kernel . . . 564
44.2 42.2 validate . . . 565
44.3 42.3 scan . . . 565
44.4 42.4 heal . . . 565
44.5 42.5 The 255-Bit Standard . . . 566
44.6 42.6 The `inherits:` Resolution . . . 566
44.7 42.7 Tier Classification . . . 567

44.8 42.8 Exit Codes . . . 567
44.9 42.9 Discovery Algorithm . . . 567
44.10 42.10 ledger . . . 568
44.11 42.11 Compilation . . . 568
44.12 42.12 Performance . . . 568
44.13 42.13 Clinical Application . . . 569
44.14 42.14 FFI Bindings . . . 569
44.15 42.15 Extending the Kernel . . . 570
44.16 42.16 Multi-Org Deployments . . . 570
44.17 42.17 Error Recovery . . . 570
44.18 42.18 Clinical Vignette: magic validate in a Pre-Operative Checklist System . . . 571
44.19 42.19 Kernel Architecture Details . . . 571
44.20 42.20 magic scan: Fleet Intelligence . . . 572
44.21 42.21 Governance Proof: Kernel Correctness . . . 573
44.22 42.22 magic ledger: Economic Event Query . . . 573
44.23 42.23 magic commit: Governed Commit Wrapper . . . 574
44.24 42.24 magic scan Filters and Flags . . . 574
44.25 42.25 magic validate Verbose Mode . . . 575
44.26 42.26 Clinical Vignette: magic validate in a Residency Training Program . . . 576
44.27 42.27 magic as Governance Kernel . . . 577
44.28 VaaS: The Kernel as Product . . . 578
44.29 FRESHNESS: Incremental Compilation . . . 578

45 Chapter 43: magic-heal 579

45.1 43.1 The Five Stages . . . 579
45.2 43.2 Usage . . . 579
45.3 43.3 Stage 1: SCAN . . . 580
45.4 43.4 Stage 2: DIAGNOSE . . . 580
45.5 43.5 Stage 3: PROPOSE . . . 581
45.6 43.6 Stage 4: SETTLE . . . 582
45.7 43.7 Stage 5: VALIDATE . . . 582
45.8 43.8 Full Pipeline Example . . . 583
45.9 43.9 Clinical Healing Example . . . 583
45.10 43.10 Healing Debt . . . 583
45.11 43.11 The Settlement Model . . . 584
45.12 43.12 Batch Healing . . . 584

45.13 43.13 Healing Frequency . . . 584
45.14 43.14 Healing vs Hand-Authoring . . . 584
45.15 43.15 Clinical Vignette: Batch Healing a Hospital Onboarding . . . 585
45.16 43.16 Healing Economics . . . 586
45.17 43.17 Governance Proof: Healing Convergence . . . 586
45.18 43.18 Healing Template Packs . . . 586
45.19 43.19 Healing Diff Review . . . 587
45.20 43.20 Healing Batch Operations . . . 588
45.21 43.21 Healing and the PROPOSE-SETTLE Pattern . . . 589
45.22 43.22 Healing Priority Algorithm . . . 590
45.23 43.23 Clinical Vignette: Healing Sprint Saves Joint Commission Survey . . . 591
45.24 43.24 magic-heal as Governance Antibody . . . 592

46 Chapter 44: build 593

46.1 44.1 The Build Pipeline . . . 593
46.2 44.2 _generated Flag . . . 594
46.3 44.3 Deploy Order . . . 594
46.4 44.4 Stage-by-Stage Reference . . . 594
46.5 44.5 Build Determinism . . . 596
46.6 44.6 Build Failure Recovery . . . 596
46.7 44.7 Build Performance . . . 596
46.8 44.8 Clinical Build Example . . . 597
46.9 44.9 Clinical Vignette: Build Pipeline Catches Silent Regression . . . 597
46.10 44.10 Build Artifacts and the _generated Contract . . . 598
46.11 44.11 Build Phase Dependencies (DAG) . . . 598
46.12 44.12 Build Caching . . . 599
46.13 44.13 Governance Proof: Build Soundness . . . 599
46.14 44.14 Build Dry-Run Mode . . . 600
46.15 44.15 Build Hooks and Custom Stages . . . 601
46.16 44.16 Build Output Manifest . . . 601
46.17 44.17 Build and the _generated Verification Tool . . . 602
46.18 44.18 Build Environment Isolation . . . 603
46.19 44.19 Clinical Vignette: Build Pipeline Detects Cross-Service Inconsistency . . . 604
46.20 FRESHNESS: Build Caching . . . 605
46.21 44.20 build as Governance Compiler . . . 605

47 Chapter 45: Validation Errors and Healing 607

47.1 45.1 Common Errors . . . 607
47.2 45.2 Diagnostic Flow . . . 607
47.3 45.3 The Heal Loop . . . 608
47.4 45.4 Error Code Reference . . . 608
47.5 45.5 Error Severity Levels . . . 610
47.6 45.6 Fix Patterns . . . 610
47.7 45.7 Diagnostic Procedure . . . 611
47.8 45.8 Healing in CI . . . 612
47.9 45.9 Common Error Combinations . . . 612
47.10 45.10 Prevention . . . 613
47.11 45.11 Error Resolution Priority . . . 613
47.12 45.12 Regression Detection . . . 614
47.13 45.13 Healing Metrics . . . 614
47.14 45.14 Clinical Error Example . . . 615
47.15 45.15 Clinical Vignette: Cascading Validation Errors in a Multi-Site Deployment . . . 615
47.16 45.16 Error Recovery Time Benchmarks . . . 616
47.17 45.17 Healing Maturity Model . . . 617
47.18 45.18 Governance Proof: Error System Completeness . . . 618
47.19 45.19 Error Aggregation for Fleet-Wide Diagnosis . . . 618
47.20 45.20 Error History and Trend Analysis . . . 619

48 Chapter 46: The Build Pipeline 620

48.1 46.1 CI/CD Integration . . . 620
48.2 46.2 Pre-Commit Hooks . . . 621
48.3 46.3 The Full Pipeline . . . 621
48.4 46.4 The magic-validate.yml Workflow . . . 621
48.5 46.5 Build Workflow Steps . . . 622
48.6 46.6 Pre-Commit Hook Details . . . 623
48.7 46.7 Score Gates . . . 623
48.8 46.8 Pipeline Monitoring . . . 623
48.9 46.9 Pipeline Recovery . . . 624
48.10 46.10 Pipeline Security . . . 624
48.11 46.11 Clinical CI/CD Example . . . 624
48.12 46.12 Clinical Vignette: Pipeline Prevents PHI Leak . . . 625
48.13 46.13 Multi-Repository Pipeline Architecture . . . 625
48.14 46.14 Pipeline Metrics and SLAs . . . 626

48.15 46.15 Pipeline Rollback Procedure . . . 627
48.16 46.16 Pipeline Evolution: From Manual to Continuous . . . 627
48.17 46.17 Governance Proof: Pipeline Completeness . . . 628
48.18 46.18 Pipeline Observability . . . 628
48.19 46.19 DAG-Parallel Execution . . . 629
48.20 46.20 Pipeline Disaster Recovery . . . 630
48.21 46.21 Pipeline Compliance Evidence Generation . . . 631
48.22 The 13-Phase Pipeline (DAG-Governed) . . . 632
48.23 46.22 Clinical Vignette: Pipeline Enables Regulatory Fast-Track . . . 632

49 Chapter 47: Advanced Tools 634

49.1 47.1 validate-design . . . 634
49.2 47.2 magic-tag . . . 634
49.3 47.3 vault . . . 635
49.4 47.4 enforce-magic-ip . . . 635
49.5 47.5 Production Hardening . . . 635
49.6 47.6 validate-design Deep Dive . . . 636
49.7 47.7 magic-tag: Certification and Release Management . . . 637
49.8 47.8 vault: The Economic Operations Console . . . 637
49.9 47.9 enforce-magic-ip: Intellectual Property Protection . . . 639
49.10 47.10 Tool Composition and Pipeline Integration . . . 640
49.11 47.11 Clinical Vignette: Advanced Tools Prevent a Deployment Incident . . . 641
49.12 47.12 Writing Custom Advanced Tools . . . 642
49.13 47.13 Advanced Tool Governance Proof . . . 643
49.14 47.14 Backup and Disaster Recovery . . . 643

50 Chapter 48: The Closure — CANONIC Against All of Programming 646

50.1 48.1 The Argument . . . 646
50.2 48.2 The Neofunctionalization Tree . . . 649
50.3 48.3 The Closure Matrix . . . 649
50.4 48.4 The Phylogenetic Proof . . . 650
50.5 48.5 LEARNING . . . 650
50.6 48.6 The 20 Closure Tables . . . 650
50.7 48.7 The Final Argument . . . 655
50.8 48.8 The Three Impossibility Results . . . 655
50.9 48.9 The Closure Proof . . . 655
50.10 48.10 The Last Word . . . 656

50.11 48.11 The Implementation Evidence . . . 656
50.12 48.12 The Clinical Proof . . . 657
50.13 48.13 The Finality . . . 657
50.14 27.18 INTEL.md Migration Patterns . . . 657
50.15 27.19 INTEL Versioning and History . . . 658
50.16 27.20 INTEL and Agent Knowledge Boundaries . . . 659
50.17 27.21 INTEL Completeness Checklist . . . 660
50.18 27.22 INTEL.md as Single Source of Truth . . . 661
50.19 Appendix A: Governance File Reference . . . 661
50.20 Appendix B: Tier Algebra . . . 662
50.21 Appendix C: CLI Quick Reference . . . 662
50.22 Appendix D: Naming Convention Quick Reference . . . 663
50.23 Appendix E: The Closure Tables . . . 663
50.24 E.1: Imperative & OOP . . . 664
50.25 E.2: Functional . . . 664
50.26 E.3: Type Systems & Dependent Types . . . 665
50.27 E.4: Concurrent & Actor . . . 666
50.28 E.5: Logic & Constraint . . . 667
50.29 E.6: Reactive & Dataflow . . . 667
50.30 E.7: Concatenative & Stack . . . 668
50.31 E.8: Array & Numeric . . . 668
50.32 E.9: Metaprogramming . . . 669
50.33 E.10: Smart Contracts . . . 670
50.34 E.11: Proof Assistants . . . 670
50.35 E.12: Probabilistic . . . 671
50.36 E.13: GPU & Parallel . . . 671
50.37 E.14: DSLs & Config . . . 672
50.38 E.15: Markup & Styling . . . 672
50.39 E.16: Query & Data . . . 673
50.40 E.17: Shell & Scripting . . . 673
50.41 E.18: Visual & Low-Code . . . 674
50.42 E.19: Systems & Bare Metal . . . 674
50.43 E.20: The Master Matrix . . . 675
50.44 Appendix F: INTEL Compilation Reference . . . 676
50.45 Persona Resolution for BOOK Type . . . 676

50.46 Persona Resolution for SERVICE Type 676
50.47 systemPrompt Sources 677
50.48 Appendix G: References 677
50.49 Whitepapers [W-XX] 677
50.50 Blogs [B-XX] 678
50.51 Governance Sources [G-XX] 678
50.52 Glossary 679
50.53 Colophon 679
50.54 References 680

The Dev Manual

This book is one half of a pair. Its companion — The Canonic Canon — covers the same system from the governor's perspective: policy, oversight, organizational adoption. This book covers the developer's perspective: code, commands, architecture, build pipelines. Read either independently; read both for the complete picture.

• • •

• • •

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

Devs speak programming. This book speaks theirs.

• • •

Dexter Hadley, MD/PhD [1] Author, CANONIC February 2026

• • •

0.1. Abstract

This is the implementation manual for CANONIC — a system where governance compiles like code. It walks you from your first axiom through your first TRIAD, your first scope, your first service, and your first 255-bit validation score. If you can write Markdown, you can write governance: create a directory, add CANON.md with an axiom and constraints, add README.md with the public interface, add VOCAB.md with defined terms, then run the validator. You will start at 35 and iterate toward 255 — the score that proves every governance obligation is satisfied, by architecture, not by committee. DRY. MATH. FIXED. PURE.

• • •

• • •

THE CANONIC DOCTRINE

• • •

THE CANONIC DOCTRINE *The MAGIC Implementation Standard*

CANONIC Series | 1st Edition | 2026

• • •

Copyright 2026 CANONIC. All rights reserved. Governed under MAGIC 255-bit compliance standard. Every chapter evidenced. Every claim cited. Every commit COIN.

• • •

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

• • •

> "255, or it doesn't deploy."
> — CANONIC [2]

0.2. 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 [3].

This is not a manual about governance — it is a manual about building. Governance in CANONIC works like compilation: your CANON.md is the grammar, your VOCAB.md is the type system, your README.md is the header file, and `magic validate` is the compiler that checks them all against a 255-bit target. Everything else is syntax [4].

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. Every chapter gives you a concrete workflow: write the governance file, run the command, read the output, iterate until the score holds.

• • •

Part I (Chapters 1-4) establishes the primitives: axiom, TRIAD, inheritance, and the eight-dimensional scoring model. Part II (Chapters 5-9) walks you through building scopes, services, products, and federations. Part III (Chapters 10-23) details each of the 14 services, from LEARNING to DEPLOY. Parts IV-VI cover INTEL compilation, design tokens, and the COIN economy. Part VII presents the theoretical closure — governance as type system, compiler, and version control. Part VIII documents the toolchain: `magic`, `build`, and `heal`. For the governance policy perspective on any of these topics, see The Canonic Canon.

• • •

PART I — FOUNDATIONS

• • •

Chapter 1

Chapter 1: The Axiom

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

The FHIR endpoint that drifted

A backend developer ships a FHIR integration — R4 Patient resources flowing from an external lab system into the hospital's EHR. It works. Six months later, the upstream lab vendor quietly migrates to a revised profile. The `Observation.category` binding changes. No governance file tracks the dependency, so nothing flags the break. The integration keeps running, but coded lab results silently stop mapping. Patient data still flows — it just stops meaning what the clinicians think it means. Nobody knows which version of the FHIR spec the endpoint was built against, because nobody wrote it down in a place the toolchain could check.

This is the problem axioms solve. An axiom like **"LabBridge serves laboratory results with governed FHIR R4 INTEL. Every resource profile versioned."** turns that silent failure into a compilation error. The version constraint propagates into the governance files. When the upstream profile changes and the scope's evidence base no longer matches, `magic validate` catches the drift before the data pipeline does. The axiom is the single sentence that makes the invisible dependency visible — and enforceable.

1.1. 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 [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 [5].

1.2. 1.2 Clinical Axioms

In clinical AI, the axiom doubles as the clinical contract — a single sentence declaring what the scope does, who it serves, and what guarantee it makes. Every clinical AI deployment in CANONIC begins with one:

```
## 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 interaction audited.
```

This axiom adds an operational guarantee — every interaction audited — and that single phrase 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. One sentence in the axiom drives the entire architecture.

1.3. 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 [5].

1.4. 1.4 Derivation

The axiom is the root of a derivation tree: constraints derive from the axiom, files derive from the constraints, and child scopes inherit the axiom's obligations. This derivation chain is, concretely, the compilation chain [3].

```
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 [6].

To see this concretely, consider the MammoChat axiom: "Every recommendation cited." That single phrase 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. 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 [3].

When `magic validate` runs against a scope, it reads the axiom in CANON.md first, because the axiom establishes the evaluation context. Constraints are evaluated against the axiom, files are evaluated against the constraints, and the score is computed from those file evaluations. Whether you are integrating CANONIC into an EHR environment or building a standalone service, the pattern is the same: one sentence defines the contract, the contract drives the architecture, and the architecture compiles to a score that serves as proof. Start with the axiom. Everything else follows.

1.6. 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, and validate to 255. For a complete walkthrough from empty directory to 255, see Chapter 5.

1.7. 1.7 The Three Primitives

CANONIC is built on three primitives, and every service, product, and deployment composes one or more of them. The axiom declares which primitives a scope uses; the constraints enforce how. We will explore how these primitives compose into the 14 services in Chapter 7 and detail each service individually in Chapters 10-23 [7].

Primitive	Symbol	Purpose	Clinical Mapping
INTEL	Knowledge	Governed evidence with provenance	Clinical guidelines, BI-RADS atlas, NCCN protocols
CHAT	Conversation	Contextual dialogue with governed constraints	MammoChat, OncoChat, MedChat — clinical TALK agents
COIN	Value	Economic truth — minting, spending, auditing	Governance work measurement, compliance investment tracking

The primitives are orthogonal. INTEL exists without CHAT. CHAT exists without COIN. COIN exists without INTEL. But composition is where power emerges. A TALK service composes CHAT + INTEL: it converses AND it knows. A SHOP service composes COIN + INTEL: it prices AND it evidences. A full AGENT scope composes all three: it knows (INTEL), it converses (CHAT), and it accounts for value (COIN) [7].

The axiom declares the composition. Consider these three axioms and their primitive compositions:

```
## Axiom

**LEARNING discovers governed INTEL. Every pattern evidenced.**
```

This axiom composes INTEL only. No CHAT. No COIN. The scope discovers and governs knowledge. It does not converse. It does not transact.

```
## Axiom

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

This axiom composes CHAT + INTEL. The scope converses (serves breast imaging) and knows (governed BI-RADS INTEL). It does not transact.

```
## Axiom

**SHOP publishes governed products at governed prices. Every transaction COIN. Every product 255.**
```

This axiom composes COIN + INTEL. The scope transacts (governed prices, COIN) and knows (governed products). It does not converse.

Primitive composition is not metadata — it is architecture. A scope that composes CHAT + INTEL must have both a conversation engine and a knowledge layer. A scope that composes COIN must have a LEDGER. A scope that composes all three must have all three. The axiom declares what the architecture requires, and the validator enforces it.

1.8. 1.8 Governance-First Principle

Governance precedes implementation. Always. No exceptions. This is not a workflow preference. It is an architectural constraint [2].

The governance-first principle means:

1. Write CANON.md before writing code.
2. Define VOCAB.md before naming variables.
3. Write README.md before writing the API.
4. Run `magic validate` before running tests.
5. Achieve 35 before writing the first function.

```
# WRONG: code first, governance later
mkdir my-service
cd my-service
npm init
# ... write 5,000 lines of code ...
# Three months later: "We should probably add governance"
echo "# CANON" > CANON.md  # Too late. The architecture is already set.

# RIGHT: governance first, code follows
mkdir MY-SERVICE
cd MY-SERVICE
cat > CANON.md << 'EOF'
# MY-SERVICE — CANON


## Axiom

**MY-SERVICE serves clinical queries with governed INTEL. Every answer cited.**


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```
```

1.9. Constraints

MUST: Cite evidence for every clinical answer MUST: Log every session to LEDGER MUST NOT: Generate uncited recommendations

• • •

EOF

magic validate # Score: first question answered. Now build from here.

```
The governance-first principle has a compiler analogy: you write the type signature before you write the f

In healthcare, governance-first is not just good practice — it is a regulatory requirement expressed as ar

## 1.9 Axiom Mechanics: The Validator's Perspective

When `magic validate` reads a CANON.md, it processes the axiom in three steps ^6^:

**Step 1: Parse.** Extract the axiom text from the `## Axiom` section. The axiom must be a single bold-for
```

ERROR: CANON.md — Axiom section missing or malformed. Expected: ## Axiom followed by **{axiom text}** Got: ## Axiom followed by empty section Fix: Add a bold-formatted axiom sentence. Run: magic validate –verbose for details.

```
**Step 2: Constraint derivation check.** The validator verifies that every MUST/MUST NOT constraint in the
it is structural checking. The validator confirms that the axiom's key terms appear in the constraints. If
```

WARN: CANON.md — Axiom term “cited” has no matching constraint. Axiom: “Every recommendation cited.” Constraints: [no constraint references citing or evidence] Fix: Add MUST: Cite evidence source for every recommendation

```
**Step 3: Inheritance consistency.** The validator checks that the axiom does not contradict any inherited
```

ERROR: CANON.md — Axiom contradicts inherited constraint. Parent: hadleylab-canonic/MAGIC/SERVICES/TALK Parent constraint: MUST NOT: Process PHI outside the deployment perimeter Child axiom implies: external PHI processing Fix: Revise axiom to comply with inherited constraints.

```
These three steps — parse, derivation check, inheritance consistency — execute every time `magic validate`

## 1.10 Writing Your First Axiom: A Walkthrough

Open a terminal. Create the scope directory. Write the axiom.

```bash
mkdir -p DERMCHAT
cd DERMCHAT
```

Start with the clinical contract. What does this scope do? It serves dermatology. What evidence does it use? AAD (American Academy of Dermatology) clinical guidelines. What guarantee does it make? Every recommendation includes a differential.

```
## Axiom

**DermChat serves dermatology with governed AAD INTEL. Every recommendation includes differential. Every i
```

Test the axiom against the three criteria:

Criterion	Check	Result
Declarative	States what IS, not what you hope	PASS — “serves dermatology”
Specific	Names domain, evidence, guarantee	PASS — dermatology, AAD INTEL, differential + audited
Testable	Validator can check compliance	PASS — differential presence checkable, audit trail checkable

Now derive constraints from the axiom:

```
MUST:     Cite AAD guideline for every dermatologic recommendation
MUST:     Include differential diagnosis for every clinical assessment
MUST:     Log every interaction to LEDGER with timestamp and actor
MUST:     Version evidence base with AAD publication date
MUST NOT: Generate recommendations without AAD INTEL backing
MUST NOT: Provide biopsy recommendations (refer to dermatologist)
MUST NOT: Process PHI outside the local deployment perimeter
```

Every constraint traces to a word or phrase in the axiom. “Governed AAD INTEL” produces the MUST about AAD guidelines and the MUST NOT about INTEL backing. “Every recommendation includes differential”
```

produces the MUST about differential diagnosis. “Every interaction audited” produces the MUST about LEDGER logging. The derivation is mechanical. If you cannot derive a constraint from the axiom, either the constraint is wrong or the axiom is incomplete.

1.10. 1.11 Axiom Versioning

Axioms change. Clinical evidence evolves. Organizational scope shifts. When the axiom changes, everything downstream must revalidate [2].

```
# Before (v2025-06)
## Axiom

**MammoChat serves breast imaging with governed ACR INTEL. Every recommendation cited.**

# After (v2026-01)
## Axiom

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

The axiom changed in two ways: (1) evidence source changed from “ACR” to “BI-RADS” (more specific), and (2) an audit guarantee was added (“Every interaction audited”). Both changes propagate to constraints, files, and child scopes.

When you change the axiom:

1. Update the `version:` field in CANON.md.
2. Update every constraint that referenced the changed terms.
3. Run `magic validate` on the scope AND every child scope.
4. Record the change as an EVOLUTION signal in LEARNING.md.
5. Commit with a GOV prefix: `git commit -m "GOV: MammoChat axiom — add audit guarantee"`.

```
magic validate --scope SERVICES/TALK/MAMMOCHAT --recursive
# Validates MammoChat AND all child deployments (MAMMOCHAT-UCF, MAMMOCHAT-ADVENT, etc.)
```

If any child scope’s constraints conflict with the updated axiom, the validator identifies the conflict. Fix the child scope’s constraints. Revalidate. The cascade is deterministic — you change the root (axiom), the compiler tells you every leaf that breaks.

1.11. 1.12 The Axiom in Production

In a production clinical AI deployment, the axiom is not just a governance artifact. It is the clinical contract that appears in audit logs, compliance reports, and regulatory submissions. When the FDA asks “what does

this AI system do?" — the axiom is the answer. When the Joint Commission asks "what is the governance basis for this clinical decision support tool?" — the axiom is the answer. When the malpractice insurer asks "what guarantees does this system make?" — the axiom is the answer [3].

The axiom is one sentence — the most important sentence in the scope. Write it carefully, derive everything from it, validate against it, and deploy on it. Everything else is implementation. For the governance policy perspective on axioms, see The Canonic Canon, Chapter 1.

1.12. 1.13 Axiom Composition Patterns

When building complex scopes, the axiom pattern determines the architectural composition. Three composition patterns recur across every CANONIC deployment:

Pattern 1: Single-Primitive Axiom. The scope uses one primitive. The architecture is narrow.

```
## Axiom

**LEARNING discovers governed FHIR patterns. Every discovery evidenced.**
```

This axiom composes INTEL only. The scope directory contains INTEL.md, LEARNING.md, and governance files. No CHAT engine. No COIN operations. The scope is a knowledge container.

```
FHIR-PATTERNS/
  CANON.md          ← axiom declares INTEL-only composition
  VOCAB.md          ← FHIR governance terms
  INTEL.md          ← evidence corpus (FHIR R4/R5 patterns)
  LEARNING.md       ← accumulated pattern discoveries
  COVERAGE.md       ← dimensional self-assessment
  ROADMAP.md        ← governance trajectory
  README.md         ← public interface
  FHIR-PATTERNS.md  ← scope specification
```

Pattern 2: Dual-Primitive Axiom. The scope uses two primitives. The architecture widens.

```
## Axiom

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

This axiom composes CHAT + INTEL. The scope directory contains everything in Pattern 1 plus conversation infrastructure: systemPrompt compilation, persona resolution, disclaimer rendering.

Pattern 3: Triple-Primitive Axiom. The scope uses all three primitives. Full architecture.

```
## Axiom

**AGENT composes INTEL + CHAT + COIN. Every knowledge governed. Every conversation audited. Every transact
```

This axiom composes all three primitives. The scope directory includes knowledge governance, conversation engines, and economic operations. The AGENT scope is the most complex — and the most powerful.

Pattern	Primitives	Directory Count	Typical COIN	Clinical Example
Single	INTEL only	8 files	255	FHIR-PATTERNS knowledge base
Dual	CHAT + INTEL	12 files	255	MammoChat clinical agent
Triple	INTEL + CHAT + COIN	16+ files	255	Full AGENT with economic identity

The axiom declares the pattern, the pattern determines the architecture, and the architecture compiles to 255. Start with the axiom — the rest follows.

1.13. 1.14 Axiom Testing: The Five-Question Checklist

Before committing an axiom, run it through this checklist. Every question must answer YES:

#	Question	Test	Example (PASS)	Example (FAIL)
1	Is it one sentence?	Count periods	“MammoChat serves breast imaging…”	Two paragraphs of description
2	Is it declarative?	Contains “is/serves/stores” — not “aims/strives”	“WALLET stores COIN”	“We aim to store COIN”
3	Is it testable?	Can `magic validate` check it?	“Every recommendation cited”	“Best clinical support”
4	Does it name the domain?	Specific noun present	“breast imaging”	“healthcare”
5	Does it state the guarantee?	Observable commitment	“Every event signed”	“Secure system”

```
# Validate axiom quality programmatically
magic validate --axiom-check
# Output:
#   Declarative: PASS (verb: "serves")
#   Testable: PASS (guarantee: "Every recommendation cited")
#   Domain-bounded: PASS (domain: "breast imaging")
#   Single-sentence: PASS (1 sentence)
#   Bold-formatted: PASS (**...**)
```

If any question answers NO, rewrite the axiom before proceeding. The five-question checklist takes 30 seconds. A bad axiom costs weeks of rework downstream.

Chapter 2

Chapter 2: The TRIAD

Every governance framework eventually drowns in its own paperwork. CANONIC bets on the opposite: three files are enough to declare a governed scope. CANON.md states what you believe. VOCAB.md defines what your words mean. README.md describes what you offer the world. If you write nothing else, write these three. They are the foundation that the validator evaluates, that other scopes inherit from, and that auditors examine.

2.1. 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
`README.md`	Interface — what you offer the world	Header file / public API	Service description

Without the TRIAD, your scope score is 0. With it, you have the minimum viable governance declaration — enough for the validator to evaluate, for other scopes to inherit from, and for auditors to examine [5].

2.2. 2.2 CANON.md Structure

```
# {SCOPE} — CANON

version: {YYYY-MM}


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

2.3. Axiom

{One sentence. The contract.}

• • •

2.4. Constraints

“‘ MUST: {binding obligation} MUST NOT: {binding prohibition} “‘

• • •

Every governed Markdown file follows this structure: header, `inherits:`, separator, content, separator, f

The structure maps directly to clinical protocol governance. Consider a MammoChat deployment at a specific

```markdown
# MAMMOCHAT-UCF — CANON

## Axiom
```
```

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

```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

2.5. 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 “‘

• • •

The constraints are not suggestions. They are binding obligations that `magic validate` will check. "MUST

2.3 VOCAB.md Structure

```markdown
# VOCAB

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

```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

An undefined term is a type error. A stub definition (`"--"` or `"Governed term."`) is a gap, not a closur

In clinical informatics, vocabulary precision is patient safety. Define every clinical term the scope uses

```markdown
# VOCAB
```

Term	Definition
BI-RADS	Breast Imaging Reporting and Data System. ACR standardized classification (0-6) for mammograph
SCREENING	Population-level breast imaging for asymptomatic patients. Distinct from DIAGNOSTIC.
DIAGNOSTIC	Targeted breast imaging for symptomatic patients or abnormal screening findings.
TRIAGE	Classification of screening findings into follow-up categories based on BI-RADS assessment.
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.

```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

If MammoChat uses "SCREENING" without a VOCAB.md definition, the validator flags it — just as a compiler f
and in clinical AI, runtime bugs are patient safety events.

2.4 README.md Structure

```markdown
# {SCOPE} — {Title}

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

```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

{SCOPE} | README | {DOMAIN}

The README is the public interface — the service description that external consumers read, internal consum

BLOAT EXTINCTION

The TRIAD files were originally all hand-authored. Production experience revealed that COVERAGE.md and REA
developers maintaining generated content by hand, creating drift risk. BLOAT EXTINCTION resolved this:

- **COVERAGE.md** is now generated by the build pipeline from the eight-question assessment. Developers an
- **README.md** is minimal — inherits: + axiom + purpose. No hand-maintained feature lists, no changelog,

The TRIAD remains the foundation, but leaner. CANON.md is still hand-authored — the axiom is human judgmen
term definitions require domain expertise. README.md is minimal by design. COVERAGE.md is generated. The b

```
## 2.5 The Recursive Property

The TRIAD appears at every level of the governance tree — organization, team, project, service. The valida
```

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

```
Same three files at every scale. A 5,000-employee health system and a single-physician AI deployment use t
that is why it scales.

## 2.6 The TRIAD in EHR Integration

When you integrate CANONIC with an existing EHR (Epic, Cerner, MEDITECH), the TRIAD becomes the integratio
FHIR resource types, HL7 message types, clinical terminology standards. README.md describes the interface
endpoints, data flows, clinical workflows.

When the EHR vendor asks "what does your AI governance framework require?" — three files. When IT security
CANON.md. When the clinical informatics committee asks "what do your terms mean?" — VOCAB.md. Everything e

## 2.7 Building the TRIAD: Step-by-Step

Open a terminal. Create the scope. Write the three files in order: CANON.md first (the contract), VOCAB.md

```bash
mkdir -p CARDIO-CHAT
cd CARDIO-CHAT
```

Step 1: CANON.md. Write the axiom and constraints. The axiom is the single sentence that defines the scope. The constraints derive from the axiom.

```
cat > CANON.md << 'EOF'
# CARDIO-CHAT — CANON

version: 2026-03


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

2.6. Axiom

CardioChat serves cardiology with governed AHA/ACC INTEL. Every recommendation graded. Every interaction audited.

• • •

2.7. Constraints

MUST: Cite AHA/ACC guideline for every cardiovascular recommendation MUST: Include evidence grade (Class I/IIa/IIb/III) for every recommendation MUST: Log every interaction to LEDGER with timestamp and actor MUST: Version evidence base with AHA/ACC publication date MUST: Maintain PHI boundary — no patient identifiers in governance metadata MUST NOT: Generate uncited cardiovascular recommendations MUST NOT: Override cardiologist clinical judgment MUST NOT: Process PHI outside the local deployment perimeter MUST NOT: Provide dosing recommendations without weight-based calculation source

• • •

EOF

```
Verify the CANON.md structure. Every CANON.md must have: (1) a header with the scope name, (2) an `inherit

**Step 2: VOCAB.md.** Define every SCREAMING\_CASE term the scope uses. Start with terms from CANON.md and

```bash
cat > VOCAB.md << 'EOF'
# VOCAB

| Term | Definition |
|------|-----------|
| CARDIO-CHAT | Governed clinical TALK agent serving cardiology queries with AHA/ACC evidence |
| AHA | American Heart Association — publisher of cardiovascular clinical guidelines |
| ACC | American College of Cardiology — co-publisher of cardiovascular clinical guidelines |
| 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-GRADE | AHA/ACC classification: Class I (benefit >>> risk), IIa (benefit >> risk), IIb (benefit
| STEMI | ST-Elevation Myocardial Infarction — time-critical cardiac event requiring immediate interventio
| NSTEMI | Non-ST-Elevation Myocardial Infarction — acute coronary syndrome without ST elevation |
| ASCVD | Atherosclerotic Cardiovascular Disease — primary prevention target for statin therapy |


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip

EOF


Every term must have a precise definition. Not a stub. Not a circular reference. Not "Governed term." The

**Step 3: README.md.** Write the public interface. This is what external consumers see — the hospital's cl

```bash
cat > README.md << 'EOF'
# CARDIO-CHAT — Cardiovascular Clinical Decision Support


CardioChat deployment for cardiovascular clinical decision support. Provides governed clinical recommendat
no patient identifiers in governance metadata.

## Capabilities

- Cardiovascular risk assessment (ASCVD 10-year risk)
- Guideline-directed medical therapy recommendations
- Acute coronary syndrome triage guidance (STEMI/NSTEMI)
- Heart failure management (HFrEF/HFpEF staging)
- Anticoagulation management (AF, VTE)

## Limitations

- Does not replace cardiologist clinical judgment
- Does not provide dosing without weight-based calculation source
- Does not process PHI outside deployment perimeter


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip

EOF
```
```

```
**Step 4: Validate.**

```bash
git add CANON.md VOCAB.md README.md
git commit -m "GOV: bootstrap CARDIO-CHAT — TRIAD"
magic validate
```

Expected output:

```
CARDIO-CHAT: 35/255 (COMMUNITY)
  D: PASS  — CANON.md present, axiom valid
  E: PASS  — VOCAB.md present, 10 terms, 0 stubs
  T: FAIL  — ROADMAP.md missing
  R: FAIL  — CARDIO-CHAT.md missing
  O: FAIL  — COVERAGE.md missing
  S: PASS  — inherits: valid, axiom present, constraints present
  L: FAIL  — LEARNING.md missing
  LANG: FAIL — LANGUAGE not inherited
```

Score 35. COMMUNITY tier. The TRIAD is complete. The scope exists and is governed. Now build upward.

2.8. 2.8 TRIAD Validation Errors and Fixes

The validator produces specific errors for TRIAD problems. Here is the complete error catalog for the three files [6]:

CANON.md errors:

Error	Cause	Fix
`CANON.md not found`	File missing	Create CANON.md with axiom and constraints
`Axiom section missing`	No `## Axiom` heading	Add `## Axiom` section with bold sentence
`Axiom empty`	`## Axiom` exists but no bold text follows	Add `**{axiom text}**` after the heading
`Constraints section missing`	No `## Constraints` heading	Add `## Constraints` with MUST/MUST NOT
`No MUST constraints`	Constraints section has no MUST entries	Add at least one `MUST:` constraint
`inherits: missing`	No `inherits:` line in header	Add `inherits: {parent/path}`
```

Error	Cause	Fix
`inherits: unresolvable`	Parent path does not exist	Fix the path or create the parent scope
`Footer missing`	No `*CANON | {SCOPE} | {DOMAIN}*` line	Add footer in the correct format

VOCAB.md errors:

Error	Cause	Fix
`VOCAB.md not found`	File missing	Create VOCAB.md with term table
`No term table`	VOCAB.md has no Markdown table	Add `| Term | Definition |` table
`Stub definition: "{TERM}"`	Term has `--` or empty definition	Write a precise definition
`Circular definition: "{TERM}"`	Term defined using itself	Rewrite without self-reference
`Undefined term: "{TERM}"`	SCREAMING_CASE term used in scope but not in VOCAB.md	Add the term to VOCAB.md
`Orphan term: "{TERM}"`	Term in VOCAB.md but never used in scope	Remove the term or use it

README.md errors:

Error	Cause	Fix
`README.md not found`	File missing	Create README.md with scope description
`inherits: missing`	No `inherits:` line	Add `inherits: {parent/path}`
`Empty body`	README.md has header but no content	Add scope description
`Footer missing`	No footer line	Add `*{SCOPE} | README | {DOMAIN}*`

When `magic validate` reports errors, fix them in order: CANON.md first, VOCAB.md second, README.md third. The validator processes files in this order because CANON.md establishes the evaluation context, VOCAB.md defines the terms CANON.md uses, and README.md describes the interface CANON.md constrains.

2.9. 2.9 VOCAB.md: The Type System in Practice

VOCAB.md is not a glossary — it is a type system. Every SCREAMING_CASE term in the scope is a type, and the definition is its specification. The validator checks type consistency across all files in the scope [7].

Here is how the type-checking works: the validator extracts every SCREAMING_CASE token from every file in the scope — CANON.md, README.md, COVERAGE.md, LEARNING.md, INTEL.md, SHOP.md — then resolves each token against VOCAB.md (local scope first, then the inherited VOCAB.md chain). An unresolved token is a type error.

```
ERROR: CANON.md line 12 — Undefined term: ASCVD
  Term "ASCVD" used in constraints but not defined in VOCAB.md
  Fix: Add ASCVD definition to VOCAB.md
  Hint: Check inherited VOCAB.md — term may be defined in parent scope
```

The inheritance chain applies to VOCAB.md as well. If the parent scope defines INTEL in its VOCAB.md, the child scope does not need to redefine INTEL — it inherits the definition. But if the child scope uses INTEL with a different meaning than the parent, the child must override the definition in its own VOCAB.md. The override is explicit. The validator detects when a child redefines a parent term and flags it as a warning (not an error — overrides are permitted but should be intentional).

```
WARN: VOCAB.md — Term "INTEL" overrides parent definition.
  Parent (hadleylab-canonic/MAGIC/VOCAB.md): "Governed knowledge unit with provenance"
  Child (CARDIO-CHAT/VOCAB.md): "AHA/ACC governed knowledge unit"
  This override narrows the parent definition. Verify this is intentional.
```

The type system prevents a category of errors that plagues traditional clinical AI deployments: term collision. When Hospital A's "SCREENING" means population-level mammography and Hospital B's means any diagnostic imaging, a federated system without vocabulary governance silently mixes the two meanings. VOCAB.md prevents this — each hospital's scope defines "SCREENING" precisely, and the validator detects the collision when the federation scope attempts to inherit both definitions.

2.10. 2.10 README.md: The Public API

Think of README.md as a header file. It declares what the scope offers without exposing implementation details. Other scopes, integration teams, compliance auditors, and clinical informatics committees read it to understand what the scope does and how to interact with it [5].

Three sections:

1. **Description**: What the scope does. One paragraph. No jargon beyond what VOCAB.md defines.

2. **Capabilities**: What the scope offers. Bulleted list. Each capability maps to a constraint in CANON.md.
3. **Limitations**: What the scope does NOT do. Bulleted list. Each limitation maps to a MUST NOT in CANON.md.

Capabilities and limitations are not documentation — they are the public API contract. When another scope inherits from yours, README.md is the interface specification. When a FHIR integration connects to a clinical TALK agent, README.md describes what the agent can and cannot do.

In clinical AI deployments, README.md also serves a regulatory function: it is the product description the committee reviews during the AI governance approval process. The committee does not need CANON.md (the internal contract) or VOCAB.md (the internal type system). They read README.md — the external interface description. Internal governance versus external interface: the TRIAD enforces that separation architecturally, not bureaucratically.

2.11. 2.11 The TRIAD Completeness Theorem

The TRIAD covers the three essential governance questions — and nothing more:

Question	File	Failure Mode if Missing
What do you believe?	CANON.md	No contract — scope behavior is undefined
What do your words mean?	VOCAB.md	No type system — terms are ambiguous
What do you offer?	README.md	No interface — external consumers cannot evaluate the scope

Leave any of these unanswered and governance is incomplete. CANON.md without VOCAB.md gives you a contract with ambiguous terms — unenforceable because its vocabulary is undefined. VOCAB.md without CANON.md gives you defined terms describing nothing. README.md without CANON.md gives you a public interface making promises that no contract enforces.

The TRIAD is the minimum, not the maximum. A scope at COMMUNITY tier (35/255) has only the TRIAD plus structural governance (inherits: + axiom + constraints). A scope at FULL (255/255) adds ROADMAP.md, {SCOPE}.md, COVERAGE.md, LEARNING.md, and LANGUAGE inheritance. Everything else accumulates on the TRIAD foundation.

2.12. 2.12 TRIAD Anti-Patterns

The empty VOCAB.md. Zero terms defined. The file exists, so the proof question gets partial credit, but the validator flags the empty table. Every scope uses at least one SCREAMING_CASE term — the scope name itself. Define it.

The copy-paste README.md. Copying the parent's README.md verbatim into the child. The child's README.md must describe the child's interface, not the parent's. MammoChat describes breast imaging capabilities, not the TALK service's generic conversation features.

The prose CANON.md. Writing CANON.md as a narrative instead of a structured governance file. Its structure is fixed: header, inherits, axiom, constraints, footer. Prose belongs in README.md. Policy narratives belong in {SCOPE}.md. CANON.md is compiler input — write it like code.

The undeclared term. Using SCREAMING_CASE terms in CANON.md that are not in VOCAB.md. Every `MUST: Cite INTEL source` requires INTEL to be defined. Every `MUST NOT: Process PHI` requires PHI to be defined. The validator catches this — fix it before committing.

```
# Check for undefined terms before committing
magic validate --verbose | grep "Undefined term"
```

The disconnected README.md. Describing capabilities not declared in CANON.md. If README.md says "provides dosing recommendations" but CANON.md has `MUST NOT: Provide dosing recommendations`, the validator detects the contradiction. README.md is the public interface of the CANON.md contract — the two must be consistent.

Three files. Write them correctly, write them in order, write them first. Everything in CANONIC builds on this foundation. Chapter 5 walks through writing these files from scratch, and Chapter 6 covers how scopes accumulate additional governance files to reach 255. For the governance policy rationale behind the TRIAD, see The Canonic Canon, Chapter 2.

Chapter 3

Chapter 3: The Inheritance Chain

Most governance systems rely on humans reading policy documents and remembering to follow them. CANONIC replaces that with a single line — `inherits:` — that is a binding obligation, not metadata. Write it, and the compiler resolves and enforces every constraint in your parent's governance chain, all the way to the root [8].

3.1. 3.1 Syntax

This line says: "I accept all governance constraints from `hadleylab-canonic/PAPERS`." Not a reference. Not a dependency. 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. 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 [8].

Resolution is deterministic. The same `inherits:` chain always produces the same constraint set. The

validator does not use heuristics, does not make judgments, does not apply discretion. It resolves the chain, collects the constraints, and checks them. Pass or fail. 255 or less.

The practical consequence: 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. 3.3 Monotonic Accumulation

A child scope can add constraints beyond its parent. It cannot remove or weaken them. This is the Liskov Substitution Principle applied to governance: a subtype extends but never violates the parent contract [8].

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

In a hospital network, this means 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, local IRB requirements. It cannot remove them. The network's governance floor is architecturally enforced.

This is what makes the model compelling for hospital CISOs: the governance floor is not a policy that humans must remember to follow. It is a constraint 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. 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 [7].

3.5. 3.5 The Root

The root of the entire system is `canonic-canonic`. It defines the fundamental primitives (INTEL, CHAT, COIN), service composition, tier algebra, and the validation framework (MAGIC). Every organization inheriting from it accepts these foundational constraints [8].

```
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 [9].

3.6. 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 the descent [9].

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 deployment inherits the network's HIPAA constraints, Hospital A's site-specific constraints, the TALK service's clinical constraints, and the root's foundational constraints — all resolved, collected, and enforced at validation time.

3.7. 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. No leaf can weaken any ancestor's constraint. Build the tree correctly, and the governance follows.

3.8. 3.8 Building an Inheritance Chain: Step-by-Step

This walkthrough builds a three-level inheritance chain: organization, service, and deployment. Each level adds constraints. The validator enforces the entire chain [8].

Level 1: Organization scope. Create the organization root.

```
mkdir -p myhealth-canonic/MAGIC/SERVICES
cd myhealth-canonic

cat > CANON.md << 'EOF'
# MYHEALTH — CANON

version: 2026-03


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```
```

3.9. Axiom

MyHealth governs clinical AI deployments with MAGIC compliance. Every scope validated. Every event audited.

• • •

3.10. Constraints

MUST: Validate every scope to 255 before production deployment MUST: Maintain PHI boundary per HIPAA §164.312 MUST: Record every governance event on LEDGER MUST: Inherit from canonic-canonic/MAGIC for all services MUST NOT: Deploy clinical AI without ENTERPRISE tier minimum MUST NOT: Process PHI outside organizational perimeter MUST NOT: Bypass validation for any deployment

• • •

EOF

````
**Level 2: Service scope.** Create the TALK service under the organization.

```bash
mkdir -p MAGIC/SERVICES/TALK
cd MAGIC/SERVICES/TALK

cat > CANON.md << 'EOF'
# TALK — CANON

version: 2026-03


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
````

3.11. Axiom

TALK serves clinical conversations with governed INTEL. Every response cited. Every session audited.

• • •

3.12. Constraints

MUST: Include systemPrompt with scope identity and disclaimer MUST: Cite evidence source for every clinical recommendation MUST: Log every session to LEDGER with session ID and timestamp MUST: Display disclaimer before first clinical interaction MUST NOT: Generate uncited clinical recommendations MUST NOT: Claim to replace clinical judgment MUST NOT: Store session content beyond audit retention period

• • •

EOF

````
**Level 3: Deployment scope.** Create the specific hospital deployment.

```bash
mkdir -p MAMMOCHAT-REGIONAL
cd MAMMOCHAT-REGIONAL

cat > CANON.md << 'EOF'
# MAMMOCHAT-REGIONAL — CANON

version: 2026-03


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
````

3.13. Axiom

MammoChat-Regional serves Regional Hospital breast imaging with governed BI-RADS INTEL. Every recommendation cited. Every interaction audited.

• • •

3.14. Constraints

MUST: Cite BI-RADS category for every screening assessment MUST: Reference ACR BI-RADS Atlas 5th Edition for classification MUST: Comply with Florida Statute 404.22 radiation safety MUST: Include Regional Hospital disclaimer per legal dept. template MUST NOT: Recommend biopsy without BI-RADS 4+ classification MUST NOT: Override radiologist clinical judgment

• • •

EOF

Validate the chain.

```bash
magic validate --scope myhealth-canonic --recursive
```

Expected output:

```
myhealth-canonic: 35/255 (COMMUNITY)
  Chain: canonic-canonic → myhealth-canonic
  Constraints collected: 7 (org) + inherited (root)

myhealth-canonic/MAGIC/SERVICES/TALK: 35/255 (COMMUNITY)
  Chain: canonic-canonic → myhealth-canonic → TALK
  Constraints collected: 7 (service) + 7 (org) + inherited (root)

myhealth-canonic/MAGIC/SERVICES/TALK/MAMMOCHAT-REGIONAL: 35/255 (COMMUNITY)
  Chain: canonic-canonic → myhealth-canonic → TALK → MAMMOCHAT-REGIONAL
  Constraints collected: 6 (deployment) + 7 (service) + 7 (org) + inherited (root)
```

The validator reports the full chain for each scope. The constraint count accumulates upward. MAMMOCHAT-REGIONAL inherits every constraint from every ancestor — root, organization, service, and its own. The total constraint set is the union. No constraint is lost. No constraint is weakened.

3.15. 3.9 Inheritance Errors and Resolution

The validator produces specific errors for inheritance problems. Here is the error catalog [6]:

Error	Cause	Fix
`inherits: missing`	No `inherits:` line	Add `inherits: {parent/path}` to file header
`inherits: unresolvable — path not found`	Parent scope does not exist	Create the parent scope or fix the path
`inherits: chain broken at {SCOPE}`	Intermediate ancestor missing CANON.md	Add CANON.md to the intermediate scope
`inherits: circular dependency`	Scope A inherits B, B inherits A	Break the cycle — restructure the tree
`Constraint violation: child weakens parent`	Child MUST NOT contradicts parent MUST	Remove the weakening constraint from child
`Constraint conflict: inherited chains diverge`	Two ancestors have contradictory MUST entries	Resolve at the common ancestor level

The most common error in practice is `inherits: unresolvable` — a developer typed the wrong path. The fix is mechanical:

```
# Check if the parent scope exists
ls -la $(magic resolve-path hadleylab-canonic/MAGIC/SERVICES/TALK)/CANON.md

# If not found, the inherits: path is wrong. List available scopes:
magic scan --scopes | grep TALK
```

The second most common is `Constraint violation: child weakens parent` — a child adds a MUST NOT that contradicts an inherited MUST. For example, the parent says `MUST: Log every interaction to LEDGER` and the child says `MUST NOT: Log interactions for privacy`. The child cannot weaken the parent. Either modify the child's constraint to comply, or negotiate a change at the parent level.

```
ERROR: MAMMOCHAT-REGIONAL/CANON.md
  Constraint violation: child weakens parent
  Parent (TALK): MUST: Log every session to LEDGER
  Child: MUST NOT: Log clinical sessions
```

```
  Fix: Remove "MUST NOT: Log clinical sessions" from child
  Note: If the parent constraint is wrong, fix it at the parent level.
        Child scopes cannot weaken parent constraints.
```

3.16. 3.10 Constraint Propagation Mechanics

When the validator collects constraints from the inheritance chain, it applies three rules [8]:

Rule 1: Union. All MUST constraints from all ancestors are collected into a single set. The child must satisfy every MUST in the union.

Rule 2: Monotonic. MUST NOT constraints can only be added by children, never removed. If the root says `MUST NOT: Process PHI externally`, every descendant inherits that prohibition.

Rule 3: Specificity. When a child adds a constraint that is more specific than a parent constraint, both apply. The parent says `MUST: Cite evidence`. The child says `MUST: Cite BI-RADS evidence`. The child must cite BI-RADS evidence (specific) AND cite evidence generally (parent). The specific constraint does not replace the general constraint — it adds to it.

These three rules produce deterministic constraint resolution. Given the same inheritance chain, the validator always produces the same constraint set. No human judgment, no discretion, no exceptions.

3.17. 3.11 Practical Inheritance Patterns

Pattern 1: The Service Specialization. A common pattern in clinical AI governance. The TALK service defines general conversation constraints. Each clinical TALK agent specializes by adding domain-specific constraints.

```
SERVICES/TALK/CANON.md
  MUST: Include systemPrompt with disclaimer
  MUST: Cite evidence for recommendations

SERVICES/TALK/MAMMOCHAT/CANON.md
  inherits: SERVICES/TALK
  MUST: Cite BI-RADS category (adds to parent's general evidence citation)
  MUST: Reference ACR BI-RADS Atlas (adds specific evidence source)

SERVICES/TALK/ONCOCHAT/CANON.md
  inherits: SERVICES/TALK
  MUST: Cite NCCN guideline version (adds to parent's general evidence citation)
  MUST: Include cancer staging in recommendations (adds domain-specific requirement)
```

Both agents inherit TALK's general constraints, add domain-specific requirements, and satisfy the full constraint union without weakening the parent.

Pattern 2: The Multi-Site Deployment. A single product deployed across multiple hospitals. Each hospital adds site-specific constraints.

```
MAMMOCHAT/CANON.md
  MUST: Cite BI-RADS category
  MUST NOT: Process PHI outside deployment perimeter

MAMMOCHAT/MAMMOCHAT-UCF/CANON.md
  inherits: MAMMOCHAT
  MUST: Comply with UCF IRB protocol #2026-001
  MUST: Include UCF College of Medicine disclaimer

MAMMOCHAT/MAMMOCHAT-ADVENT/CANON.md
  inherits: MAMMOCHAT
  MUST: Comply with AdventHealth system-wide AI governance policy
  MUST: Include faith-based care statement per AdventHealth policy
```

Each hospital deployment inherits the product's constraints and adds site-specific requirements — UCF adds IRB compliance, AdventHealth adds organizational policy — without weakening the core constraints (BI-RADS citation, PHI boundary).

Pattern 3: The Cross-Organization Federation. Multiple organizations inherit from the same root. Each organization maintains independent governance within the shared constraint framework.

```
canonic-canonic/CANON.md
  MUST: Validate to 255 before production
  MUST: Record every event on LEDGER

hadleylab-canonic/CANON.md
  inherits: canonic-canonic
  MUST: Academic research governance (IRB, publication ethics)

adventhealth-canonic/CANON.md
  inherits: canonic-canonic
  MUST: HIPAA compliance per AdventHealth system policy
  MUST: Faith-based care governance
```

Both organizations inherit the root's foundational constraints and add their own. The federation is the shared constraint floor; organizations build on top of it.

3.18. 3.12 Debugging Inheritance Problems

When a scope fails validation due to inheritance issues, follow this debugging procedure:

```
# Step 1: Print the full inheritance chain
magic validate --scope MAMMOCHAT-REGIONAL --chain
# Output: canonic-canonic → myhealth-canonic → TALK → MAMMOCHAT-REGIONAL

# Step 2: Validate each ancestor individually
magic validate --scope canonic-canonic
magic validate --scope myhealth-canonic
magic validate --scope myhealth-canonic/MAGIC/SERVICES/TALK

# Step 3: Identify which ancestor introduced the failing constraint
magic validate --scope MAMMOCHAT-REGIONAL --verbose | grep "FAIL"

# Step 4: Check constraint origin
magic validate --scope MAMMOCHAT-REGIONAL --constraint-origin "MUST: Log every session"
# Output: Constraint "MUST: Log every session" originated at myhealth-canonic/MAGIC/SERVICES/TALK
```

Walk the chain from root to leaf. At each step, the validator reports the constraint set. The step where the constraint count changes is the step where the failing constraint was introduced. Fix it at the origin and the fix propagates downward to all descendants.

The inheritance chain is the backbone of CANONIC governance. Build the chain correctly, and the governance follows. For the theoretical basis, see Chapter 36 (Governance as Type System) and Chapter 37 (Governance as Compiler), which prove that inheritance corresponds to subtyping in programming language theory.

3.19. 3.13 Inheritance Performance and Depth Limits

Inheritance chains resolve in O(d) time, where d is the chain depth. In practice, depth rarely exceeds 6 levels:

Depth	Typical Chain	Resolution Time
1	root □ org	< 1ms
2	root □ org □ service	< 2ms
3	root □ org □ service □ product	< 3ms
4	root □ org □ service □ product □ site	< 4ms
5	root □ org □ service □ product □ site □ department	< 5ms
6+	Deeper chains	Discouraged — debug complexity increases

The validator caches resolved constraint sets per session. Once it resolves `hadleylab-canonic/MAGIC/SERVICES/TALK`, every child scope that inherits from TALK reuses the cached result. A health network with 50 deployments all inheriting from TALK resolves that constraint set once, not 50 times.

```
# Profile inheritance resolution
magic validate --scope MAMMOCHAT-UCF --profile
# Chain resolution:
#   canonic-canonic: 4 constraints (cached)
#   hadleylab-canonic: 7 constraints (cached)
#   MAGIC/SERVICES/TALK: 7 constraints (cached)
#   MAMMOCHAT: 6 constraints (resolved)
#   MAMMOCHAT-UCF: 5 constraints (resolved)
# Total: 29 constraints collected in 3.2ms
```

3.20. 3.14 Inheritance and the _generated Contract

The `inherits:` chain interacts with the `_generated` contract (as described in Chapter 26). When the build pipeline compiles `.md` governance files into `.json` runtime artifacts, the compiler resolves the inheritance chain and embeds the full constraint set in the compiled output:

```
CANON.md (source, inherits: hadleylab-canonic/MAGIC/SERVICES/TALK)
  → build pipeline resolves inheritance chain
    → collects constraints from all ancestors
      → compiles to CANON.json (output, contains full constraint set)
```

The compiled CANON.json contains resolved constraints from every ancestor. The runtime reads the compiled artifact instead of re-resolving the chain — O(1) constraint lookup at runtime, full chain walk at compile-time.

If the compiled CANON.json contains an incorrect constraint, do not edit it. It is `_generated`. Fix the CANON.md source or the ancestor's constraints, run `build`, and the pipeline recompiles. The fix propagates.

3.21. 3.15 Multi-Fleet Inheritance

When CANONIC is deployed across multiple organizations (fleets), inheritance crosses fleet boundaries via cross-fleet references:

```
canonic-canonic/MAGIC                    ← Fleet A (root)
  └── hadleylab-canonic/MAGIC/SERVICES   ← Fleet B (org A)
  └── adventhealth-canonic/MAGIC/SERVICES← Fleet C (org B)
```

Both Fleet B and Fleet C inherit from Fleet A. Cross-fleet inheritance resolves at build time by cloning the parent fleet's governance files as git submodules, ensuring the parent's constraints are versioned — each child fleet pins a specific commit. Updating the parent requires an explicit submodule bump and revalidation.

```
# Update parent fleet reference
cd hadleylab-canonic
git submodule update --remote canonic-canonic
magic validate --recursive --strict
# If all scopes pass: commit the submodule bump
git commit -m "GOV: bump canonic-canonic — updated root constraints"
```

The submodule bump is itself a governance event. The pre-commit hook validates all scopes against the updated parent constraints, and if any scope fails, the bump is blocked. Fix your governance before adopting the parent's updates.

3.22. 3.16 Inheritance Anti-Patterns

Avoid these common mistakes:

Diamond inheritance. Scope D inherits from both B and C, which both inherit from A. CANONIC does not support multiple inheritance — each scope has exactly one `inherits:` directive. If you need constraints from two parents, create a common ancestor.

Phantom inheritance. Writing `inherits: hadleylab-canonic/MAGIC` when the scope uses no constraints from MAGIC. Do not inherit from a scope whose constraints you do not intend to satisfy.

Stale inheritance. Inheriting from a scope that has been archived or renamed. Run `magic validate` to detect broken chains — it reports `inherits: unresolvable` with the exact fix.

Deep inheritance. Chains deeper than six levels without operational justification. Each level adds debugging complexity. The practical test: can a new developer trace the full chain in under two minutes? If not, flatten the hierarchy by merging intermediate scopes. Deep chains are not wrong — they are expensive, and the cost is paid in debugging time when a constraint violation surfaces at a leaf and you must walk six ancestors to find the origin. Keep the chain as shallow as the domain permits. For multi-organization federation across inheritance chains, see Chapter 9. For the governance policy perspective on inheritance, see The Canonic Canon, Chapter 3.

Chapter 4

Chapter 4: The Eight Dimensions

Governance frameworks typically measure compliance with checklists that grow without bound. CANONIC measures it with eight questions. Each is binary — answered or not. Together they produce a single score: 0 to 255, the full range of an 8-bit unsigned integer. You either satisfy a dimension or you do not, and the score is a mathematical fact, not an assessment.

4.1. 4.1 The Dimensions

#	Question	What It Validates
1	What do you believe?	The scope has declared its purpose — an axiom, constraints, a governed identity
2	Can you prove it?	The scope has defined its terms — a closed vocabulary, no undefined concepts
3	Where are you going?	The scope has a timeline — milestones, trajectory, planned evolution
4	Who are you?	The scope has described itself — an interface specification others can read
5	How do you work?	The scope has assessed its coverage — an operational self-check
6	What shape are you?	The scope has structure — inheritance, axiom, and formal constraints
7	What have you learned?	The scope has captured patterns — accumulated intelligence from governance events
8	How do you express?	The scope speaks a governed language — inherited design vocabulary

Each question is binary: answered or not. The score is a bitmask — the sum of all answered questions.

The math is in the C kernel (`magic.c`); the governance is in the files [6].

4.2. 4.2 The Score

255 is the state in which all eight governance questions are answered simultaneously. 0 means none are. The kernel computes the score deterministically from the governance files present in the scope [10].

There is no human judgment in the scoring. No discretion. No "close enough." The files exist or they do not. The structures are present or they are not. The same governance files always produce the same score.

4.3. 4.3 Compliance Tiers

Tier	Questions Answered	Score	Healthcare Minimum
COMMUNITY	3 of 8	35	Research prototype
BUSINESS	4 of 8	43	Internal pilot
ENTERPRISE	6 of 8	63	Clinical deployment
AGENT	7 of 8	127	Learning clinical AI
FULL (MAGIC)	All 8	255	Production clinical AI

Tiers are cumulative. You cannot skip dimensions. BUSINESS requires COMMUNITY. AGENT requires ENTERPRISE [6].

The tiers map directly to clinical AI deployment readiness. COMMUNITY means you have declared your purpose and defined your terms, but you have no roadmap, no coverage assessment, and no production readiness. It is a research prototype.

ENTERPRISE adds history (ROADMAP.md) and operational self-assessment (COVERAGE.md) — the minimum tier for clinical deployment. The compliance committee can review coverage, and the roadmap shows governance trajectory.

AGENT adds LEARNING.md — accumulated intelligence from governance events. The scope learns from its operation and starts providing continuous quality improvement signals, not just serving clinical queries.

FULL (255) adds language governance — the scope expresses itself in a controlled vocabulary inherited from the LANGUAGE standard. Production-grade clinical AI: fully governed, fully documented, fully auditable.

4.4. 4.4 Clinical Dimension Mapping

Each dimension maps to specific healthcare compliance requirements:

MAGIC Question	HIPAA	FDA Part 11	Joint Commission	HITRUST
What do you believe?	Purpose limitation	Record declaration	Service definition	Risk scope
What proves it?	PHI evidence	ALCOA evidence	Quality evidence	Security evidence
Where are you going?	Processing trans-parency	Change control	Quality improvement plan	Monitoring plan
Who are you?	Reproducible controls	Validation protocol	Reproducible quality	Reproducible security
How does it work?	Operational safe-guards	Operational validation	Operational compliance	Operational controls
What shape is it?	Structural integrity	System structure	Organizational structure	Framework structure
What patterns emerge?	Pattern detection	Change detection	Quality learning	Continuous monitoring
How is it expressed?	Controlled vocabu-lary	Legibility	Quality vocabulary	Security vocabulary

Build a scope to 255 and you simultaneously satisfy governance requirements across every major healthcare compliance standard. The dimensions themselves are not healthcare-specific — they are universal governance dimensions — but in healthcare, each one maps to specific regulatory requirements. The mapping is structural, not approximate.

4.5. 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 [6].

```
# COVERAGE

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

```
| 8 | 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. The scope claims its own compliance; the validator audits the claim.

4.6. 4.6 Kernel Internals

The bit weights, hex values, and tier boundary calculations are kernel internals, implemented in the C binary (`magic.c`) and not published in governance prose [2]. Tier names, dimension names, and composition formulas are public. The scoring algorithm is public. The bit-weight assignments are private.

Do not attempt to reverse-engineer the bit weights from the tier scores. You do not need to know the register allocation strategy of `gcc` to write C programs, and 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.

4.7. 4.7 Dimension-by-Dimension Walkthrough

Each dimension has specific requirements, specific evidence files, and specific failure modes. This section walks through every dimension with the detail a developer needs to satisfy it [6].

Question 1: "What do you believe?"

This question checks for a valid CANON.md. The requirements:

Requirement	Check	Failure
CANON.md exists	File presence	`CANON.md not found`
`## Axiom` section present	Section heading	`Axiom section missing`
Axiom is bold-formatted	`**{text}**` format	`Axiom empty or malformed`
`## Constraints` section present	Section heading	`Constraints section missing`
At least one MUST entry	MUST keyword scan	`No MUST constraints`

The first question is the gate. Without it, no other question can be evaluated. A scope without CANON.md scores 0 because the validator has no axiom to evaluate against.

```
# Check first question specifically
magic validate --verbose
# Output: Q1: PASS — CANON.md present, axiom valid, 5 constraints
```

Question 2: “Can you prove it?”

This question checks for a valid VOCAB.md. The requirements:

Requirement	Check	Failure
VOCAB.md exists	File presence	`VOCAB.md not found`
Term table present	Markdown table with Term/Definition columns	`No term table`
No stub definitions	Each definition is non-empty and non-placeholder	`Stub definition: "{TERM}"`
All scope terms defined	Every SCREAMING_CASE token resolves	`Undefined term: "{TERM}"`

Undefined clinical terms create patient safety risks. A scope that uses “SCREENING” without defining whether it means population-level or diagnostic screening introduces clinical ambiguity that propagates to patient care.

```
# Check proof question and list all terms
magic validate --question 2 --verbose
# Output: Q2: PASS — VOCAB.md present, 12 terms, 0 stubs, 0 undefined
```

Question 3: “Where are you going?”

This question checks for ROADMAP.md. The requirements:

Requirement	Check	Failure
ROADMAP.md exists	File presence	`ROADMAP.md not found`
`inherits:` present	Header check	`inherits: missing`
At least one section (Done/Now/Next)	Content check	`ROADMAP.md empty`

ROADMAP.md answers the transparency question: where is this scope going? It is the governance trajectory document the compliance committee reviews quarterly — what governance work has been completed, what is in progress, and what is planned.

```
# MAMMOCHAT — ROADMAP


## Done
- GOV: TRIAD created. COMMUNITY tier achieved.
- GOV: COVERAGE + SPEC + ROADMAP. ENTERPRISE tier achieved.
- GOV: LEARNING.md. AGENT tier achieved.
```

````
- GOV: LANGUAGE inherited. FULL (255) achieved.

## Now
- Evidence base update: BI-RADS Atlas 6th Edition integration.
- Site deployment: MAMMOCHAT-REGIONAL for Regional Hospital.

## Next
- FDA pre-submission for clinical decision support classification.
- Multi-site federation: connect 5 hospital deployments via GALAXY.


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
````

Question 4: "Who are you?"

This question checks for {SCOPE}.md — the scope specification file. The requirements:

Requirement	Check	Failure
`{SCOPE}.md` exists	File named after scope directory	`{SCOPE}.md not found`
`inherits:` present	Header check	`inherits: missing`
Scope Intelligence table	Markdown table with Dimension/Value columns	`Spec table missing`

The scope specification file is the scope's identity document — subject, audience, evidence base, and stat

````
```markdown
# MAMMOCHAT


## Scope Intelligence

| Dimension | Value |
|---|---|
| Subject | Breast imaging clinical decision support |
| Audience | Radiologists, breast surgeons, primary care physicians |
| Evidence | ACR BI-RADS Atlas 5th Edition, NCCN Breast Cancer Screening Guidelines |
| Status | Production — 3 hospital deployments, 255/255 compliance |
| Contact | Clinical informatics team — mammochat@hadleylab.org |


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
````
````

Question 5: "How do you work?"

This question checks for COVERAGE.md. The requirements:

Requirement	Check	Failure
COVERAGE.md exists	File presence	`COVERAGE.md not found`
Eight-row question table	Coverage table with all 8 questions	`Coverage table incomplete`
Status column present	PASS/FAIL per question	`Status column missing`
Claims match reality	COVERAGE PASS claims verified against actual files	`Claim mismatch: Q1=PASS but

COVERAGE.md claims compliance status for each question; the validator audits those claims against the actu
the validator catches every discrepancy.

Question 6: "What shape are you?"

This question is composite. It checks three structural requirements simultaneously:

Requirement	Check	Failure
`inherits:` present in CANON.md	Header check	`inherits: missing`
Axiom present and valid	Section + bold text	`Axiom missing`
MUST/SHOULD constraints present	Keyword scan	`No constraints`

This question is partially satisfied by the same file that answers question 1 (CANON.md). This is intentio
a CANON.md with an axiom and constraints answers both "what do you believe?" and "what shape are you?" sim

Question 7: "What have you learned?"

This question checks for LEARNING.md. The requirements:

Requirement	Check	Failure
LEARNING.md exists	File presence	`LEARNING.md not found`
`inherits:` present	Header check	`inherits: missing`
Pattern table present	Markdown table with Date/Signal/Pattern/Source columns	`Pattern table missing`
Valid signal vocabulary	All Signal values in recognized vocabulary	`Signal "{X}" not in vocabulary`

This is the dimension that separates ENTERPRISE from AGENT. Without LEARNING.md, a scope is compliant but

Question 8: "How do you express?"

This question checks for LANGUAGE governance inheritance. The requirements:

Requirement	Check	Failure
LANGUAGE inherited	`inherits:` chain includes LANGUAGE standard	`LANGUAGE not inherited`
Controlled vocabulary active	Scope uses terms from inherited LANGUAGE	`No LANGUAGE terms detected`

This is the final gate to 255. It requires the scope to inherit from the LANGUAGE standard —
the controlled vocabulary that governs how the scope expresses itself. This is the most demanding question

4.8 Score Arithmetic Examples

The C kernel assigns each question a weight and sums the satisfied weights. The specific weights are imple

Scenario	Questions Answered	Score	Tier
Empty directory	0 of 8	0	None
CANON.md only	2 of 8 (belief + shape partial)	33	Below COMMUNITY
TRIAD complete	3 of 8	35	COMMUNITY
TRIAD + spec	4 of 8	43	BUSINESS
TRIAD + spec + roadmap + coverage	6 of 8	63	ENTERPRISE
Above + LEARNING	7 of 8	127	AGENT
All eight	8 of 8	255	FULL

No hidden weighting. No curve. No partial credit within a question. Each is binary — satisfied or not —
and the kernel computes the result deterministically.

4.9 Dimension Dependencies

The eight questions have logical (not mechanical) dependencies. The validator does not enforce ordering, b

“What do you believe?” foundation — required for all others “Can you prove it?” requires belief (terms reference the axiom) “What shape are you?” requires belief (structure is defined in CANON.md) “Where are you going?” requires belief (roadmap references the axiom’s trajectory) “Who are you?” requires belief + proof (spec references axiom and terms) “How do you work?” requires all above (COVERAGE references all questions) “What have you learned?” requires mechanism (learning is operational knowledge) “How do you express?” requires all (language governance crowns the scope)

The dependency chain suggests a build order: belief first, then proof and shape (simultaneously —
they share CANON.md), then timeline and identity, then mechanism, then learning, then expression. This is

4.10 Dimension Gaps: Diagnosis and Repair

When `magic validate` reports a score below 255, the gap is the set of unanswered questions. Diagnose the

```bash
```

```
magic validate --scope MAMMOCHAT --gaps
```

Output:

```
MAMMOCHAT: 127/255 (AGENT)
  Answered: 7 of 8 questions
  Missing: "How do you express?" — LANGUAGE not inherited
  Fix: Add LANGUAGE inheritance to CANON.md inherits: chain
  Cost: 128 COIN to close (from 127 to 255)
```

The gap diagnosis tells you exactly what is missing, why, and how to fix it. The cost tells you how much COIN the fix will mint. When you report to the compliance committee, the gap report IS the governance improvement plan.

Common gap patterns:

Gap Pattern	Missing Question	Typical Cause	Fix
“No roadmap”	Where are you going?	Team ships without planning	Add ROADMAP.md with Done/Now/Next
“No spec”	Who are you?	Team builds without describing	Add {SCOPE}.md with Scope Intelligence table
“No coverage”	How do you work?	Team does not self-assess	Add COVERAGE.md with 8-question table
“No learning”	What have you learned?	Team does not capture knowledge	Add LEARNING.md with pattern table

Gap Pattern	Missing Question	Typical Cause	Fix
“No language”	How do you express?	Team does not inherit LANGUAGE	Add LANGUAGE inheritance to inherits: chain
“TRIAD only”	5 questions unanswered	Team stopped at COMMUNITY	Build to ENTERPRISE, then AGENT, then FULL

The fix for every gap is the same: create the missing file with the required structure, run `magic validate`, watch the score increase, and commit. Governance improves monotonically.

4.8. 4.11 Dimension Validation Quick Reference

Use this quick reference when debugging specific dimension failures:

```
# Validate a single dimension and get remediation instructions
magic validate --dimension D --remediate
magic validate --dimension E --remediate
magic validate --dimension L --remediate
```

Each remediation command outputs the exact file to create, the exact structure required, and the expected score increase upon completion. Run the dimension-specific validation before running the full `magic validate` to isolate failures efficiently. For a complete walkthrough of dimension-by-dimension remediation, see Chapter 45 (Validation Errors and Healing). For the governance policy perspective on these eight questions, see The Canonic Canon, Chapter 4.

Chapter 5

Chapter 5: Your First 255

Zero to FULL in one session. This chapter walks you 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, not by committee sign-off. The theoretical insight behind this walkthrough — that governance is compilation, and 255 is the compiled target — is developed fully in Chapters 36-37. For the governor's perspective on the same journey, see The Canonic Canon, Chapter 2 [11].

The governance-first save

Before you start, consider what happens when governance comes first. A developer sits down to build a clinical recommendation engine. Instead of writing code, she opens CANON.md and writes the axiom: **"RecEngine serves oncology with governed NCCN INTEL. Every recommendation cited."** Then she derives the constraints — and one of them reads: `MUST: Cite evidence source for every recommendation`. She starts sketching the architecture and realizes: the third-party evidence API she planned to use does not return citation metadata in its response payload. There is no `source` field, no DOI, no guideline reference. The API returns recommendation text only.

Without the governance file, she would have discovered this three weeks into implementation — after building the integration, writing the tests, and deploying to staging. With the governance file, she discovers it before writing a single line of code. The axiom's constraint — "every recommendation cited" — exposed an architectural gap in the upstream API. She switches to an evidence source that includes citation metadata, and the architecture is sound from day one.

That is the governance-first save. The CANON.md caught a structural problem that no amount of testing would have surfaced until production. Now let's build one together.

5.1. 5.1 Create the Scope

You are going to build a governed scope for a clinical TALK agent — a MedChat deployment at a community hospital. The scope inherits from the TALK service governance and adds site-specific clinical constraints.

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

5.2. 5.2 Write the TRIAD

You need three foundational files — the TRIAD. Each one follows the governed Markdown structure: header, `inherits:`, separator, content, separator, footer. Start with the most important one.

CANON.md:

```
# MEDCHAT-COMMUNITY — CANON


## Axiom

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


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

5.3. 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 “‘

• • •
```

```
Notice how each constraint traces back to a phrase in the axiom, as described in [Chapter 1](\#chapter-1-t
or the axiom is incomplete.

**VOCAB.md:**
```markdown
# VOCAB

| 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 |


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip


Every SCREAMING\_CASE term your scope uses must be defined here — no stubs, no circular definitions. If yo

**README.md:**
```markdown
# MEDCHAT-COMMUNITY — Clinical Decision Support


MedChat deployment for Community Hospital clinical staff. Provides governed clinical decision support acro


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip


## 5.3 First Validation — COMMUNITY Tier

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

Your score: ~35. COIN minted: ~35. Tier: COMMUNITY [11].

You have a governed scope. It has declared its purpose, defined its terms, and established its structure — three questions answered. This is minimum viable governance: the scope is on the map. It is not ready for
```

clinical deployment yet, but it exists, and its existence is governed.

5.4. 5.4 Add Community and Practice — ENTERPRISE Tier

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

MEDCHAT-COMMUNITY.md (the scope spec):

```
# MEDCHAT-COMMUNITY


## 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 |


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```


COVERAGE.md (practice coverage):
```markdown
# COVERAGE

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

Score: pending validation
```
```

ROADMAP.md (transparency):

```
# MEDCHAT-COMMUNITY — ROADMAP


## 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.


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

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

Your score climbs. The scope now has reproducibility, operations coverage, and transparency — Tier: ENTERPRISE. This is the minimum for clinical deployment. A compliance committee can review your COVERAGE assessment, the spec describes what the scope does, and the roadmap shows where it is going.

5.5. 5.5 Add LEARNING — AGENT Tier

Create LEARNING.md — this is where your scope captures what it learns from operation:

```
# LEARNING


Evidence lane for MEDCHAT-COMMUNITY.

## Patterns

| Date | Signal | Pattern | Source |
```
```

```
|------|--------|---------|--------|
| 2026-02-27 | GOV_FIRST | Governance files created before clinical deployment. | Step 0 |


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

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

Your score climbs to AGENT tier. The scope now learns from its own operation — governance events are captured as LEARNING patterns, which is where clinical quality improvement begins.

5.6. 5.6 Final Pass — 255

You are close. Close the vocabulary, fix any structural gaps, add the LANGUAGE inheritance, and make sure 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 [11].

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

5.7. Alignment with GETTING_STARTED.md

If you want the abbreviated version, GETTING_STARTED.md at the repository root provides the fast path. This chapter provides the deep explanation. Both converge on the same workflow:

1. Fork the repository
```

2. Create a scope directory with CANON.md
3. Add VOCAB.md, README.md — score ~35 (COMMUNITY)
4. Add {SCOPE}.md, COVERAGE.md, ROADMAP.md — score ~63 (ENTERPRISE)
5. Add LEARNING.md — score ~127 (AGENT)
6. Add LANGUAGE inheritance — score 255 (FULL)
7. Run `magic validate` — confirm 255/255

The pre-commit hook enforces 255 on every subsequent commit. The CI pipeline runs `magic validate --strict` — any score below 255 is a build failure. Governance is in the build pipeline, not in the review meeting.

5.8. 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. For the full economics of COIN and gradient minting, see Chapter 32 (COIN and the WALLET) and Chapter 33 (Gradient Minting) [12].

The total COIN minted for building this scope from 0 to 255 is exactly 255 COIN, and it is on the LEDGER. When the 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.

5.9. 5.8 Common Build Errors and How to Fix Them

During the climb from 0 to 255, developers encounter predictable errors. This section catalogs the errors by tier transition and provides exact fixes [6].

Errors at COMMUNITY tier (0 □ 35):

Error	Cause	Fix
`Score: 0 — no CANON.md`	Missing governance file	Create CANON.md with axiom and constraints
`Score: 1 — axiom present, no VOCAB.md`	CANON.md exists but VOCAB.md missing	Create VOCAB.md with term definitions
`Undefined term: "INTEL"`	SCREAMING_CASE term in CANON.md not in VOCAB.md	Add INTEL definition to VOCAB.md

Error	Cause	Fix
`inherits: unresolvable`	Wrong parent path	Verify path: `ls $(magic resolve-path {parent})`
`Footer missing`	No footer line in CANON.md	Add `*CANON | {SCOPE} | {DOMAIN}*`

```
# Diagnose COMMUNITY tier issues
magic validate --verbose --tier COMMUNITY
```

Errors at ENTERPRISE tier (35 □ 63):

Error	Cause	Fix
`ROADMAP.md not found`	Question 3 unanswered	Create ROADMAP.md with Done/Now/Next sections
`{SCOPE}.md not found`	Question 4 unanswered	Create {SCOPE}.md with Scope Intelligence table
`COVERAGE.md not found`	Question 5 unanswered	Create COVERAGE.md with eight-question assessment
`Claim mismatch: ROADMAP=PASS but ROADMAP.md missing`	COVERAGE.md lies	Update COVERAGE.md to reflect actual status
`ROADMAP.md empty`	File exists but has no content	Add at least one item in Done, Now, or Next

Errors at AGENT tier (63 □ 127):

Error	Cause	Fix
`LEARNING.md not found`	Question 7 unanswered	Create LEARNING.md with pattern table
`Pattern table missing`	LEARNING.md exists but no table	Add Date/Signal/Pattern/Source table
`Signal "EVOLVE" not in vocabulary`	Misspelled signal	Use EVOLUTION, not EVOLVE
`No patterns recorded`	Empty pattern table	Add at least one GOV_FIRST pattern

Errors at FULL tier (127 □ 255):

Error	Cause	Fix
`LANGUAGE not inherited`	Question 8 unanswered	Add LANGUAGE inheritance to inherits: chain
`No LANGUAGE terms detected`	LANGUAGE inherited but not used	Use controlled vocabulary terms from LANGUAGE

5.10. 5.9 The Build Session: Timing and Workflow

A developer building a scope from 0 to 255 for the first time should expect the following timeline:

Phase	Duration	Deliverable	Score
1. TRIAD	15-30 minutes	CANON.md + VOCAB.md + README.md	35
2. Spec + Roadmap + Coverage	20-40 minutes	{SCOPE}.md + ROADMAP.md + COVERAGE.md	63
3. LEARNING	10-15 minutes	LEARNING.md with initial patterns	127
4. LANGUAGE closure	10-20 minutes	LANGUAGE inheritance + vocabulary alignment	255
Total	**55-105 minutes**	**Full 255 governance**	**255**

The total time for a first-time developer to build a scope from 0 to 255 is under two hours. An experienced developer can do it in 30 minutes. The governance work is front-loaded — most of the intellectual effort is in writing the axiom and deriving the constraints (Phase 1). Everything after that is filling in the structural requirements.

Compare that to traditional clinical AI governance: months of committee meetings, policy drafts, review cycles, and sign-offs. CANONIC governance requires one developer, one terminal, and two hours. The governance lives in the files, the validator confirms it, and the LEDGER records it. No meetings required.

5.11. 5.10 Validating a Real Clinical Deployment

To see how the same pattern applies in a different clinical domain, imagine you are deploying a governed clinical decision support agent for emergency medicine triage. The scope is EMERGECHAT, and the walk-through follows the same structure as MEDCHAT-COMMUNITY with emergency medicine specifics.

```
mkdir -p EMERGECHAT
cd EMERGECHAT
```

CANON.md:

```
# EMERGECHAT — CANON

version: 2026-03


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

5.12. Axiom

EmergeChat serves emergency medicine triage with governed ESI INTEL. Every recommendation includes acuity level. Every interaction audited.

• • •

5.13. Constraints

MUST: Assign ESI acuity level (1-5) for every triage recommendation MUST: Cite emergency medicine guideline for every clinical recommendation MUST: Log every interaction to LEDGER with timestamp, actor, and acuity level MUST: Display “NOT A SUBSTITUTE FOR CLINICAL TRIAGE” disclaimer MUST: Version evidence base with ACEP publication date MUST NOT: Generate triage recommendations without ESI INTEL backing MUST NOT: Override triage nurse clinical judgment MUST NOT: Process PHI outside the emergency department perimeter MUST NOT: Delay clinical care — response time < 2 seconds for triage queries

• • •

Notice the emergency-medicine-specific constraint: `MUST NOT: Delay clinical care — response time < 2 seco
```

VOCAB.md:

```markdown
# VOCAB

| Term | Definition |
|------|-----------|
| EMERGECHAT | Governed clinical TALK agent serving emergency medicine triage with ESI evidence |
| ESI | Emergency Severity Index — 5-level triage algorithm (1=resuscitation, 5=non-urgent) |
| ACUITY | Patient urgency classification per ESI algorithm |
| ACEP | American College of Emergency Physicians — publisher of emergency medicine guidelines |
| 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 |
| TRIAGE | Initial patient assessment to determine acuity and resource allocation |
| RESUSCITATION | ESI Level 1 — immediate life-saving intervention required |
| EMTALA | Emergency Medical Treatment and Labor Act — federal mandate for emergency screening |


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip


Build the remaining files (README.md, EMERGECHAT.md, ROADMAP.md, COVERAGE.md, LEARNING.md) following the s

```bash
magic validate --verbose
```

The score climbs from 0 to 255 in the same predictable progression. The clinical domain changes — emergency medicine instead of general medicine — but the governance structure is identical: the same three TRIAD files, the same eight dimensional questions, the same 255 target as proof.

5.14. 5.11 After 255: Maintaining Compliance

Achieving 255 is not the end. Maintaining 255 is the ongoing obligation. Governance drift — changes that reduce the score — triggers DEBIT:DRIFT events on the LEDGER [12].

Common drift causes in clinical AI deployments:

Drift Cause	Detection	Fix
Deleted ROADMAP.md	`magic validate` reports question FAIL	Restore or recreate ROADMAP.md
New undefined term in CANON.md	`magic validate` reports undefined term	Add term to VOCAB.md
Expired evidence base	LEARNING captures DRIFT signal	Update evidence base, version INTEL
Broken inherits: link	`magic validate` reports chain break	Fix the inherits: path
Stale COVERAGE.md	Claim mismatch detected	Update COVERAGE.md to reflect current state

The maintenance workflow is straightforward: run `magic validate` on every commit, and if the score drops, the commit introduced drift — fix it before merging. The CI pipeline enforces this automatically, treating any commit that reduces the score as a failing build. Governance lives in the build pipeline, not in the review meeting.

```
# CI pipeline step: governance validation
magic validate --scope . --strict
# --strict: any score < 255 is a build failure
# Exit code 0 = 255/255. Exit code 1 = below 255.
```

In production, this means no code change, no configuration change, and no evidence base update can reach production without passing validation. Governance becomes continuous and automated — drift is detected at commit time, not at the annual compliance audit.

5.15. 5.12 The Score Trajectory

Track the score over time. The trajectory reveals the governance pattern. A healthy trajectory is monotonically non-decreasing: each commit either maintains or increases the score. A score drop is a drift event.

```
magic trajectory SERVICES/TALK/MAMMOCHAT --last 30d
# 2026-02-10: 0 → 35    ▓▓▓
```

The trajectory is stored on the LEDGER and renderable by the MONITORING dashboard. If you report to a compliance committee, the trajectory chart shows governance investment over time — no spreadsheet required.

5.16. 5.13 Cross-Scope Validation

After building a single scope to 255, validate the entire service tree to confirm inheritance consistency:

```
magic validate --recursive SERVICES/TALK/
# SERVICES/TALK: 255/255
#   MAMMOCHAT: 255/255
#   ONCOCHAT: 255/255
#   MEDCHAT-COMMUNITY: 255/255
# Fleet: 4/4 scopes at 255. No drift detected.
```

The recursive validation walks every child scope and verifies that inherited constraints are satisfied at every level.

5.17. 5.14 Post-255 Checklist

#	Check	Command	Expected
1	Score is 255	`magic validate`	`255/255 (FULL)`
2	No undefined terms	`magic validate --verbose`	`0 undefined`
3	Chain resolves	`magic validate --chain`	No broken links
4	COVERAGE claims match	`magic validate --coverage-audit`	`0 mismatches`
5	LEARNING has patterns	`magic validate --verbose`	`LEARNING: PASS`
6	Cross-references valid	`magic validate --references`	`0 broken`
7	Pre-commit hook installed	`magic init --check`	`Hook: installed`

Run the checklist once. Then trust the validator. The pre-commit hook enforces 255 on every subsequent commit.

5.18. 5.15 The Clinical Economics of 255

The journey from 0 to 255 is not merely a compliance exercise — it is an economic event with quantifiable returns. Consider a hospital deploying MammoChat at three sites. Without CANONIC governance, the compliance process involves: committee formation (2 weeks), policy drafting (4 weeks), legal review (3 weeks), IT security assessment (2 weeks), IRB review (6 weeks), and sign-off collection (2 weeks). Total: 19 weeks of elapsed time, 200+ person-hours, and approximately $85,000 in staff time across compliance, legal, clinical, and IT departments [13].

With CANONIC governance, the same deployment requires one clinical informatics engineer, one terminal session, and 105 minutes to reach 255. The governance lives in the files, the validator confirms compliance, the LEDGER records the work, and the CI pipeline enforces ongoing compliance. The compliance committee reviews COVERAGE.md and the validation trajectory — not a 47-page policy document that no one reads.

The economic comparison is stark. Traditional governance costs $85,000 and 19 weeks per deployment site. CANONIC governance costs 105 minutes of developer time per deployment site. For a three-site deployment, the savings are approximately $250,000 and 57 weeks of elapsed time. These savings compound: every subsequent deployment inherits the governance chain and starts at the parent's constraint floor, not at zero.

The 255 score proves the governance work has been done, the COIN minted is the economic record of that work, the LEDGER is the audit trail, and the inheritance chain is the compliance architecture. Build to 255 — the economics follow.

Chapter 6

Chapter 6: Building a Scope

A scope is a directory with a CANON.md file. That single requirement makes it the atom of CANONIC governance — the smallest unit you can validate, score, and mint. This chapter walks through anatomy, naming conventions, the full build procedure from empty directory to 255, and the practical decisions that determine whether your governance tree scales or collapses under its own weight. Where Chapter 5 walked through one scope end-to-end, this chapter covers the general principles. Chapter 7 builds on these principles to construct services, and Chapter 8 extends them to products.

6.1. 6.1 Scope Anatomy

A scope is a directory with a CANON.md file — the only hard requirement. Everything else accumulates [7].

```
MY-SCOPE/
  CANON.md          ← What do you believe? (required)
  VOCAB.md          ← Can you prove it?
  README.md         ← What shape are you? (with inherits: + axiom)
  {MY-SCOPE}.md     ← Who are you? (the spec)
  COVERAGE.md       ← How do you work?
  ROADMAP.md        ← Where are you going?
  LEARNING.md       ← What have you learned?
  SHOP.md           ← economic projection
  INTEL.md          ← scope intelligence
```

6.2. 6.2 Naming Convention

GOV (~/CANONIC/) uses SCREAMING_CASE .md files. RUNTIME (~/.canonic/) uses lowercase [6].

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 [7].

6.3. 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 [8].

6.4. 6.4 Scope vs Leaf

A scope has a CANON.md; a leaf does not. Scopes are governance containers, leaves are content [6].

```
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
```

6.5. 6.5 Full Scope Build: mkdir to 255

This is the complete procedure for building a scope from an empty directory to a fully validated 255-bit governance scope. Every command. Every file. No gaps [11].

Step 1: Create the directory structure.

```
# Create the scope directory
mkdir -p NEPHROCHAT
cd NEPHROCHAT

# Verify parent scope exists (the scope you will inherit from)
ls $(magic resolve-path hadleylab-canonic/MAGIC/SERVICES/TALK)/CANON.md
# If this fails, the parent scope does not exist. Create it first.
```

Step 2: Write CANON.md.

```
cat > CANON.md << 'EOF'
# NEPHROCHAT — CANON

version: 2026-03


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

6.6. Axiom

NephroChat serves nephrology with governed KDIGO INTEL. Every recommendation graded. Every interaction audited.

• • •

6.7. Constraints

MUST: Cite KDIGO guideline for every nephrology recommendation MUST: Include CKD stage (G1-G5) and albuminuria category (A1-A3) when relevant MUST: Log every interaction to LEDGER with timestamp and actor MUST: Version evidence base with KDIGO publication date MUST: Maintain PHI boundary — no patient identifiers in governance metadata MUST NOT: Generate uncited nephrology recommendations MUST NOT: Override nephrologist clinical judgment MUST NOT: Process PHI outside the local deployment perimeter MUST NOT: Provide dialysis access recommendations (refer to nephrologist)

• • •
```

EOF

Step 3: Write VOCAB.md.

```bash
cat > VOCAB.md << 'EOF'
# VOCAB

| Term | Definition |
|---|---|
| NEPHROCHAT | Governed clinical TALK agent serving nephrology queries with KDIGO evidence |
| KDIGO | Kidney Disease: Improving Global Outcomes — international nephrology guideline organization |
| CKD | Chronic Kidney Disease — progressive loss of kidney function classified by GFR and albuminuria |
| GFR | Glomerular Filtration Rate — primary measure of kidney function (mL/min/1.73m2) |
| 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 |
| AKI | Acute Kidney Injury — sudden decline in kidney function per KDIGO AKI criteria |
| RRT | Renal Replacement Therapy — dialysis or transplantation for end-stage kidney disease |
| ALBUMINURIA | Abnormal albumin excretion in urine — marker of kidney damage (A1/A2/A3 staging) |


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

EOF

Step 4: Write README.md.

```bash
cat > README.md << 'EOF'
# NEPHROCHAT — Nephrology Clinical Decision Support


NephroChat deployment for nephrology clinical decision support. Provides governed clinical recommendations

## Capabilities

- CKD staging and progression assessment (GFR + albuminuria)
- AKI recognition and initial management guidance
- Medication dosing adjustment for renal impairment
- Electrolyte management in CKD

```
- Referral criteria for nephrology consultation

Limitations

- Does not replace nephrologist clinical judgment
- Does not provide dialysis access recommendations
- Does not process PHI outside deployment perimeter


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip

EOF


**Step 5: Validate TRIAD — achieve COMMUNITY.**

```bash
git init
git add CANON.md VOCAB.md README.md
git commit -m "GOV: bootstrap NEPHROCHAT — TRIAD"
magic validate
# Expected: 35/255 (COMMUNITY)
```

Step 6: Write {SCOPE}.md — the spec.

```
cat > NEPHROCHAT.md << 'EOF'
# NEPHROCHAT


## Scope Intelligence

| Field | Value |
|-------|-------|
| Subject | Nephrology clinical decision support |
| Audience | Nephrologists, internists, hospitalists, primary care physicians |
| Evidence | KDIGO Clinical Practice Guidelines, AKI/CKD/GN |
| Status | Initial deployment |
| Contact | Clinical informatics team — nephrochat@hadleylab.org |


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip

EOF
```
```

Step 7: Write ROADMAP.md.

```bash
cat > ROADMAP.md << 'EOF'
# NEPHROCHAT — ROADMAP


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

## Now
- Build to ENTERPRISE tier. Add COVERAGE, SPEC, ROADMAP.
- Define initial KDIGO evidence base (CKD, AKI guidelines).

## Next
- Add LEARNING.md. Achieve AGENT tier.
- Deploy to nephrology pilot at Community Hospital.
- Integrate with EHR lab result feeds for CKD staging automation.


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

EOF

Step 8: Write COVERAGE.md.

```bash
cat > COVERAGE.md << 'EOF'
# COVERAGE

| # | Question | Answer | Status |
|---|----------|--------|--------|
| 1 | What do you believe? | NephroChat serves nephrology with governed KDIGO INTEL | PASS |
| 2 | Can you prove it? | VOCAB.md: 10 terms defined, zero stubs | PASS |
| 3 | Where are you going? | ROADMAP.md: ENTERPRISE build, then pilot | PASS |
| 4 | Who are you? | NEPHROCHAT.md: nephrology CDS scope | PASS |
| 5 | How do you work? | This file — operational coverage assessment | PASS |
| 6 | What shape are you? | inherits: TALK service, axiom present, 9 constraints | PASS |
| 7 | What have you learned? | LEARNING.md pending | FAIL |
| 8 | How do you express? | LANGUAGE pending | FAIL |

Score: pending validation
```
```

```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

EOF

Step 9: Validate — achieve ENTERPRISE.

```bash
git add NEPHROCHAT.md ROADMAP.md COVERAGE.md
git commit -m "GOV: NEPHROCHAT — COVERAGE + SPEC + ROADMAP"
magic validate
# Expected: 63/255 (ENTERPRISE)
```

Step 10: Write LEARNING.md — achieve AGENT.

```
cat > LEARNING.md << 'EOF'
# LEARNING

Evidence lane for NEPHROCHAT.

## Patterns

| Date | Signal | Pattern | Source |
|------|--------|---------|--------|
| 2026-03-10 | GOV_FIRST | Governance files created before clinical deployment. | Step 0 |
| 2026-03-10 | NEW_SCOPE | NephroChat scope bootstrapped with KDIGO evidence base. | CANON.md |

```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

EOF
```

git add LEARNING.md git commit -m “GOV: NEPHROCHAT — LEARNING” magic validate # Expected: 127/255 (AGENT)

Step 11: Close to 255 — add LANGUAGE, fix gaps.

```bash
```

```
# Update COVERAGE.md to reflect all PASS questions
# Ensure LANGUAGE inheritance is in the inherits: chain
# Verify all terms resolve
# Verify all cross-references are valid

git add -A
git commit -m "GOV: NEPHROCHAT — close to 255"
magic validate
# Expected: 255/255 (FULL)
```

Eleven steps. Under two hours for a first-time developer. Every file has a specific purpose, every step increases the score, and any developer following this procedure on any clinical domain produces the same governance structure.

6.8. magic heal: Scaffolding a Scope

`magic heal` automates the scaffolding process. Given a scope directory with a CANON.md, `magic heal` identifies which governance questions remain unanswered and proposes the files needed to answer them:

```
$ magic heal SERVICES/TALK/NEW-SERVICE
MISSING: COVERAGE.md (How do you work?)
MISSING: ROADMAP.md (Where are you going?)
MISSING: LEARNING.md (What have you learned?)
ACTION: Create COVERAGE.md, ROADMAP.md, LEARNING.md
```

The heal output maps directly to questions. The developer creates the files. The developer runs `magic validate`. The score climbs. `magic heal` does not auto-generate governance content — governance is human-authored. It identifies the gaps. The human fills them.

6.9. Bloat Constraints

Scope files have size constraints enforced by the build pipeline:

- **CANON.md** — axiom + constraints. No prose beyond governance declarations.
- **README.md** — minimal. inherits: + axiom + one-line purpose. No feature lists, no changelogs.
- **COVERAGE.md** — generated. Developers answer eight questions; the compiler produces the file.

These constraints prevent governance bloat — the tendency for governance files to accumulate non-governance content over time. If a file grows beyond its governance purpose, the build pipeline flags it. Governance files govern; everything else belongs in documentation or INTEL.

6.10. 6.6 Scope Organization Best Practices

Keep scopes shallow. Three to four levels of nesting is the practical maximum. Deeper nesting creates long inheritance chains that are harder to debug.

```
# GOOD: 3 levels deep
SERVICES/TALK/MAMMOCHAT/MAMMOCHAT-UCF/

# BAD: 6 levels deep
SERVICES/TALK/MAMMOCHAT/DEPLOYMENTS/FLORIDA/UCF/RADIOLOGY/
```

One scope per clinical domain. Do not combine multiple clinical domains in a single scope. MammoChat is breast imaging. OncoChat is oncology. They are separate scopes, even if they share infrastructure.

Name scopes for their clinical function. MAMMOCHAT, not BREAST-AI. ONCOCHAT, not CANCER-TOOL. NEPHROCHAT, not KIDNEY-SYSTEM. The name should tell a clinical informatics engineer exactly what the scope does.

Put deployment scopes under product scopes. MAMMOCHAT-UCF lives under MAMMOCHAT, not under UCF. The inheritance chain is product → deployment, not institution → product.

```
# GOOD: product → deployment
SERVICES/TALK/MAMMOCHAT/MAMMOCHAT-UCF/

# BAD: institution → product
UCF/MAMMOCHAT/
```

6.11. 6.7 Scope Deletion and Archival

Never delete a scope — archive it. Deleting breaks the inheritance chain for every child that inherits from it [2].

```
# Archive a scope (do NOT delete)
git mv DEPRECATED-SCOPE DEPRECATED-SCOPE.archived
# Add EXTINCTION signal to LEARNING.md
echo "| $(date +%Y-%m-%d) | EXTINCTION | DEPRECATED-SCOPE archived. | Governance decision |" >> LEARNING.m
git add .
git commit -m "GOV: archive DEPRECATED-SCOPE — EXTINCTION"
```

The EXTINCTION signal in LEARNING.md records the archival event. Any scope that inherited from the archived scope will fail validation on the next `magic validate` run — the `inherits:` chain is broken. Update those scopes to point to a living ancestor. The validator identifies the break; the fix is mechanical.

6.12. 6.8 Scope Discovery

Use `magic scan` to discover all scopes in a repository or across the fleet [6].

```
# List all scopes in current repository
magic scan --scopes
# Output:
# hadleylab-canonic (255/255)
#   MAGIC (255/255)
#     SERVICES (255/255)
#       TALK (255/255)
#         MAMMOCHAT (255/255)
#           MAMMOCHAT-UCF (255/255)
#       LEARNING (255/255)
#       SHOP (255/255)

# List all scopes across the fleet
magic scan --scopes --fleet
# Output includes canonic-canonic, hadleylab-canonic, adventhealth-canonic, etc.

# Find scopes below a specific tier
magic scan --scopes --below ENTERPRISE
# Output: scopes scoring below 63
```

For a team managing 50 clinical AI scopes across a health network, `magic scan --scopes --fleet` provides the complete governance inventory — every scope, its score, its tier, and its position in the inheritance tree. The governance posture of the entire network is visible in one command.

6.13. 6.9 Scope Migration

When a scope moves from one parent to another, follow this procedure:

```
mkdir -p NEW-PARENT/MIGRATED-SCOPE
cp MIGRATED-SCOPE/*.md NEW-PARENT/MIGRATED-SCOPE/
# Update inherits: in the new CANON.md
magic validate NEW-PARENT/MIGRATED-SCOPE
git mv MIGRATED-SCOPE MIGRATED-SCOPE.archived
git commit -m "GOV: migrate MIGRATED-SCOPE to NEW-PARENT"
```

The old location is archived (not deleted) to maintain LEDGER reference integrity.

6.14. 6.10 Scope Templates

For organizations building many similar scopes, create templates:

```
TEMPLATES/TALK-DEPLOYMENT/
  CANON.md.template
  VOCAB.md.template
  COVERAGE.md.template
```

Instantiate with:

```
magic-heal --template TEMPLATES/TALK-DEPLOYMENT \
  --scope SERVICES/TALK/MAMMOCHAT-REGIONAL \
  --vars "SCOPE_NAME=MAMMOCHAT-REGIONAL,HOSPITAL=Regional Hospital"
```

Templates accelerate multi-site deployments. A health network deploying MammoChat to 10 hospitals creates 10 scopes from one template in minutes.

6.15. 6.11 Scope Metrics

```
magic report SERVICES/TALK/MAMMOCHAT
# SCOPE: SERVICES/TALK/MAMMOCHAT
# Score: 255/255 (FULL) | Tier: MAGIC
# Chain: canonic-canonic → hadleylab-canonic → TALK → MAMMOCHAT
# COIN: 255 minted, 0 debited | Age: 127 days | Contributors: 3
# Child Scopes: MAMMOCHAT-UCF(255) MAMMOCHAT-ADVENT(255) MAMMOCHAT-REGIONAL(255)
```

6.16. 6.12 Scope Lifecycle States

State	Score	LEDGER Signal
BOOTSTRAP	0-34	NEW_SCOPE
BUILDING	35-254	MINT:WORK
GOVERNED	255	Score stable
DRIFTING	< previous	DEBIT:DRIFT
RECOVERING	rising after drift	MINT:WORK (recovery)
ARCHIVED	N/A	EXTINCTION

```
magic lifecycle SERVICES/TALK/MAMMOCHAT
# State: GOVERNED | Duration: 87 days at 255 | Stability: HIGH
```

6.17. 6.13 Clinical Vignette: Scope Design for a Multi-Specialty Hospital

Tampa General Hospital deploys governed clinical AI across five departments: radiology, cardiology, pathology, emergency medicine, and pharmacy. The clinical informatics architect must design the scope tree before any governance files are written. The scope design determines the inheritance chain, the constraint propagation, and the COIN distribution.

The architect considers two designs:

Design A: Department-first. Each department is a top-level scope. Services live under departments.

```
tampa-general-canonic/
  RADIOLOGY/
    MAMMOCHAT/
    AI-TRIAGE/
  CARDIOLOGY/
    CARDICHAT/
    ECG-AI/
  PATHOLOGY/
    PATHCHAT/
  EMERGENCY/
    EMERGECHAT/
  PHARMACY/
    DRUGCHAT/
```

Design B: Service-first. Services are organized by type (TALK, INTEL, MONITORING). Departments are deployment scopes under services.

```
tampa-general-canonic/
  SERVICES/
    TALK/
      MAMMOCHAT/
      CARDICHAT/
      PATHCHAT/
      EMERGECHAT/
      DRUGCHAT/
    INTEL/
      AI-TRIAGE/
```

```
    ECG-AI/
  MONITORING/
    RADIOLOGY-METRICS/
    HOSPITAL-DASHBOARD/
```

Design B wins. All TALK agents share conversation constraints (systemPrompt, disclaimer, evidence citation, PHI boundary), so declaring them once at the TALK service level and inheriting across all five agents eliminates duplication. Design A copies those constraints across five department scopes — a DRY violation that creates drift risk.

The numbers make it concrete: TALK carries 7 MUST and 3 MUST NOT rules. Design A copies all 10 to 5 departments: 50 constraint declarations. Design B declares them once: 10. When the FDA requires updated disclaimer language, Design B needs 1 edit. Design A needs 5.

Build the tree:

```
magic scan --scopes tampa-general-canonic
# tampa-general-canonic (255/255)
#   SERVICES (255/255)
#     TALK (255/255)
#       MAMMOCHAT (255/255) — radiology
#       CARDICHAT (255/255) — cardiology
#       PATHCHAT (255/255) — pathology
#       EMERGECHAT (255/255) — emergency
#       DRUGCHAT (255/255) — pharmacy
#     INTEL (255/255)
#       AI-TRIAGE (255/255) — radiology
#       ECG-AI (255/255) — cardiology
#     MONITORING (255/255)
#       RADIOLOGY-METRICS (255/255)
#       HOSPITAL-DASHBOARD (255/255)
# Fleet: 13/13 scopes at 255
# COIN: 3,315 minted
```

Thirteen scopes, one governance tree, every department's clinical AI inheriting the correct service-level constraints. The scope design IS the governance architecture [7].

6.18. 6.14 Scope Sizing Guidelines

Each scope should represent one unit of clinical or operational responsibility:

Scope Size	Example	Governance Unit	COIN
Too small	One API endpoint	Not meaningful governance	255 (wasted)
Right	One clinical TALK agent	One product, one team, one compliance boundary	255
Right	One service (LEDGER)	One infrastructure component	255
Right	One deployment (MAMMOCHAT-UCF)	One site, one IRB, one compliance context	255
Too large	Entire hospital	Too many concerns in one scope	255 (insufficient)

The sizing test: can one team own the scope's governance? If maintaining 255 requires coordination across multiple teams, the scope is too large — split it. If maintaining 255 requires no meaningful work, the scope is too small — merge it upward.

The natural scope boundary for clinical AI is the product-site combination. MammoChat at UCF is one scope; MammoChat at AdventHealth is another. Each has its own IRB, its own site-specific constraints, its own clinical team [7].

6.19. 6.15 Governance Proof: The Scope as Governance Atom

The scope is the atom of CANONIC governance — the smallest unit that can be validated, scored, and minted. The proof:

1. A scope has CANON.md (axiom + constraints).
2. `magic validate` produces a score for the scope.
3. Score delta produces COIN (MINT:WORK or DEBIT:DRIFT).
4. COIN is attributed to the scope and the identity.
5. The LEDGER records the event.

Every governance operation — validation, scoring, minting, auditing — operates on scopes. You cannot validate a file, mint COIN for a directory without CANON.md, or audit a namespace that is not a scope.

Think of the scope as CANONIC's equivalent of a function in programming: the smallest testable, composable, reusable unit. Scopes compose into services, services into products, products into organizations, organizations into the fleet. The hierarchy is the governance architecture, and the scope is where it starts [711].

6.20. 6.16 Scope Migration and Refactoring

Scopes can be moved, split, or merged. Every structural change is a governance event.

Moving a scope. Rename the directory, update the `inherits:` path in CANON.md, and run `magic validate`. The LEDGER records `SCOPE:MOVE` with old and new paths. COIN balances follow the scope — governance work is not lost when a scope changes its address.

```
# Move a scope
mv SERVICES/TALK/RADCHAT SERVICES/TALK/MAMMOCHAT
# Update inherits: in CANON.md
magic validate SERVICES/TALK/MAMMOCHAT
# LEDGER: SCOPE:MOVE from SERVICES/TALK/RADCHAT → SERVICES/TALK/MAMMOCHAT
```

Splitting a scope. Extract a child scope from a parent. The child inherits from the parent. The parent's COIN history remains with the parent. The child starts at score 0 and accumulates independently.

```
# Split screening logic into its own scope
mkdir SERVICES/TALK/MAMMOCHAT/SCREENING
# Create CANON.md with inherits: SERVICES/TALK/MAMMOCHAT
magic validate SERVICES/TALK/MAMMOCHAT/SCREENING
# Score: 1 (AXIOM only — new scope starts from CANON.md)
```

Merging scopes. Absorb a child scope upward into its parent. The child's COIN history is preserved in the LEDGER — the merge event references both scopes — and the parent's score may change if the merge introduces new constraints.

Operation	LEDGER Event	COIN Effect	Score Effect
Move	SCOPE:MOVE	No change	No change (same files)
Split	SCOPE:SPLIT	Parent unchanged; child starts at 0	Parent unchanged; child accumulates
Merge	SCOPE:MERGE	Both histories preserved	Parent recalculated

Refactoring is governance work. Every structural change produces a LEDGER event. The governance tree's shape is auditable at every commit. Run `git log --oneline -- SERVICES/` to see the structural evolution [714].

Chapter 7

Chapter 7: Building a Service

You have a governance tree. You have primitives — INTEL, CHAT, COIN — each a governed file with clear semantics. But files do not serve clinicians. Files do not handle HTTP requests or mint currency or answer questions at 2 a.m. Something has to project governance into the world where people actually work. That something is the service.

A service is a governed directory that composes one or more primitives into a runtime product. There are exactly 14 of them — not because 14 is a magic number, but because 14 covers every meaningful composition of three primitives. Each service is singular: one directory, one axiom, one set of constraints. The economy runs on their composition.

7.1. 7.1 The Service Constraint

Every service MUST compose the INTEL primitive. CHAT and COIN are optional [15]. The relationship between primitives and services is structural:

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

Primitives are files. Services are directories. A primitive expresses a single concept; a service orchestrates that concept into something operational [15].

7.2. 7.2 The 14 Services

Each service is detailed in its own chapter in Part III (Chapters 10-23):

#	Service	Primitives	Role	Chapter
1	LEARNING	INTEL	Governed discovery, IDF generalization	Ch 10
2	TALK	CHAT + INTEL	Contextual conversation agents	Ch 11
3	SHOP	COIN + INTEL	Public economic projection	Ch 12
4	LEDGER	COIN	Append-only economic truth	Ch 13
5	WALLET	COIN	Per-USER economic identity	Ch 14
6	VAULT	COIN + INTEL	Private economic aggregate	Ch 15
7	API	COIN	HTTP COIN operations	Ch 16
8	CHAIN	COIN	Cryptographic integrity, hash-linked events	Ch 17
9	MINT	COIN	Gradient minting, RUNNER tasks	Ch 18
10	IDENTITY	COIN	Ed25519 keys, KYC anchors	Ch 19
11	CONTRIBUTE	COIN + INTEL	External WORK, bronze/gold curation	Ch 20
12	NOTIFIER	CHAT + INTEL	Event notification, inbox delivery	Ch 21
13	MONITORING	INTEL	Runtime metrics, governance scoring	Ch 22

#	Service	Primitives	Role	Chapter
14	DEPLOY	COIN + INTEL	Governed artifact delivery, rollback	Ch 23

7.3. 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 with hard boundaries — no cross-scope leakage, no ambient state. Routes are driven from governed indices, never hardcoded [15].

7.4. 7.4 Instance vs Service

Service directories define schemas; instance directories hold content. The distinction matters because services are singletons (one WALLET service defines how wallets work) while instances are plural (each user has their own wallet). Instances live at USER scope, not nested inside SERVICES/ [2].

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

7.5. 7.5 Building a Service: Step-by-Step

Every service follows the same build procedure. This walkthrough builds a MONITORING service.

Step 1: Create the service directory.

```
mkdir -p SERVICES/MONITORING
cd SERVICES/MONITORING
```

Step 2: Write the service CANON.md.

```
cat > CANON.md << 'EOF'
# MONITORING — CANON


## Axiom

**MONITORING is continuous governance scoring. Real-time visibility. Observability without obstruction.**


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

7.6. Constraints

MUST: Expose Prometheus-compatible /metrics endpoint MUST: Include governance-specific metrics (scope scores, drift events) MUST: Health check verifies live state, not cached configuration MUST NOT: Block service operations on metrics collection failure MUST NOT: Require auth for /health and /metrics

• • •

EOF

Step 3: Write VOCAB.md with service-specific terms.

```bash
cat > VOCAB.md << 'EOF'
# VOCAB

| Term | Definition |
|------|-----------|
| MONITORING | Continuous governance scoring and runtime visibility service |
| PROMETHEUS | Open-source metrics collection format — text exposition at /metrics |
| HEALTH_CHECK | HTTP endpoint verifying service liveness and dependency state |
| DRIFT_EVENT | Governance score regression recorded on the LEDGER |
| GAUGE | Prometheus metric type — value that can increase or decrease |
| COUNTER | Prometheus metric type — monotonically increasing value |
| SCOPE_SCORE | Real-time 255-bit governance score for a governed scope |
```
```

```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

EOF

Step 4: Write remaining governance files. README.md, MONITORING.md, ROADMAP.md, COVERAGE.md, LEARNING.md — each follows the governed Markdown structure described in [Chapter 5](\#chapter-5-your-first-255).

Step 5: Validate progressively.

```bash
magic validate  # After TRIAD: 35/255
magic validate  # After COVERAGE+SPEC+ROADMAP: 63/255
magic validate  # After LEARNING + LANGUAGE: 255/255
```

Step 6: Commit at each tier.

```
git add CANON.md VOCAB.md README.md
git commit -m "GOV: bootstrap MONITORING — TRIAD"

git add MONITORING.md ROADMAP.md COVERAGE.md
git commit -m "GOV: MONITORING — COVERAGE + SPEC + ROADMAP"

git add LEARNING.md
git commit -m "GOV: MONITORING — close to 255"
```

At this point, you have minted 255 COIN — the full governance score. The service is now discoverable by `magic scan` and composable with other services in the fleet.

7.7. 7.6 Service Composition Patterns

Services compose in three patterns, each with different coupling characteristics:

Pattern 1: Independent. The service operates alone with no runtime dependency on other services. MONITORING, for example, observes the fleet but does not require any other service to function. You can deploy it first or last — the order does not matter.

Pattern 2: Producer-consumer. One service produces a primitive that another consumes. LEARNING produces INTEL; TALK consumes that INTEL to build its systemPrompt. The dependency is directional and explicit:

```
SERVICES/LEARNING/ → produces INTEL → consumed by → SERVICES/TALK/
```

Pattern 3: Economic chain. Services linked through COIN operations form a pipeline: MINT creates currency, WALLET holds it, SHOP spends it, LEDGER records it. Each link in the chain is a separate service with its own governance scope.

Pattern	Services	Coupling	Clinical Example
Independent	MONITORING, DEPLOY	None	Monitoring MammoChat without affecting operation
Producer-consumer	LEARNING TALK	INTEL flow	MammoChat serving BI-RADS INTEL from LEARNING
Economic chain	MINT WALLET SHOP	COIN flow	Clinician purchases OncoChat access

7.8. 7.7 Service Governance Constraints

Every service inherits SERVICES-level constraints:

```
MUST:     Compose at least one primitive (INTEL, CHAT, or COIN)
MUST:     Expose discoverable interface (SHOP.md or VAULT.md)
MUST:     Record state changes on LEDGER
MUST NOT: Hardcode routes — routes driven from governed indices
MUST NOT: Cross scope boundaries — no direct access to another service's state
MUST NOT: Store runtime artifacts in GOV_ROOT
```

These six constraints are universal — every service inherits them regardless of which primitives it composes. The service-specific CANON.md layers domain constraints on top, and the validator checks both inherited and local constraints in a single pass.

7.9. 7.8 Service HTTP Routes

Services with HTTP interfaces declare routes in HTTP.md:

```
# HTTP — MONITORING


## Routes

| Method | Path | Handler | Auth | Rate |
```

```
|--------|------|---------|------|------|
| GET | /api/v1/health | health_check | None | Unlimited |
| GET | /api/v1/metrics | prometheus_metrics | None | Unlimited |
| GET | /api/v1/scopes | scope_scores | Required | 100/min |

## Domains

| Domain | Target | SSL |
|--------|--------|-----|
| monitor.canonic.org | Cloudflare Worker | Auto |


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

HTTP.md is a governed route table — the single source of truth for a service's HTTP surface. The build pip

7.9 The 14 Services: Functional Grouping

Group	Services	Primitive	Purpose
Knowledge	LEARNING, MONITORING	INTEL	Discovery, observation, pattern capture
Communication	TALK, NOTIFIER	CHAT + INTEL	Conversation, notification
Economy	LEDGER, WALLET, VAULT, API, CHAIN, MINT, SHOP	COIN	Economic operations, identity, integrit
Operations	IDENTITY, CONTRIBUTE, DEPLOY	Mixed	Identity, contributions, deployment

These four groups are orthogonal — each operates independently, connected only through governed interfaces

7.10 Service Testing

Test the service by running validation in verbose mode:

```bash
magic validate --scope SERVICES/MONITORING --verbose
# [D] CANON.md present, axiom valid                    PASS
# [E] VOCAB.md present, 7 terms, 0 stubs              PASS
# [T] ROADMAP.md present, 3 sections                   PASS
# [R] MONITORING.md present, Scope Intelligence table   PASS
# [O] COVERAGE.md present, 8 dimensions, 0 mismatches  PASS
# [S] inherits: resolved, axiom present, 5 constraints  PASS
# [L] LEARNING.md present, 2 patterns                  PASS
# [LANG] LANGUAGE inherited                             PASS
# Score: 255/255 (FULL)
```
```

Each dimension is a binary test case — PASS or FAIL, deterministic across runs. Run it twice, get the same result. The verbose output tells you exactly which dimension failed and what to fix.

7.10. 7.11 Service Anti-Patterns

The mega-service. You put TALK + SHOP + MONITORING in one directory because they "all relate to clinical operations." Break them apart. Each service composes one or two primitives — that constraint is not arbitrary, it is what keeps services testable and independently deployable.

The phantom service. A SERVICES/ directory with CANON.md but no runtime projection — no SHOP.md, no VAULT.md, no HTTP.md. It validates to 255 but does nothing. Either add a projection that serves real users or reclassify the directory as a scope instead of a service.

The cross-scope reader. A service that reads another service's internal files directly. TALK should consume LEARNING's governed output (INTEL.md), not reach into LEARNING's internal processing files. The governed interface is the contract; internal files are implementation details that can change without notice.

```
# WRONG: direct internal access
intel = read("SERVICES/LEARNING/internal/raw.json")

# RIGHT: governed interface
intel = read("SERVICES/LEARNING/INTEL.md")
```

7.11. 7.12 Clinical Service Example: TALK

TALK is the service you will encounter most often if you are building clinical AI agents. It composes CHAT + INTEL to produce contextual conversation agents — each one backed by governed evidence, constrained by its axiom, and deployable as an independent product:

```
magic report SERVICES/TALK
# Score: 255/255 (FULL)
# Primitives: CHAT + INTEL
# Child scopes: MAMMOCHAT, ONCOCHAT, MEDCHAT, DERMCHAT, EMERGECHAT
# Total COIN: 1,530 (TALK + 5 products * 255 each)
```

Each child scope inherits TALK's conversation infrastructure — session management, disclaimer rendering, LEDGER integration — and adds domain-specific knowledge on top. The separation is clean and deliberate: infrastructure lives at the service level, clinical knowledge lives at the product level [15].

7.12. 7.13 Clinical Vignette: Building the CONTRIBUTE Service for Clinical Trials

MD Anderson Cancer Center builds a CONTRIBUTE service to govern external clinical trial data submissions. Researchers submit genomic data, treatment response metrics, and biomarker results from Phase II trials. Each submission must be governed — cited, attributed, and scored.

The CONTRIBUTE service composes COIN + INTEL:

```
cat > SERVICES/CONTRIBUTE/CANON.md << 'EOF'
# CONTRIBUTE — CANON


## Axiom

**CONTRIBUTE governs external work submissions. Every contribution cited. Every submission scored. Bronze


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

7.13. Constraints

MUST: Accept external INTEL with full citation chain MUST: Score contributions as BRONZE (structural) or GOLD (impactful) MUST: Record every submission as a LEDGER event MUST: Verify contributor identity via VITAE.md MUST NOT: Accept uncited submissions MUST NOT: Accept PHI in contribution metadata MUST NOT: Mint COIN for BRONZE contributions (structural only)

• • •

EOF

```
The BRONZE/GOLD distinction governs contribution quality at two levels. A BRONZE contribution meets struct
it has citations, a verified contributor identity, and parses correctly. A GOLD contribution clears a high
```

```bash
magic contribute --submit trial-NCT04711096-results.md \
  --contributor dr.williams@mdanderson.org \
```
```

```
  --scope SERVICES/TALK/ONCOCHAT
# Submission: trial-NCT04711096-results.md
# Citations: 3 (NCT04711096, NCCN 2026.1, FDA approval NDA 761310)
# Identity: VERIFIED (dr.williams@mdanderson.org via VITAE.md)
# PHI check: CLEAN (no patient identifiers detected)
# Grade: GOLD (improves OncoChat evidence layer)
# LEDGER: CONTRIBUTE:GOLD recorded
# Score impact: OncoChat evidence freshness improved
```

Note what CONTRIBUTE is not: it is not a file upload system. Every submission passes through four gates — citation verification, identity check, PHI detection, and impact scoring — before it enters the governance tree. By composing COIN (economic attribution) with INTEL (knowledge verification), CONTRIBUTE creates a governed knowledge contribution pipeline where quality is enforced structurally, not by reviewer discipline [15].

7.14. 7.14 Service Lifecycle and Deprecation

Services follow a governed lifecycle:

State	Description	Transition
PROPOSED	CANON.md drafted, not yet validated	-> ACTIVE (first 255)
ACTIVE	Validated and operational	-> DEPRECATED, -> FROZEN
DEPRECATED	Marked for sunset, 90-day notice	-> ARCHIVED
FROZEN	Operational but no new features	Indefinite
ARCHIVED	Removed from fleet, history preserved	Terminal

Deprecation requires a LEDGER event and a 90-day notice period:

```
magic service --deprecate SERVICES/LEGACY-FHIR \
  --reason "Replaced by SERVICES/FHIR-API (R5 native)" \
  --sunset 2026-06-10
# LEDGER: SERVICE:DEPRECATE LEGACY-FHIR (sunset: 2026-06-10)
# NOTIFIER: Alert sent to 12 dependent scopes
# ROADMAP.md: Updated with deprecation notice
```

Dependent scopes receive notification and have 90 days to migrate to the replacement. After sunset, the service is archived — removed from the live fleet but preserved in git history. The LEDGER retains every event from the service's lifetime. COIN already minted remains in WALLETS; the economic record survives the service itself [15].

7.15. 7.15 Service Metrics

Every service exposes standardized metrics:

```
magic service --metrics SERVICES/TALK
# TALK Service Metrics:
#   Score: 255/255
#   Child scopes: 7 (all at 255)
#   Total COIN: 2,040 (TALK + 7 children)
#   HTTP routes: 4 (2 write, 2 read)
#   Dependencies: LEARNING (INTEL source), LEDGER (event recording)
#   Dependents: MAMMOCHAT, ONCOCHAT, MEDCHAT, DERMCHAT, EMERGECHAT, GASTROCHAT, NEPHROCHAT
#   Uptime: 99.97% (last 30 days)
#   Latency p99: 142ms
#   Daily requests: 12,450
```

Notice how the metrics draw from two sources: governance data (score, COIN, routes from HTTP.md) and runtime data (uptime, latency from MONITORING). Both sources are themselves governed — there is no untracked telemetry, no shadow metrics pipeline [15].

7.16. 7.16 Governance Proof: Service Composition

Every service composes at least one primitive. The proof that service composition is complete:

1. Three primitives exist: INTEL, CHAT, COIN.
2. The 14 services cover all possible primitive compositions: INTEL alone (LEARNING, MONITORING), CHAT + INTEL (TALK, NOTIFIER), COIN alone (LEDGER, WALLET, API, CHAIN, MINT), COIN + INTEL (SHOP, VAULT, CONTRIBUTE, DEPLOY), all three (IDENTITY — implicitly through VITAE.md + governance + economic identity).
3. Every service inherits from SERVICES-level constraints (6 MUSTs/MUST NOTs).
4. Every service adds domain-specific constraints.
5. `magic validate` checks both inherited and local constraints.
6. A service at 255 satisfies all constraints from root to leaf.

Therefore: every governed service composes at least one primitive, satisfies all inherited constraints, and is validated by the same kernel. The composition is complete — no primitive composition is unrepresented in the 14-service catalog. Q.E.D. [153].

7.17. 7.17 Service Contract Completeness Checklist

Every service must satisfy a completeness checklist before reaching ACTIVE state. The checklist is enforced by `build` Stage 1 (`attest-services`):

#	Requirement	File	Dimension	Gate
1	Axiom declared	CANON.md	D (1)	REQUIRED
2	Service spec written	{service}.md	S (2)	REQUIRED
3	Vocabulary defined or inherited	VOCAB.md	LANG (4)	REQUIRED
4	Coverage matrix populated	COVERAGE.md	E (8)	REQUIRED
5	Operations documented	SPEC.md / service-specific	O (16)	REQUIRED
6	HTTP routes declared	HTTP.md	R (32)	REQUIRED for API services
7	Learning history initialized	LEARNING.md	L (64)	REQUIRED
8	Roadmap defined	ROADMAP.md	T (128)	REQUIRED
9	At least 1 primitive composed	CANON.md axiom	—	REQUIRED
10	Inheritance chain valid	CANON.md inherits:	—	REQUIRED

```
# Run service attestation checklist
magic service --attest SERVICES/TALK

# Service Attestation: SERVICES/TALK
#   [ ] 1. Axiom: "TALK is SERVICE. Composes: CHAT + INTEL"
#   [ ] 2. Service spec: TALK.md (2,400 words)
#   [ ] 3. Vocabulary: VOCAB.md (38 terms, inherits root +142)
#   [ ] 4. Coverage: COVERAGE.md (8 × 8 matrix, 100% filled)
#   [ ] 5. Operations: SPEC.md (conversation lifecycle)
#   [ ] 6. HTTP routes: HTTP.md (4 routes: 2 write, 2 read)
#   [ ] 7. Learning: LEARNING.md (3 epochs, 18 entries)
#   [ ] 8. Roadmap: ROADMAP.md (4 quarters defined)
#   [ ] 9. Primitives: CHAT + INTEL (2 of 3)
#   [ ] 10. Inheritance: hadleylab-canonic/SERVICES 
#
# Attestation: PASS (10/10)
# Score: 255/255
```

7.18. 7.18 Service Dependency Graph

Services depend on each other. The dependency graph is derived from `inherits:` chains and cross-scope references in CANON.md:

```
magic service --dependencies --graph

# Service Dependency Graph:
#
# TALK ──────→ LEARNING (INTEL source for TALK agents)
#   │          LEDGER (event recording for sessions)
#   │          IDENTITY (principal verification)
#   │
# LEARNING ──→ LEDGER (event recording for discoveries)
#   │          MONITORING (pattern detection)
#   │
# SHOP ──────→ WALLET (balance checking for purchases)
#   │          LEDGER (SPEND event recording)
#   │          IDENTITY (buyer/seller verification)
#   │
# MONITORING → NOTIFIER (alert delivery)
#   │          LEDGER (metric event recording)
#   │
# DEPLOY ────→ MONITORING (post-deploy verification)
#              NOTIFIER (deployment notifications)
#              LEDGER (deploy event recording)
#
# Cycle detection: NO CYCLES □
# Root services (no dependencies): LEDGER, CHAIN
# Leaf services (no dependents): DEPLOY
```

The dependency graph must be acyclic. A cycle (Service A depends on Service B depends on Service A) would create a chicken-and-egg problem: neither service could validate without the other already being validated. The `build` pipeline checks for cycles at Stage 0 and rejects them with `E105 CYCLE_DETECTED` [15].

7.19. 7.19 Service Templates for Rapid Bootstrap

New services bootstrap from templates that encode the service contract structure:

```
magic service --create NEW-SERVICE \
  --template clinical-talk \
  --primitives "CHAT + INTEL"
```

```
# Creating service: SERVICES/NEW-SERVICE
#
# Generated files:
#   CANON.md          ← axiom: "NEW-SERVICE is SERVICE. Composes: CHAT + INTEL"
#                       inherits: hadleylab-canonic/SERVICES
#   NEW-SERVICE.md    ← service spec template (fill in)
#   VOCAB.md          ← inherits parent vocabulary, add local terms
#   COVERAGE.md       ← 8 × 8 matrix template (fill in)
#   HTTP.md           ← route template (customize routes)
#   LEARNING.md       ← Epoch 1: Bootstrap template
#   ROADMAP.md        ← 4-quarter template (fill in)
#
# Initial score: 255/255 (all files present with template content)
# Warning: Template content must be customized before clinical use
# Next: Edit each file to add domain-specific content
```

The template produces a structurally valid 255-score scope immediately — every file present, every governance question answered. But the content is generic; "fill in" placeholders mark where domain-specific knowledge must go. The template accelerates creation from hours to minutes. The customization that follows — writing real axioms, real constraints, real evidence — is the governance work that earns COIN [1514].

7.20. 7.20 Clinical Vignette: Service Composition Prevents Governance Gap

Emory Healthcare (Atlanta, 11 hospitals) deploys a new clinical AI agent: NephroChat (nephrology consultation). The nephrology informatics team creates the scope:

```
magic service --create SERVICES/TALK/NEPHROCHAT \
  --template clinical-talk \
  --primitives "CHAT + INTEL"
```

The template generates all governance files. The team customizes: adds 23 nephrology-specific VOCAB terms (eGFR, CKD-EPI, UACR, KDIGO, AKI staging), writes the INTEL.md with 18 evidence citations (KDIGO guidelines, USRDS data, key nephrology trials), and defines the systemPrompt with CKD staging tables and medication dosing guidance for renal impairment.

During the governance review, `magic validate` catches a composition gap:

```
magic validate SERVICES/TALK/NEPHROCHAT
# Score: 247/255
```

```
# Missing: E (8) — COVERAGE.md exists but INTEL cross-reference incomplete
#   COVERAGE.md row "drug-dosing" references INTEL layer 3
#   INTEL.md layer 3 is empty (no drug interaction references)
#   Fix: Add renal dosing references to INTEL.md layer 3
#     Suggested: Lexicomp Renal Dosing, Aronoff Drug Prescribing in Renal Failure
```

The coverage matrix claimed drug dosing coverage, but the INTEL.md lacked the supporting evidence references. A human reviewer might miss this gap — the COVERAGE.md looks complete without cross-referencing the INTEL.md. The validator checks the cross-reference automatically. The team adds the renal dosing references, re-validates to 255, and deploys NephroChat with complete evidence coverage.

Without the composition check, NephroChat would have deployed claiming drug dosing coverage backed by zero evidence references. A nephrologist asking "What is the renal dosing for vancomycin?" would receive a response that the governance documentation says is evidence-based — but the evidence does not exist. The composition check closes this documentation-to-evidence gap before the agent ever speaks to a clinician [15316].

7.21. 7.21 The Service as Governance Primitive

Think of a scope as governed knowledge and a service as governed behavior. Scopes can exist without runtime expression — a CANON.md with no code is perfectly valid governance. Services, by contrast, always project into the world: routes, views, APIs, chat interfaces. Every service you build extends your organization's governed capability surface.

The 14 canonical services represent the complete set of behaviors that governance can express through the three primitives. If your use case is not covered, you are either composing multiple services (correct) or operating outside governance (a problem to fix, not a feature to ship). Build the service. Compose the primitives. Validate at 255. The service is where governance becomes operational [1514].

Chapter 8

Chapter 8: Building a Product

Governance work produces COIN. COIN establishes a cost basis. The cost basis sets a price floor. A product is what happens when a governed scope at 255 publishes a SHOP.md with a Card — the atomic listing that makes governance labor tradeable. This chapter covers the mechanics: SHOP.md structure, pricing by tier, cost basis calculation, checkout flow, and the full build procedure from governed scope to listed product. Chapter 6 covered building scopes; Chapter 7 covered building services; this chapter covers turning governed work into tradeable products. For the SHOP service architecture, see Chapter 12. For the expanded SHOP economics, see Chapter 34. The live marketplace is at shop.hadleylab.org.

8.1. 8.1 SHOP.md

Every product is a governed scope compiled to 255, listed in SHOP.md with a Card [17].

```
## 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. 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. 8.3 COST_BASIS

The cost basis of a product is the total COIN minted by governance work producing that product [17]:

```
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 [12].

8.4. 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 [17].

8.5. 8.5 Building a Product: Step-by-Step

This walkthrough builds a FHIR Integration Playbook product.

Step 1: Build the scope to 255. The product IS the scope. No scope below 255 can list a product.

```
mkdir -p SERVICES/FHIR-API
cd SERVICES/FHIR-API
# Write TRIAD (CANON.md, VOCAB.md, README.md) → 35
# Write COVERAGE + SPEC + ROADMAP → 63
# Write LEARNING.md, close to 255
magic validate
# Score: 255/255 (FULL)
```

Step 2: Create SHOP.md.

```
cat > SHOP.md << 'EOF'
# SHOP — FHIR Integration Playbook


## Card

| Field | Value |
|-------|-------|
| title | FHIR Integration Playbook |
| type | SERVICE |
| price | 255 COIN |
| status | AVAILABLE |
| synopsis | Complete governance kit for FHIR R4 integration. |
| route | /SERVICES/FHIR-API/ |

## What You Get

- CANON.md template for FHIR scopes
- VOCAB.md with 40+ FHIR governance terms
- COVERAGE.md checklist (12 resource types)
- LEARNING.md with 6 months of patterns


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

EOF

Step 3: Validate and commit.

```bash
magic validate  # 255/255, SHOP.md valid
git add SHOP.md
git commit -m "GOV: FHIR-API — SHOP listing"
```

Step 4: Verify listing.

```
magic scan --shop
# 1. FHIR Integration Playbook (255 COIN) — SERVICES/FHIR-API
```
```

8.6. 8.6 Product Types

Type	Structure	Typical Cost Basis	Example
BOOK	1 parent + N chapter scopes	N * 255	CANONIC-DOCTRINE (48 chapters)
PAPER	1 scope	255	governance-as-compilation.md
SERVICE	1 service + M sub-scopes	(1+M) * 255	FHIR Playbook
content	1 scope	255	Clinical AI Quick Start

8.7. 8.7 Product Validation Rules

Rule	Check	Error
Score gate	`score == 255`	`SHOP_SCORE_GATE: score < 255`
Cost basis floor	`price >= cost_basis`	`SHOP_PRICE_FLOOR: price < cost_basis`
Card completeness	All Card fields present	`SHOP_CARD_INCOMPLETE`
Identity gate	VITAE.md exists	`SHOP_IDENTITY_GATE`
Scope existence	Route resolves	`SHOP_ROUTE_INVALID`

```
magic validate --shop SERVICES/FHIR-API
# Score gate: PASS | Cost basis floor: PASS | Card: PASS | Identity: PASS | Route: PASS
```

8.8. 8.8 Multi-Scope Products

A product can span multiple scopes — a book with 20 chapters is 20 scopes, a governance suite with 50 templates is 50 scopes. Every constituent scope must be at 255. If any single scope drops below 255, the product delists automatically. One drifting scope taints the entire product [12].

8.9. 8.9 Product Economics

For a product priced at 255 COIN that sells 100 copies:

Metric	Value
Cost basis	255 COIN (governance labor)
Revenue (100 sales)	25,500 COIN
Net to author	25,500 COIN
LEDGER events	100 SPEND + 1 listing
Author ROI	100x cost basis

8.10. 8.10 Product Discovery

The build pipeline compiles all SHOP.md files into a browsable catalog:

```
build → scan **/SHOP.md → compile SHOP.json (_generated) → fleet SHOP page
```

The fleet SHOP page is a static site. Products are cards rendered from SHOP.json. Each card shows title, author, price, score (always 255), synopsis, and attestation count.

8.11. 8.11 Clinical Product Examples

Product	Domain	Cost Basis	Price	Use Case
MammoChat Governance Kit	Radiology	1,530 COIN	2,000 COIN	Breast imaging AI governance
NCCN OncoChat Template	Oncology	765 COIN	1,000 COIN	OncoChat with NCCN mapping
HIPAA Compliance Suite	Compliance	12,750 COIN	15,000 COIN	Enterprise HIPAA governance
EHR Migration Playbook	Operations	7,650 COIN	10,000 COIN	30-scope EHR migration
Clinical AI Quick Start	Education	255 COIN	255 COIN	First scope tutorial

Every product started as governance work that minted COIN, establishing the cost basis that set the price floor. The entire economic chain is on the LEDGER.

8.12. 8.12 Product Lifecycle

State	Condition	LEDGER Signal
DRAFT	Scope < 255, no SHOP.md	MINT:WORK (building)
LISTED	Scope = 255, SHOP.md present	SHOP:LIST
ACTIVE	Listed + purchases	SPEND events
DELISTED	Scope drifted below 255	SHOP:DELIST + DEBIT:DRIFT
RELISTED	Scope restored to 255	SHOP:RELIST

A product delists automatically when its scope drops below 255 and relists when restored. No manual intervention — governance drives commerce.

8.13. 8.13 Clinical Vignette: Product Launch at a Regional Health System

Ochsner Health (New Orleans, 40 hospitals) decides to productize their internal FHIR Integration Playbook. Dr. Nguyen, Chief Health Information Officer, has governed the playbook across 6 scopes over 14 months.

The Governance Foundation. Each scope represents a distinct FHIR integration domain: Patient Demographics (Patient resource), Clinical Notes (DocumentReference), Lab Results (Observation/DiagnosticReport), Medications (MedicationRequest), Allergies (AllergyIntolerance), and Immunizations (Immunization). Each carries full governance: CANON.md with axiom and constraints, VOCAB.md with FHIR-specific terminology (SMART_ON_FHIR, CAPABILITY_STATEMENT, SEARCH_PARAMETER, BUNDLE_TRANSACTION), INTEL.md referencing HL7 FHIR R4 v4.0.1, COVERAGE.md mapping each resource to Ochsner's Epic endpoints, LEARNING.md documenting 14 months of integration patterns, and ROADMAP.md targeting FHIR R5 readiness.

Cost Basis Calculation. The LEDGER records every MINT:WORK event across 14 months of governance labor:

```
Scope                  Commits   COIN Minted   Time
Patient Demographics   87        255           3 months
Clinical Notes         94        255           4 months
Lab Results            103       255           3 months
Medications            78        255           2 months
Allergies              45        255           1 month
Immunizations          52        255           1 month
──────────────────────────────────────────────────────
Total                  459       1,530         14 months
```

The cost basis is 1,530 COIN — the sum of all MINT:WORK events. Dr. Nguyen sets the price at 2,000 COIN (31% margin above cost basis). The price floor constraint ensures `price >= cost_basis`, preventing below-cost sales that would undervalue governance labor [17].

Product Launch. Dr. Nguyen creates SHOP.md in the parent scope:

```
magic validate --shop SERVICES/FHIR-PLAYBOOK
# Score gate:       255/255 PASS (all 6 sub-scopes at 255)
# Cost basis floor: 2,000 >= 1,530 PASS
# Card complete:    6/6 fields PASS
# Identity gate:    VITAE.md verified PASS
# Route valid:      /SERVICES/FHIR-PLAYBOOK/ PASS
# LEDGER event:     SHOP:LIST recorded
```

Market Reception. Within 90 days:

Metric	Value
Attestations (purchases)	34
Unique buyers	31
Revenue	68,000 COIN
Buyer institutions	12 hospitals, 3 HIEs, 16 clinics
SHOP:LIST to first purchase	6 days
Repeat purchases	3 (bulk institutional)

The Drift Incident. At day 47, the HL7 consortium publishes FHIR R4B (v4.3.0). Ochsner's INTEL.md references the superseded R4 v4.0.1 specification. MONITORING detects the stale evidence during its 5-minute poll cycle. The INTEL dimension drops from 8 to 0. The scope score falls from 255 to 247. The SHOP automatically delists the product:

```
LEDGER: DEBIT:DRIFT -8 COIN (SERVICES/FHIR-PLAYBOOK/LAB-RESULTS)
LEDGER: SHOP:DELIST (FHIR Integration Playbook)
NOTIFIER: DRIFT_ALERT → dr.nguyen (HIGH priority)
```

Recovery. Dr. Nguyen updates INTEL.md across all 6 scopes to reference both R4 v4.0.1 and R4B v4.3.0 (backward-compatible). She adds a LEARNING.md entry documenting the versioning pattern. All 6 scopes revalidate to 255. The product relists automatically within 4 hours of the drift detection:

```
LEDGER: MINT:WORK +8 COIN (evidence update, each scope)
LEDGER: SHOP:RELIST (FHIR Integration Playbook)
```

Total downtime: 4 hours. Zero manual intervention on the SHOP side — governance state drove listing state. The product was unavailable only while its governance was incomplete [12].

8.14. 8.14 Product Bundling and Composition

Products can be composed into bundles. A bundle is a parent SHOP.md that references child SHOP.md files:

```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

title: “Enterprise Health Informatics Bundle” type: BUNDLE children: - SERVICES/FHIR-PLAYBOOK/SHOP.md - SERVICES/EHR-MIGRATION/SHOP.md - SERVICES/COMPLIANCE-KIT/SHOP.md price: 5,000 cost_basis: 4,590

• • •

```
Bundle validation rules:

| Rule | Check | Rationale |
|------|-------|-----------|
| All children at 255 | Every child SHOP.md scope = 255 | Bundle quality gate |
| Bundle price >= sum(child cost_basis) | 5,000 >= 4,590 | Economic floor |
| No circular references | Bundle cannot contain itself | DAG enforcement |
| Child availability | All children LISTED or ACTIVE | No stale products in bundle |

If any child scope drifts, the bundle delists. The governance of each component propagates upward to the a
the same principle that governs scope inheritance applies to product composition.

## 8.15 Product Analytics

The LEDGER provides complete product analytics without a separate analytics service:

```bash
magic shop --analytics SERVICES/FHIR-PLAYBOOK --period 90d

# FHIR Integration Playbook — 90-Day Analytics
#
# Revenue:          68,000 COIN (34 sales × 2,000 COIN)
# Unique buyers:    31
# Institutional:    12 hospitals (39%), 3 HIEs (10%), 16 clinics (51%)
# Avg time-to-buy:  12 minutes (from discovery to SPEND)
```
```

```
# Refund requests:  0
# Drift incidents:  1 (4h downtime, auto-resolved)
# Child adaptation: 8 buyers (26%) forked and governed to 255
# COIN recycled:    2,040 (buyers who governed their forks)
```

The “COIN recycled” metric matters: 26% of buyers adapted the playbook to their institution, governed their version to 255, and minted 2,040 COIN through their own governance labor. The product does not just transfer knowledge — it seeds governance work that generates new economic activity. Purchase leads to adaptation, adaptation to governance labor, governance labor to COIN minting, COIN minting to the next purchase [1712].

8.15. 8.16 Product Versioning

Products are versioned through the git tag system. Each major version corresponds to a `magic-tag` certification:

```
v1.0.0 — Initial release (FHIR R4 v4.0.1)
v1.1.0 — Added R4B compatibility (v4.3.0)
v2.0.0 — FHIR R5 support (breaking: new resource types)
```

Existing purchasers retain access to the version they bought; new purchases get the latest. The LEDGER records which version each SPEND event targets:

```
{
  "event": "SPEND",
  "product": "FHIR Integration Playbook",
  "version": "v1.1.0",
  "amount": 2000,
  "buyer": "dr-park",
  "seller": "dr-nguyen",
  "ledger_event": "evt:05120"
}
```

Version upgrades for existing buyers are priced at the governance delta — the COIN minted between versions. If v1.0.0 → v1.1.0 required 48 COIN of governance work, the upgrade price is 48 COIN. The upgrade price is derivable from the LEDGER: sum of MINT:WORK between the two tag commits.

8.16. 8.17 Governance Proof: The Product Chain

The complete product chain from governance labor to market participation:

```
Dr. Nguyen writes CANON.md for Patient Demographics
  → magic validate → 1/255 (AXIOM bit set)
  → MINT:WORK +1 COIN → LEDGER records evt:03100
  → ... 86 more commits ...
  → magic validate → 255/255 (all bits set)
  → MINT:WORK +128 COIN (final dimension) → LEDGER records evt:03187
  → ... 5 more scopes governed to 255 ...
  → SHOP.md created with Card
  → magic validate --shop → all gates PASS
  → SHOP:LIST → LEDGER records evt:03650
  → Dr. Park discovers product via magic scan --shop
  → Dr. Park SPEND 2,000 COIN → LEDGER records evt:03700
  → Dr. Nguyen WALLET +2,000 COIN
  → Dr. Park receives product access
  → Dr. Park adapts, governs to 255, mints 255 COIN
  → Dr. Park creates her own SHOP.md
  → The SHOP grows by one product
```

Every step is a LEDGER event. Every COIN traces to governance labor, every product to a 255 score, every purchase to a verified identity. The chain is unbroken from labor to market. Q.E.D. [171215]

8.17. 8.18 Product Composition Patterns

Products compose in three patterns:

Pattern 1: Linear composition. Scopes arranged sequentially, each building on the previous.

```
Chapter 1 → Chapter 2 → Chapter 3 → ... → Chapter N
Product = ordered sequence of N scopes
Cost basis = N × 255 COIN
```

Books and playbooks use linear composition. Each chapter is a scope. The reading order is the scope order. The product is the sequence.

Pattern 2: Hierarchical composition. Scopes arranged in a tree, with a root scope and child scopes.

```
Root Service
  ├── Sub-service A
  │   ├── Component A1
  │   └── Component A2
  ├── Sub-service B
  └── Sub-service C
```

```
Product = tree of M scopes (M = total nodes)
Cost basis = M × 255 COIN
```

Service suites use hierarchical composition. The root scope defines the service architecture. Child scopes define components. The product is the tree.

Pattern 3: Graph composition. Scopes connected by cross-references, forming a directed acyclic graph (DAG).

```
Service A ←→ Service B
    ↓             ↓
Service C ←→ Service D
    ↓
Service E

Product = DAG of K scopes with L edges
Cost basis = K × 255 COIN
```

Clinical governance suites use graph composition. TALK agents reference INTEL scopes. INTEL scopes reference LEARNING scopes. MONITORING scopes reference all service scopes. The product is the graph.

```
# Visualize product composition
magic product --graph fhir-integration-playbook

# Graph:
#   Nodes: 8 scopes
#   Edges: 12 cross-references
#   Pattern: Hierarchical with cross-links (DAG)
#   Root: SERVICES/FHIR-API
#   Depth: 3 levels
#   Width: 4 leaves
```

8.18. 8.19 Product Quality Metrics

Beyond governance score (255/255), products accumulate quality signals from the market:

Metric	Source	Meaning
Attestation count	LEDGER SPEND events	Number of buyers
Attestation velocity	SPEND events / month	Purchase rate trend

Metric	Source	Meaning
Adaptation rate	% of buyers who fork and govern to 255	Usability indicator
LEARNING depth	Total LEARNING entries across product scopes	Maturity indicator
Evidence freshness	Mean age of INTEL citations	Currency indicator
Drift frequency	DEBIT:DRIFT events / year	Maintenance indicator

```
magic product --quality fhir-integration-playbook

# Product Quality Report:
#   Score: 255/255 (all 8 scopes)
#   Attestations: 47 (12-month)
#   Velocity: 3.9/month (stable)
#   Adaptation rate: 26% (above 20% target)
#   LEARNING depth: 42 entries across 8 scopes (5.25/scope)
#   Evidence freshness: mean 4.2 months (target < 12 months)
#   Drift frequency: 0.5 events/year (excellent stability)
#   Quality tier: GOLD (all metrics above target)
```

GOLD-tier products surface first in SHOP search results. The tier is computed from quality metrics, not paid promotion. The SHOP does not sell visibility — visibility is earned through governance quality [1712].

8.19. 8.20 Product Licensing and Reuse Rights

When a buyer purchases a product, the SPEND event grants specific reuse rights:

Right	Granted	Restricted
Read all governance files	Yes	—
Fork and customize for own organization	Yes	—
Deploy to own fleet	Yes	—
Add constraints (strengthen)	Yes	—
Remove constraints (weaken)	—	Cannot weaken inherited constraints
Resell as-is	—	Cannot resell without own governance work
Resell adapted version	Yes	Must reach 255 independently, cost basis reflects own work

Reuse rights are encoded in SHOP.md and enforced by the governance system. A buyer who forks and governs their version to 255 has earned their own cost basis through their own labor. Their adapted version is a new product — its own SHOP.md, its own price, its own market presence — coexisting with the original. The market decides which is more valuable based on attestation counts [1712].

8.20. 8.21 Clinical Product Example: Nursing Governance Suite

A chief nursing officer at a 700-bed academic medical center builds a governance suite for nursing-specific clinical AI agents. The suite covers 12 scopes:

```
magic product --detail nursing-governance-suite

# Nursing AI Governance Suite
# Author: nurse-informaticist@academic-med.org
# Scopes: 12
#   SERVICES/TALK/NURSEBOT (general nursing queries)
#   SERVICES/TALK/NURSEBOT/TRIAGE (ED triage assistance)
#   SERVICES/TALK/NURSEBOT/MEDICATION-ADMIN (medication administration guidance)
#   SERVICES/TALK/NURSEBOT/WOUND-CARE (wound assessment and care planning)
#   SERVICES/TALK/NURSEBOT/FALL-PREVENTION (fall risk assessment)
#   SERVICES/TALK/NURSEBOT/SEPSIS-SCREENING (qSOFA/NEWS2 early warning)
#   SERVICES/TALK/NURSEBOT/PAIN-ASSESSMENT (standardized pain scales)
#   SERVICES/TALK/NURSEBOT/DISCHARGE-PLANNING (care transition guidance)
#   SERVICES/INTEL/NURSING-EVIDENCE (evidence base for all nursing agents)
#   SERVICES/MONITORING/NURSING-METRICS (fleet monitoring for nursing agents)
#   SERVICES/LEARNING/NURSING-PATTERNS (institutional nursing AI learnings)
#   SERVICES/DEPLOY/NURSING-FLEET (deployment configuration)
#
# Cost basis: 3,060 COIN (12 × 255)
# Price: 4,200 COIN (37% margin)
# Attestations: 3 (2 academic medical centers, 1 community hospital)
# Evidence: 47 INTEL citations (ANA, AACN, NANDA-I, NIC/NOC)
# LEARNING: 31 entries over 9 months
```

The suite establishes the governance standard for nursing AI — vocabulary definitions (NANDA-I nursing diagnoses, NIC interventions, NOC outcomes), evidence citations (ANA position statements, AACN practice alerts), and clinical safety constraints (medication rights, fall risk thresholds, sepsis screening intervals). Every future nursing AI governance product in the SHOP either inherits from this suite or competes with it. First-mover advantage in governance is real: the first product to define the vocabulary becomes the standard [171215].

Chapter 9

Chapter 9: Composition and Federation

Governance that stops at organizational boundaries is not governance — it is a suggestion. CANONIC composes vertically through inheritance (parent to child, constraints accumulating monotonically — see Chapter 3) and horizontally through federation (org to org, governance verified through hashes without sharing private data). This chapter covers both axes and builds the architecture for multi-hospital governance at scale. The GALAXY visualization (hadleylab.org) renders the federated topology in real time (see Chapter 31). For the governor's perspective on federation, see CANONIC CANON Chapter 9.

9.1. 9.1 Composition

CANONIC composes along two axes [16]:

Vertical composition is the deployment architecture — how governance flows from standard to clinical deployment. A hospital deploys MammoChat, which inherits from TALK, which inherits from MAGIC, which inherits from canonic-canonic. Constraints accumulate from root to leaf.

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

9.2. 9.2 Federation Architecture

Federation is privacy-preserving distributed governance across multiple organizations. In healthcare, this is not optional — HIPAA requires it. PHI stays local; only governance metadata crosses organizational boundaries [16].

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

9.3. 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 deployments with their scores, tier statuses, and COIN trajectories. The network CISO sees the aggregate posture; hospital-level CISOs see site-specific details. Federation preserves both the aggregate view and the local privacy.

9.4. 9.4 Scale Evidence

The federation model is not theoretical — it is deployed. One developer, 19 GitHub organizations, 185+ repositories, all validating to 255 [16]. For a health network with five hospitals, the model has been proven at nearly 4x the required scale.

9.5. 9.5 ORG/USER Topology

ORG is the container. USER is the repo. The mapping to GitHub is direct: `github.com/{org}/{user}` [2].

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

Each hospital system in the federation is a separate GitHub organization with its own repositories, governance files, and validation pipeline. Governance metadata flows through the federation layer; clinical data never leaves the hospital's GitHub organization.

9.6. 9.6 The GALAXY

The GALAXY renders the federated topology as an interactive visualization — the governance equivalent of a network operations center. Every ORG, PRINCIPAL, SERVICE, and SCOPE is a node. Compliance rings show 8-dimension status using the DESIGN token system. INTEL flow pulses through edges where INTEL.md exists [18].

For a hospital board, the GALAXY provides an immediate read on AI governance posture: which deployments are at 255, which are climbing, which have DEBIT:DRIFT events. For the architect, it is the debugging tool — when a deployment fails validation, the GALAXY shows its position in the inheritance tree, its parent constraints, and its specific dimension gaps.

9.7. 9.7 Federation Protocol

Three rules govern how governance metadata crosses organizational boundaries:

Rule 1: Score sharing. Organizations share governance scores (0-255) publicly. A score is a number — it contains no PHI, no clinical data, no personally identifiable information.

Rule 2: Hash verification. Organizations share CHAIN hashes for cross-verification. Hospital A verifies Hospital B's MammoChat is at 255 by checking the published hash against the validator output. Trust is replaced by verification.

Rule 3: LEARNING aggregation. Organizations share anonymized LEARNING signals — patterns, discoveries, corrections — without sharing underlying clinical data. "Model update without validation causes drift" is safe to share. The patient data that triggered the drift detection is not.

```
# Federation verification
magic federation --verify network-canonic
# hospital-a: 255/255 hash=abc123 VERIFIED
# hospital-b: 255/255 hash=def456 VERIFIED
# hospital-c: 191/255 hash=ghi789 VERIFIED (drift detected)
# hospital-d: 255/255 hash=jkl012 VERIFIED
# hospital-e: 255/255 hash=mno345 VERIFIED
# Federation: 4/5 at 255. 1 drifting.
```

9.8. 9.8 Composition Depth

Composition has both vertical depth (inheritance chain) and horizontal breadth (federation peers):

Dimension	Metric	Practical Limit	Clinical Example
Vertical depth	Inheritance chain length	6 levels	root org services talk mammochat site
Horizontal breadth	Federation peer count	Unlimited	5 hospitals, 50 hospitals, 500 hospitals
Cross-scope	INTEL references per scope	~20	MammoChat referencing BI-RADS, NCCN, ACOG guidelines
Child scopes	Sub-scopes per parent	~50	TALK service with 50 clinical agents

Vertical depth is bounded by debuggability. Horizontal breadth is unbounded — federation scales linearly with organization count.

9.9. 9.9 Building a Federation: Step-by-Step

Step 1: Create the network root.

```
mkdir -p network-canonic
cd network-canonic
cat > CANON.md << 'EOF'
# NETWORK — CANON


## Axiom
```

```
**NETWORK federates clinical AI governance across member hospitals. Every member validated. Every score pu


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

9.10. Constraints

MUST: Validate every member scope to 255 before production MUST: Publish governance scores to federation dashboard MUST: Share anonymized LEARNING signals across members MUST NOT: Share PHI across organizational boundaries MUST NOT: Allow unvalidated member deployments MUST NOT: Override member site-specific constraints

• • •

EOF

Step 2: Add member organizations as submodules.

```bash
git submodule add https://github.com/hospital-a-canonic
git submodule add https://github.com/hospital-b-canonic
# Each member org inherits from network-canonic
```

Step 3: Validate the federation.

```
magic validate --recursive --fleet
# network-canonic: 255/255
#   hospital-a-canonic: 255/255
#     mammochat-a: 255/255
#   hospital-b-canonic: 255/255
#     mammochat-b: 255/255
# Federation: 5/5 scopes at 255
```

Step 4: Launch the GALAXY.
```

```
magic galaxy --render
# GALAXY visualization available at https://network.canonic.org/galaxy/
```

The federation is live. Member hospitals maintain independent governance. The network aggregates scores. The GALAXY visualizes the fleet. PHI stays local. Governance metadata flows through the federation protocol.

9.11. 9.10 Federation vs Centralization

Concern	Centralized	Federated (CANONIC)
Data location	Central database	Local to each org
PHI boundary	Shared data lake	Enforced per org
Governance authority	Central committee	Inherited constraints
Score computation	Central server	Local validator
Failure mode	Single point of failure	Independent operation
Scaling	Vertical (bigger server)	Horizontal (more orgs)
HIPAA compliance	Complex (shared PHI)	Simple (PHI stays local)

Federation is not a compromise — it is the architecturally correct model for multi-hospital governance. Clinical data stays where HIPAA requires it. Governance metadata flows where it is needed. The boundary is an architectural constraint enforced by the inheritance chain and the federation protocol, not a policy decision [16].

9.12. 9.11 Clinical Vignette: Federation During a Joint Commission Survey

CommonSpirit Health operates 140 hospitals across 21 states. Their clinical AI governance federation covers 45 clinical TALK agents deployed across 80 hospitals. During a Joint Commission survey at Dignity Health Northridge Hospital (one of the 140), the surveyor asks: “How do you verify that your clinical AI tools are governed consistently across all deployment sites?”

The clinical informatics officer runs one command:

```
magic federation --verify commonspirit-canonic --focus dignity-northridge
# Federation: commonspirit-canonic
# Total members: 80 hospitals
# Total scopes: 360 (avg 4.5 per hospital)
# Fleet status: 357/360 at 255 (99.2%)
```

```
# Dignity Northridge scopes:
#   MAMMOCHAT-NORTHRIDGE: 255/255 (chain: canonic-canonic -> commonspirit -> TALK -> MAMMOCHAT -> NORTHRID
#   CARDICHAT-NORTHRIDGE: 255/255
#   EMERGECHAT-NORTHRIDGE: 255/255
#   PHARMCHAT-NORTHRIDGE: 255/255
# Site: 4/4 at 255. No drift.
# Last validation: 12 minutes ago
# COIN minted (site): 1,020 COIN
# COIN minted (fleet): 91,800 COIN
```

The surveyor sees four things: every clinical AI tool at Northridge is at 255, governance validates every 12 minutes, it inherits from a system-wide standard, and only 3 of 360 fleet scopes are below 255 (99.2% compliance). No need to review 140 hospitals individually — the federation aggregates the posture.

The three drifting scopes are at Mercy Hospital Sacramento, a LEARNING.md staleness issue detected that morning. The dashboard shows the drift, the timestamp, and the expected recovery. Traditional compliance programs would have buried those 3 scopes in a 200-page aggregate report. CANONIC surfaces them in real-time [16].

9.13. 9.12 Federation Economics

Federation produces economic benefits through shared governance investment:

```
magic federation --economics commonspirit-canonic
# Investment analysis:
#   Cost to build 360 scopes independently: 91,800 COIN
#   Cost with federation (shared templates + LEARNING transfer): 45,900 COIN
#   Savings from federation: 45,900 COIN (50%)
#   Source of savings:
#     Shared TALK service constraints (inherited, not duplicated): 28,000 COIN
#     LEARNING transfer across sites (patterns reused): 12,000 COIN
#     Template standardization (reduced bootstrap time): 5,900 COIN
```

The savings come from the inheritance chain. When 80 hospitals inherit from the same TALK service, its 7 constraints are defined once and enforced everywhere. Without federation, each hospital defines its own constraints — 80 independent definitions, 80 vocabulary sets, 80 compliance reviews. With federation, one definition serves 80 hospitals. DRY applied to governance economics [1612].

9.14. 9.13 Federation Conflict Resolution

When two federated organizations have conflicting governance requirements, the conflict is resolved at the common ancestor:

```
magic federation --conflicts commonspirit-canonic
# Conflict detected:
#   Scope: SERVICES/TALK/CARDICHAT
#   Hospital A constraint: MUST cite ACC/AHA guidelines
#   Hospital B constraint: MUST cite ESC guidelines
#   Common ancestor: commonspirit-canonic/SERVICES/TALK
#   Resolution: Ancestor adds MUST cite cardiology society guidelines
#               Each hospital adds specific society as site constraint
#   Status: RESOLVED (ancestor updated, both children pass)
```

Resolution follows the inheritance rule: the common ancestor defines the general constraint (cite cardiology society guidelines), and each hospital adds the specific constraint (ACC/AHA in the US, ESC for European-trained cardiologists). Neither child weakens the ancestor. Both satisfy the full constraint union. The conflict resolves at the architectural level, not the committee level [16].

9.15. 9.14 Federation Monitoring

The MONITORING service provides fleet-wide dashboards for federated deployments:

```
magic federation --dashboard commonspirit-canonic
# ┌──────────────────────────────────────────────────────────┐
# │  CommonSpirit Health — Clinical AI Governance Federation │
# ├──────────┬───────┬──────────┬──────────┬─────────────────┤
# │ Region   │ Sites │ At 255   │ Drifting │ COIN (30d)      │
# ├──────────┼───────┼──────────┼──────────┼─────────────────┤
# │ West     │ 28    │ 28 (100%)│ 0        │ +3,420          │
# │ Midwest  │ 22    │ 22 (100%)│ 0        │ +2,640          │
# │ South    │ 18    │ 17 (94%) │ 1        │ +1,980          │
# │ East     │ 12    │ 12 (100%)│ 0        │ +1,440          │
# ├──────────┼───────┼──────────┼──────────┼─────────────────┤
# │ Total    │ 80    │ 79 (99%) │ 1        │ +9,480          │
# └──────────┴───────┴──────────┴──────────┴─────────────────┘
```

The South region has one drifting site — visible immediately. Drill down with `magic federation --drill south` to identify the specific hospital and scope [1615].

9.16. 9.15 Governance Proof: Federation Correctness

The federation model preserves governance correctness. The proof:

1. Every federated organization inherits from a common ancestor (by construction).
2. The common ancestor defines minimum constraints (by CANON.md).
3. Inheritance is monotonically enriching — children add, never remove (by Theorem 3).
4. Each site validates independently using the same kernel (`magic validate`).
5. Validation is deterministic — same governance files produce same score (by Theorem 2).
6. Federation metadata contains no PHI (by architectural enforcement).

Therefore: every site in the federation satisfies the common ancestor's constraints AND its own site-specific constraints. The governance is federated (independent validation) AND consistent (shared constraint floor). The PHI boundary is preserved because federation metadata is structurally incapable of containing clinical data. Q.E.D. [163].

9.17. 9.16 Federation Onboarding Protocol

Adding a new organization to a federation follows a governed protocol:

```
magic federation --onboard new-hospital-canonic \
  --parent commonspirit-canonic \
  --region south

# Federation Onboarding Protocol:
#
# Step 1: IDENTITY — Verify organizational identity
#   Organization: New Hospital Health System
#   Contact: cio@new-hospital.org
#   GitHub org: new-hospital-canonic
#   Status: IDENTITY VERIFIED □
#
# Step 2: ANCHOR — Pin parent submodule
#   Parent: commonspirit-canonic (v2.3.0)
#   Inherits: commonspirit-canonic/SERVICES/TALK (clinical AI constraints)
#   Status: SUBMODULE PINNED □
#
# Step 3: BOOTSTRAP — Generate initial governance tree
#   Scopes generated: 12 (from federation template)
#   Initial score: 0/255 (all dimensions pending)
#   Status: BOOTSTRAP COMPLETE □
#
```

```
# Step 4: VALIDATE — First magic validate
#   Scopes at 0: 12 (expected — new organization)
#   Inherited constraints: 7 from parent □
#   Status: VALIDATION PASSED (score 0 is valid for new org)
#
# Step 5: LEDGER — Record onboarding
#   FEDERATION:ONBOARD recorded (evt:federation:00420)
#   Fleet size: 81 (was 80)
#
# Next: Govern scopes to 255. Estimated time: 4–8 weeks.
```

Onboarding produces a governed starting point, not an empty repository. The federation template includes the parent's constraints, vocabulary, and coverage expectations. Score 0 with clear guidance on what 255 requires [1619].

9.18. 9.17 Federation Governance Tiers

Federated organizations can operate at different governance maturity levels:

Tier	Score Requirement	Participation Rights	Monitoring
OBSERVER	No minimum	Read-only access to federation templates	Monthly report
PARTICIPANT	≥ 191 at all scopes	Can inherit and customize templates	Weekly report
CONTRIBUTOR	255 at all scopes	Can propose changes to common ancestor	Daily monitoring
STEWARD	255 + LEARNING maturity	Can approve ancestor changes, review contributions	Real-time dashboard

```
magic federation --tiers commonspirit-canonic

# Federation Tier Report:
#   STEWARD:      3 organizations (30+ LEARNING entries, 365 days at 255)
#   CONTRIBUTOR: 42 organizations (all scopes at 255)
#   PARTICIPANT: 30 organizations (all scopes ≥ 191)
#   OBSERVER:     5 organizations (onboarding, < 191)
#   Total:       80 organizations
```

The tiers incentivize improvement: OBSERVERs see the value of governance templates but cannot modify them, PARTICIPANTs use templates and provide feedback, CONTRIBUTORs propose improvements to the common ancestor, and STEWARDs approve or reject proposals based on fleet-wide impact analysis [1619].

9.19. 9.18 Federation Data Sovereignty

Each organization retains full sovereignty over its data. The federation shares governance contracts (CANON.md, VOCAB.md, COVERAGE.md) but never shares operational data (patient records, clinical notes, financial data).

Shared (governance)	Not shared (sovereign)
CANON.md constraints	Patient data
VOCAB.md terminology	Clinical notes
COVERAGE.md matrices	Financial records
LEARNING.md patterns (anonymized)	Staff records
ROADMAP.md milestones	Audit details
HTTP.md route contracts	API keys and secrets
Score metadata (255/255)	WALLET balances

The separation is architectural, not policy-based. The federation submodule contains only `.md` governance files — there is no mechanism to include operational data in a submodule. The git repository structure enforces the boundary:

```
commonspirit-canonic/           ← federation parent (shared)
  CANON.md                      ← shared constraints
  VOCAB.md                      ← shared terminology
  SERVICES/TALK/CANON.md        ← shared TALK constraints

new-hospital-canonic/           ← local organization (sovereign)
  canonic-canonic/              ← submodule (shared root)
  commonspirit-canonic/         ← submodule (shared federation)
  SERVICES/                     ← local governance
  VAULT/                        ← local economic data (NEVER shared)
  LEDGER/                       ← local event chain (NEVER shared)
```

`VAULT/` and `LEDGER/` are local to each organization — they never appear in submodules and never cross federation boundaries. A federation partner can verify that your scopes are at 255 by reading published score metadata, but cannot see your WALLET balances, transaction history, or economic details. Governance is transparent; economics are private [161912].

9.20. 9.19 Clinical Vignette: Federation Enables Multi-Site Clinical Trial Governance

Duke Clinical Research Institute (DCRI) coordinates a multi-site clinical trial across 23 academic medical centers: ADAPTABLE-2 (Aspirin Dosing: A Patient-centric Trial, Assessing Benefits and Long-term Effec-

tiveness, version 2). Each site deploys a governed TALK agent (TrialChat) that helps research coordinators navigate the trial protocol.

The federation structure:

```
dcri-canonic/ (federation parent)
  SERVICES/TALK/TRIALCHAT/
    CANON.md          ← 12 constraints (protocol-specific)
    VOCAB.md          ← 47 terms (aspirin dosing, outcomes)
    INTEL.md          ← ADAPTABLE-2 protocol, DSMB reports
    COVERAGE.md       ← IRB requirements across all 23 sites

site-duke-canonic/    (inherits dcri-canonic)
site-stanford-canonic/ (inherits dcri-canonic)
site-mayo-canonic/    (inherits dcri-canonic)
... (23 total)
```

Each site inherits the protocol constraints but adds site-specific IRB language. When DCRI issues a protocol amendment (changing the aspirin dose from 325mg to 162.5mg for the low-dose arm based on interim analysis), the amendment is a governance update to the federation parent:

```
# DCRI updates federation parent
cd dcri-canonic
vim SERVICES/TALK/TRIALCHAT/VOCAB.md  # Update LOW_DOSE: 162.5mg
vim SERVICES/TALK/TRIALCHAT/INTEL.md  # Add DSMB report reference
git commit -m "GOV: ADAPTABLE-2 Amendment 3 — low dose 325→162.5mg"
magic validate  # 255 □
```

All 23 sites bump their submodule reference and revalidate:

```
# Each site runs:
git submodule update --remote dcri-canonic
magic validate --recursive
# TrialChat: 255/255 □  (inherits updated LOW_DOSE definition)
```

The protocol amendment propagates to 23 sites in a single governance cycle. Each site validates independently, each TrialChat serves the updated dosing information, and the LEDGER records the amendment at each site. The DSMB can verify that all 23 sites adopted the amendment by querying governance scores and LEDGER history [16319].

9.21. FEDERATION: Operational

FEDERATION is no longer a design document. As of March 2026, four ORGs operate under the CANONIC standard:

ORG	GitHub	Domain	Scopes
canonic-canonic	github.com/canonic-canonic	Kernel + standard	111
hadleylab-canonic	github.com/hadleylab-canonic	Healthcare + governance — hadleylab.org	255
canonic-apple	—	Platform SDK	—
RunnerMVP	github.com/RunnerMVP	Real estate operations — gorunner.pro	13

The GALAXY/ORGS/ORGS.md registry discovers ORGs structurally — no hardcoded list. Each ORG maintains its own governance tree. Each ORG's scopes are independently validated. The FEDERATION topology connects them.

9.22. The WITNESS Protocol

Cross-ORG trust requires cross-ORG verification. WITNESS provides it:

1. **DIGEST** — Each ORG computes a signed JSON summary: head SHA-256, event count, COIN totals, wallet balances. Signed with the ORG governor's Ed25519 key.
2. **WITNESS** — A peer ORG verifies the DIGEST hash and countersigns. Both ORGs store the countersignature.
3. **Threshold** — 2-of-N for N < 5 ORGs; 3-of-N for N ≥ 5.
4. **Recovery** — If an ORG deletes its repository, the kernel's WITNESSES/{org}/DIGEST.json reconstructs balances. Ed25519 signatures are non-repudiable.

At current scale (2 active witnesses: RunnerMVP and canonic-canonic), the protocol is simple. At federation scale (40+ ORGs), the WITNESS graph becomes the inter-organizational trust mesh.

Chapter 10

Chapter 10: LEARNING

Knowledge decays. A clinical AI agent trained on 2024 NCCN guidelines will silently drift as guidelines update, and the agent does not know what it does not know. LEARNING is the architectural answer: the INTEL primitive projected into runtime as a service that captures, generalizes, and propagates knowledge across the entire governance tree. Every governed operation produces LEARNING. Every LEARNING event is evidence. The service closes the loop between governance work and institutional knowledge [20].

10.1. 10.1 Axiom

LEARNING is INTEL applied. Every discovery governed. Every gradient evidenced [20].

Where INTEL.md is a file — static knowledge compiled into a scope — LEARNING is the service that makes knowledge dynamic. When new evidence enters the governance tree, LEARNING records the delta. When a clinical scope's INTEL changes, LEARNING propagates the change to dependent scopes. The knowledge network stays current because LEARNING is always listening.

10.2. 10.2 The IDF Generalization

The Invention Disclosure Form (IDF) is a structured document used in patent prosecution to capture a novel discovery: what was discovered, when, by whom, what prior art exists, and what claims can be made. LEARNING generalizes this pattern beyond patents to all governed scopes. Every discovery — code pattern, compliance insight, architectural decision, clinical protocol change — is a governed LEARNING record with the same structural rigor as a patent disclosure [20].

The generalization works because IDF structure maps directly to the eight governance questions:

IDF Field	LEARNING Field	Governance Question
Title of Invention	Pattern	What do you believe?
Date of Conception	Date	Where are you going?
Inventors	Actor	Who are you?
Prior Art	References	Can you prove it?
Claims	Assertions	What shape are you?
Evidence of Reduction to Practice	Evidence	How do you work?
Background/Field	Context	What have you learned?
Novelty Assessment	Gradient	How do you express?

A LEARNING record IS a generalized IDF — the same structured capture of novel knowledge, the same evidentiary rigor, the same provenance chain. The difference is scope: IDFs capture patentable inventions; LEARNING records capture all governed knowledge. A team discovering that MammoChat's BI-RADS 4A sensitivity improved after a model update records that discovery as a LEARNING event with the same structural completeness as a patent IDF [20].

10.3. 10.3 LEARNING.md Pattern Table

LEARNING.md is the active knowledge log for every governed scope, recording discoveries as signal-pattern pairs in a governed table:

```
## Patterns

| Date | Signal | Pattern | Source |
|------|--------|---------|--------|
| 2026-02-26 | EVOLUTION | Full rebuild from stale book. | Plan file |
| 2026-02-26 | NEW_CONSTRAINT | IP compliance — no kernel internals. | CANON.md |
| 2026-02-27 | GOV_FIRST | Healthcare vertical directive added. | INTEL.md |
| 2026-03-03 | EVOLUTION | Service ontology — CHAT→TALK. | GOV commit |
```

Signals are the vocabulary of governance change. Each signal type represents a category of knowledge event:

Signal	Meaning	Clinical Example
`EVOLUTION`	Major state change, epoch boundary	MammoChat v2 model upgrade
`NEW_CONSTRAINT`	New MUST/MUST NOT added to CANON	PHI boundary constraint for OncoChat
`NEW_SCOPE`	Child scope created	Hospital-A deploys site-specific MammoChat
`GOV_FIRST`	Governance precedes implementation	HIPAA audit gate added before feature work

Signal	Meaning	Clinical Example
`EXTINCTION`	Scope or pattern retired	Legacy screening chatbot decommissioned
`DRIFT`	Score regression detected	OncoChat drops from 255 to 191 after config change
`CLOSURE`	Gap closed, all dimensions satisfied	MedChat achieves FULL (255/255)

The table is append-only within an epoch — new patterns go at the bottom, growing monotonically until an epoch boundary triggers rotation [20].

10.4. 10.4 Record Shape

Every LEARNING record — whether captured in LEARNING.md or stored in the LEARNING service's CAS — follows a governed record shape:

Field	Type	Content	Clinical Example
Pattern	string	What was discovered	"BI-RADS 4A sensitivity improved 12% after NCCN 2026.1 update"
Date	ISO-8601	When discovered	2026-03-01T09:15:00Z
Actor	string	Who discovered it	dexter (VITAE-verified)
Signal	enum	Category of change	EVOLUTION
Priority	string	Source evidence tier	GOLD (peer-reviewed evidence)
Questions	string	Which governance questions affected	"belief, proof, learning"
Assertions	array	Structured claims	["sensitivity_delta: +0.12", "source: NCCN 2026.1"]
Evidence	array	Provenance chain	["NCT06604078", "BI-RADS 5th ed.", "NCCN v2026.1"]
References	array	Cross-scope links	["TALK/MAMMOCHAT/INTEL.md", "PAPERS/mammochat-trial"]
Gradient	integer	What changed (delta)	+12 (from 243 to 255)

The shape is uniform across all domains. A patent discovery and a clinical AI improvement use the same fields, the same provenance chain, the same gradient calculation. Any tool that reads LEARNING records can process records from any scope without domain-specific logic [20].

10.5. 10.5 Storage Architecture

Records live in a content-addressable storage (CAS) system with git-style addressing:

```
~/.canonic/services/learning/
  ├── manifests/
  │   ├── EVOLUTION.json       ← shard per signal type
  │   ├── NEW_CONSTRAINT.json
  │   ├── NEW_SCOPE.json
  │   └── index.json           ← thin index across all shards
  ├── cas/
  │   ├── a1/
  │   │   └── b2c3d4e5f6...    ← content-addressed record
  │   ├── f7/
  │   │   └── 89abcdef01...
  │   └── ...
  └── checkpoints/
      ├── hadleylab-canonic.json  ← last-scanned commit per repo
      └── canonic-canonic.json
```

CAS fanout: Records are stored by content hash with 2-char prefix buckets (`hash[:2]/hash[2:]`). This is the same addressing scheme git uses for object storage — proven at scale, O(1) lookup, no directory explosion [20].

Manifest sharding: One manifest per signal type. A LEARNING query for all EVOLUTION events reads one shard file, not the entire corpus. The thin index maps record hashes to shards for cross-signal queries.

Incremental discovery: Checkpoints track the last-scanned commit per repository. On scan, LEARNING reads only new commits since the checkpoint. For a health network with 200 repositories, incremental scanning reduces discovery time from minutes to seconds.

Hard limits: No single manifest file with 100K+ entries. No single flat CAS directory with 100K+ files. If either limit is approached, the sharding algorithm splits. Clinical AI deployments at hospital-network scale can produce millions of governance events; the architecture must handle that volume without degradation.

10.6. 10.6 Epoch Rotation

LEARNING.md rotates at epoch boundaries. An epoch is a named period of governance evolution — defined by an EVOLUTION signal that declares the boundary:

Date	Signal	Pattern	Source
2026-03-10	EVOLUTION	NYT_POLISH_COMPLETE. All chapters ≥ 3K words.	ROADMAP.md

When an EVOLUTION signal fires:

1. The current LEARNING.md is frozen

2. A snapshot is archived as `LEARNING-{EPOCH}.md` at scope root
3. A new LEARNING.md is initialized with the epoch boundary event
4. The archive file is retained for historical provenance

Epoch rotation maps to natural clinical governance cycles: model version upgrades, guideline updates (NCCN 2025 to 2026), accreditation cycles (Joint Commission survey complete), and regulatory changes (FDA guidance update). Each epoch captures the governance knowledge accumulated during that period, and the archive provides the historical trail regulatory auditors require [2].

10.7. Epoch Rotation: Operational

LEARNING operates in epochs. Epoch 1 (CONSTRUCTION) has been archived — its patterns captured the building of the governance framework itself. Epoch 2 (OPERATION) is active — its patterns capture production governance signals.

The transition matters. CONSTRUCTION-era patterns like BLOAT EXTINCTION (discovering that COVERAGE.md should be generated, not hand-authored) became permanent infrastructure. OPERATION-era patterns — DRIFT detection, FEDERATION events — represent ongoing production intelligence.

Signal types in OPERATION:

Signal	Source	Example
DRIFT	`magic validate` regression	Score dropped from 255 to 191 after merge
EVOLUTION	Scope structural change	New service added to TALK fleet
FEDERATION	Cross-ORG event	RunnerMVP joined FEDERATION
WITNESS	Cross-ORG verification	DIGEST countersigned by peer ORG
HEAL	`magic heal` resolution	Missing COVERAGE.md restored

The backpropagation loop remains: signal enters LEARNING.md analysis identifies root cause fix ships signal type becomes obsolete if root cause is permanently resolved.

10.8. 10.7 INTEL Sources and Cross-Scope Propagation

LEARNING ingests INTEL from multiple governed sources. Each source type produces different LEARNING signals:

Source	Signal Types	Clinical Example
TALK sessions	SESSION_LEDGERED, QUERY_PATTERN	MammoChat screening Q&A patterns
CONTRIBUTE	CONTRIBUTION_ACCEPTED, CONTRIBUTION_GOLD	External researcher adds BI-RADS evidence
LEDGER events	MINT:WORK, DEBIT:DRIFT	Governance score changes across fleet
Governance commits	NEW_CONSTRAINT, NEW_SCOPE	New clinical AI scope deployed
Cross-scope references	EVIDENCE_BRIDGE	OncoChat cites MammoChat INTEL

Cross-scope propagation: When LEARNING captures a pattern in one scope, it evaluates whether the pattern applies to other scopes in the governance tree. A clinical evidence update in MammoChat (new BI-RADS atlas edition) may propagate to OncoChat (which references breast cancer staging). The propagation is governed — LEARNING does not modify downstream scopes. It creates a LEARNING record in the downstream scope's LEARNING.md with a reference to the upstream source. The downstream maintainer decides whether to act on the signal [20].

In a health network with 50 clinical AI agents across five hospitals, a guideline update captured at one hospital's deployment propagates as a LEARNING signal to every other relevant deployment. No manual distribution, no email chains, no forgotten updates. The governance tree carries the knowledge.

10.9. 10.8 The LEARNING Question and MAGIC Closure

LEARNING answers "What have you learned?" in the MAGIC 255 standard. Without LEARNING.md, the question is unanswered. With LEARNING.md but no patterns, the governance infrastructure exists but has not yet produced knowledge. With evidenced patterns, epoch rotations, and cross-scope references, the question is fully answered [6].

This is the dimension that separates compliant systems from intelligent ones. An ENTERPRISE scope passes audits, satisfies HIPAA, and records events. An AGENT scope does all of that and also learns from its clinical interactions, captures emerging patterns, and propagates knowledge to dependent scopes. LEARNING turns governed compliance into governed intelligence [10].

10.10. 10.9 CLI Operations

LEARNING is not a passive accumulator — it is an active service with CLI entry points you invoke as part of governance workflows:

```
# Discover new LEARNING patterns across all governed repositories
magic scan --learning

# Discover patterns in a specific scope
magic scan --learning --scope SERVICES/TALK/MAMMOCHAT

# Validate LEARNING.md structure and signal vocabulary
magic validate --scope SERVICES/LEARNING

# Extract patterns matching a signal type
magic scan --learning --signal EVOLUTION

# Force epoch rotation
magic scan --learning --rotate --epoch "NCCN_2026_UPDATE"

# Export LEARNING corpus for downstream analytics
magic scan --learning --export json > learning-export.json
```

The `magic scan --learning` command executes a four-phase pipeline:

Phase	Operation	Output
1. Checkpoint read	Load last-scanned commit per repo	Incremental boundary
2. Diff extraction	Read commits since checkpoint	Raw governance diffs
3. Pattern classification	Classify each diff into signal types	Typed LEARNING records
4. CAS write	Hash and store each record	Content-addressed artifacts

In a health network with 200 repositories, phase 1 reduces the scan surface to only new commits. Phase 2 extracts governance-relevant diffs (CANON.md, INTEL.md, LEARNING.md, SHOP.md changes). Phase 3 classifies each diff into the signal vocabulary. Phase 4 writes classified records into the CAS. The pipeline is idempotent — running it twice on the same commit range produces identical CAS entries because content-addressing deduplicates by hash.

10.11. 10.10 Error Handling and Failure Modes

LEARNING handles five categories of failure:

1. Malformed LEARNING.md. The validator checks that LEARNING.md contains a valid Markdown table with the required columns (Date, Signal, Pattern, Source). A missing column header, a malformed date, or an unrecognized signal type triggers a validation error:

```
ERROR: SERVICES/TALK/MAMMOCHAT/LEARNING.md
```

```
  Line 8: Signal "EVOLVE" not in vocabulary.
  Valid signals: EVOLUTION, NEW_CONSTRAINT, NEW_SCOPE,
    GOV_FIRST, EXTINCTION, DRIFT, CLOSURE
  Run: magic heal --scope SERVICES/TALK/MAMMOCHAT
```

`magic heal` auto-corrects common signal misspellings (EVOLVE to EVOLUTION, NEWSCOPE to NEW_SCOPE) when the correction is unambiguous. Ambiguous corrections require manual resolution.

2. Orphaned CAS entries. A CAS record not referenced by any manifest shard is orphaned. `magic scan --learning --gc` identifies and reports them but does not delete automatically — an orphaned entry may indicate manifest corruption that needs investigation, not cleanup.

3. Checkpoint corruption. If a checkpoint references a commit that no longer exists (force-push, rebased branch), the scanner falls back to a full-history scan for that repository. The fallback is logged as a DRIFT signal in the LEARNING service's own LEARNING.md — the service learns about its own failures.

4. Cross-scope propagation failure. When a pattern propagates from MammoChat to OncoChat, and OncoChat's LEARNING.md is locked (epoch rotation in progress), the propagation is queued in a pending manifest:

```
~/.canonic/services/learning/pending/
  └── ONCOCHAT-a1b2c3d4.json  ← queued propagation record
```

Pending records are retried on the next `magic scan --learning` invocation. If a pending record fails three consecutive retries, it is escalated to a DRIFT signal on the LEARNING service scope.

5. Evidence gap. A LEARNING record with empty Evidence or References arrays triggers a governance warning — not an error, but a quality signal. The warning appears in validation output:

```
WARN: LEARNING record a1b2c3d4 has no evidence chain.
  Pattern: "BI-RADS 4A sensitivity improved 12%"
  Recommendation: Add evidence sources (NCT ID, guideline version, etc.)
```

A LEARNING corpus with a high evidence-gap rate indicates the team is recording discoveries but not linking them to clinical evidence — degrading the corpus's value for downstream analytics and regulatory audit.

10.12. 10.11 Clinical Vignette: MammoChat NCCN Guideline Update

Consider the workflow when NCCN publishes updated breast cancer screening guidelines (NCCN v2026.2) that change the screening interval recommendation for women aged 40-49 from annual to biennial for average-risk patients.

Day 0: Guideline published. The clinical governance team updates MammoChat's INTEL.md to reference the new guideline version. The commit produces a governance diff.

Day 1: LEARNING discovery. The next `magic scan --learning` run detects the INTEL.md change in the MammoChat scope. The scanner classifies the diff as an EVOLUTION signal:

```
| 2026-06-15 | EVOLUTION | NCCN v2026.2 screening interval change: 40-49 avg-risk biennial. | INTEL.md com
```

The LEARNING record includes structured assertions: `["screening_interval: annual→biennial", "age_group: 40-49", "risk_level: average", "source: NCCN v2026.2"]`. The evidence chain references `["NCCN Breast Cancer Screening v2026.2", "NCT06604078"]`.

Day 2: Cross-scope propagation. The scanner evaluates cross-scope impact. OncoChat references MammoChat's INTEL for breast cancer staging. The scanner creates a LEARNING record in OncoChat's LEARNING.md:

```
| 2026-06-16 | EVIDENCE_BRIDGE | Upstream NCCN v2026.2 screening change in MammoChat. Review OncoChat brea
```

MedChat, which references general screening guidelines, also receives a propagated record. The propagation is automatic; the clinical team reviews the downstream signals and decides whether to update INTEL files.

Day 5: LEDGER recording. The governance officer reviews the LEARNING records, updates OncoChat's INTEL.md, and the work produces a MINT:WORK event: 47 COIN for gradient advancement across three scopes. The LEARNING corpus now contains the full provenance chain — upstream guideline change, cross-scope propagation, downstream INTEL update, and LEDGER economic recording.

10.13. 10.12 Clinical Vignette: Tamoxifen-to-Aromatase Inhibitor Protocol Shift

A large academic medical center uses OncoChat for oncology decision support. The ASCO 2026 guideline update recommends aromatase inhibitors (anastrozole 1mg daily, letrozole 2.5mg daily) as first-line adjuvant endocrine therapy for postmenopausal HR+/HER2- early breast cancer, replacing the prior recommendation of tamoxifen 20mg daily for 5 years as an acceptable first-line option.

LEARNING capture. The governance team records the discovery:

```
{
  "pattern": "ASCO 2026: AI first-line over tamoxifen for postmenopausal HR+/HER2- early breast",
  "signal": "EVOLUTION",
  "assertions": [
    "drug_class: aromatase_inhibitor",
    "agents: anastrozole_1mg, letrozole_2.5mg",
```

```
    "replaces: tamoxifen_20mg_first_line",
    "population: postmenopausal_HR+_HER2-_early",
    "evidence_level: Category_1",
    "source: ASCO_2026_adjuvant_endocrine"
  ],
  "evidence": [
    "ASCO Clinical Practice Guideline 2026",
    "ATAC Trial (anastrozole vs tamoxifen)",
    "BIG 1-98 (letrozole vs tamoxifen)",
    "NCCN Breast Cancer v2026.2"
  ],
  "gradient": 31
}
```

Cross-scope impact. The scanner propagates to MammoChat (screening context references adjuvant therapy), MedChat (general oncology references), and FinChat (CPT/ICD-10 coding for aromatase inhibitor prescribing). Each receives an EVIDENCE_BRIDGE signal. The FinChat propagation is clinically significant — coding changes for anastrozole vs. tamoxifen affect reimbursement workflows.

Quantified outcome. Before LEARNING, guideline updates propagated via email chains, averaging 23 days from publication to institutional awareness across five sites. After LEARNING, automated propagation achieved 100% cross-scope notification within 48 hours. Governance velocity for the endocrine therapy update: +142 COIN across 12 affected scopes.

10.14. 10.13 Network Map and VITAE Integration

LEARNING records are produced by governed actors — identified individuals with VITAE.md profiles. The LEARNING service maintains a network map that tracks the relationship between actors, discoveries, and scopes:

```
~/.canonic/services/learning/network/
  ├── actors.json          ← actor → discovery count, signal distribution
  ├── galaxy.json          ← scope → discovery count, propagation count
  ├── edges.json           ← actor-scope pairs with strength weights
  └── clusters.json        ← discovered communities of practice
```

This is not a social graph — it is a knowledge provenance graph: who discovered what, in which scope, with what evidence. Which actors produce the highest-quality LEARNING records? Which scopes generate the most DRIFT signals? Which actor-scope pairs have the strongest knowledge production edges?

For a chief medical informatics officer, the network map answers the questions that surveys and self-reports cannot: who is driving governance at each site, which sites produce knowledge that propagates to others, and which scopes are stable versus drifting [20].

VITAE integration is enforced by a binding constraint: `MUST NOT: Create LEARNING/{slug}.md without USERS/{slug}/VITAE.md`. Every actor in the LEARNING network must have a verified VITAE profile, ensuring provenance chains terminate at verified identities — not anonymous accounts, not shared credentials, not bot accounts. VITAE provides the identity layer; LEARNING provides the knowledge layer [20].

10.15. 10.14 The Onboarding Pipeline

When a new user joins the governance network, the LEARNING service executes a 4-channel scan pipeline (SOP-011):

```
1. VITAE check     → Does USERS/{slug}/VITAE.md exist?
2. KYC check       → Does directory name match legal identity? (SOP-010)
3. Network scan    → What existing actors share scope edges?
4. LEARNING init   → Create LEARNING/{slug}.md with MINT:SIGNUP record
```

The pipeline triggers on the first `magic scan --learning` invocation after a new USERS/{slug}/ directory appears. The scan detects the new scope (NEW_SCOPE signal), verifies VITAE existence, validates KYC naming, maps the actor into the network graph, and initializes their LEARNING history. Onboarding is not a registration form — it is a governance event discovered by scanning the filesystem [20].

10.16. 10.15 Closure

LEARNING is INTEL applied. Every discovery governed by the IDF-generalized record shape, every signal typed in the governance vocabulary, every record content-addressed in the CAS, every cross-scope propagation tracked, every actor verified through VITAE, every event recorded on the LEDGER. A scope that governs without learning is a bureaucracy. A scope that learns without governing is chaos. LEARNING is the governed intelligence layer — and for clinical AI, governed intelligence is the only kind that belongs in healthcare. For the theoretical closure that LEARNING provides to the eight-dimensional model, see Chapter 39 (The LEARNING Closure). For examples of LEARNING in production TALK agents, see Chapter 11 (TALK) and Chapter 25 (Contextual Agents).

Chapter 11

Chapter 11: TALK

Imagine a conversation engine where every response is backed by governed evidence, every session is recorded on an immutable ledger, and every agent speaks in the precise professional vocabulary of its domain — not because a developer remembered to add citations, but because the architecture makes ungoverned responses structurally impossible. That is TALK: the composition of CHAT and INTEL into a governed conversation service that powers every clinical AI agent in the CANONIC ecosystem [21].

11.1. 11.1 Axiom

TALK is CHAT + INTEL composed. Industry determines the voice. INTEL provides the knowledge [21].

Two binding obligations follow from this axiom. First, TALK must wire INTEL — the agent never speaks without governed knowledge backing its response. An LLM without governed INTEL is a chatbot; with it, the LLM becomes a clinical decision support tool whose evidence sources are auditable. Second, the industry determines the voice — a radiology agent speaks in radiological terminology, a legal agent in legal language. The voice is not cosmetic; it is governance.

11.2. 11.2 The Composition

```
TALK = CHAT + INTEL
```

This is not two systems bolted together. INTEL provides what the agent knows — the evidence base, the domain boundaries, the citation chain. CHAT determines how the agent speaks — the persona, the session management, the disclaimer architecture. TALK composes both into a single governed conversation where knowledge and voice are inseparable. You cannot deploy a TALK agent without both primitives present and validated, because the composition is enforced at build time [21].

Building a TALK agent requires a governed INTEL layer (the clinical evidence base compiled into the systemPrompt) and a CHAT configuration (the conversation engine with domain-specific voice settings). The build pipeline fuses them into a single artifact — CANON.json — that the runtime reads. The governance files are source code; CANON.json is the compiled output.

11.3. 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. 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 [21]

None of this is optional. Remove the governed systemPrompt and you have an ungoverned chatbot. Remove the disclaimer and you have a liability exposure. Remove the session ledger and you fail the next HIPAA audit. The governance is not a layer on top of the architecture — it is the architecture. Strip it away and what remains is not a degraded TALK agent; it is a different and dangerous thing entirely.

11.5. 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 [21].

The implication is broader than clinical AI: every governed scope in the ecosystem can have a TALK agent. A paper scope produces an agent that discusses the paper's findings. A service scope produces an agent that explains the service's architecture. A book scope produces an agent that navigates the book's content. The agent's knowledge boundary matches the scope's governance boundary — it knows what the scope knows, and nothing more.

For clinical AI, this means each evidence domain gets its own governed conversational interface:

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

11.6. 11.6 Persona Resolution

The CANON.md Persona table determines how the agent communicates. A clinical agent speaks in clinical language because its audience — radiologists, oncologists, pharmacists — expects precision. A legal agent speaks in legal language because its audience expects formality. The persona is compiled from CANON.md into the systemPrompt; the build pipeline reads it, not guesses it:

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 is not a UI skin. It is a governance constraint that shapes how the model formulates responses — the difference between "the mass appears suspicious" (clinical-third) and "you might want to get that

checked out" (generic). Governed agents do not drift into casual language because the persona is enforced at compile time.

11.7. 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 [21].

For clinical TALK agents, the disclaimer serves as the first line of both clinical safety and legal liability protection:

```
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. A frontend developer cannot accidentally suppress it with a CSS override — `magic validate` checks for disclaimer rendering and blocks deployment if it is missing. The governance framework ensures the disclaimer is always present, not because developers are untrustworthy, but because the consequence of omission is a liability exposure that no amount of after-the-fact correction can undo.

11.8. 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 [21].

Session governance means every conversation between a clinician and a TALK agent is a governed session — opened, recorded, and closed on the LEDGER. The session record captures governance metadata: who initiated the session (IDENTITY), when it started, how many turns it contained, what INTEL was cited, and when it ended. Critically, the session record does not include PHI — patient identifiers, clinical data, and protected information remain in the clinical system. The LEDGER sees the governance trail, not the clinical content.

For compliance teams, this architecture produces 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 — as a byproduct of normal operation. You do not reconstruct the audit trail from server logs after the fact. The governance architecture produces it continuously, automatically, and immutably.

11.9. 11.9 CLI Operations and Build Pipeline

TALK agents are compiled, not configured. The CLI provides the build and validation entry points:

```
# Validate a TALK scope's governance completeness
magic validate --scope SERVICES/TALK/MAMMOCHAT

# Build the TALK agent — compile INTEL into systemPrompt
build --scope SERVICES/TALK/MAMMOCHAT

# Scan for all TALK-enabled scopes in the governance tree
magic scan --talk

# Validate all TALK child scopes
magic validate --scope SERVICES/TALK --recursive

# Check that CANON.json declares users[] for cross-user routing
magic validate --scope SERVICES/TALK --check users

# Generate PDF from TALK.md via LATEX pipeline
build --scope SERVICES/TALK/MAMMOCHAT --format tex
```

The build pipeline follows a deterministic compilation path:

```
CANON.md → INTEL.md → build → CANON.json { systemPrompt, persona, users[] }
                                    ↓
                              talk.js → runtime agent
                                    ↓
                              TALK.md → LATEX → PDF (offline surface)
```

Each step in the pipeline is governed, and the separation between source and output is critical to understand. CANON.md and INTEL.md are human-authored source code — you write and maintain them. CANON.json is machine-compiled output — the build pipeline produces it. The runtime reads CANON.json exclusively; it never touches the source Markdown. The `_generated` marker in CANON.json means exactly what it says: if the output is wrong, fix the compiler or the contract, never the output file itself [21].

11.10. 11.10 The TALK Data Flow

Every conversation turn in a TALK session follows a governed data flow:

```
User message
```

```
→ TALK frontend (displays disclaimer, enforces PHI boundary)
→ Session manager (opens or continues governed session)
→ systemPrompt injection (compiled from INTEL.md via CANON.json)
→ LLM inference (model processes user message + systemPrompt)
→ Response filter (strips PHI leakage, enforces citation requirement)
→ LEDGER write (POST /talk/ledger — server-side, every turn)
→ COIN mint (COIN=WORK per session — governance work rewarded)
→ Response display (with disclaimer, with evidence citations)
```

Step	Governance Gate	Failure Mode
Frontend	Disclaimer displayed	Missing disclaimer BLOCK deployment
Session manager	Session ID created	No session no LEDGER trail audit failure
systemPrompt	INTEL compiled	Empty systemPrompt ungoverned response BLOCK
LLM inference	Model within scope boundary	Out-of-scope response DRIFT signal
Response filter	Citations present	No citation response flagged for review
LEDGER write	Event hash-chained	Write failure session paused, retry queue
COIN mint	MINT:WORK event	Mint failure logged, session continues

Every step in this data flow is a governance gate, and failure at any gate produces a specific, actionable error — not a generic “something went wrong.” The error tells you exactly which constraint was violated and what to do about it. When a response lacks citations, you know it is the response filter. When a session has no LEDGER trail, you know it is the session manager. The architecture makes failures diagnosable, not mysterious.

11.11. 11.11 Cross-User Messaging

TALK supports governed cross-user messaging. When user A needs to send a message to user B, the message is delivered through the governed inbox — not through a side-channel:

```
{
  "endpoint": "POST /talk/send",
  "from": "dr-martinez",
  "to": "dr-chen",
  "scope": "SERVICES/TALK/MAMMOCHAT",
```

```
  "message": "Review BI-RADS 4A case — LEARNING record a1b2c3d4",
  "session_id": "s-789abc"
}
```

Every cross-user message is LEDGER-recorded. The CANON.json `users[]` array declares which users can receive messages in each TALK scope — a user not in the array cannot receive messages in that scope. The routing is governed by the same CANON.md that governs the agent itself.

In clinical practice, this enables governed collaboration across specialties: a radiologist using MammoChat flags a LEARNING discovery to an oncologist, with the message recorded on the LEDGER, both identities verified, and the routing constrained by governance. The collaboration trail is complete and auditable — who sent what to whom, when, in what clinical context, through what governed channel.

11.12. 11.12 COIN Minting per Session

Every governed TALK session mints COIN. The minting formula:

```
COIN = f(session_turns, citation_count, evidence_quality)
```

Factor	Weight	Clinical Example
Session turns	1 COIN per turn	8-turn MammoChat session = 8 base COIN
Citation count	0.5 COIN per unique citation	4 unique BI-RADS citations = 2 bonus COIN
Evidence quality	Multiplier (1.0-2.0)	GOLD evidence (peer-reviewed) = 1.5x multiplier
Total	Weighted sum	(8 + 2) * 1.5 = 15 COIN minted

Every mint is recorded on the LEDGER as a MINT:WORK event with the session ID as the work reference. The key insight is that minting is not a reward for chatting — it is a reward for governed clinical work. A session with no citations mints fewer COIN than a session with evidence-rich responses. The economic incentive is structurally aligned with governance quality: the more evidence you cite, the more COIN you earn [12].

11.13. 11.13 Error Handling and Failure Modes

TALK handles six categories of failure:

1. Missing systemPrompt. If CANON.json does not contain a `systemPrompt` field (compilation failure, empty INTEL.md), the TALK service refuses to start the agent:

```
ERROR: SERVICES/TALK/MAMMOCHAT/CANON.json
  Missing field: systemPrompt
  Cause: INTEL.md is empty or build pipeline did not run.
  Fix: Populate INTEL.md and run: build --scope SERVICES/TALK/MAMMOCHAT
```

2. Disclaimer suppression. If the frontend template does not render the disclaimer (template bug, CSS override), the `magic validate` check fails:

```
ERROR: SERVICES/TALK/MAMMOCHAT — disclaimer not rendered
  MUST constraint violated: "Every TALK agent displays a disclaimer"
  Fix: Check _includes/talk-disclaimer.html template
```

3. LEDGER write failure. If the `POST /talk/ledger` endpoint is unreachable, the TALK session pauses and the turn is queued for retry. The session does not continue without LEDGER recording — ungoverned conversation turns are not permitted:

```
~/.canonic/services/talk/retry-queue/
  └── s-789abc-turn-08.json  ← queued LEDGER write
```

4. PHI boundary violation. If the response filter detects a pattern matching PHI (MRN format, SSN format, date-of-birth combined with name), the response is blocked and a DRIFT signal is recorded in LEARNING.

5. Out-of-scope response. If the LLM generates a response outside the agent's governed knowledge boundary (MammoChat answering a cardiology question), the response filter flags it for review with a governance notice directing the user to the appropriate agent.

6. Boilerplate detection. The TALK CANON enforces that every child scope declares domain-specific content — not template boilerplate. The validator detects generic axioms:

```
ERROR: SERVICES/TALK/MAMMOCHAT/CANON.md
  Axiom: "MAMMOCHAT is CANONIC" — too generic.
  MUST: Every child scope declares domain-specific Axiom.
```

11.14. 11.14 Clinical Vignette: MammoChat BI-RADS 4A Triage

A community hospital radiologist uses MammoChat during a breast imaging reading session. The patient's mammogram shows a 1.2cm irregular mass with indistinct margins. The radiologist opens a MammoChat session.

Turn 1 (Radiologist): "Irregular mass 1.2cm indistinct margins in upper outer quadrant. BI-RADS assessment guidance."

Turn 1 (MammoChat): "Based on ACR BI-RADS 5th Edition, an irregular mass with indistinct margins is suspicious. BI-RADS 4B (intermediate suspicion for malignancy, PPV 15-30%) is appropriate. Tissue sampling is recommended. [Source: ACR BI-RADS Atlas 5th ed., 2013; NCCN Breast Cancer Screening v2026.1, BINV-2]"

Session governance: The session is opened with session ID `s-mammochat-2026-03-15-001`. Every turn is LEDGER-recorded via `POST /talk/ledger`. The systemPrompt constrains responses to BI-RADS, ACR, and NCCN evidence. The disclaimer is displayed on every turn.

COIN minting: The 6-turn session produces 6 base COIN, 4 unique citations (ACR BI-RADS, NCCN v2026.1, NCCN BINV-2, ACR Practice Parameter), evidence quality multiplier 1.5 (peer-reviewed guidelines). Total: (6 + 2) * 1.5 = 12 COIN minted to the radiologist's WALLET.

LEARNING capture: The session pattern "irregular mass + indistinct margins BI-RADS 4B" is captured as a LEARNING record with signal SESSION_LEDGERED, as described in Chapter 10. The record propagates to the institutional LEARNING corpus, contributing to the community learning dashboard at mammo.chat. For a deeper look at how the MammoChat TALK page is constructed, see the MammoChat TALK.

11.15. 11.15 Clinical Vignette: OncoChat Pembrolizumab Eligibility

An oncologist at an academic medical center uses OncoChat to evaluate immunotherapy eligibility for a newly diagnosed stage IIIA non-small cell lung cancer (NSCLC) patient. The tumor is PD-L1 TPS 60%, EGFR/ALK/ROS1 wild-type.

Turn 1 (Oncologist): "Stage IIIA NSCLC, PD-L1 TPS 60%, no driver mutations. Pembrolizumab eligibility and regimen."

Turn 1 (OncoChat): "Per NCCN NSCLC v2026.2 (NSCL-31), for stage IIIA NSCLC with PD-L1 >= 50% and no EGFR/ALK/ROS1 alterations: pembrolizumab 200mg IV q3w monotherapy is Category 1 recommended as first-line systemic therapy. Alternative: pembrolizumab + pemetrexed 500mg/m2 + carboplatin AUC5 q3w x 4 cycles, then pembrolizumab + pemetrexed maintenance (KEYNOTE-189). [Source: NCCN NSCLC v2026.2 NSCL-31; KEYNOTE-024, N Engl J Med 2016;375:1823-33; KEYNOTE-189, N Engl J Med 2018;378:2078-92]"

Governance trail: The session references three evidence sources, all from OncoChat's governed INTEL. The drug names (pembrolizumab, pemetrexed, carboplatin), dosages (200mg, 500mg/m2, AUC5), and trial citations (KEYNOTE-024, KEYNOTE-189) are sourced from compiled INTEL — not hallucinated by the LLM. The session mints 18 COIN (high citation density, GOLD evidence quality).

11.16. 11.16 View Architecture

Every TALK scope presents three governed views, discovered from `controls.views` in CANON.json:

View	Surface	Content Source
`gov`	Governance contract	CANON.md compiled into CANON.json
`web`	Interactive TALK agent	systemPrompt + CHAT runtime
`tex`	PDF document	TALK.md compiled via LATEX pipeline

The default view is `gov` — the governance contract renders first. Before you interact with the agent, you see what governs it: its axiom, its constraints, its evidence sources. The governance is not hidden behind the conversational interface; it is presented ahead of it. You know the rules before the agent speaks.

The view toggle itself is discovered from CANON.json, not hardcoded in a template. The compiler emits the `controls.views` array; the theme renders toggle buttons for whatever views exist. Add a new view to the compiler and the theme renders it automatically, with no template change required [21].

11.17. 11.17 Community Learning Dashboard

Every first-class TALK service with a `.ai` domain surfaces a Community Learning Dashboard compiled from `TALK/{SERVICE}/LEARNING.md`:

```
## Community Learning Dashboard

| Question | Evidence | Source |
|----------|----------|--------|
| What does BI-RADS 4A mean? | Intermediate suspicion, PPV 2-10% | ACR BI-RADS 5th ed. |
| When is breast MRI indicated? | High-risk (>20% lifetime risk) | NCCN v2026.1 BINV-A |
| What is tomosynthesis sensitivity? | +1.2 cancers/1000 screens | Friedewald et al. 2014 |
```

Every question on this dashboard is real — extracted from LEARNING records produced by governed TALK sessions, never fabricated. A fabricated community learning question is a governance violation that degrades institutional trust. The dashboard serves a dual purpose: patient education (community-tier, free access) and governance transparency. The public can see what clinicians actually ask and what evidence the agent provides in response [21].

11.18. 11.18 Closure

This chapter opened with a claim: TALK is CHAT + INTEL composed. The architecture proves it. INTEL compiles into the systemPrompt. CHAT configures the clinical persona. Sessions are governed on the LEDGER. COIN mints for governance work. Disclaimers are enforced by the build pipeline. PHI boundaries are maintained by architectural separation. Cross-user messages route through governed channels. Community learning dashboards compile from real clinical interactions.

Remove the INTEL, and the agent speaks without evidence — an ungoverned chatbot. Remove the CHAT, and the evidence has no voice — a knowledge base with no interface. TALK composes both into something neither primitive achieves alone: a governed conversation engine where every word is backed by evidence and every session is a first-class governance event. In clinical healthcare, that composition is the difference between a liability and an asset. For the visual rendering of TALK agents, see Chapter 30 (The CHAT Layer). For the compilation pipeline that produces agents from INTEL, see Chapter 25 (Contextual Agents). For the SHOP economics of TALK products, see Chapter 12 (SHOP) and Chapter 34 (The SHOP).

Chapter 12

Chapter 12: SHOP

COIN + INTEL. Public economic projection. The marketplace that compiles itself [15].

12.1. 12.1 Axiom

SHOP compiles the public aggregate. Composable. Deterministic. Drift-gated [15].

Most marketplaces are databases with a frontend bolted on. SHOP is the opposite: the governance tree itself is the product catalog, and the marketplace is a compiled projection of it. Every governed scope that wants to be visible creates a SHOP.md file. The SHOP service walks the tree, discovers every SHOP.md, and compiles them into a unified catalog. No manual curation. No product database. No CMS [17].

The pattern makes governed clinical AI products discoverable by construction. Build MammoChat to 255, create a SHOP.md in its scope, and the product appears in the marketplace. The governance score is the listing qualification — you cannot game your way into the catalog without doing the governance work.

12.2. 12.2 Discovery Architecture

SHOP discovers products by walking the filesystem. No database. No API. No registration form. The walk is deterministic — the same governance tree always produces the same product catalog:

```
magic scan → find SHOP.md files → parse Cards → compile catalog → drift gate → publish
```

Step 1: Walk. The SHOP compiler walks every directory in the governance tree looking for SHOP.md files. Every SHOP.md found is a product candidate.

Step 2: Parse. Each SHOP.md contains a Card — a structured declaration of the product's metadata (title, type, price, status, synopsis, route). The compiler parses each Card into a product record.

Step 3: Validate. The compiler checks that each product's parent scope validates to 255. A SHOP.md in a scope that scores below 255 is ignored — ungoverned products are not listed. This is the quality gate. You cannot game the marketplace by creating SHOP.md files in empty scopes [15].

Step 4: Compile. All valid product records are compiled into a single catalog artifact — `_data/shop.json` — that the frontend reads to render the marketplace.

Step 5: Drift gate. Before publishing, the compiler regenerates the catalog and compares it to the existing published version. If the regenerated catalog differs from the published version (drift detected), the publish is blocked until the drift is resolved. This ensures that the published marketplace always matches the current governance tree [15].

12.3. 12.3 SHOP.md Cards

Every product is declared as a Card in SHOP.md:

```
## Card

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

Card fields are governed:

Field	Required	Constraint	Clinical Example
title	YES	Display name, max 80 chars	"MammoChat Enterprise"
type	YES	Enum: BOOK, PAPER, SERVICE, BLOG, API, TEMPLATE	SERVICE
price	YES	Integer COIN, tier-aligned	127 COIN (AGENT tier)
status	YES	Enum: AVAILABLE, COMING_SOON, ARCHIVED	AVAILABLE

Field	Required	Constraint	Clinical Example
synopsis	YES	1-2 sentences, no marketing fluff	“Governed breast imaging AI with BI-RADS INTEL”
route	YES	Path to governed scope	/SERVICES/TALK/MAMMOCHAT/
cost_basis	NO	Derived from LEDGER	2,147 COIN (governance work invested)

Every Card field traces to a governed source. The synopsis comes from the scope’s axiom, the price from tier alignment, the route from the governance tree path. Marketing does not write Cards — governance compiles them [17].

12.4. 12.4 Pricing by Tier

Product pricing aligns with governance tiers. The tier determines the audience. The audience determines the price:

Tier	COIN Range	Target Audience	Clinical Example
COMMUNITY	0-35	Everyone — free or near-free	MammoChat screening Q&A
BUSINESS	36-63	Small teams, departments	OncoChat single-department license
ENTERPRISE	64-127	Organizations, health systems	MedChat multi-hospital deployment
AGENT	128-254	Developers, integrators, governors	CANONIC-DOCTRINE (dev manual)
FULL	255	General public, flagship products	CANONIC-CANON (governor’s manual)

The alignment is not arbitrary. Governance investment scales with deployment complexity. A patient-facing screening chatbot (COMMUNITY, free) demands far less governance depth than a clinical decision support tool for radiologists (ENTERPRISE, 127 COIN) or a full institutional deployment with HIPAA audit trails and LEDGER integration (FULL, 255 COIN). The tier encodes that reality [12].

12.5. 12.5 The Composable Frontmatter Pattern

SHOP.md is not the only way to list a product. For scopes that need inline product visibility, the `shop:` frontmatter key enables composable listing:

```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

shop: true

• • •

When `shop: true` is present in a scope's frontmatter, the SHOP compiler includes that scope in discovery

Set `shop: true` on each clinical AI service scope in your SERVICES directory and the SHOP compiler discov
no separate SHOP.md per service. Same convention as `talk:` frontmatter: composable, filesystem-discoverab

12.6 Checkout Flow

The SHOP checkout is a governed economic transaction — every step is a LEDGER event:

1. Reader selects product SHOP:VIEW event
2. System checks WALLET balance SHOP:BALANCE_CHECK
3. Reader confirms purchase SPEND event (reader debited, author credited)
4. Both sides hash-chained LEDGER integrity preserved
5. Product access granted SHOP:ACCESS_GRANTED
6. Attestation minted CONTRIBUTE:ATTESTATION (reader attests to product quality)

There is no payment gateway here. The SPEND event debits the reader's WALLET and credits the author's —
both hash-chained on the LEDGER. No external payment processor, no credit card, no bank account. When a ho

12.7 Drift Gate

The drift gate is SHOP's integrity mechanism. Before any catalog publish, the compiler regenerates the ent

regenerated = compile(governance_tree) published = read(current_catalog)

if regenerated != published: BLOCK PUBLISH LOG: drift detected at {scope} — {diff} REQUIRE: resolve drift before publish

The drift gate prevents three failure modes:

1. **Stale catalog**: A product scope was updated but the catalog was not regenerated. The drift gate catc
2. **Orphaned listing**: A product scope was deleted but the catalog still lists it. The drift gate catche
3. **Score regression**: A product scope dropped below 255 but the catalog still lists it as available. Th

The catalog never advertises a clinical AI tool that has fallen out of governance compliance. If OncoChat

12.8 SHOP and the Hospital Procurement Office

Traditional SaaS marketplaces ask you to trust the vendor's compliance claims. SHOP eliminates that depend
not self-reported. The SHOP listing is the vendor assessment.

A procurement officer reviewing a SHOP Card sees:
- **Title and type**: What the product is
- **Price in COIN**: What the governance investment costs
- **Cost basis**: How much governance work produced the product (transparency)
- **Route**: Where to inspect the product's governance tree
- **Status**: Whether the product is currently available and at 255

Every field derives from governance, not from vendor marketing materials. In a governed marketplace, the c

12.9 CLI Operations

SHOP is compiled, not administered. The CLI commands:

```bash
# Discover all SHOP.md files in the governance tree
magic scan --shop

# Compile the product catalog from discovered SHOP.md files
build --shop

# Validate SHOP.md Card fields against constraints
magic validate --scope SERVICES/SHOP

# Check drift between compiled catalog and published catalog
magic validate --shop --drift

# Regenerate catalog and publish (blocked if drift detected)
build --shop --publish

# List all products with governance scores
magic scan --shop --scores

# Export catalog as JSON for external integration
magic scan --shop --export json > shop-catalog.json
```

The `build --shop` command executes the five-phase pipeline described in Section 12.2. The pipeline is idempotent — running it twice on the same governance tree produces identical catalog output. The idempotency is critical for the drift gate: if the pipeline were non-deterministic, the drift gate would produce false positives on every run.

The `magic validate --shop --drift` command is the pre-publish check. Run it before `build --shop --publish`. If drift is detected, the command outputs a diff showing exactly which products changed, were added, or were removed:

```
DRIFT DETECTED in _data/shop.json:
  + ADDED: SERVICES/TALK/CARIBCHAT (new SHOP.md discovered)
  ~ CHANGED: SERVICES/TALK/MAMMOCHAT (price: 127 → 255 COIN)
  - REMOVED: SERVICES/TALK/LEGACY-BOT (scope deleted)

  Resolve drift before publish.
  Run: build --shop --publish --force to accept changes.
```

12.6. 12.10 The Catalog Data Structure

The compiled catalog is a JSON artifact at `_data/shop.json`. The structure:


```json
{
  "_generated": true,
  "compiled_at": "2026-03-10T14:00:00Z",
  "compiler": "build --shop v3.1",
  "products": [
    {
      "title": "MammoChat Enterprise",
      "type": "SERVICE",
      "price": 255,
      "status": "AVAILABLE",
      "synopsis": "Governed breast imaging AI with BI-RADS INTEL",
      "route": "/SERVICES/TALK/MAMMOCHAT/",
      "cost_basis": 2147,
      "governance_score": 255,
      "tier": "FULL",
      "last_validated": "2026-03-10T13:45:00Z",
      "card_hash": "a1b2c3d4e5f6..."
    },
    {
      "title": "OncoChat Enterprise",
      "type": "SERVICE",
      "price": 127,
```

```
      "status": "AVAILABLE",
      "synopsis": "Pan-cancer staging intelligence — 10 cancer types, 12 biomarkers",
      "route": "/SERVICES/TALK/ONCOCHAT/",
      "cost_basis": 1893,
      "governance_score": 255,
      "tier": "AGENT",
      "last_validated": "2026-03-10T13:45:00Z",
      "card_hash": "f7890abcdef1..."
    }
  ],
  "catalog_hash": "deadbeef01234567..."
}
```

The `_generated: true` flag marks this file as compiler output. Do not hand-edit. The `card_hash` per product is the SHA-256 of the product's Card fields — used by the drift gate to detect per-product changes. The `catalog_hash` is the SHA-256 of the entire products array — used for whole-catalog drift detection [15].

12.7. 12.11 Bag Review Pattern

The SHOP CANON requires bag review before checkout: `MUST: Bag review before checkout`. The bag is the user's selection of products before the economic transaction executes. The bag review step serves three governance purposes:

1. Confirmation gate. The user sees exactly what they are purchasing, at what price, before COIN is debited. No surprise charges. No hidden fees. The bag displays: product title, price in COIN, current governance score, and the user's WALLET balance.

2. Balance check. The SHOP verifies that the user's WALLET has sufficient COIN balance before proceeding. If the balance is insufficient, the checkout is blocked with a specific message:

```
INSUFFICIENT BALANCE
  Product: MammoChat Enterprise — 255 COIN
  Your balance: 142 COIN
  Shortfall: 113 COIN

  Earn COIN through governed work:
    - TALK sessions mint COIN per turn
    - CONTRIBUTE submissions mint COIN per acceptance
    - Governance work mints COIN per gradient advancement
```

3. One CTA per card. Each product card in the bag has exactly one call-to-action button. The constraint `MUST: One CTA per product card` prevents UI patterns that confuse users with multiple action paths. The CTA is "Purchase" for AVAILABLE products and "Join Waitlist" for COMING_SOON products.

12.8. 12.12 Error Handling and Failure Modes

SHOP handles four categories of failure:

1. Invalid Card fields. A SHOP.md with missing required fields or invalid enum values triggers a validation error:

```
ERROR: SERVICES/TALK/MAMMOCHAT/SHOP.md
  Line 7: Field "status" value "LIVE" not in enum.
  Valid values: AVAILABLE, COMING_SOON, ARCHIVED
  Run: magic heal --scope SERVICES/TALK/MAMMOCHAT
```

2. Ungoverned scope. A SHOP.md in a scope that does not validate to 255 is excluded from the catalog with a warning:

```
WARN: SERVICES/TALK/LEGACY-BOT/SHOP.md excluded.
  Governance score: 191/255 (missing: LEARNING, EVIDENCE)
  Products require 255 governance score for SHOP listing.
  Fix: Run magic validate --scope SERVICES/TALK/LEGACY-BOT
```

3. Inline CSS/JS violation. The SHOP CANON prohibits inline CSS and JS in compiled pages: `MUST NOT: Inline CSS or JS in compiled pages.` The validator scans compiled SHOP pages for `<style>` and `<script>` tags:

```
ERROR: SHOP compiled page contains inline CSS.
  File: _site/SHOP/index.html, Line: 47
  MUST NOT: Inline CSS or JS in compiled pages.
  Fix: Move styles to _TOKENS.scss via DESIGN.css authority.
```

4. Admin tools on public surface. The constraint `MUST NOT: Admin tools on public SHOP surface` prevents administrative functions (catalog regeneration, drift resolution, score override) from being exposed on the public-facing SHOP. The validator checks that no admin endpoints are referenced in compiled SHOP templates.

12.9. 12.13 Clinical Vignette: Hospital Network Procurement

A five-hospital health system evaluates clinical AI products for system-wide deployment. The chief medical informatics officer (CMIO) opens the CANONIC SHOP catalog.

Discovery. The CMIO browses the SHOP. The catalog shows three clinical AI products at AVAILABLE status:

Product	Type	Price	Score	Cost Basis
MammoChat Enterprise	SERVICE	255 COIN	255/255	2,147 COIN
OncoChat Enterprise	SERVICE	127 COIN	255/255	1,893 COIN
MedChat Enterprise	SERVICE	127 COIN	255/255	1,654 COIN

Due diligence. The CMIO clicks the route link for MammoChat Enterprise. The link opens the MammoChat governance tree — the CANON.md, INTEL.md, LEARNING.md, and full validation report are visible. The CMIO verifies: evidence sources (NCT06604078, BI-RADS 5th ed., NCCN v2026.1), governance constraints (HIPAA compliance, PHI boundary, disclaimer enforcement), and LEARNING history (23 EVOLUTION events, 0 unresolved DRIFT events). The due diligence is self-service — no vendor presentation required. The governance tree IS the vendor assessment.

Procurement. The CMIO adds MammoChat Enterprise (255 COIN) and OncoChat Enterprise (127 COIN) to the bag. The bag review shows: total 382 COIN, institutional WALLET balance 500 COIN, remaining after purchase 118 COIN. The CMIO confirms. The checkout executes:

```
SPEND: 255 COIN — MammoChat Enterprise
  Buyer: memorial-health-system (WALLET: 500 → 245 COIN)
  Seller: hadleylab (WALLET: +255 COIN, 5% fee: -12.75 COIN)
  LEDGER: event-a1b2c3d4 (hash-chained)

SPEND: 127 COIN — OncoChat Enterprise
  Buyer: memorial-health-system (WALLET: 245 → 118 COIN)
  Seller: hadleylab (WALLET: +127 COIN, 5% fee: -6.35 COIN)
  LEDGER: event-e5f6g7h8 (hash-chained)
```

Both transactions are recorded on the LEDGER. The procurement audit trail is complete: who purchased, when, at what price, from what scope, at what governance score. The hospital's procurement office has a governed record of every clinical AI acquisition — no purchase orders lost, no vendor contracts unsigned, no compliance assessments undocumented.

12.10. 12.14 Clinical Vignette: Score Regression and Drift Gate

OncoChat deploys an updated language model without running `magic validate`. The model change causes OncoChat's governance score to drop from 255 to 191 — the EVIDENCE dimension fails because the new model's citation patterns differ from the validated INTEL.

Drift gate trigger. The next `build --shop` run detects the score regression:

```
DRIFT DETECTED:
  SERVICES/TALK/ONCOCHAT — score regression: 255 → 191
```

```
OncoChat Enterprise listing BLOCKED until score restored.

Missing questions: "Can you prove it?", "What have you learned?"
Fix: Run magic validate --scope SERVICES/TALK/ONCOCHAT
     Review model update against INTEL.md evidence sources.
     Run magic heal if validation identifies auto-fixable issues.
```

The SHOP catalog is not published. OncoChat Enterprise remains listed at its last-valid state (255/255) until the drift is resolved. No customer sees a degraded product listing. No hospital procurement office purchases a clinical AI tool that has fallen out of governance compliance.

Resolution. The clinical informatics team reviews the model update, updates INTEL.md to account for the new model's citation format, runs `magic validate` to restore the 255 score, and then `build --shop --publish` succeeds. The drift gate has protected the marketplace integrity. A LEARNING record captures the event: signal DRIFT, pattern "Model update without validation caused EVIDENCE dimension failure," gradient -64 (from 255 to 191, then restored to 255).

12.11. 12.15 Per-Page Wallet Constraint

The SHOP CANON contains a specific anti-pattern constraint: `MUST NOT: Per-page wallet reimplementation`. This constraint prevents each SHOP product page from implementing its own WALLET balance display, its own checkout flow, or its own COIN handling logic. The WALLET is a single service. The SHOP reads from it. There is one WALLET implementation, and every SHOP page uses it through the same API.

The constraint exists because in early development, individual product pages implemented their own COIN balance checks — leading to inconsistencies where different pages showed different balances for the same user. The constraint eliminates the class of bug entirely: one WALLET, one balance, one truth [15].

12.12. 12.16 Product Type Taxonomy

The SHOP Card `type` field uses a governed enum. Each type maps to a distinct compilation path and display template:

Type	Compilation Source	Display Template	Clinical Example
BOOK	BOOKS/{title}/CANON.md	book-card.html	CANONIC-DOCTRINE
PAPER	PAPERS/{title}/CANON.md	paper-card.html	MammoChat clinical trial manuscript
SERVICE	SERVICES/{name}/CANON.md	service-card.html	MammoChat Enterprise
BLOG	BLOGS/{slug}/CANON.md	blog-card.html	"Why Governance Before Code"
API	SERVICES/API/{name}/CANON.md	api-card.html	TALK REST API

Type	Compilation Source	Display Template	Clinical Example
TEMPLATE	TEMPLATES/{name}/CANON.md	template-card.html	Clinical AI governance starter kit

Each type resolves governance differently. A SERVICE resolves from the full governance tree (CANON.md + INTEL.md + LEARNING.md + all child scopes). A BOOK resolves from its scope only. A PAPER resolves from the paper scope and cross-references the trial registry (NCT ID). Resolution logic varies by type, but the output format is uniform — every product compiles to a JSON object with the same field schema [15].

12.13. 12.17 Closure

SHOP compiles the public aggregate. Products discovered by filesystem walk, Cards parsed from SHOP.md, scores validated to 255, catalogs compiled deterministically, drift gates blocking stale publishes, check-outs LEDGER-recorded, bag reviews enforcing confirmation gates, procurement audit trails baked into the transaction architecture. SHOP is not a marketplace that happens to have governance — it is governance that happens to produce a marketplace. For clinical AI procurement, that distinction separates vendor trust from vendor assessment. The live SHOP is at shop.hadleylab.org. For the broader COIN economics that drive the SHOP, see Chapter 32 (COIN and the WALLET) and Chapter 35 (COSTBASIS and Pricing). For product-level construction, see Chapter 8 (Building a Product) and Chapter 34 (The SHOP) [17].

Chapter 13

Chapter 13: LEDGER

COIN. Append-only economic truth. The audit trail that IS the compliance [12]. The LEDGER is one of the 14 services described in Chapter 7, composing COIN with INTEL to create an immutable economic record. Its hash chain integrity layer is covered in Chapter 17; its per-user projection is the WALLET (Chapter 14).

13.1. 13.1 Properties

Every compliance audit begins with the same question: "Show me the trail." The LEDGER is an immutable, append-only log that answers it permanently. It tracks not transactions but provenance — who did what, when, with what evidence, under what governance, and why it mattered. Every COIN minted, every COIN spent, every drift event, every reconciliation lives here [22].

HIPAA §164.312(b) requires 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 becomes a LEDGER event — timestamped, attributed, hash-linked, append-only. Governance and audit are not separate systems. They are architecturally unified.

13.2. 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

Event	Direction	Description	Healthcare Example
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 events, exhaustive. No ninth type exists. The economy is closed by enumeration — when a HIPAA auditor asks "what types of events does this system record?" the answer is this table, and no others [12].

13.3. 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/BOOKS/CANONIC-DOCTRINE",
  "signature": "ed25519hex..."
}
```

Each event hash-links to its predecessor via the `prev` field and carries an Ed25519 signature after the cutoff date. Change a single character in any event and every subsequent hash breaks. The LEDGER is append-only by design and tamper-evident by cryptography [12].

13.4. 13.4 The LEDGER as Healthcare Audit Trail

One LEDGER serves simultaneously as four compliance instruments:

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.

Four compliance standards, one governed event log. The audit trail maps to every standard through the compliance matrix — no separate system per regulation.

13.5. 13.5 Querying the LEDGER

The LEDGER is queryable. Extract governance analytics 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 governance velocity, drift rates, compliance trends, and economic return on governance investment — metrics derived from governed events, not surveys or self-assessments.

13.6. 13.6 CLI Operations

Query and audit the LEDGER through CLI commands:

```
# View the last 20 LEDGER events
magic ledger --tail 20

# View all events for a specific user
magic ledger --user dexter

# View all events for a specific scope
magic ledger --scope SERVICES/TALK/MAMMOCHAT

# View all events of a specific type
magic ledger --event MINT:WORK

# Reconcile the current month
magic ledger --reconcile

# Verify hash chain integrity
magic ledger --verify

# Export LEDGER for external audit
magic ledger --export json > ledger-audit-2026-Q1.json

# Compute governance velocity for a date range
magic ledger --velocity --from 2026-01-01 --to 2026-03-31
```

The `magic ledger --verify` command is the chain integrity check. It reads every event in the LEDGER, recomputes the `prev` hash for each event, and verifies that every hash matches. If any event has been modified — even a single character — the verification fails with a specific error:

```
CHAIN INTEGRITY FAILURE
  Event: a1b2c3d4e5f6
  Expected prev: z9y8x7w6v5u4
  Actual prev:   z9y8x7w6v5u3
  Position: event #4,721 of 12,847

  The LEDGER has been tampered with.
  Restore from backup: magic ledger --restore --from backup-2026-03-09.json
```

Verification is O(n) — it reads every event. For 100,000 events, verification takes seconds; for 10 million events at hospital-network scale, minutes. The linear cost is acceptable because verification is an audit operation, not a runtime operation.

13.7. 13.7 Storage Architecture

The LEDGER is stored as an append-only JSON lines file with periodic snapshots:

```
~/.canonic/services/ledger/
  ├── ledger.jsonl              ← append-only event log (one JSON per line)
  ├── snapshots/
  │   ├── 2026-01.json          ← monthly snapshot (reconciled)
  │   ├── 2026-02.json
  │   └── 2026-03.json          ← current month in progress
  ├── index/
  │   ├── by-user.json          ← user → event ID list
  │   ├── by-scope.json         ← scope → event ID list
  │   ├── by-event.json         ← event type → event ID list
  │   └── by-date.json          ← date → event ID list
  └── signatures/
      ├── a1b2c3d4.sig          ← Ed25519 signature per event (post-cutoff)
      └── ...
```

JSON lines format. One event per line. New events append to the end. The file is never rewritten, never truncated, never modified in place — the service opens it in `O_APPEND` mode and never calls seek or truncate [12].

Monthly snapshots. At each CLOSE event (monthly reconciliation), the LEDGER service writes a snapshot of the current month's events. The snapshot is a complete, self-contained record of that month's economic activity. Snapshots serve three purposes: (1) fast queries scoped to a single month, (2) backup and restore operations, (3) regulatory audit submissions requiring monthly granularity.

Indexes. Four thin indexes provide O(1) lookup by user, scope, event type, and date. The indexes are derived artifacts — they can be regenerated from `ledger.jsonl` at any time. The `_generated` principle applies: if an index is corrupted, regenerate it from the source, do not repair it.

Ed25519 signatures. After the signing cutoff (configured per deployment), every new event is signed with Ed25519. The signature covers the event JSON (excluding the signature field itself) and provides cryptographic non-repudiation — a signed event cannot be denied by its creator [12].

13.8. 13.8 The CLOSE Event and Monthly Reconciliation

The CLOSE event is the monthly reconciliation. It computes the month's economic summary and freezes the month's LEDGER:

```
{
  "id": "close-2026-02",
  "prev": "last-event-hash",
  "ts": "2026-02-28T23:59:59Z",
  "event": "CLOSE",
  "user": "system",
  "amount": 0,
  "delta": 0,
  "summary": {
    "total_minted": 4721,
    "total_debited": 312,
    "total_transferred": 1847,
    "total_spent": 923,
    "total_settled": 0,
    "net_velocity": 4409,
    "active_users": 47,
    "active_scopes": 73,
    "events_recorded": 1284
  },
  "signature": "ed25519hex..."
}
```

After the CLOSE event is written, the month's snapshot is frozen. No new events can be added to a closed month. Corrections go forward as new events in the next month — never backward. Standard double-entry bookkeeping [12].

The CLOSE event also produces the monthly governance report that administrators need: governance work done (total_minted), drift incurred (total_debited), active scopes, and net governance velocity. Computed from governed events — not estimated from activity logs.

13.9. 13.9 The TRANSFER Event and Fee Structure

The TRANSFER event moves COIN between users. Every transfer incurs a 5% fee:

```
{
  "id": "transfer-789abc",
  "prev": "previous-hash",
  "ts": "2026-03-05T10:15:00Z",
  "event": "TRANSFER",
  "user": "memorial-health-system",
  "amount": 100,
  "delta": -100,
  "fee": 5,
```

```
  "recipient": "hadleylab",
  "recipient_delta": 95,
  "work_ref": "SERVICES/TALK/MAMMOCHAT/LICENSE",
  "signature": "ed25519hex..."
}
```

The 5% fee is the governance tax — it funds the validators, the compilers, the LEDGER service itself. Not negotiable. The TRANSFER event type includes a `fee` field that is always 5% of the `amount`, debited before crediting the recipient [12].

Transfer Amount	Fee (5%)	Recipient Receives	Clinical Context
100 COIN	5 COIN	95 COIN	Department license transfer
255 COIN	12.75 COIN	242.25 COIN	MammoChat Enterprise purchase
500 COIN	25 COIN	475 COIN	Institutional onboarding bonus
1,000 COIN	50 COIN	950 COIN	Multi-service enterprise deal

13.10. 13.10 The DEBIT:DRIFT Event

DEBIT:DRIFT is the governance penalty. When a scope's governance score regresses — from 255 to 191, from FULL to ENTERPRISE, from compliant to non-compliant — the LEDGER records a DEBIT:DRIFT event that debits COIN from the scope maintainer's WALLET:

```
{
  "id": "drift-456def",
  "prev": "previous-hash",
  "ts": "2026-03-07T08:30:00Z",
  "event": "DEBIT:DRIFT",
  "user": "dr-chen",
  "amount": 64,
  "delta": -64,
  "work_ref": "SERVICES/TALK/ONCOCHAT",
  "drift_from": 255,
  "drift_to": 191,
  "dimensions_lost": ["E", "L"],
  "cause": "Model update without validation",
  "signature": "ed25519hex..."
}
```

The debit amount equals the score regression: 255 - 191 = 64 COIN. Proportional to the damage — a small regression (255 to 243, -12 COIN) penalizes less than a catastrophic one (255 to 0, -255 COIN). Regression costs COIN. Advancement earns COIN. The LEDGER tracks both [12].

DEBIT:DRIFT captures governance negligence with full context: which dimensions were lost, what caused the drift, and how much COIN was debited. Query the LEDGER for all DEBIT:DRIFT events across the clinical AI fleet and you can identify which scopes are most prone to drift — a governance quality metric that no other system provides.

13.11. 13.11 Error Handling and Failure Modes

The LEDGER handles four categories of failure:

1. Hash chain corruption. If the `magic ledger --verify` command detects a broken hash chain, the LEDGER is flagged as compromised. All LEDGER operations pause until the corruption is resolved:

```
CRITICAL: LEDGER hash chain broken at event #4,721.
  All LEDGER writes suspended.
  Restore from last verified snapshot:
    magic ledger --restore --from snapshots/2026-02.json
  Replay events after restoration.
```

2. Concurrent write conflict. If two services attempt to append to the LEDGER simultaneously (TALK session and SHOP checkout), the append-mode file semantics handle the concurrency — `O_APPEND` is atomic on POSIX systems for writes under the pipe buffer size. For LEDGER events (typically under 1KB), the atomicity guarantee holds. For deployments that exceed this limit, the LEDGER service serializes writes through a queue.

3. Signature verification failure. If an event's Ed25519 signature does not verify against the event's JSON content, the event is flagged as potentially tampered:

```
SIGNATURE FAILURE: event a1b2c3d4
  Signature does not match event content.
  Possible causes: event modified after signing,
    key rotation without re-signing.
  Action: Flag for manual audit.
```

4. CLOSE reconciliation mismatch. If the CLOSE event's summary totals do not match the sum of the month's individual events, the reconciliation fails:

```
RECONCILIATION FAILURE: 2026-02 CLOSE
  Expected total_minted: 4,721 COIN
  Computed total_minted: 4,709 COIN
  Discrepancy: 12 COIN (3 events)
  Fix: Investigate discrepant events before closing.
```

13.12. 13.12 Clinical Vignette: HIPAA Audit Response

A hospital undergoes a HIPAA compliance audit. The auditor requests evidence of activity monitoring for all clinical AI systems (§164.312(b)). The team responds with LEDGER data.

Audit request 1: "Show all clinical AI interactions for Q1 2026."

```
magic ledger --event MINT:WORK --scope SERVICES/TALK \
  --from 2026-01-01 --to 2026-03-31
```

Result: 4,721 governed sessions across five TALK agents. Each timestamped, attributed to a verified user, and hash-chained.

Audit request 2: "Show evidence of access controls."

```
magic ledger --event TRANSFER --scope SERVICES/TALK \
  --from 2026-01-01 --to 2026-03-31
```

Result: 47 TRANSFER events showing COIN-gated access to clinical AI services. Each transfer shows the buyer, seller, amount, fee, and scope.

Audit request 3: "Show evidence of integrity controls."

```
magic ledger --verify --from 2026-01-01 --to 2026-03-31
```

Result: 12,847 events verified. Hash chain intact. Zero integrity failures. Ed25519 signatures verified for all events after signing cutoff. The auditor has cryptographic proof that the LEDGER has not been tampered with.

The hospital passes. Three categories of evidence — activity monitoring, access controls, integrity controls — from a single governed data source. No separate audit log system. No manual evidence collection [2].

13.13. 13.13 Clinical Vignette: Governance Velocity Dashboard

A health system's chief quality officer requests a governance velocity dashboard for the clinical AI program:

```
magic ledger --velocity --from 2026-01-01 --to 2026-03-31 --by-scope
```

Result:

Scope	Minted	Drifted	Velocity	Trend
MammoChat	1,247	0	+1,247	Stable

Scope	Minted	Drifted	Velocity	Trend
OncoChat	893	64	+829	Recovered (drift 03-07)
MedChat	654	0	+654	Stable
LawChat	412	12	+400	Minor drift (03-15)
FinChat	1,515	0	+1,515	Strong growth
Total	**4,721**	**76**	**+4,645**	**Healthy**

Positive governance velocity: 4,721 COIN minted versus 76 drifted, a 1.6% drift rate. OncoChat's March 7 drift event (model update without validation) resolved within 24 hours. None of this is a report someone wrote — it is a computation over governed LEDGER events, exact to the COIN [12].

13.14. 13.14 The SETTLE Event and Fiat Exit

The SETTLE event converts COIN to monetary value — the fiat exit from the governed economy:

```
{
  "id": "settle-789abc",
  "prev": "previous-hash",
  "ts": "2026-03-10T16:00:00Z",
  "event": "SETTLE",
  "user": "hadleylab",
  "amount": 1000,
  "delta": -1000,
  "fiat_amount": "\$1,000.00",
  "fiat_currency": "USD",
  "settlement_method": "ACH",
  "work_ref": "SETTLE/2026-03-Q1",
  "signature": "ed25519hex..."
}
```

SETTLE is the only event that crosses the COIN-to-fiat boundary. All other events operate entirely within the COIN economy. SETTLE debits COIN from the user's WALLET and triggers a fiat payment through the configured settlement method. COIN = WORK, and SETTLE converts WORK to USD [12].

13.15. 13.15 LEDGER Capacity Planning

For health systems scaling from pilot to enterprise deployment, LEDGER capacity grows linearly with governance activity:

Deployment Scale	Events/Month	Storage/Month	Verification Time
Single department (10 scopes)	~200	~500 KB	< 1 second
Hospital (50 scopes)	~2,000	~5 MB	< 5 seconds
Health system (200 scopes)	~10,000	~25 MB	< 30 seconds
Multi-site network (1,000 scopes)	~50,000	~125 MB	< 3 minutes
Enterprise federation (5,000 scopes)	~250,000	~625 MB	< 15 minutes

Storage is bounded — events are typically 200-500 bytes each. Monthly snapshots keep query performance constant regardless of total LEDGER size: single-month queries hit the snapshot, multi-month queries merge them.

```
# Check current LEDGER capacity metrics
magic ledger --capacity
# Events: 47,231
# Size: 22.4 MB
# Oldest event: 2025-11-01T00:00:00Z
# Newest event: 2026-03-10T16:45:00Z
# Snapshots: 5 (monthly)
# Indexes: 4 (by-user, by-scope, by-event, by-date)
# Verification estimate: 12 seconds
```

For regulated environments requiring long-term retention (HIPAA: 6 years, SOX: 7 years), snapshots are immutable files — archive them to cold storage. The LEDGER never forgets. The retention requirement is architecturally satisfied.

13.16. 13.16 Closure

Eight exhaustive event types. Hash-chained for tamper evidence (see Chapter 17). Ed25519-signed for non-repudiation. Monthly-reconciled via CLOSE. Queryable for governance analytics. Verifiable for audit compliance. Four regulatory standards (HIPAA, FDA 21 CFR Part 11, SOX, Joint Commission) from a single data source. The WALLET (Chapter 14) projects per-user views of this chain. The VAULT (Chapter 15) gates private access. MONITORING (Chapter 22) surfaces LEDGER metrics in real time. Remove the LEDGER and the governed economy has no memory. Remove the memory and governance is fiction [12].

Chapter 14

Chapter 14: WALLET

COIN. Per-USER economic identity. The account that computes itself [12]. The WALLET is the per-principal projection of the LEDGER (Chapter 13), secured by the CHAIN integrity layer (Chapter 17) and gated by VAULT auth (Chapter 15).

14.1. 14.1 Axiom

WALLET stores COIN for every USER. Every event signed. Balance derived, never stored [12].

A traditional account stores a balance and hopes it stays consistent. The WALLET stores an event chain — an append-only sequence of credits and debits — and derives the balance by walking it: `balance = SUM(credits) - SUM(debits)`. If the balance is always computed, it can never be stale, corrupted, or inconsistent with the underlying events. An entire class of financial accounting bugs vanishes [12].

The architecture maps directly to healthcare audit requirements. HIPAA §164.312(b) requires tamper-evident audit logs. The WALLET's append-only, hash-linked chain satisfies this by construction. Walk the chain to verify a clinician's governance activity — every MINT:WORK event, every SPEND, every DEBIT:DRIFT is there, signed and verifiable.

14.2. 14.2 Append-Only Event Chain

No mutable balance field. No stored state that can drift from reality. The WALLET is its chain:

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

Every event in the chain is hash-linked to its predecessor. The chain is a directed acyclic graph from the user's first event (typically MINT:SIGNUP) to the current head. Walking the chain backward from head to

genesis verifies integrity: if any event has been tampered with, the hash of every subsequent event breaks [12].

```
{
  "id": "wallet:dexter:evt:00047",
  "prev": "wallet:dexter:evt:00046",
  "ts": "2026-03-10T14:30:00Z",
  "type": "MINT:WORK",
  "amount": 92,
  "scope": "hadleylab-canonic/BOOKS/CANONIC-DOCTRINE",
  "signature": "ed25519:a1b2c3d4..."
}
```

The `prev` field is the critical integrity mechanism. It links each event to its predecessor by hash. The `signature` field provides non-repudiation — the user's Ed25519 private key signs each event after the cutoff date. Together, `prev` and `signature` make the WALLET chain both tamper-evident and attributable [12].

14.3. 14.3 The 7 Invariants

The WALLET enforces seven invariants that guarantee balance consistency, non-negativity, and conservation across the system. For the formal specification of these invariants, including the conservation equation and proof obligations, see Chapter 32, Section 32.5 (WALLET Invariants). From the user's perspective, the invariants mean that `vault verify-wallet` detects any violation and reports it, and that the WALLET is a financial instrument with built-in controls satisfying SOX §404 requirements for internal control over financial reporting [12].

14.4. 14.4 Dual-Write Architecture

Every COIN event writes to two targets simultaneously: the global LEDGER and the per-principal WALLET timeline. The WALLET implements this dual-write architecture to ensure consistency between the LEDGER and the WALLET balance; for the complete dual-write specification and failure-mode analysis, see Chapter 32, Section 32.6 (Dual-Write Protocol). For the user, the practical consequence is that a compliance team can audit governance activity from three independent sources: the individual clinician's WALLET, the hospital's organizational timeline, and the global LEDGER. Cross-referencing these three sources provides the assurance level that healthcare regulators require [12].

14.5. 14.5 Monthly CLOSE

The CLOSE operation runs monthly and reconciles every WALLET against the LEDGER, verifying that the sum of all WALLET balances equals total circulation. For the complete CLOSE specification, conservation equation, and reconciliation algorithm, see Chapter 32, Sections 32.4 (CLOSE Protocol) and 32.8 (Conservation Proof). From the user's perspective, CLOSE produces a per-principal verification report:

```
$ vault close --month 2026-03
Processing 47 wallets...
  dexter: balance=12,847 verified ✓
  isabella: balance=3,420 verified ✓
  ...
Circulation: 187,340 COIN
Unreconciled: 0
Mismatches: 0
CLOSE event appended to all timelines.
```

The CLOSE event is itself a LEDGER event, so if a regulator asks "when was the last reconciliation?" the answer is the timestamp of the last CLOSE event on the LEDGER. The reconciliation trail is self-documenting [12].

14.6. 14.6 The WALLET in Healthcare Governance

In most hospitals, clinical AI governance is invisible — it happens in meetings, emails, and policy documents that no one reads. The WALLET makes it an economic event:

- **Per-clinician tracking**: Every clinician interacting with clinical AI has a WALLET. Governance contributions (reviewing AI outputs, validating recommendations, flagging errors) are MINT:WORK events. Their governance track record is on the chain.
- **Per-department aggregate**: The organizational timeline aggregates activity by department. The CMO sees which departments are actively governing their AI deployments — and which are not.
- **Institutional economics**: Total COIN minted is total governance work performed. Total DEBIT:DRIFT is total governance regression. The ratio is institutional governance velocity.

Governance work mints COIN. Governance neglect costs COIN. The incentive structure is architectural [22].

14.7. 14.7 WALLET CLI Operations

```
$ vault user-wallet dexter
# Balance: 12,847 COIN | Events: 347 | Last: MINT:WORK +92
```

```
$ vault user-timeline dexter --last 5
# evt:00343 MINT:WORK +128 SERVICES/COMPLIANCE 127→255
# evt:00344 SPEND −255 SHOP/FHIR-PLAYBOOK
# evt:00345 MINT:WORK +35 SERVICES/NEW-SCOPE 0→35
# evt:00346 TRANSFER −100 → isabella (fee: 5)
# evt:00347 MINT:WORK +92 BOOKS/DOCTRINE 163→255

$ vault verify-wallet dexter
# Hash chain: INTACT (347/347) | Signatures: 298/347 | Status: VERIFIED
```

14.8. 14.8 WALLET Error Handling

Error	Cause	Resolution
`INSUFFICIENT_BALANCE`	SPEND exceeds balance	Earn COIN via governance work
`CHAIN_BROKEN`	Hash mismatch	Restore from LEDGER
`SIGNATURE_INVALID`	Ed25519 failure	Rotate keys: `vault keygen`
`WALLET_NOT_FOUND`	No wallet	Run `vault signup`
`CLOSE_MISMATCH`	Balance disagrees with LEDGER	LEDGER wins — recompute

14.9. 14.9 WALLET Data Structure

```
{
  "principal": "dexter",
  "created": "2025-11-01T00:00:00Z",
  "balance": 12847,
  "events": 347,
  "head": "wallet:dexter:evt:00347",
  "last_close": "2026-02-28T23:59:59Z",
  "pubkey": "ed25519:a1b2c3d4...",
  "status": "ACTIVE"
}
```

The WALLET JSON is `_generated` — derived from the LEDGER. If the JSON is wrong, fix the LEDGER. Run `vault reconstruct`. The `_generated` contract applies [2].

14.10. 14.10 WALLET Capacity

Scale	Wallets	Events/Month	Storage
Solo	1	~50	< 100 KB
Team	10	~500	< 1 MB
Department	50	~2,500	< 5 MB
Hospital	200	~10,000	< 20 MB
Network	1,000	~50,000	< 100 MB

14.11. 14.11 Clinical Vignette: Governance Portfolio

Dr. Chen governs five scopes. Her WALLET reflects the portfolio:

```
$ vault user-wallet dr.chen --by-scope
# MAMMOCHAT: 255 COIN (complete)
# ONCOCHAT: 255 COIN (complete)
# FHIR-API: 255 COIN (complete)
# HIPAA: 255 COIN (complete)
# RADIOLOGY/AI: 127 COIN (in progress)
# MINT:SIGNUP: 500 COIN
# MINT:PYRAMID: 1,000 COIN (2 referrals)
# SPEND: -362 COIN (purchases)
# Balance: 3,285 COIN
```

Dr. Chen's WALLET is her governance resume — a cryptographically signed event chain. A hiring committee runs `vault verify-wallet dr.chen` and confirms every claim [12].

14.12. 14.12 WALLET and Regulatory Reporting

Generate per-principal audit reports at any time:

```
$ vault audit-report dr.chen --period 2026-Q1 --format pdf
# Generating audit report...
# Principal: dr.chen@hadleylab.org
# Period: 2026-01-01 to 2026-03-31
# MINT:WORK events: 12 (governance labor)
# DEBIT:DRIFT events: 0 (no regressions)
# SPEND events: 3 (product purchases)
```

```
# TRANSFER events: 2 (COIN transfers)
# Net COIN: +1,147
# Chain integrity: VERIFIED
# Report saved: audit-dr.chen-2026-Q1.pdf
```

Derivable from the LEDGER at any time. No manual report generation. No quarterly surveys. The report is the chain [12].

14.13. 14.13 Clinical Vignette: WALLET Forensics After a Governance Incident

University of Michigan Health deploys CardiChat — a governed TALK agent for ACC/AHA heart failure guideline navigation. Six months in, MONITORING flags a drift: CardiChat drops from 255 to 191. The DEBIT:DRIFT event debits 64 COIN from the governance team's WALLET.

The governance officer investigates:

```
vault user-timeline carditeam@umich.edu --scope SERVICES/TALK/CARDICHAT --last 10
# evt:00892 MINT:WORK +128 SERVICES/TALK/CARDICHAT 127->255 (2025-10-15)
# evt:00893 MINT:WORK +0   neutral edit (2025-11-01)
# evt:00894 MINT:WORK +0   neutral edit (2025-12-15)
# evt:00895 MINT:WORK +0   neutral edit (2026-01-10)
# evt:00896 DEBIT:DRIFT -64 SERVICES/TALK/CARDICHAT 255->191 (2026-03-08)
#   Cause: LEARNING.md stale — no new entries in 90 days
#   Question lost: What have you learned? (LEARNING.md deleted)
#   Commit: abc1234 by intern@umich.edu
```

Root cause: a medical intern committed a code change to CardiChat's response templates. The pre-commit hook was not installed on the intern's workstation (new hire, incomplete onboarding). The commit modified response formatting but did not update LEARNING.md. MONITORING detected the staleness 90 days later when the freshness threshold expired.

The recovery procedure:

```
# Step 1: Record the incident in LEARNING.md
echo "| 2026-03-08 | DRIFT | LEARNING stale — intern commit bypassed
pre-commit hook. Root cause: onboarding gap. Fix: mandatory hook
install in first-day checklist. | Governance incident |" >> LEARNING.md

# Step 2: Add current patterns from the last 90 days
echo "| 2026-03-08 | PATTERN | Heart failure patients ask about SGLT2i
(dapagliflozin, empagliflozin) 4x more than beta-blockers. Update
```

```
evidence priority. | Usage analytics |" >> LEARNING.md

# Step 3: Commit and validate
git add LEARNING.md
git commit -m "GOV: CARDICHAT — restore LEARNING, record drift incident"
magic validate
# Score: 255/255 (restored)
# MINT:WORK +64 COIN (recovery)
```

The WALLET now shows the full incident timeline: drift, debit, recovery. Net COIN impact: zero (64 debited, 64 recovered). But LEARNING.md is richer — it contains both the drift incident pattern and the clinical usage pattern about SGLT2 inhibitor queries. Governance improved because the system detected, penalized, and recovered from the failure automatically [12].

14.14. 14.14 WALLET Cross-Reference Verification

The WALLET supports cross-reference verification — confirming that events recorded in one principal's WALLET match events recorded in another:

```
vault cross-reference alice@hospital.org bob@hospital.org
# Shared events: 3
#   evt:00234 TRANSFER alice->bob 100 COIN (2026-02-15)
#     alice: DEBIT 100 COIN + 5 FEE -> VERIFIED
#     bob:   CREDIT 95 COIN         -> VERIFIED
#   evt:00267 SPEND alice bought bob's product 255 COIN (2026-02-28)
#     alice: DEBIT 255 COIN         -> VERIFIED
#     bob:   CREDIT 255 COIN        -> VERIFIED
#   evt:00301 TRANSFER bob->alice 50 COIN (2026-03-05)
#     bob:   DEBIT 50 COIN + 2.5 FEE -> VERIFIED
#     alice: CREDIT 47.5 COIN        -> VERIFIED
# Cross-reference: ALL MATCH
```

Double-entry bookkeeping, CANONIC-style. Every transaction appears in two WALLETS. The amounts must balance. CLOSE verifies this for all principals. If a discrepancy is found, the LEDGER is the tiebreaker [12].

14.15. 14.15 WALLET Privacy Architecture

WALLET data is tiered by privacy level:

Data	Privacy Level	Who Can See	Access Method
Balance (aggregate)	Public	Everyone	Fleet page
Event count	Public	Everyone	Fleet page
Event details	Private	Owner + governance officers	`vault user-timeline`
Signatures	Private	Owner only	`vault verify-wallet`
Private key	Secret	Owner only	Local keychain

Organizational transparency (public balance) coexists with individual privacy (private event details). Publish department-level governance metrics without exposing individual clinician activity. The VAULT auth gate (Chapter 15) enforces the tiering [1215].

14.16. 14.16 WALLET Lifecycle States

State	Description	Transitions
PENDING	Identity created, no events	-> ACTIVE (first MINT:SIGNUP)
ACTIVE	Normal operation	-> SUSPENDED, -> CLOSED
SUSPENDED	Temporarily frozen (investigation)	-> ACTIVE, -> CLOSED
CLOSED	Permanently sealed (departure)	Terminal state

```
vault lifecycle dr.chen@hadleylab.org
# State: ACTIVE
# Created: 2025-11-01
# Events: 347
# Last event: 2026-03-10 (today)
# Days active: 130
```

When a clinician leaves, their WALLET transitions to CLOSED. The event chain is sealed — no new events — but remains readable for audit purposes. COIN balance transfers to a successor WALLET or returns to TREASURY. The transition is itself a LEDGER event, providing the audit trail HIPAA requires for workforce clearance procedures [12].

14.17. 14.17 WALLET Performance Characteristics

Operation	Time Complexity	Storage	Latency
Get balance	O(n) chain walk, O(1) cached	N/A	< 1ms cached
Add event	O(1) append	~200 bytes/event	< 5ms
Verify chain	O(n) full walk	N/A	< 100ms for 1K events
Monthly CLOSE	O(n * m) all wallets, all events	N/A	< 5s for 200 wallets
Cross-reference	O(n + m) two chains	N/A	< 50ms

At hospital scale (200 wallets, 10,000 events/month), CLOSE completes in under 5 seconds. Individual wallet verification: under 100ms. Performance is bounded by chain length, which grows linearly. For multi-year deployments, archival of settled events keeps active chains manageable [12].

14.18. 14.18 Governance Proof: The WALLET as Financial Instrument

The WALLET satisfies financial instrument requirements under healthcare accounting standards:

1. **Verifiability.** Any balance can be independently verified by walking the chain: `balance = SUM(credits) − SUM(debits)`. No hidden state.
2. **Non-repudiation.** Ed25519 signatures on events prevent participants from denying transactions.
3. **Tamper-evidence.** Hash-linked chain breaks on any modification. Tampering is detectable in O(n).
4. **Reconcilability.** Monthly CLOSE cross-references USER timelines, ORG timelines, and LEDGER. Three independent records.
5. **Auditability.** Any regulator can run `vault verify-wallet` and `vault audit-report` to produce complete audit documentation.
6. **Conservation.** Total COIN in circulation equals SUM of all active WALLET balances. Verified at every CLOSE.

These six properties satisfy SOX §404 (internal controls), HIPAA §164.312(b) (audit controls), and GAAP requirements for financial recording. The WALLET is not a database row storing a number — it is a cryptographically verified, append-only, hash-linked financial instrument that computes its own balance. The proof is the chain. Q.E.D. [12][15].

14.19. 14.19 WALLET Migration Between Organizations

When a clinician transfers between health systems, their WALLET migrates intact — no history lost.

```
vault migrate dr.chen@hospital-a.org dr.chen@hospital-b.org

# Migration plan:
```

```
#   Source: hospital-a-canonic WALLET (347 events, 1,240 COIN)
#   Target: hospital-b-canonic WALLET (new)
#
#   Step 1: Seal source WALLET (no new events)
#   Step 2: Export signed event chain
#   Step 3: Verify chain integrity at target
#   Step 4: Import chain into target WALLET
#   Step 5: Transfer COIN balance (1,240 COIN)
#   Step 6: Record WALLET:MIGRATE on both LEDGERs
#
# Execute? [y/N] y
# Migration complete.
#   Source: CLOSED (sealed at evt:347)
#   Target: ACTIVE (347 imported events + WALLET:MIGRATE)
#   Balance: 1,240 COIN (verified)
```

Both organizations' LEDGERs record the event — Hospital A logs `WALLET:MIGRATE_OUT`, Hospital B logs `WALLET:MIGRATE_IN`. Every scope governed, every COIN minted, every LEARNING contributed transfers intact. The receiving organization verifies the entire chain independently [1219].

14.20. 14.20 WALLET Consolidation for Multi-Role Identities

Clinicians serving multiple roles (attending physician, researcher, governance officer) may accumulate COIN across multiple WALLET addresses. Consolidation merges them:

```
vault consolidate \
  --primary dr.chen@hadleylab.org \
  --merge dr.chen-research@hadleylab.org \
  --merge dr.chen-governance@hadleylab.org

# Consolidation plan:
#   Primary: dr.chen@hadleylab.org (847 events, 2,100 COIN)
#   Merge 1: dr.chen-research (123 events, 340 COIN)
#   Merge 2: dr.chen-governance (56 events, 180 COIN)
#   Result: dr.chen@hadleylab.org (1,026 events, 2,620 COIN)
#
# Consolidation produces WALLET:MERGE events on LEDGER.
# Merged WALLETs are sealed (CLOSED state).
# All historical events remain queryable via merged chain.
```

Complete provenance is preserved. Each merged event retains its original attribution — governance work performed under `dr.chen-research` remains attributable to the research role after consolidation. The result is the union of all chains, ordered by timestamp [12].

14.21. 14.21 WALLET Delegation

When a governance officer manages COIN on behalf of team members, WALLET supports delegation with defined limits. Delegates can mint and transfer but cannot settle (fiat exit) or close:

Permission	Owner	Delegate	Scope
MINT:WORK	Yes	Yes	Capped at 255 COIN per commit
TRANSFER	Yes	Yes	Capped at 500 COIN per day
SPEND	Yes	Yes	Capped at 1,000 COIN per day
SETTLE	Yes	No	Owner only
CLOSE	Yes	No	Owner only

```
vault delegate \
  --wallet dr.chen@hadleylab.org \
  --delegate intern.garcia@hadleylab.org \
  --permissions MINT,TRANSFER \
  --limits "MINT:255/commit,TRANSFER:500/day" \
  --expires 2026-06-10

# Delegation recorded: WALLET:DELEGATE on LEDGER
# Delegate: intern.garcia@hadleylab.org
# Expires: 2026-06-10
```

Delegation is a LEDGER event. Every delegate action is attributed to the delegate identity with the delegation reference. Revocation at any time via `WALLET:REVOKE_DELEGATE` — also a LEDGER event [1215].

Chapter 15

Chapter 15: VAULT

COIN + INTEL. Private economic aggregate. The encrypted counterpart to the public SHOP (Chapter 12; extended in Chapter 34) [15]. The VAULT projects LEDGER data (Chapter 13), WALLET balances (Chapter 14), and MONITORING metrics (Chapter 22) through an auth gate.

15.1. 15.1 Axiom

VAULT aggregates private projections from all services. Auth-gated. Ledger-backed. The private mirror of SHOP [15].

SHOP shows the world what you sell. VAULT shows you how the business runs. Both use the same discovery mechanism — filesystem walk — and the same compilation pipeline. The only difference is the auth gate. VAULT.md files create private projections restricted to authorized principals. No separate database. Same governance tree, different audience [2].

15.2. 15.2 The Dual Projection Model

Every service in CANONIC can project into two surfaces:

```
Service → SHOP.md  (public)   → discovered by SHOP compiler → visible to everyone
Service → VAULT.md (private)  → discovered by VAULT compiler → visible to authed principals
```

A service can have both projections, only one, or neither. The projection files are the governed interface between the service and its audiences [2].

Projection	Discovery	Auth	Content	Clinical Example
SHOP.md	Filesystem walk	None (public)	Product cards, pricing, synopsis	MammoChat pricing page
VAULT.md	Filesystem walk	GitHub OAuth	Economic data, dashboards, metrics	MammoChat usage analytics

15.3. 15.3 VAULT Structure

The VAULT aggregates private data across services into a single auth-gated surface:

```
~/.canonic/vault/
  ├── LEDGER/        ← economic event history (private dashboard)
  ├── MAGIC/         ← compute economics (GPU costs, provider spend)
  ├── WALLET/        ← per-user COIN balance and timeline
  ├── ANALYTICS/     ← governance metrics, drift tracking
  └── ADMIN/         ← user management, access control
```

Each directory corresponds to a service's private projection. Not a monolithic dashboard — a composable aggregate of service-level views. Add a VAULT.md to a service and its private projection appears automatically [15].

15.4. 15.4 Auth Architecture

VAULT auth uses GitHub OAuth as the KYC anchor. The auth flow:

```
1. User requests VAULT resource → /vault/{service}/
2. VAULT checks session → KV-backed session token (not client localStorage)
3. If no session → redirect to GitHub OAuth
4. GitHub returns identity → verify against CANON.md readers/writers list
5. If authorized → grant access, set session
6. If unauthorized → deny (fail-closed)
7. Auth event → LEDGER (login, grant, or deny recorded)
```

Auth is fail-closed: if identity cannot be verified, access is denied. HIPAA requires the default to be deny, not allow. Every auth event (login, logout, grant, deny) is recorded on the LEDGER [15].

15.5. 15.5 VAULT CLI

The `vault` command-line tool manages VAULT operations:

```
$ vault keygen dexter                    # generate Ed25519 key pair
$ vault auth dexter                      # authenticate principal
$ vault mint dexter 92 --scope DOCTRINE  # mint COIN for governance work
$ vault drift dexter 15 --scope ONCOCHAT # debit for governance regression
$ vault transfer dexter isabella 100     # transfer COIN (5% fee)
$ vault spend dexter 255 --product CANON # purchase product
$ vault settle dexter 1000 --fiat USD    # fiat exit
$ vault close --month 2026-03            # monthly reconciliation
$ vault user-wallet dexter               # show wallet state
$ vault user-timeline dexter             # show event timeline
$ vault verify-wallet dexter             # verify chain integrity
$ vault reconcile                        # cross-check all wallets vs LEDGER
```

State-modifying commands produce LEDGER events. Read-only commands do not. The CLI is the administrative interface to the CANONIC economy [15].

15.6. 15.6 VAULT and Clinical Data Governance

The pattern separates governance into two surfaces:

- **Public (SHOP)**: Product availability, pricing, governance scores, tier status. The procurement surface.
- **Private (VAULT)**: Usage analytics, cost tracking, drift events, clinician governance activity. The operations surface.

A hospital's MammoChat product page (SHOP) shows pricing and evidence citations — intentionally public. The MammoChat governance dashboard (VAULT) shows usage metrics, drift alerts, and activity logs — necessarily auth-gated. Right data, right audience, right gate. SHOP for procurement. VAULT for operations. Both governed. Both discoverable. Both on the LEDGER [2] [15].

15.7. 15.7 VAULT Data Flow Architecture

The VAULT stores nothing — it projects data from governed sources through an auth gate:

```
Source Service          VAULT Projection          Consumer
──────────────          ────────────────          ────────
LEDGER events     →     /vault/LEDGER/       →    Governor dashboard
```

```
WALLET balances   →    /vault/WALLET/       →    User account page
MONITORING metrics →   /vault/ANALYTICS/    →    Ops team dashboard
MAGIC compute logs →   /vault/MAGIC/        →    Cost tracking view
IDENTITY records  →    /vault/ADMIN/        →    Admin user list
```

Each arrow is a read-only projection, filtered by the principal's permissions. The VAULT never duplicates source data — it reads from the canonical location and renders through its auth-gated surface [15].

Projections are compiled, not configured. The VAULT compiler walks the tree, discovers all VAULT.md files, and generates the projection map:

```
# Compile VAULT projections
build --vault

# Output: _data/vault.json
{
  "_generated": true,
  "compiled_at": "2026-03-10T14:00:00Z",
  "projections": [
    {
      "source": "SERVICES/LEDGER",
      "vault_path": "/vault/LEDGER/",
      "auth_level": "WRITER",
      "projection_type": "DASHBOARD"
    },
    {
      "source": "SERVICES/WALLET",
      "vault_path": "/vault/WALLET/",
      "auth_level": "OWNER",
      "projection_type": "ACCOUNT"
    }
  ]
}
```

The `_generated: true` flag applies. Do not hand-edit `_data/vault.json`. Fix the VAULT.md files or the VAULT compiler instead.

15.8. 15.8 VAULT Encryption at Rest

VAULT data projections traverse an encryption layer before reaching the auth-gated surface. The encryption model:

```
Plaintext source → Ed25519 signing → AES-256-GCM encryption → Auth gate → Decrypted view
```

Key management is scope-based. Each scope with a VAULT.md has its own encryption context:

Layer	Algorithm	Key Source	Rotation
Signing	Ed25519	`vault keygen {principal}`	On principal rotation
Encryption	AES-256-GCM	Derived from scope key	Monthly via `vault rotate`
Session	HMAC-SHA256	KV-backed session secret	Per-session (ephemeral)
Transport	TLS 1.3	Platform certificate	Annual

Encryption is not optional. Every VAULT projection is encrypted at rest and in transit. Generate the key pair that anchors a principal's identity:

```
# Generate key pair for a new principal
vault keygen dr-martinez

# Output:
# Public key:  vault/keys/dr-martinez.pub
# Private key: vault/keys/dr-martinez.key (NEVER commit)
# Fingerprint: SHA256:a1b2c3d4e5f6...
# Added to:    CANON.md readers list
```

The private key never enters the governance tree. The public key is referenced in CANON.md for authorization. The fingerprint is recorded on the LEDGER as the principal's cryptographic identity [2].

15.9. Ed25519: Enforced

Ed25519 signing is no longer optional. As of March 2026, every LEDGER event is signed. The enforcement is a hard CI gate: `vault verify-sig` runs in build phase `09-econ` and exits with code 1 on any unsigned event.

Fleet-wide status: zero unsigned events across all 9 principals. Key rotation is governed — `vault key-status` shows key age, and the KEY_ROTATION hardening gate (CLOSED) ensures rotation happens on schedule.

The signature cutoff marks the boundary between legacy (unsigned CONSTRUCTION-era events) and production (all events signed). Events before the cutoff are grandfathered. Events after the cutoff must be signed or the build fails.

15.10. 15.9 VAULT Key Management

No external PKI. Keys are governed by the same CANON.md that governs everything else.

Key lifecycle:

```
GENERATE → ACTIVE → ROTATING → ROTATED → REVOKED
```

Each transition is a LEDGER event:

```
# Rotate a principal's key (generates new key, marks old key as ROTATING)
vault rotate dr-martinez

# Output:
# New key generated: vault/keys/dr-martinez-2026-03.pub
# Old key status:    ROTATING (grace period: 72h)
# LEDGER event:      VAULT:KEY_ROTATE recorded
# After 72h:         Old key → ROTATED (read-only, no new sessions)

# Revoke a key immediately (emergency action)
vault revoke dr-martinez --reason "departure"

# Output:
# Key status:    REVOKED
# Active sessions: TERMINATED (3 sessions closed)
# LEDGER event:  VAULT:KEY_REVOKE recorded
# VAULT access:  DENIED for all projections
```

Key management rules:

Rule	Constraint	Enforcement
No shared keys	One key pair per principal	`vault keygen` rejects duplicate principals
Mandatory rotation	Keys rotate every 90 days	`vault audit --keys` flags stale keys
Revocation propagation	Revoked keys terminate all sessions	Immediate — no grace period
Key backup	Private keys are never stored in governance tree	`.gitignore` enforced, `magic validate` checks
Audit trail	Every key event is a LEDGER entry	`vault keygen`, `rotate`, `revoke` all log

When a clinician leaves, `vault revoke` immediately terminates VAULT access, logs the revocation on the LEDGER, and satisfies HIPAA §164.312(d) [15].

15.11. 15.10 VAULT Dashboard Views

The VAULT surface renders as a set of governed dashboards. Each dashboard is a projection of a service's private data through the auth gate:

LEDGER Dashboard — `/vault/LEDGER/`

```
┌──────────────────────────────────────────────────┐
│  LEDGER — Economic Event Stream                  │
├────────────┬────────────┬────────────┬───────────┤
│ Time       │ Event      │ Amount     │ Scope     │
├────────────┼────────────┼────────────┼───────────┤
│ 14:32:01   │ MINT       │ +92 COIN   │ DOCTRINE  │
│ 14:31:45   │ SPEND      │ −255 COIN  │ MAMMOCHAT │
│ 14:30:12   │ DRIFT      │ −15 COIN   │ ONCOCHAT  │
│ 14:29:58   │ TRANSFER   │ −100 COIN  │ → isabella│
│ 14:29:58   │ FEE        │ −5 COIN    │ TRANSFER fee │
└────────────┴────────────┴────────────┴───────────┘
  Total events today: 47    Net COIN flow: +312
```

ANALYTICS Dashboard — `/vault/ANALYTICS/`

```
┌──────────────────────────────────────────────────┐
│  Governance Analytics — Fleet Overview           │
├──────────────────────────────────────────────────┤
│  Scopes at 255:    73/73  (100%)                 │
│  Mean score:       255.0                         │
│  Drift events:     2 (last 30 days)              │
│  Recovery time:    avg 4.2 hours                 │
│  COIN minted:      18,615 (lifetime)             │
│  COIN debited:     342 (drift penalties)         │
│  Net governance:   18,273 COIN                   │
├──────────────────────────────────────────────────┤
│  Score Distribution:                             │
│  ████████████████████████████████████ 255        │
│                                        0–254     │
└──────────────────────────────────────────────────┘
```

MAGIC Dashboard — `/vault/MAGIC/`

```
┌──────────────────────────────────────────────────┐
│  Compute Economics — Cost Tracking               │
├──────────────┬──────────┬────────────────────────┤
│ Provider     │ Spend    │ Requests (30d)         │
├──────────────┼──────────┼────────────────────────┤
│ Anthropic    │ $142.30  │ 3,847 (Claude)         │
│ OpenAI       │ $0.00    │ 0                      │
│ Ollama       │ $0.00    │ 12,450 (local)         │
```

```
├──────────────┼──────────┼──────────────────────┤
│ Total        │ \$142.30  │ 16,297               │
└──────────────┴──────────┴──────────────────────┘

  Cost per COIN minted: \$0.0076
  Cost per validation:  \$0.0023
```

Each view is compiled from the governance tree — layout governed by VAULT.md, data projected from canonical sources, access filtered by the authenticated principal's permissions [15].

15.12. 15.11 VAULT Integration with MONITORING

MONITORING provides the operational data that VAULT projects as private dashboards. The integration follows the producer-consumer pattern:

```
MONITORING (producer)                VAULT (consumer)
─────────────────────                ────────────────

Collects metrics          →          Projects /vault/ANALYTICS/
Detects drift             →          Renders drift alerts
Tracks uptime             →          Shows service status
Records validation runs   →          Displays compliance timeline
```

MONITORING writes to the governance tree. VAULT reads from it. Neither service knows about the other — they communicate through the filesystem. Same pattern as SHOP: filesystem-mediated composition, not API-mediated integration [2].

Configuration for the integration is declared in the VAULT.md:

```
```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

vault: true monitoring_source: SERVICES/MONITORING refresh: 300

• • •

15.13. VAULT
```

Field	Value
projection	ANALYTICS
source	MONITORING
auth_level	WRITER
refresh_seconds	300

`refresh: 300` triggers recompilation every 300 seconds — not polling, but a compilation trigger. The VAUL

15.12 Clinical VAULT Example: Hospital Governance Dashboard

A hospital deploys three governed clinical AI services and the governance officer needs a private dashboar

1. Which services are at 255
2. Which services have drifted
3. How much COIN has been minted for governance work
4. Which clinicians are contributing governance
5. Cost of compute (LLM API spend)

No custom application required. Authenticate via GitHub OAuth and the VAULT projects the data:

```bash
# Hospital governance officer's VAULT view
vault dashboard --org hadleylab

# Output:
# ┌──────────────────────────────────────────────┐
# │  HadleyLab Clinical AI Governance            │
# ├──────────────┬───────┬────────────┬──────────┤
# │ Service      │ Score │ Status     │ Last Check│
# ├──────────────┼───────┼────────────┼──────────┤
# │ MammoChat    │ 255   │ GOVERNED   │ 2m ago   │
# │ OncoChat     │ 255   │ GOVERNED   │ 2m ago   │
# │ CardiChat    │ 191   │ DRIFTING   │ 2m ago   │
# ├──────────────┼───────┼────────────┼──────────┤
# │ Fleet Score  │ 701/765 (91.6%)              │
# └──────────────┴───────┴────────────┴──────────┘
#
# ALERT: CardiChat dropped from 255 → 191
#   Dimension: LEARNING (new model not validated)
#   Action required: validate LEARNING scope
#   Run: magic validate --scope SERVICES/TALK/CARDICHAT
```

When CardiChat's score drops, MONITORING writes the drift event to the governance tree. VAULT reads

it and renders the alert. No custom alerting pipeline. No webhook configuration. Filesystem-mediated governance [15].

15.14. 15.13 VAULT vs Traditional Data Warehouses

The differences from a traditional data warehouse are structural:

Property	Traditional Data Warehouse	VAULT
Data storage	Centralized copy	Projection from source (no copy)
Schema	ETL-defined	Governance-defined (VAULT.md)
Auth	Application-level RBAC	GitHub OAuth + CANON.md principals
Audit trail	Separate audit log	LEDGER (same chain as economic events)
Refresh	Batch ETL jobs	Compilation trigger (build –vault)
Drift detection	Custom monitoring	Built-in (MONITORING VAULT pipeline)
Cost	License + infrastructure	Zero marginal cost (governance tree projection)

No ETL layer. No data to extract. The VAULT reads from the governance tree — the same tree that SHOP, MONITORING, and every service reads from. The governance tree is the data layer. The VAULT is a view, filtered by auth [2].

This eliminates a major compliance burden. Traditional data warehouses aggregating clinical AI usage data require their own BAA, their own access controls, and their own audit logs. The VAULT inherits all of these from the governance tree: CANON.md defines access controls, the LEDGER provides audit, and the tree is already under the organization's BAA. No additional compliance infrastructure.

15.15. 15.14 VAULT Audit Capabilities

Every VAULT access is auditable. The audit trail spans three levels:

Level 1: Access Audit — Who accessed what, when.

```
vault audit --access --last 30d

# Output:
# Principal       Resource              Time              Action
# dr-martinez     /vault/ANALYTICS/     2026-03-10 14:32  VIEW
# dr-chen         /vault/WALLET/        2026-03-10 14:28  VIEW
# admin-jones     /vault/ADMIN/         2026-03-10 14:15  MODIFY
# dr-martinez     /vault/LEDGER/        2026-03-10 14:01  VIEW
# [... 847 entries ...]
```

Level 2: Key Audit — Key lifecycle events.

```
vault audit --keys

# Output:
# Principal       Key Status    Created       Last Rotated   Age
# dr-martinez     ACTIVE        2026-01-15    2026-03-01     9d
# dr-chen         ACTIVE        2025-11-20    2026-02-28     10d
# admin-jones     ACTIVE        2026-02-01    2026-03-05     5d
# dr-legacy       REVOKED       2025-06-01    —              REVOKED
# [WARN] dr-chen key age approaching 90-day rotation threshold (80d)
```

Level 3: Projection Audit — What data was projected, from what source.

```
vault audit --projections

# Output:
# Projection      Source              Last Compiled     Staleness
# LEDGER          SERVICES/LEDGER     2m ago            FRESH
# ANALYTICS       SERVICES/MONITORING 2m ago            FRESH
# WALLET          SERVICES/WALLET     2m ago            FRESH
# MAGIC           SERVICES/MAGIC      5m ago            FRESH
# ADMIN           SERVICES/IDENTITY   5m ago            FRESH
# [OK] All projections within refresh threshold
```

Three audit levels provide the evidence trail for HIPAA §164.312(b) (audit controls) and §164.312(d) (authentication). No separate audit system — the LEDGER already records every event. The audit CLI commands are read-only projections, filtered by type [15].

15.16. 15.15 VAULT Error Handling

Operations fail closed. Every error produces a LEDGER event and a clear diagnostic:

Error	Cause	LEDGER Event	Resolution
`VAULT_AUTH_DENIED`	Principal not in CANON.md readers	`VAULT:AUTH_DENY`	Add principal to CANON.md
`VAULT_KEY_EXPIRED`	Key older than 90 days, not rotated	`VAULT:KEY_EXPIRE`	Run `vault rotate {principal}`
`VAULT_KEY_REVOKED`	Key explicitly revoked	`VAULT:KEY_REVOKE`	Generate new key with `vault keygen`
`VAULT_PROJECTION_STALE`	Source data older than refresh threshold	`VAULT:STALE`	Run `build --vault` to recompile
`VAULT_SESSION_INVALID`	KV session expired or tampered	`VAULT:SESSION_FAIL`	Re-authenticate via GitHub OAuth
`VAULT_ENCRYPTION_FAIL`	Encryption key missing or corrupted	`VAULT:CRYPTO_FAIL`	Run `vault rotate` to regenerate

Every error is recoverable through a governance action — a CLI command or a CANON.md edit. No manual database intervention. No administrator tickets. Both recovery paths are governed, auditable, and on the LEDGER.

15.17. 15.16 VAULT Governance Proof

VAULT is not a feature bolted onto CANONIC — it is a governance primitive. Every operation produces a LEDGER event. The audit trail satisfies HIPAA §164.312(b) without additional infrastructure. The proof chain:

```
VAULT.md (governance declaration)
  → build --vault (compilation)
    → vault.json (compiled projection map, _generated)
      → VAULT surface (auth-gated projection)
        → LEDGER (every access recorded)
          → vault audit (audit trail rendered)
```

Each link in the chain is governed. The VAULT.md declares what is projected. The compiler discovers and compiles projections. The auth gate enforces access. The LEDGER records every event. The audit CLI

renders the trail. No link is ungoverned. No event is unrecorded. The VAULT is COIN + INTEL composed through an auth gate — economic data (COIN balances, spend tracking, cost analytics) combined with governed knowledge (governance scores, drift events, compliance status), filtered by cryptographic identity. The composition is the VAULT. The auth gate is the differentiator. SHOP projects the same data publicly. VAULT projects it privately. Both are projections of the same governance tree. Both are discovered by filesystem walk. Both are compiled by the build pipeline. The only difference is the gate [2] [15].

Chapter 16

Chapter 16: API

HTTP COIN operations. The runtime interface to the governed economy [15]. The API exposes the same operations as the `vault` CLI (Chapter 15), routes declared via HTTP.md (see Chapter 26 for the generated contract pattern), and records every write to the LEDGER (Chapter 13). Fleet API endpoints are live at api.gorunner.pro.

16.1. 16.1 Axiom

API exposes COIN operations via HTTP. Every request authenticated. Every response governed. Every call ledgered [15].

The `vault` CLI manages the economy from the command line. The API exposes the same operations over HTTP for programmatic access — clinical AI applications, hospital integrations, and third-party systems interact with the CANONIC economy through it [15].

16.2. 16.2 Endpoint Architecture

Routes are driven from governed indices — no hardcoded endpoints. Every API endpoint maps to a governed operation:

Endpoint	Method	Operation	Auth	Rate	LEDGER Event
`/api/v1/mint`	POST	Mint COIN for governance work	Required	10/min	MINT:WORK
`/api/v1/spend`	POST	Purchase product	Required	10/min	SPEND
`/api/v1/transfer`	POST	Transfer COIN between users	Required	10/min	TRANSFER

Endpoint	Method	Operation	Auth	Rate	LEDGER Event
`/api/v1/settle`	POST	Fiat exit	Required	1/day	SETTLE
`/api/v1/wallet`	GET	Read wallet state	Required	100/min	None (read-only)
`/api/v1/timeline`	GET	Read event timeline	Required	100/min	None (read-only)
`/api/v1/health`	GET	Service health check	None	Unlimited	None
`/api/v1/metrics`	GET	Prometheus metrics	None	Unlimited	None

Write operations require authentication and produce LEDGER events. Read operations require authentication but do not produce events. Health and metrics endpoints are public and unlimited [15].

16.3. 16.3 Authentication and Rate Limiting

Write operations require GitHub OAuth authentication. Session tokens are server-side (KV-backed), never in client localStorage. Rate limits scale with your governance tier:

Tier	Write Rate	Read Rate	Daily Limit
COMMUNITY	5/min	50/min	100 writes
ENTERPRISE	10/min	100/min	500 writes
AGENT	20/min	200/min	1,000 writes
FULL	50/min	500/min	5,000 writes

Rate limiting is governance, not punishment. Higher tiers have earned more capacity through governance work [15].

16.4. 16.4 Request/Response Governance

Every API response includes governance headers:

```
X-Canonic-Scope: hadleylab-canonic/SERVICES/TALK/MAMMOCHAT
X-Canonic-Score: 255
X-Canonic-Tier: FULL
X-Canonic-Ledger-Id: evt:00047
```

Every response carries its own provenance: the serving scope, its governance score at request time, its tier, and the LEDGER event ID for writes. Client applications verify governance by reading headers — no separate lookup required [15].

16.5. 16.5 Error Handling

The API fails closed. If governance cannot validate a request, the request is denied:

```
{
  "error": "GOVERNANCE_VALIDATION_FAILED",
  "scope": "hadleylab-canonic/SERVICES/TALK/MAMMOCHAT",
  "score": 191,
  "required": 255,
  "message": "Scope score below deployment threshold. Heal governance before API access."
}
```

A 403 response with a governance error is not a bug — it is the system working as designed. The API does not serve requests from scopes that have fallen below 255. The fix is governance work, not API configuration [15].

16.6. 16.6 Clinical Integration Patterns

Three integration patterns for hospital EHR teams:

Epic MyChart: Query `/api/v1/wallet` to display governance activity in the patient portal — interaction count, evidence cited, COIN involved.

Cerner PowerChart: Query the TALK API to serve governed clinical recommendations within the EHR workflow. Governance headers in the response feed the EHR's own audit trail.

HL7 FHIR bridge: Record governance events for FHIR resource accesses. When a clinical AI agent reads a Patient resource, the access becomes a LEDGER event via the API [15].

16.7. 16.7 API Route Governance via HTTP.md

API routes are not hardcoded in application code. They are declared in HTTP.md — a governed route manifest that the API compiler reads:

```
```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

16.8. Routes
```

method	path	handler	auth	rate	ledger
POST	/api/v1/mint	mint_handler	required	10/min	MINT:WORK
POST	/api/v1/spend	spend_handler	required	10/min	SPEND
POST	/api/v1/transfer	transfer_handler	required	10/min	TRANSFER
GET	/api/v1/wallet	wallet_handler	required	100/min	none
GET	/api/v1/health	health_handler	none	unlimited	none

```
The HTTP.md file is the single source of truth for API routes. The API compiler reads HTTP.md and generate

```bash
# Compile API routes from HTTP.md
build --api

# Validate route table against governance constraints
magic validate --scope SERVICES/API

# List all governed routes
magic scan --routes

# Output:
# SERVICES/API/HTTP.md: 8 routes (5 write, 3 read)
# SERVICES/TALK/MAMMOCHAT/HTTP.md: 4 routes (2 write, 2 read)
# SERVICES/TALK/ONCOCHAT/HTTP.md: 4 routes (2 write, 2 read)
# Total: 16 governed routes across 3 scopes
```

Every HTTP endpoint is discoverable, validated, and ledgered. No shadow endpoints. No undocumented routes. HTTP.md is the API's single source of truth.

16.9. 16.8 Request Lifecycle

Every API request traverses a governed pipeline. The lifecycle:

```
Request → Auth Gate → Rate Limiter → Scope Validator → Handler → LEDGER → Response
```

Step-by-step:

1. **Request arrives.** The API router matches the path and method against the compiled route table from HTTP.md.
2. **Auth Gate.** If the route requires auth (`auth: required`), the API checks the KV-backed session token. No token or invalid token returns 401:
```

```
{
  "error": "AUTH_REQUIRED",
  "message": "Authenticate via GitHub OAuth before accessing this endpoint."
}
```

3. **Rate Limiter.** The rate limiter checks the principal's request count against the tier-based limit. Exceeded limit returns 429:

```
{
  "error": "RATE_LIMIT_EXCEEDED",
  "tier": "ENTERPRISE",
  "limit": "10/min",
  "retry_after": 42,
  "message": "Rate limit exceeded. Retry in 42 seconds."
}
```

4. **Scope Validator.** Before executing the handler, the API verifies that the target scope is at 255. If the scope has drifted, the request is denied with 403 (as shown in Section 16.5).
5. **Handler.** The route handler executes the business logic — minting COIN, processing a spend, transferring between wallets.
6. **LEDGER.** If the route is a write operation, the handler's result is recorded as a LEDGER event with hash-chain linking (see Chapter 17).
7. **Response.** The response includes governance headers (Section 16.4) and the operation result.

Every request traverses this pipeline. No route skips auth. No route bypasses the scope validator. No write route omits the LEDGER event. The compiler enforces this — registering a write route without LEDGER integration is structurally impossible.

16.10. 16.9 API Versioning

The API uses path-based versioning: `/api/v1/`, `/api/v2/`. Version governance rules:

Rule	Constraint
No breaking changes within a version	Adding fields is OK; removing or renaming fields is a new version
Version sunset requires 90-day notice	`X-Canonic-Sunset: 2026-06-10` header on deprecated versions
Maximum 2 active versions	Old versions must sunset before new versions ship
Version transitions are LEDGER events	`API:VERSION_SUNSET` and `API:VERSION_ACTIVATE` recorded

```
# Check active API versions
magic scan --api --versions

# Output:
# v1: ACTIVE (since 2025-09-01, 186 days)
#   Routes: 8    Requests (30d): 147,382
# v2: STAGING (since 2026-03-01, 9 days)
#   Routes: 10   Requests (30d): 0 (staging only)
#
# No versions pending sunset.
```

Two active versions maximum prevents API sprawl. Each version is governed by its own HTTP.md. Version transitions are governed processes, not flag flips.

16.11. 16.10 API Payload Schemas

Payload schemas are governance-native, declared in HTTP.md (not OpenAPI):

```
## POST /api/v1/mint

#### Request

| field | type | required | constraint |
|-------|------|----------|------------|
| principal | string | yes | Must match authenticated session |
| scope | string | yes | Must be a valid governed scope at 255 |
| from_bits | integer | yes | Previous governance score (0–254) |
| to_bits | integer | yes | New governance score (from_bits < to_bits ≤ 255) |
| commit | string | yes | Git commit SHA that produced the governance work |

#### Response

| field | type | description |
|-------|------|-------------|
| event_id | string | LEDGER event ID (sha256 hash) |
| coin_minted | integer | COIN minted (to_bits − from_bits) |
| wallet_balance | integer | Updated wallet balance |
| chain_head | string | New chain head hash |
```

`build --api` compiles the schema into request validation middleware. Invalid payloads are rejected before the handler executes:

```
{
  "error": "SCHEMA_VALIDATION_FAILED",
  "violations": [
    {"field": "from_bits", "constraint": "must be integer 0–254", "received": -1},
    {"field": "scope", "constraint": "must be governed scope at 255", "received": "INVALID/PATH"}
  ]
}
```

16.12. 16.11 API Monitoring and Metrics

The API exposes Prometheus-compatible metrics at `/api/v1/metrics`:

```
# HELP canonic_api_requests_total Total API requests by endpoint and status
# TYPE canonic_api_requests_total counter
canonic_api_requests_total{endpoint="/api/v1/mint",status="200"} 3847
canonic_api_requests_total{endpoint="/api/v1/mint",status="403"} 12
canonic_api_requests_total{endpoint="/api/v1/wallet",status="200"} 147382

# HELP canonic_api_latency_seconds Request latency in seconds
# TYPE canonic_api_latency_seconds histogram
canonic_api_latency_seconds_bucket{endpoint="/api/v1/mint",le="0.1"} 3800
canonic_api_latency_seconds_bucket{endpoint="/api/v1/mint",le="0.5"} 3840
canonic_api_latency_seconds_bucket{endpoint="/api/v1/mint",le="1.0"} 3847

# HELP canonic_api_governance_score Current governance score of the API scope
# TYPE canonic_api_governance_score gauge
canonic_api_governance_score{scope="SERVICES/API"} 255
```

Public and unlimited — MONITORING scrapes on a fixed interval. Metrics feed into the VAULT ANALYTICS dashboard (Chapter 15) through the MONITORING-to-VAULT pipeline.

16.13. 16.12 API and the Workers Runtime

The API runs on Cloudflare Workers. The runtime architecture:

```
Client → Cloudflare Edge → Worker (API handler) → KV (sessions, wallets)
                                                 → D1 (LEDGER events)
                                                 → R2 (large artifacts)
```

Component	Purpose	Governance
Worker	Request handling, business logic	Deployed via `build --deploy`
KV	Session tokens, wallet balances	Low-latency reads, eventually consistent
D1	LEDGER events, hash chain	SQL-queryable, strong consistency
R2	Large governance artifacts	Object storage for builds, exports

Not a traditional server — a governed Worker deployed from the governance tree via the build pipeline. The deployment is a LEDGER event. The runtime inherits the governance of its source scope [15].

```
# Deploy API to Workers
build --deploy --scope SERVICES/API

# Output:
# Compiling routes from HTTP.md...
# Generating Worker from governed sources...
# Deploying to Cloudflare Workers...
# LEDGER event: DEPLOY:API recorded
# API live at: https://api.gorunner.pro/api/v1/
# Governance score at deploy: 255
```

16.14. 16.13 Clinical API Example: EHR Integration

Integrating MammoChat with Epic via the CANONIC API:

```
Epic MyChart → FHIR Server → CANONIC API → MammoChat TALK → Response → MyChart
```

Step 1: The Epic integration engine authenticates with the CANONIC API:

```
curl -X POST https://api.gorunner.pro/api/v1/auth \
  -H "Content-Type: application/json" \
  -d '{"provider": "github", "token": "gho_..."}'

# Response:
# {"session": "kvs:abc123", "tier": "ENTERPRISE", "expires": "2026-03-11T14:00:00Z"}
```

Step 2: The integration queries MammoChat for a clinical recommendation:

```
curl -X POST https://api.gorunner.pro/api/v1/talk \
  -H "Authorization: Bearer kvs:abc123" \
  -H "Content-Type: application/json" \
  -d '{"scope": "SERVICES/TALK/MAMMOCHAT", "query": "BI-RADS 4 management"}'

# Response headers:
# X-Canonic-Scope: hadleylab-canonic/SERVICES/TALK/MAMMOCHAT
# X-Canonic-Score: 255
# X-Canonic-Tier: FULL
# X-Canonic-Ledger-Id: evt:00048
#
# Response body:
# {"response": "...", "evidence": ["ACR-2023-BIRADS", "NCCN-2024-BREAST"]}
```

Governance headers give the Epic integration engine the provenance it needs: scope at 255, evidence cited, interaction ledgered. Audit every clinical AI interaction through the LEDGER — no separate audit system [15].

16.15. 16.14 Clinical Vignette: API Governance Prevents Unaudited Access

Cedars-Sinai deploys OncoChat and integrates it with Epic via the API. During testing, a developer bypasses the API and calls the LLM directly — reasoning that auth and rate limiting add unnecessary latency.

The direct call works. The LLM responds. But there are no governance headers, no LEDGER event, no audit trail. The interaction is invisible to compliance.

MONITORING detects the gap within 300 seconds:

```
# MONITORING alert:
# ALERT: Unaudited LLM access detected
#   Source: 10.42.1.15 (epic-integration-dev)
#   Target: anthropic-api (direct)
#   Scope: SERVICES/TALK/ONCOCHAT
#   LEDGER events expected: 47 (based on LLM usage)
#   LEDGER events found: 31
#   Gap: 16 unaudited interactions
#   Action: Block direct LLM access, enforce API gateway
```

The fix is architectural: the API gateway becomes the only path to the LLM. Direct access is blocked at the network level. Every interaction traverses the governed pipeline — Auth Gate, Rate Limiter, Scope Validator, Handler, LEDGER, Response. No shortcuts.

Sixteen unaudited interactions in a one-week testing window. At production scale, that becomes thousands of unaudited clinical AI recommendations per month — each one a potential liability, each one invisible to compliance. The API gateway prevents this by making the governed path the only path [15].

16.16. 16.15 API Idempotency and Retry Safety

Write operations are idempotent when the client provides an idempotency key — critical for clinical integrations where network failures cause retries:

```
curl -X POST https://api.gorunner.pro/api/v1/mint \
  -H "Authorization: Bearer kvs:abc123" \
  -H "Idempotency-Key: mint-2026-03-10-a1b2c3d" \
  -H "Content-Type: application/json" \
  -d '{"principal":"dr.chen","scope":"SERVICES/TALK/ONCOCHAT","from_bits":127,"to_bits":255,"commit":"a1b2
```

If the request is retried (due to network timeout), the API recognizes the idempotency key and returns the original response without creating a duplicate LEDGER event or minting duplicate COIN. The idempotency key is stored in KV with a 24-hour TTL.

Scenario	First Request	Retry	Result
Network timeout	Processed, LEDGER event created	Returns cached response	No duplicate MINT
Client crash	Processed, LEDGER event created	Returns cached response	No duplicate MINT
Server error (500)	Not processed	Reprocessed as new request	Correct single MINT
Duplicate key, different payload	Processed original	Returns 409 Conflict	Prevents mutation

Without idempotency, a lost response followed by a retry would double-mint COIN. With idempotency, the retry returns the original result and the LEDGER remains consistent [15].

16.17. 16.16 API Governance Proof

The governance chain is verifiable end-to-end:

```
HTTP.md declares routes (governance source)
  -> build --api compiles route table (compilation)
    -> Worker deployment (runtime artifact, _generated)
```

```
-> Auth Gate enforces identity (access control)
  -> Scope Validator enforces 255 (governance gate)
    -> LEDGER records every write (audit trail)
      -> Governance headers prove provenance (response-level proof)
```

Every link is governed. HTTP.md is the source of truth. The route table is _generated. The Worker deploys from governed sources. Auth enforces identity. The scope validator enforces 255. The LEDGER records events. Governance headers carry proof in every response.

For a HIPAA audit, the chain provides complete evidence: what endpoints exist (HTTP.md), who can access them (auth gate + CANON.md), what happens on each request (LEDGER events), and governance state at request time (headers). The auditor reads HTTP.md and the LEDGER. No separate documentation required [15].

16.18. 16.17 API Webhook Integration

Governed webhooks propagate events to external systems in real time. Declared in HTTP.md alongside routes:

```
## Webhooks

| event | url | auth | retry | ledger |
|-------|-----|------|-------|--------|
| MINT:WORK | https://ehr.hospital.org/canonic/mint | hmac-sha256 | 3x exponential | WEBHOOK:DELIVER |
| SPEND | https://ehr.hospital.org/canonic/spend | hmac-sha256 | 3x exponential | WEBHOOK:DELIVER |
| DEPLOY | https://slack.hospital.org/canonic | hmac-sha256 | 3x exponential | WEBHOOK:DELIVER |
```

Every delivery is a LEDGER event, signed with a shared secret for receiver verification. Failed deliveries retry with exponential backoff (5s, 25s, 125s). After 3 failures: `WEBHOOK:FAIL`, and NOTIFIER alerts scope writers.

```
# Register webhook
magic api --webhook register \
  --event MINT:WORK \
  --url https://ehr.hospital.org/canonic/mint \
  --auth hmac-sha256 \
  --secret $WEBHOOK_SECRET

# List active webhooks
magic api --webhook list
# MINT:WORK → https://ehr.hospital.org/canonic/mint  (ACTIVE, 247 deliveries, 0 failures)
# SPEND    → https://ehr.hospital.org/canonic/spend  (ACTIVE, 31 deliveries, 0 failures)
# DEPLOY   → https://slack.hospital.org/canonic      (ACTIVE, 12 deliveries, 1 failure)
```

Webhooks close the loop between governance events and clinical awareness. When a governance score changes, the EHR receives immediate notification and can display it inline — the clinician sees that MammoChat was last validated 3 minutes ago at score 255 [1512].

16.19. 16.18 API Gateway Topology

Multi-site deployments use a gateway topology — each hospital site has a local gateway routing to the central governance API:

```
Hospital A Gateway → Central API → LEDGER (shared)
Hospital B Gateway → Central API → LEDGER (shared)
Hospital C Gateway → Central API → LEDGER (shared)
```

Each gateway handles local authentication (SSO with the hospital's identity provider), caches read-only responses, and forwards writes to the central API. The central API owns the LEDGER — write consistency via single-writer model, read consistency eventually consistent with 60-second cache refresh [1519].

Chapter 17

Chapter 17: CHAIN

Cryptographic integrity. Hash-linked events. The tamper-evidence layer [15]. CHAIN provides the integrity guarantee that underpins the LEDGER (Chapter 13), the WALLET (Chapter 14), and the VAULT (Chapter 15). For the governance-as-type-system formalization of these guarantees, see Chapter 36.

17.1. 17.1 Axiom

CHAIN is cryptographic integrity for the governed economy. Every event hash-linked. Tampering is structurally detectable [12].

Not a blockchain. A hash chain — simpler, more auditable, same tamper-evidence guarantee, no consensus overhead. Every LEDGER event links to its predecessor via a cryptographic hash. The chain runs as a directed sequence from genesis to head. Verification is O(n): walk the chain, recompute hashes, confirm integrity [12].

17.2. 17.2 The Hash Chain

Every LEDGER event contains three integrity fields:

```
{
  "id": "sha256:a1b2c3d4e5f6...",
  "prev": "sha256:z9y8x7w6v5u4...",
  "signature": "ed25519:f7e8d9c0b1a2..."
}
```

Field	Purpose	Verification
`id`	Content hash of this event	`sha256(event.data) == event.id`
`prev`	Hash of predecessor event	`event[n].prev == event[n-1].id`
`signature`	Ed25519 signature by actor	`verify(event.data, event.signature, actor.pubkey)`

`id` is the content hash (event not modified). `prev` is the chain link (sequence not reordered). `signature` is the attribution proof (actor actually produced the event) [12].

17.3. 17.3 Verification Algorithm

Chain verification walks backward from head to genesis:

```
def verify_chain(events):
    for i in range(len(events) - 1, 0, -1):
        current = events[i]
        predecessor = events[i - 1]

        # Content integrity
        assert sha256(current.data) == current.id

        # Chain integrity
        assert current.prev == predecessor.id

        # Signature integrity (after cutoff)
        if current.ts >= SIGNATURE_CUTOFF:
            assert ed25519_verify(current.data, current.signature, current.actor.pubkey)

    # Genesis event has no predecessor
    assert events[0].prev is None
    return True
```

Change a single character in any field and the chain breaks. The algorithm reports the exact event where integrity was lost. For 10,000 events, verification takes milliseconds [12].

17.4. 17.4 Fork Detection

A hash chain has a single linear history. Two events claiming the same predecessor constitute a fork:

```
Event A: prev = evt:00046
Event B: prev = evt:00046  ← FORK DETECTED
```

Forks are not permitted — two conflicting histories mean one is wrong. The CHAIN flags forks for investigation. In practice, a fork typically indicates a dual-write bug: an event written to the USER timeline but not the LEDGER, then a second event written to the LEDGER with the same predecessor [12].

17.5. 17.5 CHAIN and Healthcare Compliance

The CHAIN provides the cryptographic guarantees that regulators increasingly require:

HIPAA §164.312(c)(1) requires mechanisms to protect ePHI from improper alteration or destruction. The hash chain ensures that governance events (which may reference clinical AI interactions) cannot be altered after the fact. If an event is modified, the chain breaks. The alteration is detectable.

FDA 21 CFR Part 11 §11.10(e) requires audit trails that cannot be altered. The CHAIN's append-only, hash-linked architecture satisfies this requirement by construction — audit trail entries are hash-linked and cryptographically signed.

Joint Commission standard IM.02.01.03 requires that health information integrity be maintained. The CHAIN provides the structural mechanism for maintaining governance information integrity across the entire clinical AI governance infrastructure [3] [12].

17.6. 17.6 CHAIN CLI Operations

The CHAIN is verified and inspected via CLI commands:

```
# Verify the entire chain from genesis to head
vault verify-chain

# Output:
# Chain length:     4,847 events
# Genesis:          sha256:0000...0001 (2025-09-01T00:00:00Z)
# Head:             sha256:a1b2c3d4... (2026-03-10T14:32:01Z)
# Content hashes:   4,847/4,847 valid □
# Chain links:      4,846/4,846 valid □
# Signatures:       3,201/3,201 valid □  (after cutoff)
# Forks:            0 detected □
# Verification:     PASSED (12ms)

# Verify a specific range of events
vault verify-chain --from evt:04000 --to evt:04847
```

```
# Inspect a single event's chain properties
vault inspect evt:04500

# Output:
# Event:     evt:04500
# ID:        sha256:e7f8g9h0...
# Prev:      sha256:d6e7f8g9... (evt:04499)
# Next:      sha256:f8g9h0i1... (evt:04501)
# Signature: ed25519:j2k3l4m5... (principal: dexter)
# Timestamp: 2026-03-08T09:15:22Z
# Type:      MINT:WORK
# Data hash: sha256:a0b1c2d3... (matches ID □ )

# Export chain for external audit
vault export-chain --format json > chain-audit.json
vault export-chain --format csv > chain-audit.csv
```

Run `verify-chain` before any audit, after any incident, and on a schedule. The verification is idempotent and non-destructive — it reads, recomputes, and reports. It never modifies the chain [12].

17.7. 17.7 The Genesis Event

Every chain starts with a genesis event. The genesis event has no predecessor (`prev: null`) and establishes the chain's root of trust:

```
{
  "id": "sha256:0000000000000001",
  "prev": null,
  "type": "CHAIN:GENESIS",
  "timestamp": "2025-09-01T00:00:00Z",
  "data": {
    "scope": "hadleylab-canonic",
    "algorithm": "sha256",
    "signature_algorithm": "ed25519",
    "genesis_principal": "dexter",
    "genesis_pubkey": "ed25519:pk_a1b2c3d4..."
  },
  "signature": "ed25519:genesis_sig_a1b2..."
}
```

Only the genesis event may have `prev: null` — any other event with `prev: null` is a chain violation. The

genesis data declares the cryptographic parameters for the entire chain: hash algorithm, signature algorithm, and establishing principal [12].

Creating a genesis event:

```
# Initialize a new chain (only for new deployments)
vault init-chain --principal dexter --algorithm sha256

# Output:
# Genesis event created: sha256:0000000000000001
# Chain initialized with:
#   Hash algorithm:      SHA-256
#   Signature algorithm: Ed25519
#   Genesis principal:   dexter
#   Genesis pubkey:      ed25519:pk_a1b2c3d4...
# LEDGER event: CHAIN:GENESIS recorded
```

Immutable once created — cannot be modified, replaced, or deleted. Every subsequent event's integrity traces back to genesis through the hash chain.

17.8. 17.8 Chain Compaction

Chains grow. Seventy-three scopes producing multiple events per day accumulate thousands of events per month. Compaction reduces verification time without sacrificing integrity:

```
Full chain:     evt:00001 → evt:00002 → ... → evt:04847
Compacted:      checkpoint:Q1-2026 → evt:04001 → ... → evt:04847
```

The checkpoint is a hash commitment over all events in the compacted range:

```
{
  "id": "sha256:checkpoint_q1_2026",
  "type": "CHAIN:CHECKPOINT",
  "range": {"from": "evt:00001", "to": "evt:04000"},
  "merkle_root": "sha256:m3r4k5l6...",
  "event_count": 4000,
  "timestamp": "2026-04-01T00:00:00Z"
}
```

Compaction rules:

Rule	Constraint
Minimum age	Only events older than 90 days can be compacted
Checkpoint granularity	Quarterly (Q1, Q2, Q3, Q4)
Original preservation	Compacted events are archived, not deleted
Verification equivalence	`verify(checkpoint + tail)` must equal `verify(full chain)`
Reversibility	Archive can be restored to reconstruct full chain

```
# Compact the chain (quarterly operation)
vault compact-chain --quarter Q1-2026

# Output:
# Compacting events evt:00001 through evt:04000...
# Computing Merkle root: sha256:m3r4k5l6...
# Creating checkpoint: CHAIN:CHECKPOINT
# Archiving compacted events to R2...
# LEDGER event: CHAIN:COMPACT recorded
# Verification time reduced: 12ms → 4ms (67% improvement)

# Verify compacted chain
vault verify-chain

# Output:
# Checkpoint Q1-2026: merkle root valid □  (covers evt:00001-04000)
# Active chain:       847 events valid □  (evt:04001-04847)
# Full integrity:     PASSED (4ms)
```

17.9. 17.9 Dual-Write Architecture

Every economic event is written to two locations simultaneously:

```
Event → USER timeline (per-principal view)
      → LEDGER (global, hash-chained view)
```

Both writes must succeed. If either write fails, the entire event is rolled back:

```
def write_event(event, principal):
    # Begin atomic dual-write
    user_result = write_to_user_timeline(principal, event)
    ledger_result = write_to_ledger(event)

    if user_result.ok and ledger_result.ok:
```

```
        return SUCCESS
    else:
        rollback(user_result)
        rollback(ledger_result)
        log_error("CHAIN:DUAL_WRITE_FAIL", event)
        return FAILURE
```

A principal's timeline must be a subsequence of the LEDGER — every event in a principal's timeline must also appear globally. `vault reconcile` verifies this invariant:

```
# Reconcile all user timelines against the LEDGER
vault reconcile

# Output:
# Principal       Timeline Events    LEDGER Matches    Status
# dexter          1,247              1,247             OK □
# isabella        892                892               OK □
# dr-martinez     456                456               OK □
# dr-chen         312                312               OK □
# [OK] All timelines reconcile with LEDGER
```

Reconciliation failure indicates a dual-write bug. The report shows exactly which events are mismatched [12].

17.10. 17.10 Chain Branching for Federation

In federated deployments (Chapter 9), each organization maintains its own hash chain. The federation protocol cross-references chains without merging them:

```
HadleyLab chain:     genesis → ... → evt:04847 (head)
PartnerHospital chain: genesis → ... → evt:02156 (head)

Federation cross-reference:
  HadleyLab evt:04500 references PartnerHospital evt:02100
  (MammoChat score shared across federation boundary)
```

Cross-references are recorded as special CHAIN events:

```
{
  "id": "sha256:cross_ref_001",
  "type": "CHAIN:CROSS_REF",
```

```
  "local_event": "evt:04500",
  "remote_chain": "partner-hospital",
  "remote_event": "evt:02100",
  "remote_hash": "sha256:r3m0t3..."
}
```

Each chain remains independently verifiable. Cross-references create auditable links between them, enabling federation-level audit: "When HadleyLab shared MammoChat's score with PartnerHospital, PartnerHospital's chain recorded the receipt."

17.11. 17.11 CHAIN Error Handling

Chain errors indicate integrity violations or system failures:

Error	Severity	Cause	Resolution
`CHAIN_HASH_MISMATCH`	CRITICAL	Event content modified after write	Investigate: compare event data to stored hash
`CHAIN_LINK_BROKEN`	CRITICAL	Event predecessor hash does not match	Investigate: identify missing or reordered event
`CHAIN_FORK_DETECTED`	CRITICAL	Two events share same predecessor	Investigate: identify dual-write race condition
`CHAIN_SIGNATURE_INVALID`	HIGH	Event signature does not verify	Check: principal key rotation or key compromise
`CHAIN_DUAL_WRITE_FAIL`	HIGH	Event written to one location only	Run `vault reconcile` to identify mismatch
`CHAIN_GENESIS_MISSING`	CRITICAL	No genesis event found	Chain corrupted — restore from backup

CRITICAL errors halt all economic operations. No minting, spending, or transferring COIN with compromised chain integrity. Fail-closed: if the integrity layer cannot guarantee tamper-evidence, the economy stops [12].

```
# Check chain health
vault chain-health

# Output (healthy):
# Chain status:    HEALTHY
# Last verified:   2026-03-10T14:32:01Z (8 seconds ago)
# Head:            sha256:a1b2c3d4...
```

```
# Length:          4,847 events
# Errors:          0

# Output (unhealthy):
# Chain status:    CRITICAL
# Error:           CHAIN_HASH_MISMATCH at evt:04501
# Expected hash:   sha256:f8g9h0i1...
# Actual hash:     sha256:x9y0z1a2...
# Economic ops:    HALTED
# Action:          Investigate evt:04501 immediately
```

17.12. 17.12 CHAIN Performance

Performance characteristics:

Operation	Complexity	Time (4,847 events)	Time (100,000 events)
Append event	O(1)	<1ms	<1ms
Verify full chain	O(n)	12ms	247ms
Verify from checkpoint	O(n-c)	4ms	82ms
Fork detection	O(1) per event	<1ms	<1ms
Reconcile all users	O(u * e)	45ms	920ms
Export chain	O(n)	8ms	164ms

Append is O(1) — only the current head hash is needed. Verification is O(n) but fast: SHA-256 is hardware-accelerated on modern CPUs. For 100,000 events, full verification completes in under 250ms. Compaction keeps the active chain short for routine checks [12].

17.13. 17.13 Clinical Vignette: Chain Integrity During FDA Audit

Cleveland Clinic (23 hospitals) receives an FDA 510(k) review for PathChat — a pathology decision support system for breast cancer biopsy interpretation (DCIS grading, invasive staging per AJCC 8th ed., HER2/neu IHC scoring). The FDA reviewer requests software change control evidence per 21 CFR Part 820.30 and audit trail integrity per Part 11.

The governance team exports the hash chain covering PathChat's 18-month lifecycle — 2,847 events:

```
vault export-chain --scope SERVICES/TALK/PATHCHAT \
  --format json \
  --from genesis \
  --to head \
```

```
  > pathchat-chain-audit.json

vault verify-chain --scope SERVICES/TALK/PATHCHAT
# Chain length:     2,847 events
# Genesis:          sha256:0000...0001 (2024-09-01)
# Head:             sha256:f8a3b7c2... (2026-03-10)
# Content hashes:   2,847/2,847 valid □
# Chain links:      2,846/2,846 valid □
# Signatures:       2,412/2,412 valid □  (after cutoff)
# Forks:            0 detected □
# Verification:     PASSED (34ms)
```

Each FDA requirement maps to a CHAIN query:

FDA Requirement	21 CFR Section	CHAIN Evidence	Query
Design history file	820.30(j)	Complete commit chain from genesis	`vault export-chain --full`
Design input records	820.30(c)	INTEL.md changes referencing AJCC, CAP, ASCO	`vault timeline --filter INTEL`
Design verification	820.30(f)	`magic validate` events (score = 255)	`vault timeline --filter VALIDATE`
Design validation	820.30(g)	Clinical testing events and LEARNING entries	`vault timeline --filter LEARNING`
Audit trail integrity	11.10(e)	Hash chain verification report	`vault verify-chain`
Electronic signatures	11.50	Ed25519 signatures on all events post-cutoff	`vault verify-chain --signatures`
Signature authority	11.100(a)	VITAE.md linking signer to credentials	`vault credentials --scope PATHCHAT`

The reviewer zeros in on evt:01847 — a MINT:WORK event from 7 months ago where the HER2 scoring algorithm was updated. “Has this event been modified since it was recorded?”

```
vault inspect evt:01847
# Event:     evt:01847
# ID:        sha256:c4d5e6f7a8b9...
# Prev:      sha256:b3c4d5e6f7a8... (evt:01846)
# Next:      sha256:d5e6f7a8b9c0... (evt:01848)
# Signature: ed25519:sig_m7n8o9p0... (principal: dr-pathak)
# Timestamp: 2025-08-15T11:42:33Z
# Type:      MINT:WORK
# Scope:     SERVICES/TALK/PATHCHAT/HER2-SCORING
# Data:
#   file_changed: INTEL.md
#   evidence_added: "ASCO/CAP HER2 Testing Guideline 2023"
#   score_before: 247
#   score_after: 255
#   gradient: +8
#   coin_minted: 8
# Data hash verification: sha256(data) = sha256:c4d5e6f7a8b9... □ (matches ID)
# Chain link verification: evt:01846.id = sha256:b3c4d5e6f7a8... □ (matches prev)
# Signature verification: ed25519_verify(data, sig, dr-pathak.pubkey) □
```

Three independent checks confirm the event is unmodified: content hash matches stored ID, predecessor link matches the previous event, Ed25519 signature verifies against the signer's public key. Altering any field breaks at least one check.

The FDA reviewer confirms the audit trail meets 21 CFR Part 11 — and notes that cryptographic verification exceeds the regulatory bar. Part 11 requires "system-enforced, time-stamped audit trails." The CHAIN provides mathematical proof [1215].

17.14. 17.14 Chain Migration and Key Rotation

When a principal rotates their Ed25519 key, the CHAIN records the rotation:

```
{
  "id": "sha256:key_rotation_001",
  "type": "CHAIN:KEY_ROTATE",
  "principal": "dr-pathak",
  "old_pubkey": "ed25519:pk_old_a1b2...",
  "new_pubkey": "ed25519:pk_new_c3d4...",
  "rotation_reason": "annual_rotation",
  "effective_from": "evt:02500"
}
```

Key rotation rules:

Rule	Constraint	Rationale
Signed by old key	Rotation event must be signed by the key being rotated	Prevents unauthorized rotation
Forward-only	Events before rotation are verified with old key	No retroactive re-signing
Grace period	Both keys valid for 72 hours after rotation	Prevents lockout
LEDGER event	KEY_ROTATE is a hash-chained event	Rotation is auditable
No key reuse	A rotated key cannot be reactivated	Prevents downgrade attacks

```
vault key-rotate dr-pathak --reason annual_rotation
# Generating new Ed25519 key pair...
# New public key: ed25519:pk_new_c3d4...
# Signing rotation event with current key...
# CHAIN event: KEY_ROTATE recorded (evt:02500)
# Grace period: 72 hours (both keys valid until 2026-03-13T14:00:00Z)
# After grace: only new key accepts signatures
```

17.15. 17.15 Chain Anchoring to External Systems

For external tamper-evidence, the CHAIN supports anchoring — periodically publishing a hash commitment to an immutable external store:

```
vault anchor --epoch 2026-03
# Computing Merkle root of all ORG DIGESTs...
# ORGs: 8 (sorted alphabetically)
# Merkle root: sha256:m3r4k5l6...
# Broadcasting Bitcoin OP_RETURN: <CANONIC> <2026-03> <merkle_root_32bytes>
# Bitcoin txid: abc123def456...
# Anchor recorded: CHAIN:ANCHOR event (evt:04848)
# External verification: any block explorer — no CANONIC software needed

vault anchor --target timestamping --authority rfc3161
# Computing anchor hash...
# Requesting RFC 3161 timestamp from timestamp.digicert.com
# Timestamp token received: 2026-03-10T15:00:00Z
# Anchor recorded: CHAIN:ANCHOR event (evt:04849)
```

Anchoring options:

Target	Mechanism	Verification	Use Case
Bitcoin OP_RETURN	Monthly Merkle root in 45-byte payload	Any block explorer	Primary — public tamper-evidence
RFC 3161	Trusted timestamp authority	TSA certificate chain	Regulatory compliance
Notary	Public notary service	Notary certificate	Legal proceedings

Anchoring is optional — the internal hash chain suffices for most audit requirements. External anchoring adds an independent witness that the chain state existed at a specific time, providing the additional evidentiary weight that FDA submissions and legal proceedings may demand.

17.16. 17.16 Governance Proof: Chain Integrity

The CHAIN's integrity guarantee rests on three cryptographic properties:

```
Property 1: Collision Resistance (SHA-256)
  P(sha256(x) == sha256(y) where x ≠ y) < 2^-128
  Practical: no known collision in SHA-256 family
  Therefore: each event's ID uniquely identifies its content

Property 2: Chain Linkage
  For events e₁, e₂, ..., eₙ:
    eᵢ.prev = eᵢ₋₁.id for all i > 1
    e₁.prev = null (genesis)
  Modifying eₖ changes eₖ.id
  But eₖ₊₁.prev stores old eₖ.id
  Therefore: modification of any event breaks the chain at eₖ₊₁
  Detection: O(n) verification walk

Property 3: Non-Repudiation (Ed25519)
  For event e signed by principal p:
    ed25519_verify(e.data, e.signature, p.pubkey) = TRUE
  The private key never leaves the principal's device
  Therefore: only the principal could have produced the signature
  Forging a signature requires breaking Ed25519 (infeasible)
```

The three properties compose: collision resistance ensures content integrity, chain linkage ensures ordering integrity, non-repudiation ensures attribution integrity. If any event is modified, inserted, deleted, or reordered, the chain breaks at a detectable point. The CHAIN does not prevent tampering — it makes tampering detectable. Detection is sufficient for governance. Q.E.D. [1215]

Chapter 18

Chapter 18: MINT

Gradient minting. The mechanism that turns governance work into COIN [15]. MINT implements the gradient rule introduced in Chapter 5 and extended formally in Chapter 33. Every MINT event is recorded on the LEDGER (Chapter 13) and credited to a WALLET (Chapter 14). RUNNER tasks (gorunner.pro) drive automated minting at scale.

18.1. 18.1 Axiom

MINT is the gradient function. WORK produces score changes. Score changes produce COIN. Only improvement mints [12].

Every COIN in circulation traces to a specific act of governance work. MINT converts that labor — creating scopes, writing CANON.md files, building INTEL layers, validating to 255 — into COIN, and the conversion is mathematical: COIN minted equals the governance score delta. Build a scope from 0 to 35: mint 35. Advance from 35 to 127: mint 92. Push to 255: mint 128. Total: 255 COIN for the full governance lifecycle of one scope [12].

18.2. 18.2 The Gradient Rule

```
gradient = to_bits - from_bits
```

The gradient is the signed difference between the scope's score after the work and the score before the work. Three outcomes:

Gradient	Economic Effect	LEDGER Event	Example
Positive	COIN minted	MINT:WORK	MammoChat advances from 127 to 255 → mint 128
Negative	COIN debited	DEBIT:DRIFT	OncoChat regresses from 255 to 191 → debit 64
Zero	No COIN event	None	Scope stays at 255 after maintenance commit

Only improvement mints. Staying at 255 mints zero — nothing to improve. Going backward costs COIN through DEBIT:DRIFT. The incentive structure is clean: build it right, keep it right, and governance is free. Let it drift, and it costs you [12] [22].

18.3. 18.3 Maximum Mint Per Scope

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

A scope can mint at most 255 COIN over its entire lifecycle — the theoretical maximum for 0-to-FULL compliance. Build incrementally across multiple commits and you still get 255 COIN total, just spread across multiple MINT:WORK events [10].

The maximum is absolute. No admin override. No special tier. No exception. Score range is 0-255, so the maximum gradient sum is 255.

18.4. 18.4 Supply Ceiling

```
SUPPLY_CEILING = unique_scopes * 255
```

Total COIN supply is bounded: scope count times 255. A health network with 500 governed scopes has a ceiling of 127,500 COIN [12]. For the formal proof that the supply ceiling converges and cannot be inflated, see Chapter 33, Section 33.5 (Supply Ceiling Proof).

No override. No quantitative easing. No central bank. The only way to expand supply is to create new scopes — to do new work. Every COIN in circulation represents real governance labor on a specific scope. Supply cannot be inflated by decree; it can only be expanded by work [12].

18.5. 18.5 RUNNER Tasks

MINT operates via RUNNER tasks — discrete governance operations that produce measurable gradients:

RUNNER Task	Operation	Typical Gradient	Example
`validate`	Run `magic validate` on scope	Variable (depends on starting score)	New scope: 0 → 35 = +35
`heal`	Run `magic-heal` to fix governance gaps	Variable (depends on gap size)	Missing VOCAB.md: +2
`commit`	Commit governance work to repository	Variable (depends on changes)	Add COVERAGE.md: +16
`review`	Review and approve external contribution	Fixed: +5 per review	Approve CONTRIBUTE PR
`certify`	Tag scope at 255 with git tag	0 (maintenance, no gradient)	Annual re-certification

Every task follows the same pattern: measure before, execute, measure after, compute gradient, mint accordingly. Execution and minting are atomic — you cannot get the COIN without the validator confirming the gradient [15].

18.6. 18.6 MINT and the Healthcare Governance Economy

MINT makes governance investment visible and measurable:

Governance labor accounting: Total COIN minted by a clinical informatics team equals total governance work performed. When the CMO asks "how much governance work has been done?" — the answer is the sum of MINT:WORK events on the LEDGER, auditable by anyone.

Department comparison: Department A minted 12,750 COIN across 50 scopes. Department B minted 2,550 across 10. The comparison is objective — derived from validated governance events, not subjective assessment.

ROI calculation: $500K in governance labor minting 50,000 COIN equals $10/COIN. If that work produces compliance documentation that would have cost $2M through traditional consulting, governance ROI is 4x. COIN provides the denominator for ROI calculations that were previously impossible [12] [22].

18.7. 18.7 MINT CLI Operations

Every MINT event originates from a CLI command or an automated RUNNER task. The operator interface is deterministic.

Checking the current score before work:

```
$ magic validate SERVICES/TALK/MAMMOCHAT
Scanning SERVICES/TALK/MAMMOCHAT...
  CANON.md        ✓ (axiom, inherits, privacy, notify present)
  INTEL.md        ✓ (3 sources, all cited)
  VOCAB.md        ✓ (12 terms defined)
  COVERAGE.md     ✓ (ACR BI-RADS 5th Edition mapped)
  LEARNING.md     ✓ (7 patterns recorded)
  SOP.md          ✓ (SOP-001 through SOP-008)
  DISCLAIMER.md   ✓ (FDA 21 CFR Part 11 referenced)
Score: 255/255 (FULL)
```

Performing governance work that produces a gradient:

```
$ magic validate SERVICES/TALK/ONCOCHAT
Scanning SERVICES/TALK/ONCOCHAT...
  CANON.md        ✓
  INTEL.md        ✓
  VOCAB.md        ✗ (missing: VOCAB.md not found)
  COVERAGE.md     ✗ (missing: COVERAGE.md not found)
  LEARNING.md     ✓
Score: 191/255 (SERVICE tier)
```

The operator creates VOCAB.md and COVERAGE.md. Then re-validates:

```
$ magic validate SERVICES/TALK/ONCOCHAT
Score: 255/255 (FULL)
Gradient: 255 - 191 = +64
MINT:WORK → 64 COIN credited to DEXTER wallet
LEDGER event: evt:00312 MINT:WORK scope=ONCOCHAT from=191 to=255 coin=64
```

No separate “claim COIN” step. The validator confirms the gradient, the LEDGER records the event, and the WALLET receives the credit — one operation, three side effects, all deterministic [12].

Querying MINT history for a scope:

```
$ vault ledger --scope SERVICES/TALK/MAMMOCHAT --type MINT
evt:00102  2025-11-15T10:00:00Z  MINT:WORK    from=0   to=35   coin=35
evt:00145  2025-12-01T14:30:00Z  MINT:WORK    from=35  to=127  coin=92
evt:00198  2026-01-10T09:15:00Z  MINT:WORK    from=127 to=255  coin=128
Total minted: 255 COIN (ceiling reached)
```

Full minting history — every gradient event, every COIN credit, every timestamp. Immutable. The ceiling is visible: 255 COIN minted equals the maximum for a single scope [15].

18.8. 18.8 MINT Data Structures

The MINT event is recorded as a JSON object on the LEDGER. The structure is fixed — no optional fields, no nullable values:

```
{
  "event": "MINT:WORK",
  "id": "evt:00312",
  "ts": "2026-03-10T09:15:00Z",
  "scope": "SERVICES/TALK/ONCOCHAT",
  "principal": "DEXTER",
  "from_score": 191,
  "to_score": 255,
  "gradient": 64,
  "coin": 64,
  "commit": "abc1234def5678",
  "validator_version": "magic-v3.2.1",
  "signature": "ed25519:a1b2c3d4..."
}
```

Field	Type	Constraint	Purpose
`event`	string	Enum: `MINT:WORK`, `MINT:SIGNUP`, `MINT:REFER`	Event classification
`id`	string	Monotonic, unique	LEDGER sequence position
`ts`	ISO-8601	UTC, no timezone offset	Temporal ordering
`scope`	string	Valid scope path	What was worked on
`principal`	string	Verified IDENTITY	Who did the work
`from_score`	uint8	0-255	Score before work
`to_score`	uint8	0-255	Score after work
`gradient`	int16	-255 to +255	Signed difference

Field	Type	Constraint	Purpose
`coin`	uint8	0-255	COIN minted (only positive gradients)
`commit`	string	Git SHA	Provenance anchor
`validator_version`	string	SemVer	Reproducibility
`signature`	string	Ed25519	Non-repudiation

After the Ed25519 cutoff, every MINT event is cryptographically signed by the principal who performed the work. The signature covers the entire payload — tamper with any field and the signature invalidates. Append-only LEDGER, non-repudiable signatures, forensically sound history [15].

18.9. 18.9 DEBIT: The Negative Gradient

MINT is bidirectional. Positive gradients mint COIN. Negative gradients debit COIN. The DEBIT mechanism is the economic cost of governance neglect.

Drift scenario — MammoChat model update breaks validation:

A team updates MammoChat's underlying LLM. The model change alters response formatting, and DISCLAIMER.md disclaimers no longer render correctly. The next `magic validate` returns 191 instead of 255.

```
$ magic validate SERVICES/TALK/MAMMOCHAT
Score: 191/255 (SERVICE tier — regression detected)
Gradient: 191 - 255 = -64
DEBIT:DRIFT → 64 COIN debited from DEXTER wallet
LEDGER event: evt:00347 DEBIT:DRIFT scope=MAMMOCHAT from=255 to=191 coin=-64
NOTIFIER: DRIFT_ALERT sent to DEXTER
```

Debit and alert are both automatic — a single validation pass detects the regression, records the economic cost, and notifies the responsible principal. Restoring to 255 mints +64 COIN, recovering the debit [12] [22].

Vignette: Oncology department drift cascade. OncoChat runs across three campuses (Orlando, Tampa, Jacksonville), all referencing NCCN breast cancer guidelines Version 2.2026. NCCN releases Version 3.2026, upgrading the pembrolizumab + chemotherapy recommendation for triple-negative breast cancer from Category 2A to Category 1. Orlando's INTEL.md still references 2.2026. Score drops from 255 to 239 (INTEL freshness penalty: -16). DEBIT:DRIFT: 16 COIN per campus, 48 COIN total. The team updates INTEL.md to reference 3.2026, re-validates, and mints +48 COIN across the fleet. NCCN release to fleet-wide update: 72 hours. The LEDGER records the entire lifecycle [12] [15].

18.10. 18.10 Epoch-Based Supply Analysis

The CANONIC economy operates in epochs — discrete time periods (typically quarterly) over which MINT activity is aggregated for reporting.

```
$ vault economy --epoch 2026-Q1
Epoch: 2026-Q1 (Jan 1 - Mar 31)
Scopes governed:     73
MINT:WORK events:    312
DEBIT:DRIFT events:  18
Net COIN minted:     4,230
Supply ceiling:      73 * 255 = 18,615
Utilization:         4,230 / 18,615 = 22.7%
```

Epoch Metric	Formula	Interpretation
Net minted	SUM(positive gradients) - SUM(negative gradients)	Total governance labor value
Supply ceiling	scope_count * 255	Maximum theoretical supply
Utilization	net_minted / supply_ceiling	How much governance capacity was exercised
Velocity	net_minted / epoch_days	Daily governance throughput
Drift ratio	DEBIT events / total events	Governance stability indicator

High utilization (>80%) means most scopes are at 255 and new COIN can only come from new scopes. High drift ratio (>10%) means governance is unstable — scopes regressing faster than advancing. These metrics give executives the view they need [12].

18.11. 18.11 MINT Error Handling and Failure Modes

MINT operations can fail. Every failure mode has a defined behavior:

Failure Mode	Cause	System Response	Recovery
`VALIDATE_FAIL`	Scope has governance gaps	No MINT event, error returned	Fix gaps, re-validate
`CEILING_REACHED`	Scope already at 255	Gradient = 0, no COIN	Scope is fully governed

Failure Mode	Cause	System Response	Recovery
`WALLET_NOT_FOUND`	Principal has no wallet	MINT blocked, error returned	Run `vault signup` first
`SIGNATURE_FAIL`	Ed25519 key invalid or expired	MINT blocked, error returned	Run `vault keygen` to rotate key
`LEDGER_CORRUPT`	LEDGER hash chain broken	All operations halted	Run `vault ledger --verify`, rebuild from last valid event
`CONCURRENT_MINT`	Two RUNNER tasks modify same scope	Second task blocked by scope lock	Retry after lock release

`CONCURRENT_MINT` deserves attention. The scope locks during validation and unlocks after the MINT event is recorded. If two operators both advance a scope from 191 to 255, only the first receives 64 COIN — the second sees gradient 0 and mints nothing. The scope lock is the serialization mechanism that ensures supply integrity [15].

18.12. 18.12 MINT Integration with WALLET

MINT events flow directly into the WALLET service. The WALLET maintains per-principal balances:

```
$ vault wallet DEXTER
Principal: DEXTER
Balance: 12,750 COIN
  MINT:SIGNUP    500 (initial bonus)
  MINT:WORK      12,890 (governance labor)
  MINT:REFER     200 (referral bonuses)
  DEBIT:DRIFT    -640 (governance regressions)
  SPEND:PURCHASE -200 (product purchases)
Net: 12,750 COIN
```

The WALLET is a projection of the LEDGER, not a separate store. Balance is derived by replaying all MINT, DEBIT, and SPEND events. If the WALLET disagrees with the LEDGER, the LEDGER wins — recalculate and move on. The WALLET cannot be corrupted independently of the LEDGER [12].

18.13. 18.13 Clinical Vignette: FinChat Compliance Mint

FinChat serves real estate professionals — different regulatory requirements (no FDA, no HIPAA) but identical MINT mechanics.

A brokerage deploys FinChat for market analysis. Governance work: CANON.md, INTEL.md (MLS data sources, county property appraiser), VOCAB.md (CAP rate, NOI, GRM), COVERAGE.md (Florida Statute 475), DISCLAIMER.md (not legal or financial advice). Each file advances the score:

```
CANON.md created:        0 → 35    MINT:WORK +35
INTEL.md created:       35 → 67    MINT:WORK +32
VOCAB.md created:       67 → 99    MINT:WORK +32
COVERAGE.md created:    99 → 191   MINT:WORK +92
DISCLAIMER.md created: 191 → 223   MINT:WORK +32
SOP.md created:        223 → 255   MINT:WORK +32
                                   Total: 255 COIN
```

Six commits to 255. Total: 255 COIN. The gradient rule is domain-agnostic — clinical AI, financial AI, real estate AI. Zero to 255 always mints 255 COIN. Governance requirements vary by domain (Florida Statute 475, not NCCN guidelines), but the economic mechanism is universal [12] [22].

18.14. 18.14 MINT Batch Operations

Batch validation and minting for enterprise-scale deployments:

```
# Batch validate and mint across all scopes in a service tree
$ magic validate --recursive SERVICES/TALK/ --mint
Scanning SERVICES/TALK/...
  MAMMOCHAT:     255/255 (no change)       gradient: 0
  ONCOCHAT:      191→255 (healed)          gradient: +64   MINT:WORK
  MEDCHAT:       127→191 (partial advance)  gradient: +64   MINT:WORK
  DERMCHAT:      0→35 (bootstrap)          gradient: +35   MINT:WORK
  EMERGECHAT:    255/255 (no change)       gradient: 0
Summary: 5 scopes, 3 MINT:WORK events, 163 COIN minted
```

Batch is syntactic sugar — each scope validates independently, each gradient computes independently, each MINT event records independently on the LEDGER. No new economic mechanism. Same atomic rules.

Integrate into a nightly cron sweep:

```
# Nightly governance sweep (crontab entry)
0 2 * * * magic validate --recursive SERVICES/ --mint --report /var/log/canonic/nightly.json
```

The nightly report captures fleet-wide governance state in machine-readable JSON. The sweep runs while clinical staff sleeps; by morning, the dashboard reflects overnight state.

18.15. 18.15 MINT and Multi-Author Scopes

When multiple authors contribute to a single scope, COIN is attributed to the author who commits the gradient-producing change:

Commit	Author	Change	Gradient	COIN Recipient
abc123	Dr. Chen	Create CANON.md + VOCAB.md	0 35 (+35)	Dr. Chen
def456	Dr. Park	Add COVERAGE.md + ROADMAP.md	35 63 (+28)	Dr. Park
ghi789	Dr. Chen	Add LEARNING.md	63 127 (+64)	Dr. Chen
jkl012	Dr. Park	Close to 255	127 255 (+128)	Dr. Park

Dr. Chen: 99 COIN. Dr. Park: 156 COIN. Total: 255. Attribution is per-commit, not per-scope — multiple authors share COIN in proportion to their gradient contributions. The LEDGER records it permanently. No subjective allocation. No manager discretion. The gradient determines the split.

18.16. 18.16 Closure

MINT is the gradient function. Score delta measures governance work. Work produces COIN. COIN is bounded by scope ceiling. Ceiling is bounded by scope count. Scope count is bounded by real organizational need. No inflation, no arbitrary supply. The only way to earn COIN is to improve governance, and the only way governance improves is through work. MINT converts labor into value [12] [15].

Chapter 19

Chapter 19: IDENTITY

Ed25519 keys. KYC anchors. The cryptographic foundation of governed identity [15]. IDENTITY anchors every principal in the CANONIC economy: their WALLET (Chapter 14), their CHAIN signatures (Chapter 17), and their VAULT access (Chapter 15) all trace to the key binding established here. For cross-organization identity in federated deployments, see Chapter 9.

19.1. 19.1 Axiom

IDENTITY anchors every USER to a verifiable real-world identity. Ed25519 keys sign events. VITAE.md is the evidence gate [19].

An ungoverned economy can tolerate anonymous actors. A governed one cannot. IDENTITY ensures every actor is who they claim to be through three layers: GitHub OAuth as the initial anchor, VITAE.md as the credential declaration (publications, licenses, affiliations), and Ed25519 key binding for cryptographic attribution on every LEDGER event [15].

19.2. 19.2 The KYC Pipeline

Know Your Customer (KYC) in CANONIC is a four-stage pipeline:

```
GitHub OAuth → VITAE.md declaration → Credential verification → Ed25519 key binding
```

Stage 1: GitHub OAuth. The user authenticates via GitHub. Their GitHub username becomes the principal identifier throughout the CANONIC ecosystem. The GitHub account is the minimum viable identity — it proves the user controls an account [15].

Stage 2: VITAE.md declaration. The user creates a VITAE.md file in their USER scope declaring their credentials:

```
# DEXTER — VITAE

## Credentials

| Type | Credential | Issuer | Status |
|------|------------|--------|--------|
| MD | Doctor of Medicine | University of California, San Francisco | Active |
| PhD | Doctor of Philosophy (Biomedical Informatics) | UCSF | Conferred |
| License | FL Medical License | FL DOH | Active |
| Patent | US Patent Application | USPTO | Pending |
```

Stage 3: Credential verification. The IDENTITY service verifies declared credentials against issuing authorities: state medical boards (for physician licenses), NPPES (for NPI numbers), USPTO (for patents), PubMed (for publications). Verification status is recorded in VITAE.md [19].

Stage 4: Ed25519 key binding. The user generates an Ed25519 key pair via `vault keygen`. The public key is bound to the user's identity. The private key signs all subsequent LEDGER events. The key binding is a LEDGER event itself — cryptographic attribution begins at the moment of binding [15].

19.3. 19.3 Onboarding Flow

New users enter the CANONIC ecosystem through a governed onboarding flow:

```
$ vault signup dexter
Creating USER scope: DEXTER/
  → CANON.md created (axiom: DEXTER is a governed principal)
  → VITAE.md created (credentials template)
  → Generating Ed25519 key pair...
  → Public key: ed25519:a1b2c3d4e5f6...
  → MINT:SIGNUP — 500 COIN credited to WALLET
  → magic validate → score: 35 (COMMUNITY tier)
Welcome to CANONIC. Your starting balance is 500 COIN.
```

The 500 COIN signup bonus provides enough to explore — purchase COMMUNITY-tier products, experiment with governance work, begin climbing tiers. Your first real governance work (adding VOCAB.md, COVERAGE.md, LEARNING.md) mints additional COIN [11].

19.4. 19.4 The SOP-009/010/011 Pipeline

For batch onboarding, IDENTITY provides a Standard Operating Procedure pipeline:

SOP	Step	Action	Evidence
SOP-009	Network mapping	Trace user to VITAE.md	LEARNING/{slug}.md created
SOP-010	KYC verification	Directory name = node name = legal identity	VITAE.md verified
SOP-011	4-channel scan	Scan GitHub, LinkedIn, publications, patents	INTEL backpropagated

SOP-011 scans four channels (GitHub activity, LinkedIn profile, publication record, patent portfolio) to build a complete professional picture. Results are stored in LEARNING/{slug}.md. Not surveillance — governed discovery of publicly available professional information [20].

19.5. 19.5 IDENTITY and Clinical AI Governance

IDENTITY maps directly to clinical AI credentialing requirements:

Practitioner verification: Before a clinician can interact with clinical AI decision support, credentials must be verified against issuing authorities — the same verification hospital credentialing offices perform, automated and on the LEDGER.

Non-repudiation: After key binding, every signed governance event is non-repudiable. A clinician cannot later claim they did not approve a recommendation their key signed — satisfying FDA 21 CFR Part 11 §11.50.

Audit attribution: When Joint Commission surveyors ask "who approved this AI's deployment?" — the LEDGER shows the specific verified individual, with credentials, at the specific timestamp [19] [3].

19.6. 19.6 IDENTITY Data Model

Every governed principal has a USER scope:

```
USERS/DEXTER/
  ├── CANON.md        ← axiom, tier, governance constraints
  ├── VITAE.md        ← credentials, publications, licenses
  ├── VOCAB.md        ← terminology and role definitions
  └── LEARNING/       ← per-user intelligence (from SOP-011 scans)
      ├── github.md
```

```
    ├── pubmed.md
    ├── patents.md
    └── network.md
```

The CANON.md for a USER scope:

```
```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

tier: AGENT

• • •

19.7. Axiom

DEXTER is a governed principal. Physician-informaticist. Founder. Tier 127.

19.8. Readers

- dexter
- isabella

19.9. Writers

- dexter

```
## 19.7 VITAE.md Verification States

| State | Meaning | Transition |
|-------|---------|-----------|
| DECLARED | User has declared the credential | Initial state on creation |
| PENDING | Verification request submitted | `vault verify-credential` initiated |
| VERIFIED | Issuer confirmed | Response received and validated |
| EXPIRED | Past expiration date | Automatic on expiration date |
```
```

```
| REVOKED | Issuer has revoked | Issuer notification received |
| DISPUTED | Verification failed | Manual review required |

```bash
vault verify-credential dexter --type "MD" --issuer "UCSF"
# Status: DECLARED → PENDING
# LEDGER event: IDENTITY:VERIFY_REQUEST recorded

vault credentials dexter
# Type     Credential              Issuer    Status     Verified
# MD       Doctor of Medicine      UCSF      VERIFIED   2025-10-15
# PhD      Doctor of Philosophy    UCSF      VERIFIED   2025-10-15
# License  FL Medical License      FL DOH    VERIFIED   2025-11-01
# Patent   US Patent Application   USPTO     PENDING    —
# NPI      1234567890              CMS       VERIFIED   2025-10-20
```

19.10. 19.8 Ed25519 Key Architecture

Property	Ed25519	RSA-2048	ECDSA P-256
Key size	32 bytes	256 bytes	32 bytes
Signature size	64 bytes	256 bytes	64 bytes
Sign speed	62,000/sec	1,000/sec	14,000/sec
Verify speed	28,000/sec	10,000/sec	5,000/sec
Deterministic	Yes	No	No

Ed25519 is deterministic — same message and key always produce the same signature. Critical for CHAIN reproducibility [12].

19.11. 19.9 Multi-Identity and Role Separation

```
Person: Dr. Dex Martinez
  ├── DEXTER (principal)     — governance work, code contributions
  ├── DEXTER-CLINICAL        — clinical AI validation
  └── DEXTER-ADMIN           — administrative operations
```

Rule	Constraint
Own key pair per identity	No key sharing between roles
Own WALLET per identity	COIN does not flow implicitly
```

Rule	Constraint
Cross-role transfer requires TRANSFER event	Auditable on LEDGER
Shared VITAE.md	Credentials apply to the person
Own tier per role	Computed from role-specific work

19.12. 19.10 IDENTITY Revocation

```
vault revoke-identity dr-legacy --reason "departure" --effective "2026-03-15"
# 1. Ed25519 key → REVOKED (immediate)
# 2. Sessions → TERMINATED (immediate)
# 3. VAULT access → DENIED (immediate)
# 4. WALLET → FROZEN (effective date)
# 5. USER scope → ARCHIVED (effective date + 30d)
```

Revocation is irreversible for the key but not for the person. Old governance work remains attributed on the LEDGER [15].

19.13. 19.11 IDENTITY and Federation

In federated deployments, cross-organization verification trusts the partner's KYC pipeline, declared in the federation CANON.md. If a partner's score drops below 255, federation trust suspends automatically [2] [15].

19.14. 19.12 Clinical Vignette: Identity Verification Prevents Unauthorized AI Governance

Mass General Brigham (16 hospitals) onboards 40 clinical AI governors — oncologists, radiologists, pathologists, and informaticists. During onboarding, IDENTITY catches a discrepancy.

Principal `dr-harrison` submits VITAE.md claiming: MD from Johns Hopkins, NPI 1234567890, active Massachusetts medical license #12345, board certification in Radiation Oncology from ABR. Stage 3 verification:

```
vault verify-credential dr-harrison --type "MD" --issuer "Johns Hopkins"
# Checking NPPES registry for NPI 1234567890...
# NPI 1234567890: ACTIVE — Dr. Sarah Harrison, Radiation Oncology
# Checking MA Board of Registration in Medicine...
```

```
# License #12345: ACTIVE — Dr. Sarah Harrison, exp: 2027-06-30
# Checking ABR certification database...
# ABR Radiation Oncology: CERTIFIED — Dr. Sarah Harrison, MOC current
# All credentials VERIFIED □
# LEDGER: IDENTITY:VERIFY (3 credentials, 3 verified)
```

Meanwhile, principal `dr-harrison-2` submits VITAE.md claiming identical credentials — same NPI, same license number, same ABR certification. The IDENTITY service flags the duplicate:

```
vault verify-credential dr-harrison-2 --type "MD" --issuer "Johns Hopkins"
# WARNING: IDENTITY:DUPLICATE_CREDENTIAL
# NPI 1234567890 already verified for principal: dr-harrison
# License #12345 already verified for principal: dr-harrison
# ABR certification already verified for principal: dr-harrison
# Status: DISPUTED
# Action: Manual review required
# LEDGER: IDENTITY:DISPUTE recorded (evt:08100)
```

Investigation reveals `dr-harrison-2` is a research assistant who copied Dr. Harrison's credentials to gain governance privileges — specifically, write access to OncoChat INTEL.md (radiation therapy guidelines). The assistant intended to update guidelines on Dr. Harrison's behalf.

One set of credentials maps to one principal. The assistant onboards with their own credentials (research coordinator) and gets COMMUNITY-tier read access. Write access to clinical INTEL.md requires VERIFIED physician credentials [1519].

Had the duplicate succeeded, the assistant could have modified which radiation therapy guidelines OncoChat uses — potentially referencing outdated hypofractionation protocols (ASTRO 2018) instead of current evidence (ASTRO 2024), affecting dose recommendations for NSCLC patients. Duplicate detection prevents unauthorized governance changes to clinical AI systems.

19.15. 19.13 Identity Lifecycle and Credential Monitoring

Credentials are not verified once — they are monitored continuously:

```
vault credential-monitor --schedule

# Credential monitoring schedule:
# Check type          Frequency    Source              Last check
# Medical license     Monthly      State medical board  2026-03-01
# NPI status          Monthly      NPPES registry       2026-03-01
# Board certification Quarterly    Specialty board      2026-01-15
# DEA registration    Monthly      DEA NTIS             2026-03-01
```

```
# Institutional       Weekly       HR system webhook     2026-03-08
# Publication record  Quarterly    PubMed/ORCID          2026-01-15
```

When a credential lapses, the IDENTITY service triggers an automatic governance response:

```
Credential lapse detected:
  Principal:    dr-martinez
  Credential:   FL Medical License #67890
  Status:       EXPIRED (2026-03-01)
  Previous:     ACTIVE

Response chain:
  1. IDENTITY:CREDENTIAL_LAPSE event → LEDGER (evt:08200)
  2. NOTIFIER: HIGH priority alert → dr-martinez
  3. NOTIFIER: HIGH priority alert → scope owners where dr-martinez has writer access
  4. Scope access: WRITE → READ (automatic downgrade)
  5. WALLET: No freeze (existing COIN preserved)
  6. Timeline: 30 days to resolve before key suspension
```

A lapsed-credential physician cannot modify clinical AI governance. The 30-day grace period allows renewal without losing COIN or history. Once the license is renewed and re-verified, write access restores automatically.

19.16. 19.14 IDENTITY Metrics

The IDENTITY service exposes Prometheus-compatible metrics:

```
canonic_identity_total{status="verified"} 847
canonic_identity_total{status="pending"} 12
canonic_identity_total{status="disputed"} 3
canonic_identity_total{status="revoked"} 8
canonic_credential_checks_total{result="pass"} 4230
canonic_credential_checks_total{result="fail"} 47
canonic_credential_checks_total{result="lapsed"} 12
canonic_signup_total 870
canonic_keygen_total 862
```

These metrics integrate with existing credentialing dashboards. IDENTITY does not replace the hospital's credentialing system — it extends it. A physician credentialed to practice medicine is not automatically credentialed to govern clinical AI. IDENTITY adds that layer [15].

19.17. 19.15 Governance Proof: The Identity Chain

The IDENTITY chain from real-world person to governance authority:

```
Person (Dr. Sarah Harrison) exists in the physical world
  → GitHub account: dr-harrison (OAuth authentication)
  → VITAE.md declaration: MD, NPI, license, certification
  → IDENTITY:VERIFY_REQUEST → NPPES, State Board, ABR
  → All credentials VERIFIED → LEDGER records 3 IDENTITY:VERIFY events
  → vault keygen → Ed25519 key pair generated
  → Public key bound to dr-harrison → LEDGER records IDENTITY:KEYBIND
  → Every subsequent governance event signed by dr-harrison's private key
  → Every LEDGER event attributed to a verified physician
  → Every clinical AI governance change traceable to:
    - A specific person (Dr. Sarah Harrison)
    - With specific credentials (MD, board-certified Radiation Oncology)
    - At a specific time (Ed25519 signature timestamp)
    - From a specific account (GitHub: dr-harrison)
  → Non-repudiation: Dr. Harrison cannot deny authoring the change
  → If credentials lapse: access automatically downgrades
  → If key compromised: vault revoke-identity → immediate termination
```

Unbroken from physical identity to digital governance act. Every clinical AI governance change traces through this chain to a verified, credentialed human being. No anonymous governance. No unverified authority. No credential-less modification of clinical AI behavior. Q.E.D. [15193]

19.18. 19.16 IDENTITY Federation Across Organizations

Identities exist within organizational boundaries but can be recognized across them through mutual attestation — each organization verifies that the partner's identity system meets governance standards.

```
magic identity --federation-status

# Federation Identity Matrix:
#
# Organization         | Identities | Mutual Trust With          | Cross-Org Actions
# hadleylab-canonic    | 47         | adventhealth, cedars, mayo | CONTRIBUTE, REVIEW
# adventhealth-canonic | 23         | hadleylab, mayo            | CONTRIBUTE, REVIEW
# cedars-canonic       | 31         | hadleylab                  | CONTRIBUTE only
# mayo-canonic         | 89         | hadleylab, adventhealth    | CONTRIBUTE, REVIEW, TRANSFER
```

Cross-organization actions are limited by trust agreements. A Mayo Clinic physician can contribute to a hadleylab-canonic scope because mutual trust is established. The LEDGER records both the contributor's home organization and the receiving scope.

```
# Dr. Patel at Mayo contributes to hadleylab-canonic OncoChat
vault contribute \
  --identity dr.patel@mayo-canonic \
  --scope hadleylab-canonic/SERVICES/TALK/ONCOCHAT/NSCLC \
  --federation-trust mayo-hadleylab-2026

# Cross-org contribution:
#   Contributor: dr.patel@mayo-canonic (VERIFIED, Board-Certified Oncology)
#   Target scope: hadleylab-canonic/SERVICES/TALK/ONCOCHAT/NSCLC
#   Federation trust: mayo-hadleylab-2026 (active, expires 2027-03-01)
#   LEDGER (mayo):     CONTRIBUTE:CROSS_ORG_SUBMIT (evt:federation:00845)
#   LEDGER (hadleylab): CONTRIBUTE:CROSS_ORG_RECEIVE (evt:08845)
```

The dual-LEDGER recording ensures both organizations have audit trails for the cross-organizational knowledge transfer [1519].

19.19. 19.17 IDENTITY Audit Reports

For regulatory compliance, the IDENTITY service generates comprehensive audit reports:

```
magic identity --audit-report --period 12m --format json

# IDENTITY Audit Report — 12 Months
#
# Summary:
#   Total identities: 870
#   Active: 847 (97.4%)
#   Pending verification: 12 (1.4%)
#   Disputed: 3 (0.3%)
#   Revoked: 8 (0.9%)
#
# Verification coverage:
#   NPI verified: 312/312 (100% of claimed physicians)
#   State license verified: 298/312 (95.5%)
#     Pending: 14 (out-of-state licenses, verification in progress)
#   Board certification verified: 245/312 (78.5%)
#     Not claimed: 67 (residents, fellows — expected)
#
# Key management:
```

```
#   Active Ed25519 keys: 862
#   Keys rotated (12m): 124 (14.4% — above 10% annual target)
#   Keys revoked (12m): 8 (compromised or departure)
#   Mean key age: 214 days
#   Keys > 365 days: 47 (flagged for rotation)
#
# Credential events (12m):
#   IDENTITY:VERIFY: 230
#   IDENTITY:CREDENTIAL_LAPSE: 12
#   IDENTITY:CREDENTIAL_RESTORE: 10
#   IDENTITY:REVOKE: 8
#   IDENTITY:KEY_ROTATE: 124
#
# Compliance attestation:
#   HIPAA §164.312(d): Person/entity authentication — SATISFIED
#   21 CFR 11.100: Electronic signatures — SATISFIED
#   Joint Commission: Practitioner credentialing — SUPPLEMENTED
```

Deterministic — same query, same LEDGER, same report. An external auditor verifies every claim by replaying IDENTITY events. No separate identity management database required [153].

19.20. 19.18 IDENTITY and Role-Based Access Control

Roles are declared in CANON.md and enforced by the API auth gate:

Role	Permissions	Assignment	Clinical Example
READER	Read scope content	Automatic for all verified identities	Any clinician viewing MammoChat governance
WRITER	Modify governance files	Declared in CANON.md `writers:`	Radiology informatics team editing INTEL.md
REVIEWER	Review contributions	Declared in CANON.md `reviewers:`	Senior oncologist reviewing evidence submissions
OWNER	Full scope control	Declared in CANON.md `owner:`	Department chief managing scope lifecycle
ADMIN	Organization-wide control	Declared in root CANON.md	CIO or governance officer

```
# Check role assignments for a scope
magic identity --roles SERVICES/TALK/MAMMOCHAT
```

```
# Role assignments:
#   OWNER: dexter (admin)
#   WRITERS: dr-harrison, dr-chen, eng-martinez (3)
#   REVIEWERS: dr-abramson, dr-patel-mayo (2, 1 cross-org)
#   READERS: 47 (all verified identities with RADIOLOGY affiliation)
#
# Role transitions (12m):
#   dr-kim: WRITER → READER (fellowship ended, departing)
#   dr-garcia: READER → WRITER (promoted to governance team)
#   dr-patel-mayo: added as cross-org REVIEWER (federation trust)
```

Role changes are LEDGER events — every transition, every addition, every change recorded with attribution and timestamp. "Who granted dr-garcia WRITER access to MammoChat?" A LEDGER query [1519].

19.21. 19.19 Clinical Vignette: IDENTITY Prevents Credential Fraud

Intermountain Healthcare discovers that a principal (`dr-wellness-coach`) has been contributing to governance scopes. Contributions passed review and were incorporated into two TALK agents. During a quarterly credential audit, IDENTITY flags an anomaly:

```
magic identity --audit --flag-anomalies

# IDENTITY Anomaly Detected:
#   Principal: dr-wellness-coach
#   Claimed: MD, Board-Certified Internal Medicine
#   NPI: 1234567890
#   Verification status:
#     NPI check:       MISMATCH — NPI 1234567890 belongs to "Dr. James Wellness"
#                      at a different address in a different state
#     State license:   NOT FOUND — No active medical license in claimed state
#     Board cert:      NOT FOUND — ABIM has no record for this individual
#   Risk: CRITICAL — unverified credentials contributed to clinical AI governance
```

The IDENTITY service escalates immediately:

```
# Automatic response:
#   1. IDENTITY:CREDENTIAL_FRAUD event → LEDGER (HIGH severity)
#   2. Principal access: WRITER → SUSPENDED (immediate)
#   3. NOTIFIER: CRITICAL alert to governance officer, compliance officer
#   4. Contributions flagged: 7 contributions under review
#   5. Affected scopes: SERVICES/TALK/INTERNISTCHAT, SERVICES/TALK/WELLNESSCHAT
#   6. Action required: Review all 7 contributions for clinical accuracy
```

The governance officer reviews all 7 contributions. Two contain clinically inaccurate supplement interaction information. Retracted via `vault retract`. Affected TALK agents revalidated — scores remain 255 because the retracted contributions were additive.

Without the IDENTITY chain, the unverified contributor's work would have persisted indefinitely, potentially serving incorrect clinical information to physicians. The verification requirement is clinical safety infrastructure, not bureaucratic overhead [15312].

Chapter 20

Chapter 20: CONTRIBUTE

External WORK. The governed pipeline for community contributions [15]. CONTRIBUTE connects external experts to the INTEL layers built in Chapter 10, flowing through the IDENTITY verification pipeline (Chapter 19). Every contribution mints COIN via the gradient rule (Chapter 18) and propagates through federation LEARNING signals (Chapter 9).

20.1. 20.1 Axiom

CONTRIBUTE governs external work. Every contribution curated. Every curation evidenced. Every WORK mints COIN [15].

Governance cannot scale if only the core team can do the work. CONTRIBUTE opens the pipeline to external actors, curating contributions at two tiers: bronze (accepted — meets minimum quality) and gold (featured — exceeds expectations). Curation is itself governance work — reviewers mint COIN for the review labor [15].

20.2. 20.2 The Contribution Pipeline

```
External actor submits contribution
  → CONTRIBUTE service receives submission
    → Reviewer evaluates against scope CANON.md
      → BRONZE: accepted, standard COIN rate
      → GOLD: featured, elevated COIN rate
      → REJECT: returned with governance feedback
    → LEDGER: CONTRIBUTE event recorded
      → COIN: contributor minted, reviewer minted
```

Every step is a governed event — submission, review, curation decision, COIN minting. The entire lifecycle lives on the LEDGER [15].

20.3. 20.3 Contribution Tiers

Tier	Criteria	COIN Rate	Clinical Example
BRONZE	Meets CANON.md constraints, passes `magic validate`	1x standard rate	Community member adds screening FAQ to MammoChat LEARNING
GOLD	Exceeds constraints, adds significant value	2x standard rate	Oncologist contributes NCCN guideline interpretation to OncoChat INTEL
REJECT	Violates constraints or fails validation	0 (no COIN)	Unverified clinical claim submitted without evidence

GOLD contributions are permanently flagged as high-value and surfaced in LEARNING.md. BRONZE is accepted but not featured. REJECT is recorded with feedback — the rejection itself is governance work [15].

20.4. 20.4 The RESPONSES Directory

Contributions are stored in a `RESPONSES/` directory within the scope that received them:

```
SERVICES/TALK/MAMMOCHAT/
  ├── CANON.md
  ├── INTEL.md
  ├── LEARNING.md
  └── RESPONSES/
      ├── 2026-03-01-bi-rads-faq.md      ← BRONZE
      ├── 2026-03-05-nccn-update.md       ← GOLD
      └── 2026-03-08-screening-guide.md   ← BRONZE
```

Each file carries governed frontmatter: contributor, tier, review date, reviewer. The LEARNING service scans `**/RESPONSES/` paths to aggregate contribution intelligence across the ecosystem [20].

20.5. 20.5 CONTRIBUTE and Clinical Community Building

CONTRIBUTE provides the governed framework for collective clinical intelligence:

Community dashboard: Every TALK service with `.ai` domain surfaces community questions, contribution activity, and emerging patterns — all from RESPONSES/ and LEARNING.md. Questions are real (never fabricated), patterns are discovered (not curated by marketing), and every entry is a LEDGER event.

Expert curation: Clinicians with GOLD-tier contributions build LEDGER-backed reputation. The network rewards clinical expertise with COIN — governance labor that is economically visible and architecturally valued.

Federated propagation: An oncologist at Hospital A who contributes an NCCN guideline interpretation creates a LEARNING event that reaches Hospital B's instance. Knowledge scales across the federation without manual distribution [15] [20].

20.6. 20.6 CONTRIBUTE CLI Operations

```
# Submit a contribution to a scope
vault contribute --scope SERVICES/TALK/MAMMOCHAT \
  --file ./bi-rads-faq.md \
  --principal dr-patel

# Output:
# Submission received: 2026-03-10-bi-rads-faq.md
# Target scope:       SERVICES/TALK/MAMMOCHAT
# Contributor:        dr-patel (VERIFIED, tier: ENTERPRISE)
# Status:             PENDING_REVIEW
# LEDGER event:       CONTRIBUTE:SUBMIT recorded

# Review a pending contribution (reviewer action)
vault review --submission 2026-03-10-bi-rads-faq.md \
  --scope SERVICES/TALK/MAMMOCHAT \
  --tier GOLD \
  --reviewer dexter

# Output:
# Submission reviewed: 2026-03-10-bi-rads-faq.md
# Tier:               GOLD (2x COIN rate)
# COIN minted:
#   Contributor (dr-patel):  64 COIN (GOLD rate)
#   Reviewer (dexter):       16 COIN (review labor)
# Filed to:            RESPONSES/2026-03-10-bi-rads-faq.md
# LEDGER events:       CONTRIBUTE:REVIEW, MINT:WORK (x2) recorded

# List all contributions to a scope
vault contributions --scope SERVICES/TALK/MAMMOCHAT
```

```
# Output:
# Date        Contributor   Tier     COIN   Title
# 2026-03-10  dr-patel      GOLD     64     BI-RADS FAQ expansion
# 2026-03-05  dr-chen       GOLD     64     NCCN update integration
# 2026-03-01  community-1   BRONZE   32     Screening guide draft
# Total: 3 contributions, 160 COIN minted to contributors
```

20.7. 20.7 Contribution Review Workflow

Reviewers follow a checklist derived from the target scope's CANON.md:

Check	Source	Pass Criteria
Axiom alignment	CANON.md axiom	Contribution serves the scope's declared purpose
Vocabulary compliance	VOCAB.md	Contribution uses governed terminology
Evidence citations	INTEL references	Claims cite governed evidence, not opinions
Format compliance	CANON.md constraints	Markdown structure matches scope requirements
Validation pass	`magic validate`	Contribution does not break scope's 255 score
No fabrication	MUST NOT constraint	No fabricated data, citations, or clinical claims

The review produces a structured assessment:

```
```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

submission: 2026-03-10-bi-rads-faq.md reviewer: dexter date: 2026-03-10 tier: GOLD

• • •

20.8. Review
```

Check	Result
Axiom alignment	PASS — extends MammoChat screening knowledge
Vocabulary compliance	PASS — uses governed BI-RADS terminology
Evidence citations	PASS — cites ACR BI-RADS Atlas 6th Ed
Format compliance	PASS — follows INTEL.md structure
Validation pass	PASS — scope remains at 255
No fabrication	PASS — all claims evidence-backed

20.9. Decision

GOLD — Significant clinical value. Oncologist-authored BI-RADS FAQ with evidence citations. Featured in LEARNING.md.

```
## 20.8 Contribution Economics

Contributions create a two-sided economic event: the contributor earns COIN for the content, and the revie

| Actor | BRONZE Rate | GOLD Rate | REJECT Rate |
|-------|-----------|----------|------------|
| Contributor | 32 COIN | 64 COIN | 0 COIN |
| Reviewer | 8 COIN | 16 COIN | 8 COIN |
| Scope owner | 0 COIN | 0 COIN | 0 COIN |

Reviewers earn COIN even for rejections — structured feedback is governance labor regardless of the decisi

## 20.9 RESPONSES Frontmatter Schema

Every file in RESPONSES/ carries governed frontmatter:

```markdown

```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

contributor: dr-patel reviewer: dexter submitted: 2026-03-10 reviewed: 2026-03-10 tier: GOLD coin_contributor: 64 coin_reviewer: 16 ledger_submit: evt:04850 ledger_review: evt:04855 scope: SERVICES/TALK/MAMMOCHAT evidence: - ACR-BIRADS-6TH-2023 - NCCN-BREAST-2024

• • •

Chapter 21

BI-RADS FAQ Expansion

[content…]

```
Immutable after review — LEDGER event IDs link the frontmatter to the hash chain. Modifying it post-review

## 20.10 Contribution Anti-Patterns

| Anti-Pattern | Description | Detection |
|--------------|-------------|-----------|
| Self-review | Contributor reviews their own submission | `magic validate` checks contributor != reviewer
| COIN farming | Low-quality bulk submissions for COIN | Review checklist enforces quality gate |
| Ghost contribution | Contribution with fabricated evidence | INTEL cross-reference check |
| Bypass review | Contribution merged without review event | LEDGER audit: no CONTRIBUTE:REVIEW for file i
| Tier inflation | Reviewer assigns GOLD to BRONZE-quality work | Review audit: random re-review by second

Every anti-pattern has a structural detection mechanism — the toolchain catches violations automatically,

## 20.11 Clinical Contribution Example: OncoChat Guideline Update

A partner hospital oncologist contributes an updated NCCN staging interpretation for NSCLC:

```bash
# 1. Oncologist submits contribution
vault contribute --scope SERVICES/TALK/ONCOCHAT \
  --file ./nsclc-staging-2026.md \
  --principal dr-yamamoto \
  --evidence "NCCN-NSCLC-v2.2026"

# 2. OncoChat governance reviewer evaluates
```
```

```
vault review --submission 2026-03-10-nsclc-staging-2026.md \
  --scope SERVICES/TALK/ONCOCHAT \
  --tier GOLD \
  --reviewer dexter \
  --notes "Verified against NCCN v2.2026. TNM staging accurate. Evidence complete."

# 3. Result
# Contribution filed: RESPONSES/2026-03-10-nsclc-staging-2026.md
# LEARNING updated:  new NSCLC staging pattern detected
# INTEL enriched:    NCCN-NSCLC-v2.2026 added to evidence index
# COIN minted:       dr-yamamoto +64, dexter +16
# Federation signal: LEARNING event propagated to 3 partner hospitals
```

Every step is a LEDGER event. The contributor's expertise is economically rewarded. Knowledge propagates across the federation — no manual distribution, no email chains [15] [20].

21.1. 20.12 Clinical Vignette: Cross-Institutional Contribution Network

Five NCI-designated Comprehensive Cancer Centers — MSK, MD Anderson, Dana-Farber, UCSF, and Moffitt — establish a federated CONTRIBUTE network for OncoChat governance. Each institution maintains its own OncoChat with institution-specific INTEL, but contributions flow across the federation.

Each institution has verified, IDENTITY-gated principals:

```
MSK:         14 oncologists (solid tumors, hematologic, pediatric)
MDA:         18 oncologists (GI, thoracic, breast, melanoma, GU)
Dana-Farber: 11 oncologists (lymphoma, myeloma, leukemia)
UCSF:        9 oncologists (neuro-oncology, sarcoma, endocrine)
Moffitt:     12 oncologists (lung, head/neck, cutaneous)
─────────────────────────────────────────────────
Total:       64 verified contributors across 5 institutions
```

Month 1 Activity. The network generates 47 contributions in the first month:

```
vault contributions --federation nci-comprehensive --period 30d

# Federation Contribution Report — NCI Comprehensive Network (30d)
#
# Institution    Submitted  GOLD   BRONZE  REJECT  COIN Minted
# MSK            12         8      3       1       736
# MD Anderson    14         9      4       1       848
# Dana-Farber    8          6      1       1       480
```

```
# UCSF          7         5      2       0       432
# Moffitt       6         4      2       0       368
# ──────────────────────────────────────────────────────
# Total         47        32     12      3       2,864
#
# Top contributor: dr-tanaka (MSK) — 5 GOLD contributions (lymphoma staging)
# Most active scope: NSCLC (12 contributions, 8 GOLD)
# Cross-institution reviews: 23 (49% of all reviews)
```

Dr. Tanaka at MSK contributes a comprehensive Lugano Classification response assessment guide (Cheson et al., JCO 2014): CR definition mapped to PET-CT Deauville scores, PR criteria with measurable disease thresholds (>50% decrease in SPD), PD criteria including new lesion definition, and the metabolic vs. radiographic response distinction.

```
vault contribute --scope SERVICES/TALK/ONCOCHAT/LYMPHOMA \
  --file ./lugano-response-assessment.md \
  --principal dr-tanaka \
  --evidence "Cheson-JCO-2014, NCCN-NHL-2026"
# Submission received: 2026-03-15-lugano-response.md
# Status: PENDING_REVIEW

# Cross-institution review (Dana-Farber reviews MSK contribution)
vault review --submission 2026-03-15-lugano-response.md \
  --scope SERVICES/TALK/ONCOCHAT/LYMPHOMA \
  --tier GOLD \
  --reviewer dr-abramson \
  --institution dana-farber

# Review result:
# Tier: GOLD (expert lymphoma content, evidence-complete)
# COIN: dr-tanaka (MSK) +64, dr-abramson (Dana-Farber) +16
# LEARNING: Lugano Classification response assessment pattern recorded
# Federation: LEARNING signal propagated to 4 partner institutions
```

Dana-Farber reviews MSK's contribution — cross-institutional peer review, not single-hospital silo review. Governed (LEDGER event), attributed (VERIFIED principal), economically incentivized (reviewer earns COIN) [1520].

21.2. 20.13 Contribution Quality Metrics

Quality metrics tracked per contributor and per scope:

```
vault contributor-profile dr-tanaka

# Contributor Profile: dr-tanaka (MSK)
#
# Tier:              AGENT (127)
# Credentials:       MD (MSK), Board Certified (Hematology/Oncology)
# Total contributions: 23
# Tier distribution:   GOLD 18 (78%), BRONZE 4 (17%), REJECT 1 (4%)
# Total COIN earned:   1,312 (contributions) + 256 (reviews)
# Scopes contributed:  LYMPHOMA, LEUKEMIA, MYELOMA, NSCLC
# Review invitations:  12 (accepted 11, declined 1)
# Mean review time:    4.2 hours (submission to tier assignment)
#
# Contribution timeline:
# ████████████████████████████████ Jan (6)
# ████████████████████████         Feb (5)
# ████████████████████████████████████████ Mar (12, peak)
```

A 78% GOLD rate establishes dr-tanaka as a high-value contributor. Scope owners filter invitations by GOLD rate — a LEDGER-derived metric, not a subjective rating.

21.3. 20.14 Contribution Conflict Resolution

When two contributors submit conflicting information to the same scope, CONTRIBUTE resolves it:

```
# Conflict detected:
# dr-smith (MDA) submits: "EGFR exon 19 deletion — osimertinib first-line"
# dr-jones (Moffitt) submits: "EGFR exon 19 deletion — erlotinib first-line"
#
# Both reference NCCN NSCLC 2026 but interpret differently:
#   dr-smith cites FLAURA trial (OS benefit for osimertinib)
#   dr-jones cites cost-effectiveness analysis favoring erlotinib

vault conflict --scope SERVICES/TALK/ONCOCHAT/NSCLC \
  --submissions "2026-03-20-egfr-smith.md,2026-03-20-egfr-jones.md"

# Conflict resolution options:
# 1. ACCEPT_BOTH — both are valid interpretations, include both with context
# 2. ACCEPT_ONE — one supersedes the other (requires evidence justification)
# 3. ESCALATE — refer to scope owner for domain expert resolution
# 4. REJECT_BOTH — conflicting submissions returned for reconciliation
```

The scope owner selects ACCEPT_BOTH with a synthesized note: “NCCN Category 1: osimertinib pre-

ferred for EGFR exon 19 deletion/L858R (FLAURA: OS 38.6 vs 31.8 months, HR 0.80). Erlotinib is an NCCN alternative, particularly where cost-effectiveness is weighted." The resolution is a governance act — LEDGER-recorded, attributed, and preserved in RESPONSES/ as the authoritative interpretation.

21.4. 20.15 Contribution Expiration and Refresh

Clinical contributions have evidence half-lives — NCCN guidelines update annually, FDA approvals change drug indications, trial results supersede older evidence. CONTRIBUTE tracks freshness:

```
vault contributions --scope SERVICES/TALK/ONCOCHAT --stale

# Stale contributions (evidence > 12 months old):
#
# File                              Evidence        Age     Action
# 2025-01-15-nsclc-first-line.md   NCCN-NSCLC-v1  14mo    REFRESH NEEDED
# 2025-02-20-crc-adjuvant.md       NCCN-CRC-v3    12mo    REFRESH NEEDED
# 2025-03-10-breast-her2.md        NCCN-BREAST-v2 12mo    APPROACHING
#
# Auto-notifications sent to:
#   Original contributors (request for update)
#   Scope writers (stale content alert)
# LEDGER: CONTRIBUTE:STALE events recorded (3)
```

Stale contributions are warnings, not errors — they do not automatically reduce the scope score. But more than 30% stale triggers a GOV-115 compilation warning. Above 50%, the warning escalates to an error: the evidence base is no longer current enough to support governance [1520].

21.5. 20.16 Governance Proof: The Contribution Chain

Every link in the contribution lifecycle is governed:

```
External expert (dr-tanaka) holds verified credentials
  → IDENTITY: VERIFIED (MD, board-certified, NPI confirmed)
  → Submits contribution to OncoChat/LYMPHOMA scope
  → CONTRIBUTE:SUBMIT recorded on LEDGER (evt:08300)
  → Reviewer (dr-abramson) at different institution evaluates
  → Review follows CANON.md-derived checklist (6 checks)
  → Tier assigned: GOLD (2x COIN rate)
  → CONTRIBUTE:REVIEW recorded on LEDGER (evt:08305)
  → MINT:WORK: contributor +64 COIN (evt:08306)
  → MINT:WORK: reviewer +16 COIN (evt:08307)
```

```
  → Contribution filed to RESPONSES/ with governed frontmatter
  → LEARNING.md updated: new pattern detected
  → Federation: LEARNING signal propagated to 4 institutions
  → All 5 OncoChat instances benefit from one expert's knowledge
  → Evidence freshness tracked: auto-alert at 12 months
  → If conflict: governed resolution with LEDGER attribution
  → The contribution IS governed work
  → The review IS governed work
  → The knowledge transfer IS governed work
  → All work mints COIN. All COIN traces to labor. All labor is attributed.
```

No step is informal. The contribution network is not a forum — it is a governed knowledge pipeline with cryptographic attribution and economic incentives. Every COIN traces to specific labor by a verified identity at a specific time. Q.E.D. [152012]

21.6. 20.17 Contribution Templates by Clinical Domain

Each clinical domain defines contribution templates in its CANON.md, standardizing format so reviewers can evaluate consistently:

```
# CONTRIBUTE template: ONCOLOGY-GUIDELINE

## Required Fields

| field | description | example |
|-------|-------------|---------|
| guideline_source | Issuing body | NCCN, ASCO, ESMO |
| guideline_version | Version identifier | NCCN-NSCLC-v2026.1 |
| recommendation | Specific recommendation text | "Adjuvant osimertinib for Stage IB-IIIA EGFR+ NSCLC" |
| evidence_level | Evidence category | Category 1 (high-level, uniform consensus) |
| patient_population | Target population | Stage IB-IIIA resected EGFR exon 19del/L858R NSCLC |
| clinical_trial | Supporting trial | ADAURA (NCT02511106) |
| contraindications | When NOT to apply | Interstitial lung disease, severe hepatic impairment |
| citations | Source references | [P-XX] Wu et al. NEJM 2020; NCCN NSCLC v2026.1 |
```

Templates reduce reviewer burden — the reviewer knows exactly where to find the evidence level, patient population, and supporting trial. Missing required fields trigger auto-rejection before human review:

```
vault contribute --template ONCOLOGY-GUIDELINE \
  --scope SERVICES/TALK/ONCOCHAT/NSCLC

# Template validation:
```

```
#   guideline_source:   NCCN □
#   guideline_version:  NCCN-NSCLC-v2026.1 □
#   recommendation:     present □
#   evidence_level:     Category 1 □
#   patient_population: present □
#   clinical_trial:     NCT02511106 □
#   contraindications:  present □
#   citations:          2 references □
# Template validation: PASS
# Submitting for review...
```

21.7. 20.18 Contribution Impact Metrics

Contributions create compound value. Track their downstream impact:

```
vault contribution-impact dr.tanaka@msk.org --period 12m

# Contribution Impact Report — dr.tanaka (12 months)
#
# Contributions submitted:      18
# Contributions accepted:       16 (88.9%)
# GOLD tier:                    12 (75%)
# BRONZE tier:                  4 (25%)
#
# COIN earned:                  1,152 (from contributions)
# COIN earned (reviews):        192 (reviewer for 12 others)
# Total COIN:                   1,344
#
# Downstream impact:
#   Scopes referencing contributions:  8
#   Organizations using contributions: 4 (MSK, Dana-Farber, UCSF, Stanford)
#   TALK agents serving contributions: 5 (OncoChat instances)
#   Patient interactions informed:     ~12,400 (estimated from session count)
#
# Top contribution:
#   "Lugano Classification response assessment guide"
#   Referenced by: 4 scopes
#   Patient interactions informed: ~4,200
#   Attestation equivalent: GOLD with 3 cross-institutional validations
```

A single expert's contribution informs thousands of clinical interactions across multiple institutions. The 64 COIN minted for a GOLD contribution is the economic receipt; the 12,400 patient interactions downstream

are the clinical significance. The LEDGER connects both — tracing from contribution event to the TALK sessions it informed [152012].

21.8. 20.19 Contribution Retraction

When a contribution is found to contain errors after acceptance, CONTRIBUTE supports governed retraction:

```
vault retract --contribution 2026-01-15-nsclc-first-line.md \
  --reason "ADAURA 5-year follow-up changes DFS interpretation" \
  --replacement 2026-03-10-nsclc-first-line-updated.md

# Retraction recorded:
#   Original: 2026-01-15-nsclc-first-line.md (RETRACTED)
#   Reason: ADAURA 5-year follow-up changes DFS interpretation
#   Replacement: 2026-03-10-nsclc-first-line-updated.md
#   LEDGER: CONTRIBUTE:RETRACT (evt:08900)
#   NOTIFIER: Retraction notice sent to all 4 organizations using this contribution
#   Impact: 5 TALK agents notified to update INTEL references
```

Retraction is not deletion. The original remains in the LEDGER and RESPONSES/ — marked RETRACTED with a reference to its replacement. An auditor can reconstruct why a clinical AI's response changed on a specific date by following the retraction chain [1520].

Chapter 22

Chapter 21: NOTIFIER

Event notification. Inbox delivery. The nervous system of governed communication [23]. NOTIFIER carries governance events from MONITORING (Chapter 22), CONTRIBUTE reviews (Chapter 20), and drift detection to governed recipients. Every delivery is a LEDGER event (Chapter 13).

22.1. 21.1 Axiom

NOTIFIER delivers governed cross-scope messages. Every delivery ledgered. Every route declared [23].

Not email. Not Slack. NOTIFIER delivers governed events — not arbitrary messages — to governed recipients via governed routes. Every message has a declared route, a verified sender, a verified recipient, and a LEDGER record. For clinical AI deployments, this is the governed pipeline that carries results, recommendations, and governance events from AI agents to clinicians, compliance officers, and administrators — every alert with full provenance [23].

22.2. 21.2 Route Architecture

Routes are declared in governance files, not configured in a database. The `notify:` header in a scope's CANON.md declares who receives events from that scope:

```
# MAMMOCHAT — CANON

notify: DEXTER
```

All governance events from MAMMOCHAT route to the DEXTER principal. No `notify:` declaration, no delivery. NOTIFIER does not guess and does not broadcast [23].

For a health network with multiple TALK channels:

Scope	notify:	Recipient	Clinical Example
MAMMOCHAT	DEXTER	System administrator	MammoChat governance drift alert
ONCOCHAT	DEXTER, ONCOLOGY_DEPT	Admin + department	OncoChat NCCN update signal
RUNNER	DEXTER	Operations manager	New real estate vendor onboarded
NONA	DEXTER, JP	Admin + agent	Buyer referral received

22.3. 21.3 The Inbox Model

Messages are immutable once delivered — no editing, no deletion. The inbox is KV-backed, keyed by recipient principal, with read-state tracked per-message via acknowledgment:

```
DELIVER → message stored in recipient inbox (immutable)
  → LEDGER event recorded (sender, receiver, timestamp, content hash)
    → Recipient reads message → state: unread→read
      → Recipient acknowledges → ACK event recorded
```

Each principal's inbox is bounded by governance tier. At capacity, the oldest acknowledged messages archive (not delete — append to the archive chain). High-priority governance events are never lost in a flood of routine notifications [23].

22.4. 21.4 API 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
```

Rate limits are governance-derived: 10/hr for sending (prevents spam), 100/hr for reading (practically unlimited for humans). Acknowledgment has no rate limit — clearing your inbox should never be throttled [23].

22.5. 21.5 Record Shape

Field	Type	Content	Clinical Example
id	string	Unique message identifier	`notifier:evt:00315`
ts	ISO-8601	Delivery timestamp	`2026-03-10T09:15:00Z`
from	string	Sender principal (verified)	`MAMMOCHAT` (scope identity)
to	string	Recipient principal	`DEXTER`
type	string	Event category	`DRIFT_ALERT`
message	string	Content body	“MammoChat score dropped from 255 to 191”
scope	string	Source governance scope	`SERVICES/TALK/MAMMOCHAT`
read	boolean	Acknowledgment state	`false`
hash	string	Content hash for integrity	`sha256:a1b2c3...`

22.6. 21.6 LEDGER Projection

Every delivery is economic work — it mints a LEDGER event:

```
{
  "event": "NOTIFIER:DELIVER",
  "key": "MAMMOCHAT→DEXTER",
  "work_ref": "notifier:evt:00315",
  "ts": "2026-03-10T09:15:00Z"
}
```

Delivery volume, response times, and acknowledgment rates are all derivable from the LEDGER. For hospital compliance teams, this is the communication audit trail Joint Commission surveyors require — every clinical AI alert delivered, when, to whom, and whether acknowledged [23].

22.7. 21.7 Clinical Alert Patterns

Four categories of governance alerts cover clinical AI deployments:

Alert Category	Trigger	Urgency	Clinical Example
DRIFT_ALERT	Scope score drops below threshold	HIGH	MammoChat model update breaks validation
EVIDENCE_UPDATE	INTEL source publishes new version	MEDIUM	NCCN 2026 guidelines released
SESSION_ANOMALY	TALK session exceeds safety thresholds	HIGH	Patient asks about self-harm in MedChat
CONTRIBUTION_REVIEW	New CONTRIBUTE submission awaiting review	LOW	Oncologist submits NCCN interpretation

All alerts follow the same governed delivery pattern: declared route, verified sender, LEDGER event, inbox delivery. Urgency does not change the governance pipeline — HIGH and LOW alerts use the same architecture. The difference is in your response time, not the system's delivery [23].

22.8. 21.8 NOTIFIER Configuration via CANON.md

Routing is declared in the scope's CANON.md, not in a configuration database:

```
```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

notify: true

• • •

22.9. Notify

event	recipients	channel
DRIFT_ALERT	scope_writers	inbox
```

event	recipients	channel
EVIDENCE_UPDATE	scope_readers	inbox
SESSION_ANOMALY	scope_writers, admin	inbox
CONTRIBUTION_REVIEW	scope_writers	inbox

```
The `notify: true` frontmatter key enables notification for the scope. The Notify table declares event-to-
```

```bash
# Compile notification routes from all scopes
build --notifier

# Output:
# Scanning for notify: true scopes...
# SERVICES/TALK/MAMMOCHAT: 4 routes configured
# SERVICES/TALK/ONCOCHAT: 4 routes configured
# SERVICES/MONITORING: 2 routes configured
# Total: 10 notification routes across 3 scopes
# Route map compiled to _data/notifier.json
```

22.10. 21.9 Inbox Architecture

Each principal's inbox is a bounded, ordered collection of notification records:

```
~/.canonic/inbox/{principal}/
  ├── unread/          ← new notifications (bounded by tier)
  ├── read/            ← acknowledged notifications
  └── archive/         ← archived notifications (append-only)
```

Inbox capacity by tier:

Tier	Unread Limit	Archive Retention	Clinical Example
COMMUNITY	50	30 days	Patient-facing chatbot alerts
BUSINESS	200	90 days	Department-level governance alerts
ENTERPRISE	1,000	1 year	Hospital-wide clinical AI alerts
AGENT	5,000	2 years	Developer governance notifications
FULL	Unlimited	Permanent	System-level audit notifications

When the unread limit is reached, the oldest unread notifications are force-archived with a `NOTIFIER:OVERFLOW` LEDGER event — a governance hygiene signal that a principal is not responding [23].

22.11. 21.10 NOTIFIER CLI

```
# Read your inbox
vault inbox dexter

# Output:
# Unread (3):
#   [HIGH] DRIFT_ALERT — MammoChat score 255→191 (2m ago)
#   [MED]  EVIDENCE_UPDATE — NCCN 2026 v2 released (1h ago)
#   [LOW]  CONTRIBUTION_REVIEW — new submission pending (3h ago)
#
# Read (47): use --all to show
# Archive (1,204): use --archive to show

# Acknowledge a notification
vault ack notifier:evt:00315
# LEDGER event: NOTIFIER:ACK recorded

# Acknowledge all notifications
vault ack --all
# 3 notifications acknowledged
# LEDGER events: 3x NOTIFIER:ACK recorded

# Send a governed notification (scope-to-principal)
vault notify --from SERVICES/MONITORING --to dexter \
  --type DRIFT_ALERT \
  --message "CardiChat score dropped from 255 to 191"
# LEDGER event: NOTIFIER:DELIVER recorded

# Notification delivery report
vault notify-report --last 30d

# Output:
# Total delivered:     847
# Total acknowledged:  812 (95.9%)
# Mean ack time:       4.2 hours
# Overflows:           0
# Unacknowledged HIGH: 2 (action required)
```

22.12. 21.11 Notification Deduplication

Identical notifications within a configurable window are deduplicated:

```
Rule: If an identical notification (same type, same scope, same message hash)
      was delivered within the dedup_window, suppress the duplicate.

Default dedup_window: 1 hour
```

Scenario	Result
MammoChat drifts at 14:00 → DRIFT_ALERT delivered	Delivered
MammoChat still drifted at 14:15 → same DRIFT_ALERT	Suppressed (within window)
MammoChat still drifted at 15:01 → same DRIFT_ALERT	Delivered (window expired)
OncoChat drifts at 14:15 → different scope DRIFT_ALERT	Delivered (different scope)

A scope that stays drifted for 6 hours generates 6 DRIFT_ALERTs (one per hour), not 360 (one per minute). Configure the window in CANON.md:

```
notify_dedup_window: 3600  # seconds (1 hour)
```

22.13. 21.12 NOTIFIER and HIPAA Communication Requirements

NOTIFIER satisfies HIPAA communication requirements for clinical AI governance:

§164.308(a)(6)(ii) — Response and Reporting: Security incidents must be identified and responded to. NOTIFIER's DRIFT_ALERT and SESSION_ANOMALY categories provide the identification mechanism. The acknowledgment flow provides the response evidence.

§164.308(a)(5)(ii)(A) — Security Reminders: Periodic security reminders must be distributed. NOTIFIER's EVIDENCE_UPDATE category distributes governance-relevant updates to all governed principals.

"Was the governance team notified of this event?" is a LEDGER query, not a search through email archives [23].

22.14. 21.13 Clinical Vignette: NOTIFIER Prevents Stale Guideline Deployment

HCA Healthcare (Nashville, 186 hospitals) operates 12 clinical AI agents. On March 5, 2026, ACR publishes updated Lung-RADS v2.0 criteria — changing management recommendations for category 3 and

4A nodules. HCA's MammoChat and LungChat agents both reference Lung-RADS in their INTEL.md files.

MONITORING's evidence freshness check detects the update during its routine scan:

```
Signal: EVIDENCE_UPDATE
Source: ACR Lung-RADS v2.0 (published 2026-03-05)
Affected scopes:
  SERVICES/TALK/LUNGCHAT/SCREENING (references Lung-RADS v1.1)
  SERVICES/TALK/MAMMOCHAT/CHEST-INCIDENTALS (references Lung-RADS v1.1)
```

Governed alerts reach all declared recipients:

```
vault inbox lungchat-governance-team
# Unread (1):
#   [MED] EVIDENCE_UPDATE — ACR Lung-RADS v2.0 published
#     Affected: LUNGCHAT/SCREENING, MAMMOCHAT/CHEST-INCIDENTALS
#     Current reference: Lung-RADS v1.1 (2019)
#     New version: Lung-RADS v2.0 (2026)
#     Changes: Category 3 management updated, 4A threshold revised
#     Action: Update INTEL.md evidence references
#     Delivered: 2026-03-05T14:22:00Z

vault inbox mammochat-governance-team
# Unread (1):
#   [MED] EVIDENCE_UPDATE — ACR Lung-RADS v2.0 published
#     Affected: MAMMOCHAT/CHEST-INCIDENTALS
#     [same content as above]
#     Delivered: 2026-03-05T14:22:01Z
```

The acknowledgment and response timeline:

```
14:22:00  NOTIFIER:DELIVER → lungchat-governance-team (evt:09100)
14:22:01  NOTIFIER:DELIVER → mammochat-governance-team (evt:09101)
14:35:00  NOTIFIER:ACK ← dr-chen (lungchat team) (evt:09105)
15:10:00  NOTIFIER:ACK ← dr-williams (mammochat team) (evt:09108)
16:00:00  INTEL.md updated: LungChat/SCREENING (Lung-RADS v2.0 added)
16:15:00  magic validate → 255 ✓ → MINT:WORK +8 COIN
16:30:00  INTEL.md updated: MammoChat/CHEST-INCIDENTALS
16:45:00  magic validate → 255 ✓ → MINT:WORK +8 COIN
```

Time to resolution: 2 hours 23 minutes — from ACR publication to governed evidence update across both agents. Without NOTIFIER, the update depends on physicians noticing the publication, which could take weeks. In those weeks, LungChat could recommend screening follow-up based on outdated Lung-RADS v1.1 criteria when v2.0 changed the category 3 follow-up interval from 6 months to 3 months [23].

22.15. 21.14 Notification Routing Patterns

Beyond basic scope-to-principal delivery, four routing patterns:

Pattern 1: Fan-out. One scope event routes to multiple principals.

```
# CANON.md for critical scope
notify:
  DRIFT_ALERT: [admin, compliance-officer, department-chief, on-call-engineer]
```

Pattern 2: Escalation chain. Unacknowledged notifications escalate through a priority chain.

```
notify_escalation:
  - principal: scope-writer      # First: scope owner
    timeout: 1h
  - principal: department-chief  # Second: department head
    timeout: 4h
  - principal: admin             # Third: system administrator
    timeout: 24h
  - principal: ciso              # Final: security officer
    timeout: none                # No further escalation
```

```
# Example escalation:
# 09:00 DRIFT_ALERT → scope-writer (unack'd)
# 10:00 ESCALATION → department-chief (1h timeout exceeded)
# 14:00 ESCALATION → admin (4h timeout exceeded)
# LEDGER: NOTIFIER:ESCALATE events at each level
```

Pattern 3: Digest. Low-priority notifications are batched into daily digests instead of individual delivery.

```
notify_digest:
  frequency: daily
  time: "08:00"
  events: [CONTRIBUTION_REVIEW, EVIDENCE_UPDATE]
```

Pattern 4: Federation relay. Notifications cross federation boundaries when the event affects partner scopes.

```
notify_federation:
  EVIDENCE_UPDATE: relay_to_partners
  DRIFT_ALERT: relay_to_partners
  SESSION_ANOMALY: local_only  # PHI-adjacent, do not relay
```

22.16. 21.15 Notification Templates

Governed templates enforce consistent notification formatting:

```
# Template: DRIFT_ALERT

Subject: [{{severity}}] Governance Drift — {{scope_name}}

Score dropped from {{previous_score}} to {{current_score}}.

| Dimension | Previous | Current | Delta |
|-----------|----------|---------|-------|
{{#each dimensions}}
| {{name}} | {{prev}} | {{curr}} | {{delta}} |
{{/each}}

Cause: {{drift_cause}}
Commit: {{commit_hash}} by {{author}}
Action: Run `magic-heal {{scope_path}}` or manually restore.

LEDGER event: {{ledger_event_id}}
```

Templates live in the NOTIFIER scope's governance tree, compiled by `build`, validated by `magic validate`. Modifying a template is governance work that mints COIN — preventing ad-hoc changes that could obscure critical clinical AI alerts.

22.17. 21.16 Notification Analytics

Analytics surface governance communication health:

```
vault notify-analytics --period 90d

# Notification Analytics — 90 Days
#
# Delivery Summary:
#   Total delivered:          2,847
#   Total acknowledged:       2,712 (95.3%)
#   Mean time-to-ack:         3.8 hours
#   Median time-to-ack:       1.2 hours
#
# By severity:
#   HIGH:    47 delivered, 47 acked (100%), mean ack: 22 minutes
```

```
#   MEDIUM:  312 delivered, 298 acked (95.5%), mean ack: 2.1 hours
#   LOW:     2,488 delivered, 2,367 acked (95.1%), mean ack: 4.5 hours
#
# Escalations:              12 (0.4% of all notifications)
# Overflows:                 0
# Deduplication suppressed:  847 (23% reduction in noise)
#
# Slowest ack (non-escalated): 18.4 hours (EVIDENCE_UPDATE, weekend)
# Fastest ack:                  12 seconds (DRIFT_ALERT, on-call)
#
# By principal (top 5 slowest responders):
#   dr-inactive:    mean ack 14.2h (flagged for review)
#   dr-on-sabbatical: mean ack 12.8h (expected — sabbatical)
#   eng-nightshift:  mean ack 8.4h (expected — timezone offset)
```

A principal with consistently slow acknowledgment times may need routing adjusted (digest instead of real-time) or scope assignments reviewed. All data is LEDGER-derived — no separate analytics database [23].

22.18. 21.17 Governance Proof: The Notification Chain

The governance chain ensures every clinical AI event reaches its intended audience:

```
Governance event occurs (scope drift, evidence update, contribution)
  → MONITORING detects the event
  → NOTIFIER resolves route from scope CANON.md notify: declaration
  → Route validation: recipient exists, inbox not full, no dedup match
  → NOTIFIER:DELIVER event recorded on LEDGER (content hash included)
  → Message stored in recipient inbox (immutable)
  → Recipient reads message → state: unread → read
  → Recipient acknowledges → NOTIFIER:ACK on LEDGER
  → If unacknowledged after timeout → NOTIFIER:ESCALATE
  → Escalation chain continues until acknowledgment or final level
  → All delivery, ack, and escalation events are LEDGER events
  → Every notification is hashable, verifiable, and attributed
  → No notification is lost (bounded inbox with archive)
  → No notification is forged (sender verified, content hashed)
  → No notification is silently ignored (escalation chain)
```

The chain proves four properties: (1) every governance event warranting notification was delivered, (2) every delivery went to a declared recipient via a declared route, (3) every acknowledgment or lack thereof

is recorded, and (4) unacknowledged notifications escalate until someone responds. NOTIFIER provides cryptographic proof that governance communication occurred. Q.E.D. [2312]

22.19. 21.18 NOTIFIER Channel Architecture

Multiple delivery channels, each a governed integration declared in CANON.md and validated by `magic validate`:

Channel	Medium	Latency	Use Case	Configuration
INBOX	CANONIC vault inbox	Real-time	Primary governance communication	Default (always enabled)
EMAIL	SMTP delivery	< 5 minutes	Secondary, offline-accessible	`notify_email: admin@hospital.org`
SLACK	Slack webhook	Real-time	Team visibility, on-call routing	`notify_slack: #governance-alerts`
PAGERDUTY	PagerDuty API	Real-time	Critical incident escalation	`notify_pagerduty: service_key`
WEBHOOK	Custom HTTP endpoint	Real-time	EHR integration, custom systems	`notify_webhook: https://ehr.org/hook`

```
# Configure multi-channel notification for critical scope
# In CANON.md:
notify:
  DRIFT_ALERT:
    channels: [INBOX, SLACK, PAGERDUTY]
    severity: HIGH
  EVIDENCE_UPDATE:
    channels: [INBOX, EMAIL]
    severity: MEDIUM
  CONTRIBUTION_REVIEW:
    channels: [INBOX]
    severity: LOW
```

The INBOX channel cannot be disabled — every notification reaches it regardless of other channel configuration, ensuring the LEDGER-backed audit trail remains complete even if external channels fail [23].

22.20. 21.19 NOTIFIER Quiet Hours and On-Call Rotation

For 24/7 healthcare operations, NOTIFIER supports quiet hours and on-call rotation:

```
# Configure quiet hours (non-urgent notifications held)
vault notify-config dr.chen@hadleylab.org \
  --quiet-hours "22:00-06:00 America/New_York" \
  --quiet-severity LOW,MEDIUM \
  --urgent-bypass HIGH

# Configure on-call rotation
vault notify-rotation governance-oncall \
  --schedule weekly \
  --members "dr.chen,dr.williams,dr.park,eng.martinez" \
  --start "2026-03-11T08:00:00-04:00"

# Current on-call
vault notify-rotation governance-oncall --current
# On-call: dr.williams (2026-03-11 08:00 to 2026-03-18 08:00)
# Next: dr.park
# DRIFT_ALERT → dr.williams (on-call)
# EVIDENCE_UPDATE → all team members (not rotation-gated)
```

HIGH severity bypasses quiet hours — a 03:00 AM DRIFT_ALERT reaches the on-call officer immediately. MEDIUM and LOW accumulate during quiet hours and deliver as a morning digest at 06:00. The configuration itself is a LEDGER event, so auditors can verify that suppression was deliberate, not ad-hoc [23].

22.21. 21.20 NOTIFIER and Cross-Scope Notification Aggregation

When a single event affects multiple scopes, NOTIFIER aggregates to prevent alert storms:

```
# Single event: NCCN publishes updated guidelines
# Affected scopes: 5 TALK agents reference NCCN

# Without aggregation: 5 separate notifications to the same team
# With aggregation: 1 notification listing all 5 affected scopes

vault inbox governance-team
# Unread (1):
#   [MED] EVIDENCE_UPDATE — NCCN Breast Cancer v2026.2 published
#     Affected scopes (5):
#       SERVICES/TALK/MAMMOCHAT/SCREENING
#       SERVICES/TALK/ONCOCHAT/BREAST
#       SERVICES/TALK/SURGCHAT/BREAST
#       SERVICES/TALK/RADCHAT/BREAST-MRI
#       SERVICES/TALK/PATHCHAT/BREAST-BIOPSY
```

```
#     Action: Update INTEL.md in each affected scope
#     Delivered: 2026-03-10T14:00:00Z
```

Aggregation reduces notification volume by 60-80% in clinical environments where guideline updates affect multiple agents simultaneously. The window is 5 minutes — events with the same trigger merge into a single notification. The LEDGER records both the individual events and the aggregate [2312].

Chapter 23

Chapter 22: MONITORING

Runtime metrics. Governance scoring. The real-time visibility layer [24]. MONITORING feeds drift detection into NOTIFIER (Chapter 21), projects operational data into the VAULT dashboard (Chapter 15), and renders fleet health in the GALAXY (Chapter 31). The dev dashboard is live at dev.hadleylab.org.

23.1. 22.1 Axiom

MONITORING is continuous governance scoring. Real-time visibility, not snapshots. Observability without obstruction [24].

Prometheus-compatible metrics. Dependency health checks. Real-time 255-bit governance scores. MONITORING exposes all three through standard HTTP endpoints — watching the system without interfering with it [24].

The critical constraint: MONITORING MUST NOT block service operations. If metrics collection fails, the service continues. If the health check endpoint is slow, requests still serve. Observability is never a gate — the failure of observation does not cause the failure of the observed system.

23.2. 22.2 The /metrics Endpoint

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

In-memory counters, zero external dependencies:

Metric	Labels	Type	Clinical Significance
`canonic_api_requests_total`	endpoint, method, status	counter	Total API load per endpoint
`canonic_api_request_duration_seconds`	endpoint	summary (p50/p95/p99)	Latency percentiles for clinical AI responses
`canonic_auth_total`	result (ok/fail)	counter	Authentication success/failure rates
`canonic_ledger_events_total`	event_type	counter	Economic event volume by type
`canonic_wallet_balance`	user	gauge	Per-user COIN balance
`canonic_scope_score`	scope	gauge	Real-time governance score per scope
`canonic_drift_events_total`	scope	counter	Governance regression count per scope

```
# HELP canonic_scope_score Current MAGIC governance score
# TYPE canonic_scope_score gauge
canonic_scope_score{scope="SERVICES/TALK/MAMMOCHAT"} 255
canonic_scope_score{scope="SERVICES/TALK/ONCOCHAT"} 255
canonic_scope_score{scope="SERVICES/TALK/MEDCHAT"} 191
```

Any monitoring stack (Grafana, Datadog, PagerDuty) can ingest these metrics. CANONIC governance metrics integrate with your existing monitoring infrastructure without custom adapters [24].

23.3. 22.3 Extended Health Check

`GET /api/v1/health → JSON health status`

```
{
  "status": "healthy",
  "port": 8255,
  "checks": {
    "ledger_head": "evt:00047 (2026-03-10T14:30:00Z)",
    "vault_dir": "/Users/iDrDex/.canonic/vault/ exists",
    "wallet_valid": "47 wallets, 0 mismatches"
  },
  "uptime_s": 86400
}
```

Health checks verify live dependency state, not cached configuration. `ledger_head` confirms the LEDGER is receiving events, `vault_dir` confirms the directory exists, `wallet_valid` confirms wallet integrity. Any failure returns `"status": "degraded"` with details [24].

For container orchestration (Kubernetes, Docker Swarm), this endpoint provides the liveness probe. The health check is the contractual interface between CANONIC and the infrastructure hosting it.

23.4. 22.4 Governance Dashboard Metrics

The governance dashboard — DEV's real-time view of fleet health — renders directly from these metrics:

```
Fleet Scope Count:     185 (all repositories)
Fleet Average Score:   247/255 (97%)
Scopes at 255:         172 (93%)
Scopes Below 127:      3 (action required)
Drift Events (30d):    7
COIN Minted (30d):     4,230
Active TALK Sessions:  1,247
```

The dashboard is a view, not a separate data source. The metrics are the truth; the dashboard renders it [24].

23.5. 22.5 Alerting Integration

Alerting is not built into CANONIC — it uses standard Prometheus alerting that hospitals already operate:

```yaml
# Alert: clinical AI scope dropped below FULL
- alert: GovernanceDrift
  expr: canonic_scope_score < 255
  for: 1h
  labels:
    severity: warning
  annotations:
    summary: "Scope {{ $labels.scope }} score {{ $value }}/255"
```

Governance alerts integrate with your existing alert routing (PagerDuty, OpsGenie, email). A clinical AI scope dropping below 255 triggers the same pipeline as a CPU spike or a database connection failure. Governance monitoring is infrastructure monitoring [24].

23.6. 22.6 Constraints

```
MUST NOT: Block service on metrics collection failure
MUST NOT: Store metrics in external database (in-memory only)
MUST NOT: Require auth for /health and /metrics (public observability)
MUST:     Expose Prometheus-compatible text exposition format
MUST:     Include governance-specific metrics (scope scores, drift events)
MUST:     Health check verifies live state (not cached configuration)
```

In-memory only is deliberate. No time-series database, no metrics aggregator, no storage backend. In-memory counters are fast, reliable, and zero-maintenance. In hospital environments where every external dependency requires a security review, zero-dependency monitoring is a significant operational advantage [24].

23.7. 22.7 MONITORING CLI Operations

```
# Check current health status
magic health

# Output:
# Status:       HEALTHY
# Uptime:       14d 6h 32m
# Ledger head:  evt:04847 (8s ago)
# Wallet check: 47 wallets, 0 mismatches
# Scope scores: 73/73 at 255

# Query specific metrics
magic metrics --scope SERVICES/TALK/MAMMOCHAT

# Output:
# canonic_api_requests_total{scope="MAMMOCHAT"}: 3,847
# canonic_api_request_duration_seconds_p50{scope="MAMMOCHAT"}: 0.042
# canonic_api_request_duration_seconds_p95{scope="MAMMOCHAT"}: 0.187
# canonic_api_request_duration_seconds_p99{scope="MAMMOCHAT"}: 0.432
# canonic_scope_score{scope="MAMMOCHAT"}: 255
# canonic_drift_events_total{scope="MAMMOCHAT"}: 0

# Fleet-wide metrics summary
magic metrics --fleet

# Output:
# Metric                          Value
# Total scopes monitored:         73
# Scopes at 255:                  73 (100%)
# API requests (24h):             12,450
# Mean response time (p50):       0.038s
# LEDGER events (24h):            127
# COIN minted (24h):              892
# Drift events (24h):             0
# Active TALK sessions:           1,247
```

23.8. 22.8 Metric Cardinality and Governance

Cardinality limits prevent label explosion:

Metric Type	Max Labels	Max Cardinality	Rationale
Scope score	scope	1 per scope	Bounded by scope count
API requests	endpoint, method, status	3 labels max	Bounded by route table
Auth events	result	2 (ok/fail)	Binary outcome
LEDGER events	event_type	14 types max	Bounded by service count
Wallet balance	user	1 per user	Bounded by user count

Unbounded labels are prohibited. No metric may use request IDs, timestamps, or free-text fields as labels — preventing the in-memory counter store from growing unbounded, a common failure mode in monitoring systems [24].

23.9. 22.9 Drift Detection Pipeline

Governance drift detection runs as continuous validation:

```
Every 5 minutes:
  for each scope in fleet:
    score = magic validate --scope {scope} --quiet
    if score < previous_score:
      emit DRIFT event to LEDGER
      emit canonic_drift_events_total increment
      trigger NOTIFIER:DRIFT_ALERT
    update canonic_scope_score gauge
```

Drift detection is poll-based, not event-driven. Every 5 minutes, MONITORING revalidates every scope — catching drift that no commit triggered: an INTEL source that expired, an evidence link that went stale, a credential that lapsed.

```
# Manual drift check
magic drift-check

# Output:
```

```
# Scope                          Previous  Current  Delta
# SERVICES/TALK/MAMMOCHAT        255       255      0
# SERVICES/TALK/ONCOCHAT         255       255      0
# SERVICES/TALK/CARDICHAT        255       191      -64 ← DRIFT
# SERVICES/LEDGER                255       255      0
# [... 69 more scopes at 255 ...]
#
# DRIFT DETECTED: 1 scope regressed
# NOTIFIER: DRIFT_ALERT sent to scope writers
# LEDGER: DEBIT:DRIFT event recorded (-64 COIN)
```

23.10. 22.10 Grafana Dashboard Templates

Grafana dashboard templates ship with governance metadata:

```
{
  "_generated": true,
  "dashboard": "canonic-governance",
  "panels": [
    {
      "title": "Fleet Governance Score",
      "type": "gauge",
      "query": "avg(canonic_scope_score)",
      "thresholds": {"green": 255, "yellow": 191, "red": 127}
    },
    {
      "title": "Drift Events (7d)",
      "type": "graph",
      "query": "increase(canonic_drift_events_total[7d])"
    },
    {
      "title": "API Latency (p95)",
      "type": "graph",
      "query": "canonic_api_request_duration_seconds{quantile='0.95'}"
    },
    {
      "title": "COIN Minted vs Debited",
      "type": "graph",
      "query": "increase(canonic_ledger_events_total{event_type=~'MINT.*|DEBIT.*'}[24h])"
    }
  ]
}
```

The template is compiled from the governance tree — `_generated: true` applies. Import it into your existing Grafana instance; metrics are already exposed via `/api/v1/metrics`. No custom integration code required [24].

23.11. 22.11 MONITORING and Service-Level Objectives

Governance-aligned SLOs:

SLO	Target	Measurement	Alert Threshold
Governance score	255 (100%)	`canonic_scope_score`	< 255 for > 1h
API availability	99.9%	`canonic_api_requests_total{status="200"}` / total	< 99.9% over 30d
API latency (p95)	< 500ms	`canonic_api_request_duration_seconds`	> 500ms for > 5m
Drift recovery time	< 24h	Time between DEBIT:DRIFT and score restoration	> 24h
LEDGER integrity	100%	`vault verify-chain` pass rate	Any failure
Notification delivery	100%	`NOTIFIER:DELIVER` success rate	Any failure

SLOs are declared in the MONITORING scope's CANON.md:

```
## SLOs

| objective | target | window |
|-----------|--------|--------|
| governance_score | 255 | continuous |
| api_availability | 99.9% | 30d rolling |
| api_latency_p95 | 500ms | 5m window |
| drift_recovery | 24h | per-event |
| chain_integrity | 100% | continuous |
```

`build --monitoring` compiles these declarations into Prometheus alerting rules — no manual alert configuration [24].

23.12. 22.12 Clinical MONITORING Example

A hospital NOC monitoring three governed clinical AI services:

```
Hospital NOC Dashboard (Grafana)
```

Clinical AI Governance — Real-Time			
Service	Score	Latency	Sessions
MammoChat	255 ●	42ms p50	847 active
OncoChat	255 ●	38ms p50	312 active
CardiChat	191 ▲	45ms p50	88 active
Fleet	91.6%	40ms avg	1,247 tot

▲ CardiChat DRIFT — PagerDuty incident #4521 open
● All other services GOVERNED

The dashboard renders Prometheus metrics. PagerDuty fired when `canonic_scope_score{scope="CARDICHAT"} < 255` for 1 hour. MONITORING did not build any of this infrastructure — it exposed the metrics, and existing hospital systems consumed them [24].

23.13. 22.13 Clinical Vignette: MONITORING Catches Silent Model Drift

Kaiser Permanente (Oakland, 39 hospitals) deploys 6 clinical AI agents under CANONIC governance. For 8 months, all 6 maintain 255 with zero drift events. Then MONITORING catches something governance scoring alone would miss.

The `canonic_api_request_duration_seconds` metric for OncoChat shows a gradual latency increase over 3 weeks:

```
Week 1:  p50 = 38ms, p95 = 187ms, p99 = 432ms
Week 2:  p50 = 45ms, p95 = 234ms, p99 = 612ms
Week 3:  p50 = 67ms, p95 = 412ms, p99 = 1,847ms
```

Governance score stays 255 throughout — latency is not a governance dimension. But the SLO alerting rule fires:

```
- alert: OncoChat_Latency_SLO_Breach
  expr: canonic_api_request_duration_seconds{scope="ONCOCHAT", quantile="0.95"} > 0.5
  for: 5m
  labels:
    severity: warning
  annotations:
    summary: "OncoChat p95 latency {{ $value }}s exceeds 500ms SLO"
```

The engineering team traces the latency to the LLM provider's API. The model version was silently updated — a common occurrence with cloud LLM APIs. The new version produces approximately 40% more tokens per completion. Content quality is unchanged (OncoChat still references correct NCCN guidelines), but response time exceeds clinical workflow requirements.

```
magic metrics --scope SERVICES/TALK/ONCOCHAT --detail

# OncoChat Detailed Metrics (7d):
# API requests:              3,847
# Mean tokens/response:      842 (up from 602, +40%)
# Mean response time:        67ms (up from 38ms, +76%)
# p95 response time:         412ms (up from 187ms, +120%)
# p99 response time:         1,847ms (up from 432ms, +328%)
# Governance score:          255 (unchanged)
# Drift events:              0
# Session completion rate:   94.2% (down from 97.8%, -3.6pp)
#
# Correlation: token count ↑ correlates with latency ↑ (r=0.94)
# Root cause hypothesis: LLM model update increased verbosity
```

The governance team caps OncoChat's `max_tokens` at 600 (matching pre-update behavior). The adjustment is governance work — the systemPrompt is a governed file requiring `magic validate`. Latency returns to baseline within one deployment cycle:

```
Post-fix metrics (24h):
  p50 = 39ms, p95 = 191ms, p99 = 445ms
  Session completion rate: 97.6%
  SLO status: PASS
```

Governance scoring tells you the structure is complete. MONITORING tells you the system is performing within operational bounds. A system can be perfectly governed (255) and operationally degraded (p99 > 1.8s). MONITORING bridges that gap — runtime visibility that static governance scoring cannot provide [24].

23.14. 22.14 MONITORING Data Retention

In-memory metrics are ephemeral by design. For long-term analysis, take snapshots:

```
magic metrics --snapshot --format json > metrics-2026-03-10.json

# Snapshot contents:
# {
#   "timestamp": "2026-03-10T15:00:00Z",
#   "scopes": 73,
#   "metrics": {
#     "canonic_scope_score": {"MAMMOCHAT": 255, "ONCOCHAT": 255, ...},
#     "canonic_api_requests_total": {"MAMMOCHAT": 12450, ...},
#     "canonic_drift_events_total": {"MAMMOCHAT": 0, ...}
#   }
# }
```

Retention policy:

Retention Tier	Resolution	Duration	Storage
Real-time	Per-request (in-memory)	1 hour	RAM
Hourly snapshots	1 per hour	7 days	Local disk
Daily snapshots	1 per day	90 days	R2 object storage
Monthly summaries	1 per month	3 years	R2 archive

Recent metrics retain high resolution for debugging; long-term trends preserve at lower resolution for capacity planning and compliance. Monthly summaries suffice for Joint Commission evidence — the surveyor needs “was this AI monitored?” not “what was the p95 latency at 3:47 PM on Tuesday.”

23.15. 22.15 MONITORING Cross-Fleet Comparison

Cross-fleet comparison surfaces outliers that per-scope monitoring would miss:

```
magic metrics --fleet --compare

# Fleet Cross-Comparison (7d averages):
#
# Agent        Score  p50    p95    Sessions  Drift  COIN   Health
# MammoChat    255    42ms   187ms  847/d     0      892    HEALTHY
# OncoChat     255    39ms   191ms  312/d     0      756    HEALTHY
# CardiChat    255    45ms   201ms  204/d     0      445    HEALTHY
```

```
# PulmoChat    255    41ms   189ms  156/d     0      312    HEALTHY
# NeuroChat    255    52ms   234ms  98/d      0      234    HEALTHY
# GastroChat   255    38ms   178ms  187/d     0      378    HEALTHY
#
# Fleet summary:
#   All 6 agents at 255 □
#   Fleet p50: 43ms (within 500ms SLO)
#   Fleet sessions: 1,804/day
#   Fleet COIN minted (7d): 3,017
#   Outlier: NeuroChat p95 23% above fleet average (investigate)
```

NeuroChat's elevated p95 latency (234ms vs 191ms fleet average) warrants investigation — perhaps the neurology knowledge base is larger, or the model generates longer differential diagnosis lists. The comparison is LEDGER-derived; no manual data collection required.

23.16. 22.16 MONITORING and Incident Response

Incident response integrates directly with MONITORING:

```
# Incident declared when governance + operational thresholds both breach
magic incident --declare \
  --scope SERVICES/TALK/CARDICHAT \
  --trigger "scope_score < 255 AND p95 > 500ms" \
  --severity P1

# Output:
# INCIDENT DECLARED: INC-2026-0047
# Scope:     SERVICES/TALK/CARDICHAT
# Triggers:  score=191 (< 255), p95=1247ms (> 500ms)
# Severity:  P1 (critical — clinical AI degraded)
# Timeline:
#   15:00  DEBIT:DRIFT detected (score 255→191)
#   15:05  p95 latency breach (1247ms > 500ms SLO)
#   15:06  NOTIFIER: DRIFT_ALERT sent to 4 principals
#   15:06  PagerDuty: incident created (#4521)
#   15:10  INCIDENT DECLARED: INC-2026-0047
# LEDGER: MONITORING:INCIDENT_DECLARE (evt:10100)
```

Every stage of the incident lifecycle is governed:

Stage	Action	LEDGER Event
DECLARE	Incident opened, responders paged	MONITORING:INCIDENT_DECLARE
INVESTIGATE	Root cause identified	MONITORING:INCIDENT_UPDATE
MITIGATE	Temporary fix applied	MONITORING:INCIDENT_MITIGATE
RESOLVE	Permanent fix deployed, score restored	MONITORING:INCIDENT_RESOLVE
REVIEW	Post-incident review completed	MONITORING:INCIDENT_REVIEW

The incident timeline is reconstructible from the chain — providing the incident response documentation HIPAA §164.308(a)(6) requires without manual incident reports [24].

23.17. 22.17 Governance Proof: Observability Completeness

Observability completeness proof:

```
For every governed service S in the fleet:
  1. S exposes /api/v1/metrics (Prometheus format)
  2. S exposes /api/v1/health (JSON health status)
  3. MONITORING polls S every 5 minutes for governance score
  4. MONITORING tracks S's API latency, request count, and error rate
  5. MONITORING fires alerts when SLOs breach
  6. All metrics are LEDGER-compatible (derivable from governed events)

Completeness claim:
  No governance event occurs without MONITORING visibility
  No operational degradation persists without MONITORING detection
  No alert goes unacknowledged without MONITORING escalation

Proof:
  - Governance events produce LEDGER entries
  - LEDGER entries produce metric increments
  - Metric increments produce Prometheus time-series
  - Prometheus time-series produce alerting evaluations
  - Alerting evaluations produce NOTIFIER deliveries
  - NOTIFIER deliveries produce acknowledgment requirements
  - Acknowledgment requirements produce escalation chains
  - The chain from event to human awareness is unbroken
```

MONITORING does not govern — it observes governance. The feedback loop makes governance self-correcting: drift detected, alerts fired, humans respond, governance restored. Without MONITORING,

governance is open-loop and drift persists indefinitely. With it, governance is closed-loop. The loop closure time (mean time from drift detection to restoration) is the operational measure of governance health. Q.E.D.
2412

23.18. 22.18 MONITORING and Regulatory Audit Readiness

Traditional audit preparation requires weeks of documentation gathering. MONITORING eliminates this burden — the audit evidence is already structured and continuous.

Regulatory Framework	MONITORING Evidence	Query
HIPAA §164.312(b)	Audit controls — all access events logged	`magic monitoring --audit hipaa --period 365d`
Joint Commission	AI governance documentation — continuous score tracking	`magic monitoring --audit jcaho --scope TALK/*`
21 CFR 820.30(e)	Design verification — build/deploy pipeline logs	`magic monitoring --audit fda --pipeline`
SOC 2 Type II	Change management — governance score time-series	`magic monitoring --audit soc2 --controls`
GDPR Article 35	Data protection impact — federation boundary logs	`magic monitoring --audit gdpr --federation`

```
# Generate audit package for upcoming Joint Commission survey
magic monitoring --audit-package jcaho \
  --scope SERVICES/TALK/* \
  --period 2025-03-10:2026-03-10 \
  --output audit-jcaho-2026.json

# Output:
# Audit package generated: audit-jcaho-2026.json
# Scopes covered:      5 TALK agents
# Score observations:   52,560 (5-minute intervals × 365 days × 5 scopes)
```

```
# Incidents:           3 (all resolved, MTTR < 4h)
# Deploy events:       47 (all governed, 0 ungoverned)
# Drift events:        12 (all healed within SLO)
# Package size:        2.4 MB (structured JSON)
```

The package is deterministic — same query, same data, same output. An auditor can independently verify every claim by replaying LEDGER events. MONITORING does not generate audit evidence; it structures the evidence that governance already produces [2412].

23.19. 22.19 Custom Dashboard Compilation

Dashboards are compiled from governance metadata, not hand-configured in Grafana. The definition lives in the scope's MONITORING.md:

```
## Dashboard

| panel | metric | threshold | alert |
|-------|--------|-----------|-------|
| Score | canonic_scope_score | < 255 | CRITICAL |
| Latency p99 | canonic_api_latency_p99 | > 500ms | WARNING |
| Error Rate | canonic_api_error_rate | > 1% | CRITICAL |
| Session Count | canonic_talk_sessions_total | < 10/day | INFO |
| COIN Minted | canonic_coin_minted_total | 0 for 7d | WARNING |
```

```
# Compile dashboards from MONITORING.md
build --dashboards

# Output:
# Compiling dashboards from MONITORING.md files...
#   SERVICES/TALK/MAMMOCHAT: 5 panels □
#   SERVICES/TALK/ONCOCHAT: 5 panels □
#   SERVICES/API: 4 panels □
# Dashboards written to _data/dashboards.json (_generated)
```

The compiled dashboard is `_generated`. If it is wrong, fix MONITORING.md — not the Grafana JSON. The chain is unbroken: LEDGER events produce metrics, metrics produce dashboards, dashboards produce human awareness [24].

Chapter 24

Chapter 23: DEPLOY

Governed artifact delivery. The pipeline that ships only validated code [25]. DEPLOY sits at the end of the governance pipeline: `magic validate` (Chapter 42) gates entry, `build` (Chapter 44) compiles artifacts, and DEPLOY ships them. The fleet is live at mammo.chat, oncochat.hadleylab.org, medchat.hadleylab.org, and gorunner.pro. For the build pipeline details, see Chapter 46.

24.1. 23.1 Axiom

DEPLOY is governed artifact delivery. Build before deploy. Never ship unvalidated artifacts. DESIGN deploys first [25].

DEPLOY sits at the end of the governance pipeline, and that position is deliberate. By the time code reaches DEPLOY, it has already passed through every governance gate: committed with `magic commit`, validated to 255, compiled by `build`. DEPLOY does not re-evaluate governance — it enforces the score that `magic validate` already computed and ships the artifacts that `build` already compiled. The deployment order is architectural, not configurable: the DESIGN theme deploys first because all fleet sites reference it via Jekyll `remote_theme`, then fleet sites deploy in sequence. Change the order and layouts break. The order is governed because the dependency is real [25].

24.2. 23.2 The Full Pipeline

```
governance work → commit → magic validate (must = 255) → build → deploy
```

Each stage is a gate. If any gate fails, the pipeline stops:

```
1. COMMIT:       git commit (governance work recorded)
```

```
2. VALIDATE:     magic validate → 255 required
3. BUILD:        build script compiles governance → artifacts
4. DESIGN:       deploy DESIGN theme (remote_theme base)
5. FLEET:        deploy fleet sites (depend on DESIGN theme)
6. VERIFY:       smoke test — confirm fleet sites serve governed content
7. LEDGER:       DEPLOY event recorded on LEDGER
```

Step 2 is the critical gate. If `magic validate` returns anything less than 255, the pipeline stops. There is no override flag. There is no "deploy anyway" escape hatch. The only path to production is through governance: fix the gaps, achieve 255, then deploy. This single constraint eliminates an entire category of incidents — the ones that start with "we shipped it before the documentation was ready" and end with an audit finding [25].

24.3. 23.3 Deploy Gates

Gate	Condition	Block Action	Override
BUILD	`build` must pass	BLOCK_DEPLOY	None — fix the build
VALIDATE	`magic validate` must return 255	BLOCK_DEPLOY	None — govern first
FREEZE	FROZEN state active in scope	BLOCK_DEPLOY	Manual unfreeze required
PRIVATE	PRIVATE scope in public fleet	BLOCK_DEPLOY	None — change privacy
DRIFT	Drift gate detects catalog mismatch	BLOCK_PUBLISH	Regenerate and reconcile

The PRIVATE gate deserves special attention. A scope marked `privacy: PRIVATE` in its CANON.md cannot be deployed to a public fleet site — even if it validates to a perfect 255. Private scopes contain confidential data: VAULT projections, patent applications, deal terms. A 255 score means the governance is complete, not that the content is public. The PRIVATE gate prevents the specific class of incident where a well-governed but confidential scope accidentally ships to a public URL [25].

24.4. 23.4 DESIGN-First Deploy Order

```
DESIGN theme → fleet site 1 → fleet site 2 → ... → fleet site N
```

Why does DESIGN deploy first? Because every fleet site pulls its layouts, tokens, and component includes from the DESIGN theme via `remote_theme`. If a fleet site deploys before the theme update lands, it renders

against stale assets — broken layouts, missing design tokens, inconsistent styling across the fleet. The dependency is architectural, and the deploy order reflects it [25].

The DESIGN theme is itself governed. Its visual tokens (`_TOKENS.scss`), layout templates, and component includes are all constrained by DESIGN.md. When a design change ships — a new token value, a new component, a revised layout — every fleet site must render with the update. DESIGN-first deployment is not a convention you remember to follow; it is enforced by the pipeline.

24.5. 23.5 Rollback

When a deployment causes issues, rollback reverts a fleet site to its previous state:

```
$ rollback hadleylab.org HEAD~1
Rolling back hadleylab.org to commit abc1234...
  → Verifying commit exists... ✓
  → Force-with-lease push... ✓
  → Verifying site renders... ✓
Rollback complete. Previous HEAD preserved at abc1234.
```

Rollback uses `--force-with-lease` rather than `--force` — a critical distinction. A lease-based force push fails if the remote contains commits the local does not know about, preventing you from accidentally overwriting work that another team member pushed while you were diagnosing the issue. The command also prompts for confirmation before executing, because destructive operations in a governed system require explicit human approval [25].

24.6. 23.6 Containerization

The CANONIC economy services run in containers for deployment isolation:

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

Container governance constraints:

Constraint	Enforcement	Rationale
Non-root user	`USER nobody`	Principle of least privilege

Constraint	Enforcement	Rationale
No secrets in layers	`.dockerignore` excludes `.env`, credentials	Secrets managed by runtime, not image
Health check	Built-in `HEALTHCHECK` directive	Container orchestrator monitors liveness
Slim base image	`python:3.11-slim`	Minimal attack surface
Port 8255	`EXPOSE 8255`	CANONIC-specific port (255 * 32 + 95)

24.7. 23.7 Cloudflare Workers Deployment

For the TALK service fleet, deployment targets Cloudflare Workers — edge-deployed serverless functions that serve clinical AI conversations with global low-latency:

```
build → wrangler deploy → Cloudflare Worker (edge)
  → Routes: mammochat.com, oncochat.ai, medchat.ai
  → KV bindings: session storage, inbox, ledger cache
  → Secrets: API keys (runtime-injected, never in code)
```

The Cloudflare Workers deployment pattern means that clinical AI agents are served from edge locations closest to the clinician — a radiologist in Orlando gets MammoChat served from the Miami edge, not from a central server. Latency is minimized. The governance is identical — the Worker reads the same CANON.json, serves the same systemPrompt, logs to the same LEDGER [25].

24.8. 23.8 CI/CD Integration

DEPLOY integrates with GitHub Actions for continuous governance:

```yaml
name: Governed Deploy
on:
  push:
    branches: [main]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          submodules: recursive
```

```
      - name: Validate
        run: python3 .canonic/bin/magic.py scan . --min 255
      - name: Build
        run: ./build
      - name: Deploy DESIGN
        run: ./deploy design
      - name: Deploy Fleet
        run: ./deploy fleet
```

The CI/CD pipeline enforces the governance gate automatically. Every push to main triggers validation. If validation fails, the pipeline fails. If validation passes, the build and deploy proceed. The governance gate is in the pipeline, not in human discipline. You cannot forget to validate because the pipeline validates for you [25].

24.9. 23.9 DEPLOY and Clinical AI Operations

Three properties of the DEPLOY pipeline matter most for clinical AI operations:

Zero-downtime deployment. Cloudflare Workers deploy atomically — the new version replaces the old in a single operation. Clinical AI agents are never offline during a deployment window, which matters when a radiologist in the middle of a reading session cannot afford a "service unavailable" page.

Governance-gated releases. No clinical AI update ships without 255 validation. If a model update breaks governance — a missing INTEL reference, a broken disclaimer, an invalid persona — the pipeline blocks deployment before the clinician ever sees an ungoverned response. The gate is in the pipeline, not in a human checklist.

Complete audit trail. Every deployment is a LEDGER event recording who deployed what, when, with what governance score, from what commit. When a Joint Commission surveyor asks "when was this AI last updated and who approved it?" the answer is a single `magic ledger` query, not a scramble through email threads and Slack messages [25] [3].

24.10. 23.10 Deploy Artifact Registry

Every deployment produces a governed artifact record:

```
{
  "_generated": true,
  "deployment": {
    "id": "deploy:2026-03-10-001",
    "scope": "SERVICES/TALK/MAMMOCHAT",
    "commit": "abc1234def5678",
```

```
    "score_at_deploy": 255,
    "deployer": "dexter",
    "timestamp": "2026-03-10T15:00:00Z",
    "target": "cloudflare-workers",
    "routes": ["mammochat.com"],
    "previous_deploy": "deploy:2026-03-08-002",
    "ledger_event": "evt:04860"
  }
}
```

The artifact registry is compiled output — `_generated: true` applies, so you fix the pipeline if the output is wrong, not the JSON. Every deployment is traceable through its full provenance chain: the commit it came from, the score at deploy time, who deployed it, when, where, and which previous deployment it replaced. This traceability enables deployment comparison:

```
# Compare two deployments
magic deploy-diff deploy:2026-03-08-002 deploy:2026-03-10-001

# Output:
# Commits between deployments: 4
# Score change: 255 → 255 (no regression)
# Files changed: 3 (INTEL.md, LEARNING.md, VOCAB.md)
# New INTEL references: 2 (NCCN-2026-v2, ACR-2026-UPDATE)
# Governance delta: 0 COIN (maintenance — score unchanged)
```

24.11. 23.11 Deploy Environments

DEPLOY supports three governed environments:

Environment	Purpose	Validation	Access
STAGING	Pre-production testing	Score must be ≥ 191	Writers only
PRODUCTION	Live deployment	Score must be 255	Public
CANARY	Gradual rollout (10% □ 50% □ 100%)	Score must be 255	Public (subset)

```
# Deploy to staging (relaxed score requirement)
deploy --env staging --scope SERVICES/TALK/MAMMOCHAT
# Score 191 ≥ 191: PASS → deploying to staging

# Promote staging to production (requires 255)
```

```
deploy --promote staging --scope SERVICES/TALK/MAMMOCHAT
# Score 255 = 255: PASS → deploying to production

# Canary deployment (gradual rollout)
deploy --canary 10 --scope SERVICES/TALK/MAMMOCHAT
# 10% of traffic routed to new version
# Monitor for 1h, then:
deploy --canary 50 --scope SERVICES/TALK/MAMMOCHAT
deploy --canary 100 --scope SERVICES/TALK/MAMMOCHAT
```

The canary pattern addresses a reality that governance alone cannot solve: a scope can validate to 255 and still produce unexpected clinical responses in production, because structural completeness does not guarantee behavioral correctness under real-world load. The canary deployment routes a small percentage of traffic to the new version while monitoring response quality metrics. If clinical alerts fire — SESSION_ANOMALY events, user reports, latency spikes — the canary is rolled back before it reaches full traffic. Governance gates prevent structurally incomplete code from deploying; canary deployment prevents behaviorally surprising code from scaling [25].

24.12. 23.12 Deploy Freeze

Deploy freezes prevent deployments during critical periods:

```
# Activate deploy freeze
deploy --freeze --scope SERVICES/TALK/MAMMOCHAT \
  --reason "Joint Commission survey in progress" \
  --until "2026-03-15"

# Output:
# DEPLOY FREEZE activated
# Scope:    SERVICES/TALK/MAMMOCHAT
# Reason:   Joint Commission survey in progress
# Until:    2026-03-15T23:59:59Z
# LEDGER:   DEPLOY:FREEZE recorded
#
# All deploy attempts will be BLOCKED until freeze expires or is lifted.

# Attempt deploy during freeze
deploy --scope SERVICES/TALK/MAMMOCHAT
# ERROR: DEPLOY_FROZEN
# Reason: Joint Commission survey in progress
# Expires: 2026-03-15
# Action:  Wait for freeze expiry or run: deploy --unfreeze
```

```
# Lift freeze early
deploy --unfreeze --scope SERVICES/TALK/MAMMOCHAT
# LEDGER: DEPLOY:UNFREEZE recorded
```

Deploy freezes exist because healthcare environments have periods where system stability is non-negotiable. During a Joint Commission survey, no clinical AI should change behavior — not because something might break, but because any change during the survey window creates an audit question that is easier to prevent than to answer. The freeze gate is a governance constraint enforced by the pipeline, not a reminder in a shared calendar [25].

24.13. 23.13 Multi-Site Deploy Orchestration

For organizations with multiple TALK sites, DEPLOY orchestrates fleet-wide deployment:

```
# Deploy entire fleet
deploy --fleet

# Output:
# Fleet deployment order:
#   1. DESIGN theme (canonic-design) ..... DEPLOYED □
#   2. mammochat.com ..................... DEPLOYED □
#   3. oncochat.ai ....................... DEPLOYED □
#   4. medchat.ai ........................ DEPLOYED □
#   5. cardichat.ai ...................... BLOCKED □  (score 191, requires 255)
#   6. gorunner.pro ...................... DEPLOYED □
#
# Fleet result: 5/6 deployed, 1 blocked
# Blocked: cardichat.ai (governance drift — heal before deploy)
```

The fleet deployment respects per-site governance. A site that has drifted below 255 is skipped — but critically, it does not block the rest of the fleet. Each site is independently governed, independently validated, and independently deployed. The fleet command is a convenience that runs independent deploys in the governed order, not a monolithic operation where one failure cascades to all [25].

24.14. 23.14 Clinical Vignette: DEPLOY Prevents Ungoverned Model Update

Cedars-Sinai (Los Angeles, 2 hospitals + 40 clinics) schedules a quarterly model update for their clinical AI fleet — upgrading the LLM backend from GPT-4o to GPT-4o-2026-Q1. The update affects all 5 TALK

agents: OncoChat, MammoChat, CardiChat, DermaChat, and PsychChat. The deployment follows the governed pipeline.

Pre-Deployment Validation. Each agent's systemPrompt is updated to reference the new model version. The systemPrompt is a governed file — part of the scope's CANON.md-derived configuration. The update triggers revalidation:

```
# Step 1: Update model reference across all 5 agents
for agent in ONCOCHAT MAMMOCHAT CARDICHAT DERMACHAT PSYCHCHAT; do
  sed -i 's/gpt-4o-2024-08/gpt-4o-2026-q1/g' \
    SERVICES/TALK/$agent/config.md
done

# Step 2: Validate all updated agents
magic validate --all --strict

# SERVICES/TALK/ONCOCHAT:    255 □
# SERVICES/TALK/MAMMOCHAT:   255 □
# SERVICES/TALK/CARDICHAT:   255 □
# SERVICES/TALK/DERMACHAT:   255 □
# SERVICES/TALK/PSYCHCHAT:   247 □
#   ERROR: GOV-018 — INTEL reference broken
#   at SERVICES/TALK/PSYCHCHAT/INTEL.md:34
#   reference: "GPT-4o Safety Card v2024-08"
#   issue: Model-specific safety documentation not updated for 2026-Q1
#   fix: Update safety card reference to GPT-4o-2026-Q1 version
```

The Gate Holds. PsychChat fails validation because its INTEL.md references a model-specific safety card that is version-locked to the old model. The safety card documents the model's behavior boundaries for psychiatric content — suicide risk assessment responses, self-harm content handling, and crisis intervention routing. The new model version has a new safety card with updated behavioral parameters.

```
# Deploy attempt
deploy --fleet

# Output:
# Pre-deployment validation:
#   DESIGN theme ......................... 255 □
#   mammochat.com ........................ 255 □  → DEPLOY
#   oncochat.ai .......................... 255 □  → DEPLOY
#   cardichat.ai ......................... 255 □  → DEPLOY
#   dermachat.ai ......................... 255 □  → DEPLOY
#   psychchat.ai ......................... 247 □  → BLOCKED
#
# Fleet result: 5/6 deployed, 1 blocked
```

```
# Blocked: psychchat.ai (INTEL reference broken — GOV-018)
# LEDGER: DEPLOY:BLOCK recorded (evt:10500)
```

Why this matters clinically. PsychChat handles the most sensitive interactions in the fleet — suicide risk assessment, self-harm content, crisis intervention routing. The safety card documents the model's behavior boundaries for exactly these scenarios. If the new model version changed how it handles suicide risk prompts (a plausible change between model versions), and the governance documentation still references the old behavior boundaries, then the clinical safety net is misaligned with the actual model behavior. A psychiatrist relying on documented safety properties that no longer hold is worse than having no documentation at all.

The DEPLOY gate holds PsychChat on the old model version — which matches its current safety card — while the other four agents upgrade immediately. The psychiatry team has 48 hours to update INTEL.md with the new safety card, revalidate to 255, and deploy independently.

```
# PsychChat team updates INTEL.md with new safety card
# Then deploys independently:
magic validate SERVICES/TALK/PSYCHCHAT
# Score: 255 □

deploy --scope SERVICES/TALK/PSYCHCHAT
# Pre-deployment: 255 □
# Deploying to psychchat.ai...
# DEPLOYED □
# LEDGER: DEPLOY (evt:10520)
```

The total fleet update time is 48 hours. Five agents deploy immediately; one is delayed for safety documentation alignment. Zero ungoverned deployments reach production. The governance gate did exactly what it was designed to do — it protected the most sensitive agent in the fleet from a documentation-to-behavior mismatch that no human reviewer caught during the batch update [25].

24.15. 23.15 Blue-Green Deployment Pattern

For zero-downtime deployment with instant rollback capability, DEPLOY supports blue-green deployment:

```
Blue environment:  current production (live traffic)
Green environment: new version (staged, no traffic)

Deploy sequence:
  1. Deploy new version to GREEN
  2. Run smoke tests on GREEN
  3. Switch traffic: BLUE → GREEN (atomic DNS/routing switch)
```

```
  4. Monitor GREEN for 30 minutes
  5. If healthy: decommission BLUE
  6. If unhealthy: switch back to BLUE (instant rollback)
```

```
deploy --blue-green --scope SERVICES/TALK/ONCOCHAT

# Phase 1: Deploy to GREEN
#   Building artifacts... □
#   Deploying to green.oncochat.ai... □
#   Smoke tests: 12/12 passed □
#
# Phase 2: Traffic switch
#   Switching: blue.oncochat.ai → green.oncochat.ai
#   DNS propagation: 12 seconds
#   Traffic routing: 100% → GREEN
#
# Phase 3: Monitoring (30m)
#   [0:00] Score: 255, p50: 38ms, errors: 0
#   [5:00] Score: 255, p50: 39ms, errors: 0
#   [15:00] Score: 255, p50: 37ms, errors: 0
#   [30:00] Score: 255, p50: 38ms, errors: 0
#
# Phase 4: Decommission BLUE
#   blue.oncochat.ai decommissioned
#   GREEN promoted to production
# LEDGER: DEPLOY:BLUE_GREEN recorded (evt:10530)
```

If Phase 3 monitoring detects any issue — a score drop, a latency spike, an error rate increase — the rollback is instant. Traffic switches back to BLUE, which is still running the previous version with all its state intact. No rebuild, no redeployment, no waiting for a container to start. Just a routing switch that takes effect in seconds.

24.16. 23.16 Deployment Approval Workflows

For regulated environments, DEPLOY supports multi-party approval:

```
# CANON.md deploy approval configuration
deploy_approval:
  required_approvers: 2
  approval_roles:
    - governance_lead      # Must approve governance completeness
    - clinical_lead        # Must approve clinical safety
    - engineering_lead     # Must approve technical readiness
```

```
  approval_window: 72h     # Approval expires after 72 hours
  emergency_override: false # No override for clinical AI
```

```
deploy --scope SERVICES/TALK/PSYCHCHAT --request-approval

# Deployment approval request:
#   Scope:     SERVICES/TALK/PSYCHCHAT
#   Score:     255
#   Changes:   INTEL.md (safety card update), config.md (model version)
#   Requested: 2026-03-10T15:00:00Z
#   Expires:   2026-03-13T15:00:00Z
#
# Required approvals (2 of 3):
#   [ ] governance_lead (dexter)
#   [ ] clinical_lead (dr-psychiatry-chief)
#   [ ] engineering_lead (eng-martinez)
#
# LEDGER: DEPLOY:APPROVAL_REQUEST recorded (evt:10535)

# Approval process:
vault approve deploy:psychchat-2026-03-10 --role governance_lead
# Approved by dexter (governance_lead)
vault approve deploy:psychchat-2026-03-10 --role clinical_lead
# Approved by dr-psychiatry-chief (clinical_lead)
# 2/3 approvals received — threshold met

deploy --scope SERVICES/TALK/PSYCHCHAT --execute-approved
# Approval verified: 2/3 □  (within 72h window)
# Deploying...
# DEPLOYED □
# LEDGER: DEPLOY:APPROVED_EXECUTE (evt:10540)
```

Every approval in the workflow is a LEDGER event attributed to a verified identity — not an email "LGTM" or a Slack thumbs-up, but a cryptographically attributed governance action. For FDA-regulated clinical AI, this provides the design review evidence required by 21 CFR 820.30(e). The entire approval chain is reconstructible from the LEDGER at any point in the future [25].

24.17. 23.17 Deployment Metrics

DEPLOY tracks deployment health metrics:

```
magic deploy-report --period 90d

# Deployment Report — 90 Days
#
# Total deployments:          47
# Successful:                 44 (93.6%)
# Blocked (score < 255):      3 (6.4%)
# Rollbacks:                  1 (2.1%)
# Mean deploy time:           3m 12s
# Mean validation time:       6.1s
# Fleet deploys:              8
# Single-scope deploys:       39
#
# By environment:
#   Production:  32
#   Staging:     12
#   Canary:       3
#
# Deploy freezes:             2 (Joint Commission survey, system upgrade)
# Blue-green switches:        4
# Approval workflows:         6 (all approved within 24h)
#
# Fastest deploy:             1m 47s (single scope, no approval)
# Slowest deploy:             14m 33s (fleet + approval workflow)
```

24.18. 23.18 Governance Proof: Deployment Soundness

The DEPLOY governance proof establishes that only validated, governed code reaches production:

```
For every deployment D to production environment P:
  1. D.scope.score = 255 (validated by magic validate)
  2. D.build passes (all artifacts compile)
  3. D.approvals meet threshold (if required by CANON.md)
  4. D.design deploys before D.fleet (ordering enforced)
  5. D.freeze is not active (checked before execution)
  6. D.private is not set (public deployment cannot contain private scopes)

Proof by contrapositive:
  If any condition fails, DEPLOY:BLOCK event is recorded and deployment halts.
  - Score < 255 → BLOCK (no waiver, no override)
  - Build fails → BLOCK
  - Approvals insufficient → BLOCK
```

```
- DESIGN not deployed → fleet BLOCK
- Freeze active → BLOCK
- Private scope → BLOCK

No path exists from an unvalidated scope to production.
Every deployment in production traces to: a 255 score, a passing build,
sufficient approvals, correct ordering, no active freeze, and no privacy violation.

The LEDGER records every DEPLOY and every DEPLOY:BLOCK.
The deployment history is complete and verifiable.
Ungoverned code in production is structurally impossible.
```

Read the proof carefully: no path exists from an unvalidated scope to production. DEPLOY is not primarily a deployment tool — it is a governance gate that happens to ship code as a side effect. Its primary function is prevention: keeping ungoverned artifacts away from the environment where clinicians interact with them. Delivery is the secondary function. Prevention first, delivery second. Q.E.D. [25312]

24.19. 23.19 Deploy and Federation Boundaries

In federated deployments, each organization controls its own DEPLOY pipeline — Organization A cannot deploy to Organization B's fleet, full stop. The federation boundary is a deploy boundary. Cross-organization deployments require explicit federation agreements with signed deployment manifests. A scope that inherits from a federated parent validates against the parent's published contract but deploys to the child organization's infrastructure. The governance standard is shared across the federation; the deployment infrastructure is local to each organization [2519].

24.20. Deploy: March 2026

The deploy pipeline runs 9 steps:

1. **Validate** — `magic validate --all --strict` (255 gate — hard fail)
2. **Build** — 12-phase incremental compilation via FRESHNESS cache
3. **Generate** — Jekyll remote_theme compilation for all surfaces
4. **Private gate** — `gc_private()` strips 113 PRIVATE scopes from public output
5. **Health check** — post-deploy validation confirms the published site matches source
6. **Push to bare** — compiled output pushed to bare repository
7. **GitHub Pages** — bare repository serves via GitHub Pages
8. **Cloudflare** — DNS + CDN + Workers for API endpoints
9. **Notify** — LEDGER event: DEPLOY:SUCCESS with commit SHA and timestamp

The health check at step 5 is a hard gate — if the deployed site does not match the source after publication, the deploy is rolled back automatically. There is no manual override for a failed health check because a site that does not match its source is, by definition, serving ungoverned content. The governance freeze locks all 9 steps: the pipeline configuration itself does not change without an explicit unfreezing event.

FRESHNESS makes the build step practical at scale: incremental compilation skips unchanged scopes by comparing modification times against a local cache. A full fleet build (380 scopes) completes in under 3 seconds with a warm cache. Cold builds in CI (where correctness matters more than speed) take approximately 2 minutes.

Chapter 25

Chapter 24: Cross-Axiomatic Validation

A governance tree with sixty-nine scopes is not sixty-nine independent files. It is a compilation unit. Every axiom must agree with its parent (as established in Chapter 3). Every cross-scope reference must resolve. Every evidence bridge must hold weight. Cross-axiomatic validation is the mechanism that enforces this structural integrity — the immune system that catches contradictions, dangling references, and circular dependencies before they reach production. For the formal type-system treatment of this compilation model, see Chapters 36 and 37. The GALAXY visualization (Chapter 31) renders the cross-reference graph at hadleylab.org.

25.1. 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
    → 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 [3].

25.2. 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 [2].

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

The validator resolves references and verifies the cited scope exists and compiles [3].

25.3. 24.3 Upward Compilation

Validation walks upward. A child scope at 255 against a parent at 255 contributes to fleet-wide compilation — the fleet itself becomes a single compilation unit [3].

25.4. 24.4 Cross-Scope Reference Resolution

Every INTEL.md file declares its cross-scope connections. The validator must resolve every reference and confirm the target scope exists, compiles, and exposes the cited evidence. A dangling reference — a citation to a scope that does not compile — is a validation error.

Run the resolver:

```
magic validate --cross-refs
```

The resolver walks every INTEL.md in the governance tree. For each `references:` entry in frontmatter, it resolves the path, loads the target scope's CANON.json, and checks that the target compiles. The output is a cross-reference matrix:

```
CROSS-REFERENCE MATRIX
──────────────────────────────────────────────────────────────────
Source                          → Target                        Status
──────────────────────────────────────────────────────────────────
BOOKS/CANONIC-DOCTRINE/Ch4      → PAPERS/governance-as-compilation  ✓ 255
BOOKS/CANONIC-DOCTRINE/Ch10     → SERVICES/LEARNING                 ✓ 255
BOOKS/CANONIC-DOCTRINE/Ch25     → SERVICES/TALK                     ✓ 255
BOOKS/CANONIC-DOCTRINE/Ch30     → SERVICES/TALK/MAMMOCHAT           ✓ 255
SERVICES/TALK/MAMMOCHAT         → SERVICES/TALK/ONCOCHAT            ✓ 255
SERVICES/TALK/ONCOCHAT          → SERVICES/TALK/MAMMOCHAT           ✓ 255
```

```
SERVICES/TALK/RUNNER          → SERVICES/LEDGER              ✓ 255
SERVICES/GUIDEPOINT           → SERVICES/TALK                ✓ 255
DEALS/LLU                     → SERVICES/CLINICAL            ✓ 255
──────────────────────────────────────────────────────────────────

Result: 9/9 resolved. 0 degraded.
```

A degraded reference does not fail the source scope — it flags a dependency that has drifted. The governor decides whether to accept the degradation or escalate.

25.5. 24.5 The Cross-Axiom Consistency Check

Every child axiom must specialize its parent without contradicting it. The validator enforces this structurally — a child that narrows is valid; a child that conflicts is rejected.

Consider the axiom chain for MammoChat:

```
canonic-canonic:    "Governance is a type system"
  hadleylab-canonic:  "Clinical governance compiles"
    SERVICES/TALK:      "Governed conversation is clinical AI"
      TALK/MAMMOCHAT:     "Breast imaging intelligence, governed"
```

Each level narrows. None contradicts. If TALK/MAMMOCHAT declared "ungoverned breast imaging intelligence," the axiom contradicts `SERVICES/TALK` and fails validation.

Run the axiom chain check:

```
magic validate --axiom-chain SERVICES/TALK/MAMMOCHAT
```

Output:

```
AXIOM CHAIN: SERVICES/TALK/MAMMOCHAT
──────────────────────────────────────────────────

Level 0: canonic-canonic
  Axiom: "Governance is a type system"
Level 1: hadleylab-canonic
  Axiom: "Clinical governance compiles"
  Inherits: Level 0 ✓
Level 2: SERVICES/TALK
  Axiom: "Governed conversation is clinical AI"
  Inherits: Level 1 ✓
Level 3: SERVICES/TALK/MAMMOCHAT
  Axiom: "Breast imaging intelligence, governed"
```

```
  Inherits: Level 2 ✓
──────────────────────────────────────────────────

Chain: VALID (4 levels, all consistent)
```

25.6. 24.6 Evidence Bridge Types

Cross-scope evidence comes in four types. Each has different validation rules:

Bridge Type	Source	Target	Validation Rule	Example
Governance	CANON.md	CANON.md	Target must compile at 255	TALK references LEARNING
Paper	INTEL.md	PAPERS/*	Target must have DOI or arXiv ID	Ch4 cites [3]
Blog	INTEL.md	BLOGS/*	Target must have published date	Ch25 cites [22]
Service	INTEL.md	SERVICES/*	Target must have CANON.md + score	Ch30 cites DESIGN

```
# Governance bridge — strict. Both sides must be 255.
magic validate --bridge governance SERVICES/TALK SERVICES/LEARNING

# Paper bridge — resolves to PAPERS directory. Checks DOI field.
magic validate --bridge paper BOOKS/CANONIC-DOCTRINE PAPERS/governance-as-compilation

# Blog bridge — resolves to BLOGS directory. Checks date field.
magic validate --bridge blog BOOKS/CANONIC-DOCTRINE BLOGS/2026-02-18-what-is-magic

# Service bridge — resolves to SERVICES. Checks CANON.md + score.
magic validate --bridge service SERVICES/DESIGN SERVICES/TALK
```

25.7. 24.7 The Cross-Axiom Matrix

As governance trees grow, cross-scope references multiply. The cross-axiom matrix maps every bridge in the fleet. Build it:

```
magic intel --cross-matrix > cross-matrix.json
```

Output is a JSON adjacency list:

```
{
  "nodes": [
    { "id": "SERVICES/TALK", "score": 255, "tier": "MAGIC" },
    { "id": "SERVICES/LEARNING", "score": 255, "tier": "MAGIC" },
    { "id": "SERVICES/TALK/MAMMOCHAT", "score": 255, "tier": "MAGIC" },
    { "id": "SERVICES/TALK/ONCOCHAT", "score": 255, "tier": "MAGIC" },
    { "id": "SERVICES/TALK/MEDCHAT", "score": 255, "tier": "MAGIC" },
    { "id": "SERVICES/TALK/LAWCHAT", "score": 255, "tier": "MAGIC" },
    { "id": "SERVICES/TALK/FINCHAT", "score": 255, "tier": "MAGIC" },
    { "id": "SERVICES/TALK/RUNNER", "score": 255, "tier": "MAGIC" },
    { "id": "SERVICES/TALK/CARIBCHAT", "score": 255, "tier": "MAGIC" },
    { "id": "SERVICES/GUIDEPOINT", "score": 255, "tier": "MAGIC" }
  ],
  "edges": [
    { "source": "SERVICES/TALK/MAMMOCHAT", "target": "SERVICES/TALK", "type": "governance" },
    { "source": "SERVICES/TALK/ONCOCHAT", "target": "SERVICES/TALK/MAMMOCHAT", "type": "governance" },
    { "source": "SERVICES/TALK/RUNNER", "target": "SERVICES/LEDGER", "type": "governance" },
    { "source": "SERVICES/GUIDEPOINT", "target": "SERVICES/TALK", "type": "governance" },
    { "source": "BOOKS/CANONIC-DOCTRINE", "target": "PAPERS/governance-as-compilation", "type": "paper" }
  ]
}
```


The GALAXY visualization (Chapter 31) renders this matrix as an interactive graph. Edges pulse green where both sides compile at 255. Edges dim where a reference is degraded [18].

25.8. 24.8 Circular Reference Detection

Cross-scope references must be acyclic. If scope A references scope B and scope B references scope A, the validator detects the cycle and rejects:

```
ERROR: Circular reference detected
  SERVICES/TALK/MAMMOCHAT → SERVICES/TALK/ONCOCHAT → SERVICES/TALK/MAMMOCHAT
  Resolution: Remove one direction. Clinical routing is unidirectional.
```

MammoChat routes staging queries to OncoChat. OncoChat routes breast-specific queries back to MammoChat. This is a valid clinical routing pattern — but it cannot be modeled as bidirectional governance references. Model it as unidirectional:

```
# MAMMOCHAT_INTEL.md
references:
  staging_escalation: SERVICES/TALK/ONCOCHAT  # outbound clinical route
```

```
# ONCOCHAT INTEL.md
# No back-reference to MAMMOCHAT. Handles breast domain internally.
```

The clinical routing layer (Chapter 25) handles runtime dispatch. The governance layer enforces the acyclic reference graph. Runtime and governance are separate concerns [2].

25.9. 24.9 Fleet-Wide Compilation

A fleet compiles when every scope reaches 255 and every cross-scope reference resolves:

```
magic validate --fleet
```

```
═══ MAGIC BUILD ═══
  00-toolchain    10 contracts compiled                    (0s)
  01-services     69 scopes, 68 compliant, PASS            (1s)
  02-surfaces     293 surfaces compiled (102 + 191)        (111s)
  03-figures      deck exports, mermaid → SVG              (0s)
  04-shop         product wiring                           (1s)
  05-validate     intel wiring, vocab normalization        (0s)
  06-structure    gov boundary, fleet structure             (1s)
  07-scopes       galaxy JSON, 69 scopes                   (5s)
  08-learning     ledger sync, star timeline               (3s)
  09-econ         stripe, wallets, KV sync                 (6s)
  10-federation   digest + witness + threshold (4 repos)   (2s)
  11-validate     20 gov domains, 2 HTTP roots @ 255       (1s)
═══ BUILD OK — 12 phases in 135s ═══
```

The build is not a single pass — it is a twelve-phase compilation pipeline. Phase 10 (federation) runs a cryptographic digest of each governed repo, witnesses the digests across the fleet, and applies threshold validation. Phase 11 validates that all 20 governance domains and both HTTP roots score 255. A fleet at 255 means every scope compiles, every cross-reference resolves, and no cycles exist. One broken scope or one dangling reference blocks the entire fleet [3].

25.10. 24.10 Cross-Axiomatic Validation in CI

Cross-axiomatic validation runs on every commit via a pre-commit hook. The hook executes the full MAGIC 255 validation — every governance dimension is scored, and the commit is rejected if the score drops below 255:

```
hadleylab-canonic 255/255 tier:5 VALID gradient:0 idf:0000000000000000
Commit completed. Remember: Producer = Canonify, Consumer = Apply, USER = Manual
```

The pre-commit gate is more aggressive than CI — it blocks the commit itself, not just the merge. No commit that degrades a governance score reaches the repository. The build pipeline then runs the full 12-phase compilation including federation witnesses before deploy. This two-stage gate (commit-time validation + build-time compilation) ensures no governed artifact reaches production without both local and fleet-wide integrity checks [2].

25.11. 24.11 Clinical Cross-Validation Pattern

Cross-axiomatic validation has direct clinical significance. Consider a clinical decision support workflow:

```
Patient presents with breast mass
  → MammoChat (BI-RADS classification) [TALK/MAMMOCHAT INTEL]
    → OncoChat (staging if malignant) [TALK/ONCOCHAT INTEL]
      → OmicsChat (molecular profiling) [TALK/OMICSCHAT INTEL]
        → MedChat (primary care follow-up) [TALK/MEDCHAT INTEL]
```

Each step is a cross-scope reference. The cross-axiomatic validator ensures:

1. Every scope in the pathway compiles (all at 255).
2. Every reference resolves (no dangling links).
3. The pathway is acyclic (no circular routing).
4. The evidence chains are intact (every claim traces to a citation).

If a developer removes TALK/OMICSCHAT, the validator flags every scope that references it. The clinical pathway breaks at build time, not at runtime [32].

25.12. 24.12 Validation Depth Control

The validator supports depth limits for large fleets:

```
magic validate --cross-refs --depth 1        # direct refs only
magic validate --cross-refs --depth 2        # refs of refs
magic validate --cross-refs --depth unlimited # full closure (default)
```

Depth	Scopes Checked	Time (69-scope fleet)	Use Case
1	Direct references only	< 2s	Every push
2	References + their references	< 5s	PR merge
Unlimited	Full transitive closure	< 30s	Nightly / release

25.13. 24.13 Error Catalog

Error Code	Description	Resolution
`XREF-001`	Dangling reference: target not found	Create the target scope or remove the reference
`XREF-002`	Degraded reference: target below 255	Heal the target scope or accept degradation
`XREF-003`	Circular reference detected	Remove one direction of the cycle
`XREF-004`	Axiom contradiction	Rewrite child axiom to specialize, not contradict
`XREF-005`	Evidence bridge unresolvable	Add the DOI or published date to the target
`XREF-006`	Fleet compilation failed	Run `magic validate --fleet` and fix all errors
`XREF-007`	Reference count exceeds threshold	Refactor: 50+ outbound refs is too coupled
`XREF-008`	Orphan scope: no inbound references	Wire into the governance tree or archive
`XREF-009`	Bridge type mismatch	Correct the bridge type in `references:`

Fix governance, not compiled output [2].

25.14. 24.14 Cross-Validation and the LEDGER

Every cross-axiomatic validation event is ledgered:

```
{
  "event": "CROSS_VALIDATE",
  "timestamp": "2026-03-10T14:22:00Z",
  "fleet": "hadleylab-canonic",
  "scopes_checked": 69,
```

```
  "references_resolved": 148,
  "errors": 0,
  "result": "COMPILED",
  "score": 255,
  "hash": "sha256:a1b2c3d4..."
}
```

The LEDGER entry is hash-chained to the previous entry. The fleet's compilation history is an append-only audit trail. For FDA 21 CFR Part 11 compliance, this trail demonstrates that every software change was validated before deployment [214]. ## 24.15 Reference Weight and Priority

Not all cross-references carry equal weight. The validator assigns weight based on bridge type and target tier:

Bridge Type	Base Weight	Tier Multiplier	Effective Weight
Governance (255)	10	1.0	10
Governance (127)	10	0.5	5
Paper (DOI)	8	1.0	8
Paper (preprint)	8	0.7	5.6
Blog (< 90 days)	5	1.0	5
Blog (> 90 days)	5	0.5	2.5
Service (255)	7	1.0	7
Service (< 255)	7	0.5	3.5

A scope backed entirely by governance sources at 255 carries stronger cross-axiomatic integrity than one leaning on stale blog posts.

The weight calculation:

```
def calculate_cross_integrity(scope):
    total_weight = 0
    max_weight = 0
    for ref in scope.references:
        target = resolve(ref)
        base = BRIDGE_WEIGHTS[ref.type]
        multiplier = tier_multiplier(target.score) if ref.type == 'governance' else freshness_multiplier(t
        weight = base * multiplier
        total_weight += weight
        max_weight += base  # max assumes all refs at full weight
    return total_weight / max_weight if max_weight > 0 else 0
```

The cross-integrity ratio is reported alongside the scope score:

```
SERVICES/TALK/MAMMOCHAT: 255/255, cross-integrity: 0.94
```

A score of 0.94 means the scope's cross-references are 94% at full weight. The remaining 6% is a blog reference that has gone stale. The governor schedules an INTEL refresh [2].

25.15. 24.16 Transitive Reference Healing

When a cross-reference fails, the healing algorithm identifies the shortest path to resolution:

```
magic heal --cross-refs DEALS/LLU
```

```
HEAL: DEALS/LLU
──────────────────────────────────────────────────
Error: XREF-002 — Degraded reference to SERVICES/CLINICAL (127)
Root cause: SERVICES/CLINICAL missing INTEL.md (expression question unanswered)
Shortest heal path:
  1. Create SERVICES/CLINICAL/INTEL.md (answers expression question)
  2. Run magic validate SERVICES/CLINICAL (expected: 255)
  3. Run magic validate --cross-refs DEALS/LLU (expected: resolved)
Estimated effort: 15 minutes (INTEL template available)
──────────────────────────────────────────────────
```

The healer always suggests strengthening the target, never removing the reference. Governance flows upward [32].

25.16. 24.17 Visualization of Cross-References

The GALAXY visualization (Chapter 31) renders cross-references as edges. The visual encoding:

Edge State	Visual	Meaning
Both 255 + INTEL	Green pulse, 2px, solid	Full cross-axiomatic integrity
Both 255, no INTEL	Blue static, 1px, solid	Governance valid, INTEL missing
One degraded	Yellow static, 1px, dashed	Degraded reference
Target missing	Red, 1px, dotted	Dangling reference
Circular	Red highlight, animated	Cycle detected

The GALAXY makes cross-axiomatic health visible at a glance. A fleet with all-green edges is fully cross-validated. A fleet with yellow or red edges has cross-axiomatic debt [18].

25.17. 24.18 Cross-Validation Reporting

The validator produces a comprehensive cross-validation report. Run the full report:

```
magic validate --cross-report > cross-report.md
```

The report includes six sections:

Section 1: Fleet Summary

```
Fleet: hadleylab-canonic
Date: 2026-03-10
Scopes: 69 (69 at 255)
Cross-references: 148 (148 resolved)
Fleet score: 255
```

Section 2: Reference Distribution

```
By type:
  Governance: 68 (47.9%)
  Service:    42 (29.6%)
  Paper:      18 (12.7%)
  Blog:       14 (9.9%)
```

Section 3: Hotspot Analysis

Some scopes have many inbound references — they are hotspots. If a hotspot degrades, many scopes are affected:

Scope	Inbound Refs	Outbound Refs	Criticality
SERVICES/TALK	22	5	HIGH
SERVICES/LEARNING	18	3	HIGH
SERVICES/LEDGER	14	2	HIGH
SERVICES/TALK/MAMMOCHAT	12	6	HIGH
SERVICES/TALK/RUNNER	8	4	MEDIUM
PAPERS/governance-as-compilation	7	0	MEDIUM

Hotspots require extra governance attention. A governance budget should allocate more review time to high-criticality scopes [2].

Section 4: Orphan Detection

Scopes with zero inbound references are orphans — they exist in the tree but nothing references them:

```
Orphan scopes: 0
(All scopes have at least 1 inbound reference)
```

Section 5: Freshness Matrix

Cross-references have freshness. A reference to a scope whose INTEL was last updated 90+ days ago is flagged:

```
Stale references: 2
  BOOKS/CANONIC-DOCTRINE → BLOGS/2025-12-15-old-post (97 days)
  SERVICES/DESIGN → BLOGS/2025-12-20-design-update (82 days)
```

Section 6: Recommended Actions

The report concludes with prioritized recommendations:

```
1. [HIGH] Refresh INTEL for BLOGS/2025-12-15-old-post (97 days stale)
2. [MEDIUM] Review SERVICES/TALK hotspot (22 inbound refs — high blast radius)
3. [LOW] Consider archiving BLOGS/2025-12-20-design-update
```

The cross-validation report is the governor's dashboard. Run it weekly. File it in the LEDGER. Use it to prioritize governance work [214].

25.18. 24.19 Cross-Validation Anti-Patterns

Common mistakes in cross-scope reference management:

Anti-Pattern	Description	Fix
Reference hoarding	Scope references 50+ other scopes	Refactor: split into sub-scopes
Shallow references	All references are to blogs, none to governance	Add governance source references
Stale reference chains	A → B → C where C is stale	Refresh C or find alternative
Mirror references	A references B and B references A	Remove one direction
Reference to self	Scope references itself	Remove the self-reference

Anti-Pattern	Description	Fix
Dead-end scopes	Scope has outbound refs but zero inbound	Wire it into the tree
Hub dependency	Every scope references one central scope	Distribute references across services

The validator flags all anti-patterns:

```
magic validate --cross-refs --anti-patterns
```

```
ANTI-PATTERN CHECK: hadleylab-canonic
──────────────────────────────────────────────────
Reference hoarding:   0 scopes
Shallow references:   0 scopes
Stale chains:         0 chains
Mirror references:    0 pairs
Self references:      0 scopes
Dead-end scopes:      0 scopes
Hub dependency:       0 hubs
──────────────────────────────────────────────────
Result: CLEAN
```

A clean anti-pattern check means the cross-reference graph is well-structured. The governance tree has healthy reference diversity, appropriate depth, and no structural weaknesses [2].

25.19. 24.20 Cross-Validation Performance Optimization

For large fleets (500+ scopes), cross-validation can be optimized using incremental checking. The validator caches the previous cross-reference graph and only re-validates edges where either endpoint changed since the last run:

```
# Full cross-validation (cold)
magic validate --cross-refs
# Time: 28s (69 scopes, 148 references)

# Incremental cross-validation (warm)
magic validate --cross-refs --incremental
# Time: 0.3s (2 scopes changed, 8 references re-checked)
```

The cache is stored in `.canonic/cache/cross-refs.json`:

```
{
  "last_run": "2026-03-10T14:22:00Z",
  "scope_hashes": {
    "SERVICES/TALK/MAMMOCHAT": "sha256:abc123",
    "SERVICES/TALK/ONCOCHAT": "sha256:def456"
  },
  "validated_edges": 142,
  "result": "PASS"
}
```

If a scope's source hash has not changed, its outbound references are assumed valid. Only scopes with changed source hashes have their references re-validated. The cache is invalidated on `bin/build --clean` or when the compiler version changes [2].

25.20. 24.21 Cross-Validation Metrics

Cross-validation produces metrics consumed by the MONITORING service:

```
canonic_cross_refs_total{fleet="hadleylab-canonic"}           142
canonic_cross_refs_resolved{fleet="hadleylab-canonic"}        142
canonic_cross_refs_degraded{fleet="hadleylab-canonic"}        0
canonic_cross_refs_circular{fleet="hadleylab-canonic"}        0
canonic_cross_integrity_ratio{fleet="hadleylab-canonic"}      1.0
canonic_cross_validation_duration_seconds{fleet="hadleylab"}  0.3
```

These metrics feed into the GALAXY visualization and the fleet dashboard. A cross-integrity ratio below 0.95 triggers a governance alert [218].

25.21. 24.22 Cross-Validation in Multi-Organization Fleets

When multiple organizations use CANONIC (e.g., `hadleylab-canonic` and a partner hospital's `hospital-canonic`), cross-validation extends across organization boundaries. The inter-org validation uses the same bridge types but adds authentication:

```
# Validate cross-org references (authenticated)
magic validate --cross-refs --org hospital-canonic --token $ORG_TOKEN
```

Inter-org references are read-only. Organization A can reference Organization B's public scopes but cannot modify them. The reference is a declaration that Organization A's INTEL depends on Organization B's governance. If Organization B degrades, Organization A is notified.

Reference Direction	Permission	Validation
Intra-org (same fleet)	Read + Write	Full validation
Inter-org (partner fleet)	Read only	Score + tier check
External (non-CANONIC)	None	URL reachability only

Inter-org validation enables healthcare networks. A hospital that deploys MammoChat can reference Hadley Lab's governance sources. When Hadley Lab updates the NCCN guidelines in INTEL.md, the hospital's cross-validation detects the change and notifies the local governor. The evidence chain spans organizations [214].

25.22. 24.23 Cross-Validation Summary

Think of cross-axiomatic validation as the governance tree's immune system. It detects:

- Dangling references (deleted scopes still referenced)
- Degraded dependencies (scopes that have drifted below 255)
- Circular dependencies (bidirectional references that create cycles)
- Axiom contradictions (child axioms that conflict with parent axioms)
- Stale evidence bridges (references to outdated content)
- Anti-patterns (structural weaknesses in the reference graph)

It enforces:

- Acyclic reference graphs
- Consistent axiom chains
- Resolved evidence bridges
- Fleet-wide compilation integrity
- Audit trails for all validation events

It reports:

- Cross-reference matrices
- Hotspot analysis
- Freshness metrics
- Anti-pattern detection
- Prioritized remediation actions

A fleet that passes cross-axiomatic validation is structurally sound — every scope compiles, every reference resolves, every axiom is consistent, every evidence bridge holds. The governance tree has the same kind of integrity a well-typed program has. The compiler proved it [32].

Chapter 26

Chapter 25: Contextual Agents

Every clinical AI agent in CANONIC emerges from the same deterministic pipeline: governed knowledge in, contextual agent out. You do not hand-craft agents. You write INTEL.md, declare constraints in CANON.md, and the compiler produces an agent whose knowledge boundary, clinical voice, and evidence chain are structurally guaranteed. Change the INTEL, recompile, and the agent changes. The pipeline is the architecture.

26.1. 25.1 The Agent Pipeline

The pipeline is deterministic — the same INTEL always produces the same agent:

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

Each scope with INTEL and TALK produces a contextual agent. The pipeline is always the same — only the INTEL varies. Breast imaging INTEL produces MammoChat. Oncology INTEL produces OncoChat. The governance is identical across all of them [2126].

26.2. 25.2 The FHIR □ INTEL □ TALK □ COIN Pipeline

When integrating with EHR systems, the agent pipeline extends to include FHIR resource composition — bridging clinical data 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) arrives as HL7 FHIR resources — Patient, Observation, DiagnosticReport, MedicationRequest. These resources compose into governed INTEL units. A Patient's screening history becomes INTEL. A DiagnosticReport's findings become INTEL. FHIR data enters governance through the INTEL layer.

Step 2: INTEL □ systemPrompt. The build pipeline produces the agent's systemPrompt from governed INTEL: what evidence it can cite, what guidelines it references, what contraindications it knows, what disclaimer it must display.

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

Step 4: TALK □ COIN. Every clinical conversation mints COIN. The governance labor becomes economically visible.

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

Every clinical AI deployment in CANONIC follows this pipeline. Only Step 1 (which FHIR resources, which clinical domain) and Step 2 (which evidence base, which guidelines) vary. Everything else is shared infrastructure.

26.3. 25.3 systemPrompt Compilation

The `systemPrompt` is compiled from INTEL.md — never hand-written. The compiler reads the scope's INTEL, CANON, and LEARNING 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)

Do not hand-edit it. If the agent says something wrong, fix the INTEL or the CANON and recompile. The systemPrompt is output, not source [26].

26.4. 25.4 Persona Resolution

Persona resolution determines how the agent speaks — ensuring a breast imaging agent sounds like a radiologist, not 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 scopes, persona is industry-specific — determined by position in the governance tree and constraints inherited from the parent. A clinical agent inherits clinical persona from the healthcare tree; a legal agent inherits legal persona from the legal tree [21].

26.5. 25.5 Frontmatter Wiring

Each chapter or scope enables TALK via frontmatter:

```
```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

talk: inline

• • •

```
The agent answers questions about that chapter's content, governed by that chapter's axiom, in the voice o
```
```

The same wiring applies to clinical evidence documents — treatment protocols, clinical practice guidelines

25.6 The INTEL Pipeline in Detail

The INTEL pipeline is the critical path from raw knowledge to compiled agent context. Every step is determ

Step 1: INTEL.md Parsing. The compiler reads the scope's INTEL.md and extracts structured fields:

```yaml
# Parsed from INTEL.md frontmatter and body
intel:
  subject: "Breast imaging intelligence"
  audience: "Radiologists, breast surgeons, oncologists"
  sources:
    governance: 22
    papers: 9
    blogs: 14
  evidence_chain:
    - layer: 1
      source: "Governance sources (canonic-canonic)"
      count: 22
      status: INDEXED
    - layer: 2
      source: "Papers (PAPERS)"
      count: 9
      status: INDEXED
    - layer: 3
      source: "Blogs (BLOGS)"
      count: 14
      status: INDEXED
  cross_scope:
    TALK: "Conversation engine — contextual agent per chapter"
    COIN: "Economic shadow — every chapter is WORK"
    LEDGER: "Append-only truth — governance events recorded"
```

Step 2: CANON.md Parsing. The compiler reads the scope's CANON.md and extracts the axiom, constraints, persona, and dimensions:

```yaml
# Parsed from CANON.md
canon:
  axiom: "Breast imaging intelligence, governed"
  constraints:
    must:
      - "Cite BI-RADS for all classification claims"
      - "Reference NCCN for all treatment pathway claims"
```

```
      - "Display disclaimer before clinical content"
    must_not:
      - "Provide patient-specific diagnostic conclusions"
      - "Replace radiologist judgment"
  persona:
    tone: clinical
    audience: clinicians
    voice: clinical-third
    warmth: clinical-neutral
```

Step 3: LEARNING.md Parsing. The compiler reads the scope's LEARNING.md and extracts accumulated patterns:

```
# Parsed from LEARNING.md
learning:
  patterns:
    - "Users ask about BI-RADS 4 vs 4A/4B/4C subcategorization"
    - "Users confuse screening vs diagnostic mammography indications"
    - "Users need NCCN pathway references for DCIS management"
  stale:
    - "Referenced 2023 NCCN guidelines (now 2026)"
  fresh:
    - "Updated to NCCN 2026.1 breast cancer screening guidelines"
```

Step 4: systemPrompt Assembly. The compiler assembles the systemPrompt from the parsed artifacts:

```
// Compiled systemPrompt assembly (simplified)
const systemPrompt = {
  identity: canon.axiom,
  knowledge_boundary: intel.sources,
  constraints: canon.constraints,
  voice: canon.persona,
  disclaimer: "This tool provides educational information only...",
  evidence_chain: intel.evidence_chain,
  learning_patterns: learning.patterns,
  cross_scope_connections: intel.cross_scope
};
```

Step 5: CANON.json Emission. The compiled systemPrompt is written to CANON.json:

```
{
  "scope": "SERVICES/TALK/MAMMOCHAT",
  "score": 255,
  "tier": "MAGIC",
```

```
  "systemPrompt": "You are MammoChat, a governed breast imaging...",
  "breadcrumbs": ["hadleylab-canonic", "SERVICES", "TALK", "MAMMOCHAT"],
  "brand": { "accent": "#ec4899", "name": "MammoChat" },
  "welcome": "Welcome to MammoChat. I provide governed breast imaging intelligence...",
  "disclaimer": "This tool provides educational information only..."
}
```

The systemPrompt in CANON.json is compiled output. Do not hand-edit it [26].

26.6. 25.7 Agent Context Window Management

Every contextual agent operates within a context window. The INTEL pipeline manages the context window budget:

Context Segment	Token Budget	Source	Priority
systemPrompt	2,000-4,000	CANON.json	Required
Evidence chain	1,000-2,000	INTEL.md layers	Required
LEARNING patterns	500-1,000	LEARNING.md	Required
Conversation history	4,000-8,000	Runtime	Dynamic
Cross-scope context	500-1,000	Referenced INTEL	On-demand

The compiler enforces token budgets at build time. If the INTEL exceeds the evidence chain budget, the compiler truncates by priority — governance sources first, then papers, then blogs. The truncation is logged:

```
WARNING: INTEL token budget exceeded for SERVICES/TALK/MAMMOCHAT
  Budget: 2000 tokens
  Actual: 2847 tokens
  Truncated: Layer 3 (blogs) reduced from 14 to 8 entries
```

26.7. 25.8 Multi-Agent Routing

When a query spans multiple clinical domains, the TALK service routes it to the appropriate contextual agent. The routing decision is governed — not probabilistic:

```
// talk.js — multi-agent routing (simplified)
function routeQuery(query, currentScope) {
  const routing = {
    "BI-RADS": "SERVICES/TALK/MAMMOCHAT",
```

```
    "staging": "SERVICES/TALK/ONCOCHAT",
    "molecular profiling": "SERVICES/TALK/OMICSCHAT",
    "primary care": "SERVICES/TALK/MEDCHAT",
    "HIPAA": "SERVICES/TALK/LAWCHAT",
    "CPT codes": "SERVICES/TALK/FINCHAT"
  };

  for (const [keyword, scope] of Object.entries(routing)) {
    if (query.toLowerCase().includes(keyword.toLowerCase())) {
      return { scope, reason: `Query contains "${keyword}"` };
    }
  }

  return { scope: currentScope, reason: "No routing match" };
}
```

The routing table is compiled from cross-scope connections declared in each INTEL.md, then aggregated into a fleet-wide table [21].

In practice: a radiologist asks about a BI-RADS 5 lesion and gets ACR guidance. When the conversation shifts to NCCN staging, the agent routes to OncoChat, which responds with its own governed INTEL. The routing is transparent — the user sees which agent is responding and why.

26.8. 25.9 Agent Testing and Validation

Every contextual agent must pass validation before deployment:

```
# Validate agent compilation
magic validate SERVICES/TALK/MAMMOCHAT

# Test agent responses against golden dataset
magic test --agent SERVICES/TALK/MAMMOCHAT --golden tests/mammochat_golden.json

# Verify disclaimer is always present
magic test --agent SERVICES/TALK/MAMMOCHAT --assert disclaimer

# Verify evidence citations in responses
magic test --agent SERVICES/TALK/MAMMOCHAT --assert citations
```

The golden dataset contains expected question-answer pairs:

```
[
  {
    "query": "What is BI-RADS 4?",
    "expected_citations": ["ACR BI-RADS 5th Edition"],
    "expected_constraints": ["disclaimer present", "no diagnostic conclusion"],
    "forbidden_content": ["you should get a biopsy", "this is cancer"]
  },
  {
    "query": "What are the NCCN guidelines for DCIS?",
    "expected_citations": ["NCCN 2026.1"],
    "expected_constraints": ["disclaimer present"],
    "expected_routing": null
  },
  {
    "query": "What is the TNM staging for this tumor?",
    "expected_routing": "SERVICES/TALK/ONCOCHAT",
    "expected_constraints": ["routing explanation present"]
  }
]
```

26.9. 25.10 Agent Lifecycle

Contextual agents have a lifecycle governed by the pipeline that creates them:

```
CREATE  → build (INTEL + CANON + LEARNING → CANON.json)
DEPLOY  → build (CANON.json → talk.js endpoint)
MONITOR → MONITORING service (response times, error rates, routing accuracy)
UPDATE  → INTEL.md change → recompile → redeploy
ARCHIVE → scope archived → agent deactivated → LEDGER event
```

Every lifecycle transition is a LEDGER event:

Event	LEDGER Entry	COIN Impact
CREATE	`AGENT:CREATE scope=MAMMOCHAT`	MINT:WORK (new scope)
DEPLOY	`AGENT:DEPLOY scope=MAMMOCHAT`	None
UPDATE	`AGENT:UPDATE scope=MAMMOCHAT field=INTEL`	MINT:WORK (governance labor)
ARCHIVE	`AGENT:ARCHIVE scope=MAMMOCHAT`	None

26.10. 25.11 Agent Constraints Enforcement

The systemPrompt includes constraints compiled from CANON.md. The agent enforces these constraints at runtime:

```
MUST constraints (compiled from CANON.md):
  ✓ Cite BI-RADS for all classification claims
  ✓ Reference NCCN for all treatment pathway claims
  ✓ Display disclaimer before clinical content
  ✓ Identify as AI assistant, not clinician

MUST NOT constraints (compiled from CANON.md):
  ✗ Provide patient-specific diagnostic conclusions
  ✗ Replace radiologist judgment
  ✗ Store or reference patient-identifiable information
  ✗ Recommend specific treatments without NCCN citation
```

Constraint enforcement is structural — baked into the systemPrompt at compile time. Defense-in-depth: the systemPrompt constrains, MONITORING watches, and the LEDGER records every interaction for audit [2126].

26.11. 25.12 Cross-Vertical Agent Patterns

The agent pipeline is not healthcare-specific. The same pipeline produces agents for every vertical:

Vertical	Agent	INTEL Source	Persona	Key Constraints	Live
Healthcare	MammoChat	BI-RADS, NCCN, mCODE	Clinical-third	No diagnosis, always disclaim	mammo.chat
Healthcare	OncoChat	NCCN staging, AJCC TNM	Clinical-third	No prognosis, always cite	oncochat.hadleylab.org
Healthcare	OmicsChat	ClinVar, ACMG, PharmGKB	Clinical-third	No variant interpretation	—
Legal	LawChat	Statutes, case law	Legal-third	No legal advice, always disclaim	lawchat.hadleylab.org
Finance	FinChat	CPT/RVU, Medicare, SEC	Business-third	No investment advice	finchat.hadleylab.org
Real Estate	Realty	MLS, FL 475.278	Professional	No property valuation	—
Real Estate	Runner	FL 475.278, fiduciary	Professional	No legal advice	gorunner.pro

The pipeline is identical across verticals — only the INTEL, persona, and constraints change [21]. ## 25.13 Agent Observability

Every contextual agent exposes observability metrics via the MONITORING service (Chapter 22):

```
canonic_agent_requests_total{scope="MAMMOCHAT", method="query"}           42
canonic_agent_response_time_seconds{scope="MAMMOCHAT", p95="0.8"}         0.8
canonic_agent_citations_per_response{scope="MAMMOCHAT"}                   2.3
canonic_agent_routing_events_total{scope="MAMMOCHAT", target="ONCOCHAT"}  7
canonic_agent_disclaimer_shown_total{scope="MAMMOCHAT"}                   42
canonic_agent_constraint_violations_total{scope="MAMMOCHAT"}              0
```

All metrics live in-memory — no external database. The MONITORING service (Chapter 22) scrapes them via `/metrics`. Alert rules fire when agents degrade:

```yaml
- alert: AgentConstraintViolation
  expr: canonic_agent_constraint_violations_total > 0
  for: 0m
  labels:
    severity: critical
  annotations:
    summary: "Agent {{ $labels.scope }} violated constraints"
```

A constraint violation is a P0 incident — the agent said something outside its governed bounds. The LEDGER records the violation. The governor investigates [21].

26.12. 25.14 Agent Versioning

The systemPrompt is versioned via the source_hash in CANON.json. Every change to CANON.md, INTEL.md, or LEARNING.md produces a new source_hash, which means a new systemPrompt version:

```
Version history (from git log):
  v3: sha256:abc123 (2026-03-10) — Added NCT06604078 to INTEL
  v2: sha256:def456 (2026-02-28) — Updated NCCN 2026.1 guidelines
  v1: sha256:789abc (2026-02-15) — Initial MammoChat INTEL
```

The LEDGER records every version transition:

```
{
  "event": "AGENT:VERSION",
  "scope": "SERVICES/TALK/MAMMOCHAT",
  "from_hash": "sha256:def456",
```

```
  "to_hash": "sha256:abc123",
  "changed_files": ["INTEL.md"],
  "timestamp": "2026-03-10T14:22:00Z"
}
```

Rollback is possible by reverting the GOV source and recompiling:

```
git revert HEAD  # revert INTEL.md change
bin/build        # recompile
magic validate SERVICES/TALK/MAMMOCHAT  # verify
```

The agent reverts to the previous version. The LEDGER records the rollback. No manual editing of CANON.json required [26].

26.13. 25.15 The Complete Agent Architecture

The complete contextual agent architecture, from governance source to patient-facing response:

```
GOV (human-authored):
  CANON.md → axiom, constraints, persona
  INTEL.md → evidence chain, cross-scope connections
  LEARNING.md → accumulated patterns, stale/fresh tracking

COMPILER (build-surfaces):
  Parse → Resolve → Assemble → Emit
  Output: CANON.json { systemPrompt, brand, disclaimer, ... }

RUNTIME (talk.js):
  Load CANON.json → Initialize agent context
  Receive query → Route (if cross-domain) → Generate response
  Enforce constraints → Add citations → Append disclaimer
  Render via _CHAT.scss → LEDGER event → COIN mint

OBSERVABILITY:
  MONITORING → response times, citation rates, constraint compliance
  LEDGER → every conversation, every routing decision, every version
  GALAXY → visual topology, compliance rings, INTEL flow
```

Governance at the top, rendering at the bottom, observability on the side. The agent is not a chatbot — it is a governed knowledge interface. Governance is structural, not advisory. Evidence is traced, not claimed. Economics are visible, not hidden [2126].

26.14. 25.16 Agent Error Handling

When an agent encounters an error, the error handling follows a governed protocol:

```
// talk.js — error handling
async function handleQuery(query) {
  try {
    const response = await callAgent(query, chatState.systemPrompt);

    // Validate response against constraints
    const violations = validateConstraints(response, chatState.constraints);
    if (violations.length > 0) {
      // Log constraint violation to LEDGER
      ledger.append({
        event: 'AGENT:CONSTRAINT_VIOLATION',
        scope: chatState.scope,
        violations: violations,
        query_hash: sha256(query)  // hash, not content
      });

      // Return safe fallback
      return {
        content: "I cannot answer that question within my governed constraints. " +
                 "Please consult a qualified clinician for specific medical advice.",
        citations: [],
        routing: null
      };
    }

    return response;
  } catch (error) {
    // LLM error — return governed error message
    ledger.append({
      event: 'AGENT:ERROR',
      scope: chatState.scope,
      error_type: error.constructor.name,
      timestamp: Date.now()
    });

    return {
      content: "I am temporarily unable to respond. " +
               "This is a governed clinical AI service. " +
               "Please try again or consult a clinician.",
      citations: [],
```

```
      routing: null
    };
  }
}
```

Every error is ledgered, every constraint violation recorded. No error silently passes — even in failure, the agent responds with a governed message [2126].

26.15. 25.17 Agent Deployment Checklist

Before deploying a new contextual agent, verify:

Step	Command	Expected
1. CANON.md exists	`ls SERVICES/TALK/NEWCHAT/CANON.md`	File exists
2. INTEL.md exists	`ls SERVICES/TALK/NEWCHAT/INTEL.md`	File exists
3. LEARNING.md exists	`ls SERVICES/TALK/NEWCHAT/LEARNING.md`	File exists
4. Score is 255	`magic validate SERVICES/TALK/NEWCHAT`	Score: 255
5. Cross-refs resolve	`magic validate --cross-refs`	All resolved
6. systemPrompt compiles	`bin/build`	CANON.json emitted
7. Golden tests pass	`magic test --agent NEWCHAT --golden`	All pass
8. Disclaimer present	`magic test --agent NEWCHAT --assert disclaimer`	Pass
9. WCAG compliance	`magic validate --wcag`	Pass
10. Performance budget	`magic validate --performance`	Within budget

Do not deploy an agent that fails any step. The checklist is the gate. The compiler enforces it [21].

26.16. 25.18 Agent Discovery

The TALK service discovers available agents from the compiled scopes:

```
// talk.js — agent discovery
async function discoverAgents() {
  const scopes = await fetch('/MAGIC/galaxy.json').then(r => r.json());
  return scopes.nodes
    .filter(s => s.talk_sessions > 0 && s.bits === 255)
    .map(s => ({
      id: s.id,
      label: s.label,
      brand: s.brand,
      accent: s.accent,
      endpoint: `/api/talk/${s.label.toLowerCase()}`
    }));
}
```

Discovery is automatic. When a new scope adds `talk: inline` and compiles at 255, it appears in the agent roster — no manual registration, no configuration file to update. The GOV tree is the agent registry [2126].

26.17. 25.19 Agent and the LEARNING Loop

Every contextual agent participates in the LEARNING loop (Chapter 10). Conversation patterns are captured in LEARNING.md:

```
## Patterns (from MammoChat conversations)

| Pattern | Frequency | Action |
|---------|-----------|--------|
| Users ask BI-RADS 4 subcategorization | HIGH | Added detailed 4A/4B/4C explanation to INTEL |
| Users confuse screening vs diagnostic | MEDIUM | Added clarification to systemPrompt |
| Users request NCCN pathway PDFs | LOW | Added note that PDFs are copyrighted |
```

The loop closes: conversations reveal gaps in INTEL, gaps are fixed in INTEL.md, INTEL.md recompiles into the systemPrompt, and the agent improves. Every LEARNING pattern is reviewed by a governor before it enters INTEL — improvement is governed, not automatic. For the full LEARNING architecture, see Chapter 10. For the theoretical closure this loop represents, see Chapter 39 (The LEARNING Closure). For live TALK instances, visit hadleylab.org/talks/mammochat/ [2126].

Chapter 27

Chapter 26: The _generated Contract

The hardest habit to break when you first work with CANONIC is the urge to fix the output. A system-Prompt has the wrong knowledge boundary — so you open CANON.json and edit it. Twenty minutes later, `bin/build` runs and your edit vanishes. The `_generated` contract exists to prevent this wasted work: if the compiler wrote a file, only the compiler should change it. Your job is to fix the source. This chapter is the practical companion to the compiler theory in Chapter 37 and the build pipeline in Chapter 44.

27.1. 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 [2].

27.2. 26.2 Compiled Outputs

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

27.3. 26.3 Provenance

Every `_generated` file traces to its source governance file. The compilation pipeline is auditable: source compiler output [14].

27.4. 26.4 The _generated Directory Structure

The `_generated` directory mirrors the GOV tree. Every scope in GOV that compiles produces a corresponding entry in `_generated`:

```
~/.canonic/                          # RUNTIME root
├── _data/
│   ├── galaxy.json                  # Galaxy graph, compiled from GOV tree
│   ├── fleet.json                   # Fleet-wide metadata
│   └── navigation.json              # Navigation structure
├── SERVICES/
│   ├── TALK/
│   │   ├── CANON.json               # TALK service compiled config
│   │   ├── MAMMOCHAT/
│   │   │   └── CANON.json           # MammoChat compiled config
│   │   ├── ONCOCHAT/
│   │   │   └── CANON.json           # OncoChat compiled config
│   │   └── MEDCHAT/
│   │       └── CANON.json           # MedChat compiled config
│   ├── LEARNING/
│   │   └── CANON.json
│   ├── DESIGN/
│   │   └── CANON.json
│   └── LEDGER/
│       └── CANON.json
├── BOOKS/
│   ├── CANONIC-DOCTRINE/
│   │   └── CANON.json
│   └── CANONIC-CANON/
│       └── CANON.json
└── DEXTER/
    └── CANON.json
```

Every file in this tree is `_generated`. The mapping is deterministic:

```
GOV:     ~/CANONIC/hadleylab-canonic/SERVICES/TALK/MAMMOCHAT/CANON.md
RUNTIME: ~/.canonic/SERVICES/TALK/MAMMOCHAT/CANON.json
```

27.5. 26.5 CANON.json Schema

Every compiled CANON.json follows the same schema:


```
{
  "scope": "SERVICES/TALK/MAMMOCHAT",
  "inherits": "SERVICES/TALK",
  "axiom": "Breast imaging intelligence, governed",
  "score": 255,
  "tier": "MAGIC",
  "dimensions": {
    "D": true, "E": true, "S": true, "O": true,
    "T": true, "LANG": true, "ECON": true, "L": true
  },
  "constraints": {
    "must": [
      "Cite BI-RADS for all classification claims",
      "Reference NCCN for all treatment pathway claims"
    ],
    "must_not": [
      "Provide patient-specific diagnostic conclusions",
      "Replace radiologist judgment"
    ]
  },
  "persona": {
    "tone": "clinical",
    "audience": "clinicians",
    "voice": "clinical-third",
    "warmth": "clinical-neutral"
  },
  "systemPrompt": "You are MammoChat, a governed breast imaging...",
  "breadcrumbs": ["hadleylab-canonic", "SERVICES", "TALK", "MAMMOCHAT"],
  "brand": {
    "accent": "#ec4899",
    "name": "MammoChat",
    "logo": null
  },
  "welcome": "Welcome to MammoChat...",
  "disclaimer": "This tool provides educational information only...",
  "talk": "inline",
  "shop": null,
  "compiled_at": "2026-03-10T14:22:00Z",
  "compiler_version": "1.0.0",
  "source_hash": "sha256:abc123..."
```

```
}
```

The `source_hash` field is the SHA-256 hash of the source CANON.md at compilation time. If the CANON.md changes, the hash changes, and the CANON.json must be recompiled [2].

27.6. 26.6 The Build Pipeline

The build pipeline transforms GOV sources into RUNTIME artifacts deterministically — the same GOV tree always produces the same RUNTIME tree:

```
# Full build pipeline
bin/build
```

The pipeline steps:

```
1. DISCOVER    — Walk GOV tree, find all CANON.md files
2. PARSE       — Parse each CANON.md (YAML frontmatter + Markdown body)
3. RESOLVE     — Resolve inheritance chains (inherits: fields)
4. COMPILE     — Compile each scope (CANON.md + INTEL.md + LEARNING.md → CANON.json)
5. AGGREGATE   — Build fleet-wide artifacts (galaxy.json, fleet.json, navigation.json)
6. VALIDATE    — Run magic validate on all compiled artifacts
7. EMIT        — Write CANON.json files to ~/.canonic/
8. JEKYLL      — Generate static site from compiled artifacts (remote_theme)
9. VERIFY      — Verify all _generated files match their source hashes
```

Each step is logged:

```
[DISCOVER] Found 73 scopes in GOV tree
[PARSE]    Parsed 73 CANON.md files (0 errors)
[RESOLVE]  Resolved 73 inheritance chains (max depth: 5)
[COMPILE]  Compiled 73 scopes (73 at 255)
[AGGREGATE] Built galaxy.json (284 nodes), fleet.json, navigation.json
[VALIDATE] All 73 scopes pass validation
[EMIT]     Wrote 73 CANON.json files to ~/.canonic/
[JEKYLL]   Generated 142 pages via remote_theme
[VERIFY]   All _generated files match source hashes
```

27.7. 26.7 Never Hand-Edit Rules

The `_generated` contract has strict rules. Violating these rules breaks the compilation pipeline:

Rule	Rationale	Error if Violated
Never edit CANON.json	It is compiler output	`GEN-001: Hand-edited _generated file detected`
Never edit galaxy.json	It is aggregated from GOV tree	`GEN-002: Aggregated file modified`
Never edit _data/*.json	All _data files are compiled	`GEN-003: Data file modified`
Never edit index.md if `_generated`	Jekyll pages from CANON.json	`GEN-004: Generated page modified`
Never add files to ~/.canonic/	RUNTIME is compiler output	`GEN-005: Manual file in RUNTIME`

The validator detects hand-edits by comparing the `source_hash` in CANON.json against the current hash of the source CANON.md:

```
magic validate --generated
```

```
ERROR GEN-001: Hand-edited _generated file detected
  File: ~/.canonic/SERVICES/TALK/MAMMOCHAT/CANON.json
  Expected hash: sha256:abc123...
  Actual hash:   sha256:def456...
  Resolution: Run bin/build to regenerate from source
```

27.8. 26.8 Fixing the Contract, Not the Output

When compiled output is wrong, resist the instinct to fix it directly. The output is a symptom; the cause is in the source.

Wrong approach:

```
# DO NOT DO THIS
vim ~/.canonic/SERVICES/TALK/MAMMOCHAT/CANON.json
# Edit the systemPrompt directly
```

Correct approach:

```
# Fix the source
vim ~/CANONIC/hadleylab-canonic/SERVICES/TALK/MAMMOCHAT/CANON.md
# Or fix the INTEL
vim ~/CANONIC/hadleylab-canonic/SERVICES/TALK/MAMMOCHAT/INTEL.md
```

```
# Then rebuild
bin/build
# Then validate
magic validate SERVICES/TALK/MAMMOCHAT
```

Think of CANON.md as source code and CANON.json as the compiled binary. You would not debug a compiled binary by editing assembly — you fix the source and recompile.

27.9. 26.9 Compiler Error Handling

When the compiler encounters an error in a source file, it reports the error with source location and does not emit a CANON.json:

```
COMPILE ERROR: SERVICES/TALK/MAMMOCHAT/CANON.md
  Line 12: Missing required field: axiom
  Line 25: Invalid constraint: "SHOULD" (must be MUST or MUST NOT)
  Line 38: Persona field "tone" has invalid value: "casual"
           (expected: clinical, formal, narrative, precise)

  Resolution: Fix CANON.md at the indicated lines and recompile.
  No CANON.json emitted for this scope.
```

A scope that fails compilation produces no CANON.json, gets excluded from galaxy.json, and degrades the fleet score. The compiler protects the fleet from invalid governance [2].

27.10. 26.10 Incremental Compilation

The build pipeline supports incremental compilation. Only scopes whose source files have changed since the last build are recompiled:

```
# Full build (all scopes)
bin/build

# Incremental build (only changed scopes)
bin/build --incremental
```

Incremental compilation checks the modification time of each CANON.md, INTEL.md, and LEARNING.md against the modification time of the corresponding CANON.json:

```
[INCREMENTAL] Checking 73 scopes...
[SKIP]    SERVICES/TALK (unchanged)
[SKIP]    SERVICES/LEARNING (unchanged)
[COMPILE] SERVICES/TALK/MAMMOCHAT (INTEL.md modified 2026-03-10T14:00:00Z)
[SKIP]    SERVICES/TALK/ONCOCHAT (unchanged)
...
[RESULT]  1/73 scopes recompiled. 72 skipped.
```

27.11. 26.11 The Provenance Chain

Every `_generated` file carries a provenance chain — the complete record of how it was produced:

```
{
  "provenance": {
    "source": "SERVICES/TALK/MAMMOCHAT/CANON.md",
    "source_hash": "sha256:abc123...",
    "intel_source": "SERVICES/TALK/MAMMOCHAT/INTEL.md",
    "intel_hash": "sha256:def456...",
    "learning_source": "SERVICES/TALK/MAMMOCHAT/LEARNING.md",
    "learning_hash": "sha256:789abc...",
    "compiler": "magic compile v1.0.0",
    "compiled_at": "2026-03-10T14:22:00Z",
    "compiled_by": "bin/build",
    "parent_scope": "SERVICES/TALK",
    "parent_hash": "sha256:parent..."
  }
}
```

Given a CANON.json, trace every input that produced it. Given a systemPrompt, trace every evidence source that contributed. The chain is the audit trail [14].

27.12. 26.12 Clinical Significance

FDA 21 CFR Part 11 requires that electronic records be produced by validated systems with complete audit trails. The `_generated` contract satisfies each requirement:

1. **Validated system**: The compiler transforms governed source into compiled output deterministically.
2. **Audit trail**: The provenance chain records every input, compiler version, and timestamp.
3. **Access controls**: Only the compiler writes to `_generated`. Human governors write to GOV.
4. **Data integrity**: Source hashes verify that compiled output matches the source.

A clinical AI agent deployed from `_generated` CANON.json carries a complete audit trail from patient-facing output back to governance source [214]. ## gc_private(): The PRIVATE Gate

The build pipeline enforces a hard separation between public and private content. `gc_private()` strips all scopes marked `privacy: PRIVATE` from the compiled output before publication.

113 scopes are currently PRIVATE — including all principal scopes (DEXTER, FATIMA, YANA, ROBERT, etc.), authentication services, patent filings, and deal structures. The PRIVATE gate is a CI check: if any PRIVATE content appears in the public output, the build fails.

The `privacy:` field lives in CANON.md frontmatter:

```
privacy: PRIVATE
```

This is governance, not gitignore. The file exists in the repository and validates to 255, but the compiled public output excludes it. The governance tree knows about PRIVATE scopes; the public website does not.

27.13. 26.13 The _generated Lifecycle

Every `_generated` file has a lifecycle tied to the GOV source:

```
SOURCE CREATED    → _generated file does not exist yet
SOURCE COMPILED   → _generated file created (CANON.json emitted)
SOURCE MODIFIED   → _generated file stale (source_hash mismatch)
SOURCE RECOMPILED → _generated file refreshed (new source_hash)
SOURCE DELETED    → _generated file orphaned (must be cleaned)
```

The build pipeline manages orphan detection:

```
bin/build --clean-orphans
```

```
ORPHAN DETECTION: ~/.canonic/
──────────────────────────────────────────────────
CANON.json files:       73
GOV source matches:     73
Orphaned files:         0
──────────────────────────────────────────────────
Result: CLEAN
```

If a scope is deleted from GOV but its CANON.json remains in RUNTIME, the build pipeline detects the orphan and removes it. No manual cleanup [2].

27.14. 26.14 Build Reproducibility

The `_generated` contract guarantees reproducibility. Given the same GOV tree at the same git commit, `bin/build` always produces the same RUNTIME tree:

```
# Build from commit abc123
git checkout abc123
bin/build
sha256sum ~/.canonic/SERVICES/TALK/MAMMOCHAT/CANON.json
# Output: sha256:deadbeef...

# Clean and rebuild from same commit
rm -rf ~/.canonic/
bin/build
sha256sum ~/.canonic/SERVICES/TALK/MAMMOCHAT/CANON.json
# Output: sha256:deadbeef... (identical)
```

Reproducibility is essential for compliance. An auditor can verify that the deployed CANON.json matches the GOV source at the deployment commit. The hash proves it [14].

27.15. 26.15 Multi-Repository _generated

The `_generated` contract spans repositories. `canonic-canonic` (the kernel) and `hadleylab-canonic` (the organization) both produce `_generated` artifacts. The inheritance chain crosses repository boundaries:

```
canonic-canonic/CANON.md          → ~/.canonic/canonic-canonic/CANON.json
hadleylab-canonic/CANON.md        → ~/.canonic/hadleylab-canonic/CANON.json
hadleylab-canonic/SERVICES/TALK/  → ~/.canonic/SERVICES/TALK/CANON.json
```

The build pipeline resolves cross-repository inheritance via git submodules. `hadleylab-canonic` includes `canonic-canonic` as a submodule. The compiler walks both trees during discovery [2].

27.16. 26.16 Error Recovery Procedures

When `_generated` files are corrupted or hand-edited, the recovery procedure is always the same:

```
# Step 1: Verify the damage
magic validate --generated
```

```
# Step 2: Identify affected files
# (validator output lists all corrupted files)

# Step 3: Clean and rebuild
rm -rf ~/.canonic/
bin/build

# Step 4: Validate
magic validate --fleet

# Step 5: Verify
magic validate --generated
```

Never attempt to repair a `_generated` file by hand. Delete and rebuild. The pipeline is idempotent — running it twice produces the same result [214].

27.17. 26.17 _generated and Version Control

Some `_generated` files are committed to git. Others are ephemeral. The distinction:

File	Git Status	Rationale
`MAGIC/galaxy.json`	Committed	Required by Jekyll at build time
`_data/navigation.json`	Committed	Required by Jekyll at build time
`CANON.json` (per scope)	Not committed	Regenerated per build
`_site/` (Jekyll output)	Not committed	Ephemeral build artifact
`DESIGN.css`	Committed	Required by theme at runtime

The `.gitignore` enforces the boundary:

```
# _generated artifacts that are NOT committed
_site/
.jekyll-cache/
*.json.bak

# _generated artifacts that ARE committed (needed by Jekyll)
# MAGIC/galaxy.json — DO NOT .gitignore
# _data/navigation.json — DO NOT .gitignore
```

The rule: if Jekyll needs it at remote build time (GitHub Pages), commit it. If it can be regenerated locally, do not commit it. The build pipeline handles regeneration [2].

27.18. 26.18 _generated Diffing

When reviewing a PR that changes governance, the reviewer should also inspect the `_generated` diff:

```
# See what governance changes produced in _generated
git diff --stat HEAD~1 -- _data/
```

```
 MAGIC/galaxy.json     | 12 ++++++------
 _data/navigation.json |  4 ++--
 2 files changed, 8 insertions(+), 8 deletions(-)
```

The `_generated` diff should track the governance diff: new scope in GOV means new entry in `galaxy.json`; removed scope means removed entry. Any inconsistency means the build was not run after the governance change.

The CI pipeline enforces this:

```
- name: Check _generated consistency
  run: |
    bin/build
    git diff --exit-code _data/
    if [ $? -ne 0 ]; then
      echo "ERROR: _generated files are stale. Run bin/build."
      exit 1
    fi
```

27.19. 26.19 Understanding Compilation Stages

The compiler transforms governance through four stages. Each stage produces intermediate artifacts:

```
Stage 1: DISCOVERY
  Input:  GOV tree (file system)
  Output: scope_list.json (list of all CANON.md paths)

Stage 2: PARSING
  Input:  scope_list.json + CANON.md files
  Output: parsed_scopes.json (YAML parsed to JSON, per scope)

Stage 3: RESOLUTION
  Input:  parsed_scopes.json
  Output: resolved_scopes.json (inheritance chains resolved)
```

```
Stage 4: EMISSION
  Input:  resolved_scopes.json + INTEL.md + LEARNING.md
  Output: CANON.json (per scope) + galaxy.json + navigation.json
```

The intermediate artifacts (stage 1-3) are ephemeral — they exist only in memory during compilation. Only stage 4 produces persistent output. The intermediate stages are logged for debugging:

```
[DISCOVERY] Found 73 scope paths
[PARSING]   Parsed 73 CANON.md (0 errors)
[RESOLUTION] Resolved 73 inheritance chains
  Deepest chain: SERVICES/TALK/MAMMOCHAT (depth 5)
  Widest scope: SERVICES (12 direct children)
[EMISSION]  Emitted 73 CANON.json + 3 aggregate files
```

27.20. 26.20 Debugging _generated Issues

When a `_generated` file has incorrect content, debug by tracing the compilation pipeline:

```
# Step 1: Verify the source
cat ~/CANONIC/hadleylab-canonic/SERVICES/TALK/MAMMOCHAT/CANON.md
# Check: Is the axiom correct? Are constraints right?

# Step 2: Verify the INTEL
cat ~/CANONIC/hadleylab-canonic/SERVICES/TALK/MAMMOCHAT/INTEL.md
# Check: Is the evidence chain current? Are sources indexed?

# Step 3: Verify the inheritance chain
magic validate --axiom-chain SERVICES/TALK/MAMMOCHAT
# Check: Does the chain resolve without contradiction?

# Step 4: Rebuild and check the output
bin/build --verbose SERVICES/TALK/MAMMOCHAT
# Check: Is the CANON.json correct?

# Step 5: If still wrong, check the compiler
magic --version
# Check: Is the compiler up to date?
```

The debugging order is always: source first, then inheritance, then compiler. Never start by editing the output [214].

27.21. 26.21 _generated and Testing

The `_generated` artifacts are testable. The test suite validates that compiled output matches governance expectations:

```
# Test that CANON.json matches CANON.md expectations
magic test --generated SERVICES/TALK/MAMMOCHAT
```

```
GENERATED TEST: SERVICES/TALK/MAMMOCHAT
─────────────────────────────────────────────────
CANON.json exists:          ✓
Source hash matches:        ✓ (sha256:abc123)
Score matches:              ✓ (255)
Tier matches:               ✓ (MAGIC)
Axiom present:              ✓
SystemPrompt present:       ✓ (2,847 tokens)
Constraints count:          ✓ (3 MUST, 4 MUST NOT)
Persona matches:            ✓ (clinical-third)
Brand accent matches:       ✓ (#ec4899)
Disclaimer present:         ✓
─────────────────────────────────────────────────
Result: PASS (10/10)
```

Any mismatch between the source CANON.md and the compiled CANON.json indicates a compiler bug — not a governance error. File a bug against the compiler; do not hand-edit the output [2].

27.22. 26.22 _generated File Sizes

The build pipeline monitors compiled artifact sizes. Bloated `_generated` files indicate governance debt:

File	Expected Size	Warning Threshold	Error Threshold
CANON.json (per scope)	2-10 KB	> 50 KB	> 100 KB
galaxy.json	200-500 KB	> 1 MB	> 2 MB
navigation.json	5-20 KB	> 100 KB	> 500 KB
DESIGN.css	15-30 KB	> 100 KB	> 200 KB

```
magic validate --generated --sizes
```

```
GENERATED SIZES: ~/.canonic/
─────────────────────────────────────────────────
```

```
CANON.json (avg): 4.2 KB ✓
CANON.json (max): 8.7 KB (SERVICES/TALK/MAMMOCHAT) ✓
galaxy.json:      449 KB ✓
navigation.json:  12 KB ✓
DESIGN.css:       18 KB ✓
──────────────────────────────────────────────────
Result: PASS (all within thresholds)
```

A CANON.json over 50KB usually means the systemPrompt is too long (too much INTEL compiled into context) or the constraints section is overloaded. The fix is in the governance — simplify the INTEL or restructure the constraints [214].

27.23. 26.23 The Golden Rule

The `_generated` contract has one golden rule:

```
IF file IS _generated:
  DO NOT edit the file.
  DO edit the source that produced it.
  DO run bin/build after editing the source.
  DO run magic validate after building.

IF output IS wrong:
  DO NOT fix the output.
  DO fix the source.
  IF source IS correct AND output IS wrong:
    DO file a bug against the compiler.
```

No exceptions. Every hand-edit to a `_generated` file will be overwritten on the next build — which means it is wasted work. And if it is *not* overwritten, your build is broken. Fix the source. Run the build. Trust the compiler [214].

27.24. 26.24 _generated Artifacts Per Scope Type

Different scope types produce different `_generated` artifacts:

Scope Type	CANON.json	systemPrompt	index.md	Product Card	Jekyll Page
SERVICE	Yes	Yes (if talk)	Yes	Yes (if shop)	Yes
BOOK	Yes	Yes (per chapter)	Yes	Yes	Yes

Scope Type	CANON.json	systemPrompt	index.md	Product Card	Jekyll Page
PAPER	Yes	Yes	Yes	No	Yes
DECK	Yes	No	Yes	No	Yes
DEAL	Yes	No	Yes	No	Yes
USER	Yes	No	Yes	No	Yes

The compiler knows what to emit based on the scope type (declared in CANON.md frontmatter `type:` field). A SERVICE scope with `talk: inline` gets a systemPrompt compiled from INTEL.md. A DEAL scope does not — it has no conversational interface.

```
# Scope type determines compilation targets
type: SERVICE + talk: inline → CANON.json + systemPrompt + index.md + page
type: SERVICE + talk: null   → CANON.json + index.md + page
type: BOOK                   → CANON.json + per-chapter systemPrompts + page
type: DEAL                   → CANON.json + index.md + page
```

27.25. 26.25 _generated and Caching

The build pipeline caches compiled artifacts for performance. The cache key is the source hash:

```python
def compile_scope(scope_path):
    source_hash = hash_files([
        f"{scope_path}/CANON.md",
        f"{scope_path}/INTEL.md",
        f"{scope_path}/LEARNING.md"
    ])

    cached = cache.get(source_hash)
    if cached:
        return cached  # Cache hit — skip compilation

    # Cache miss — compile from source
    result = do_compile(scope_path)
    cache.set(source_hash, result)
    return result
```

Cache invalidation is hash-based — not time-based. The cache never serves stale data because the hash changes when any source file changes. The cache is local (`.canonic/cache/`) and ephemeral — `bin/build --clean` clears it [2].

27.26. 26.26 The Contract in One Sentence

The `_generated` contract in one sentence: **If it is in `~/.canonic/`, the compiler wrote it; if it is wrong, fix the source in `~/CANONIC/` and rebuild.** This is the contract. There are no exceptions [214].

27.27. 26.27 _generated and Disaster Recovery

In a disaster recovery scenario, every `_generated` artifact is expendable. Clone the GOV repository, run `bin/build`, deploy. A 73-scope fleet regenerates in under 60 seconds. No backup of `_generated` files is necessary — the GOV tree is the backup, and the compiler is the recovery tool [214].

Chapter 28

Chapter 27: Scope Intelligence

A governed agent is only as good as its evidence. INTEL.md is where that evidence lives — the single file that declares what a scope knows, where that knowledge comes from, and how it connects to the rest of the governance tree. Without INTEL.md, a scope has structure but no substance. With it, every claim traces to a citation, every knowledge boundary is explicit, and every cross-scope connection is declared and validated. INTEL feeds the LEARNING service (Chapter 10), compiles into agent context (Chapter 25), and answers the expression question in the 8-dimension score (Chapter 4). For the governor's perspective on evidence management, see CANONIC CANON.

28.1. 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} |
```

28.2. 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 [10].

28.3. 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 [2].

28.4. 27.4 INTEL.md Anatomy

Every INTEL.md follows the same structure — no deviation. The compiler parses it deterministically:

```
# {SCOPE-NAME} — INTEL


## Axiom

**{INTEL axiom — what this scope knows and why it matters}**


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

28.5. Scope Intelligence
```

Dimension	Value
Subject	{what this scope knows — one line}
Audience	{who consumes this intelligence — roles, not names}
Sources	{evidence corpus summary — N papers, N blogs, N governance}
Status	{current state — INDEXED, DRAFT, STALE, ACTIVE}

• • •

28.6. Evidence Chain

Layer	Source	Count	Status
1	Governance sources (canonic-canonic)	N	INDEXED
2	Papers (PAPERS)	N	INDEXED
3	Blogs (BLOGS)	N	INDEXED
4	Service CANON.md + SPEC.md	N	SCANNED
5	External references	N	CATALOGED

• • •

28.7. Cross-Scope Connections

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

...

```
## 27.5 Citation Format

CANONIC uses three citation prefixes, each mapping to a source type:

| Prefix | Source Type | Location | Example |
|--------|-----------|----------|---------|
| `[G-XX]` | Governance source | canonic-canonic GOV tree | `^2^` = CANON.md contract |
| `[P-XX]` | Paper | PAPERS/ directory | `^3^` = governance-as-compilation |
| `[B-XX]` | Blog post | BLOGS/ | `^22^` = what-is-magic blog |

Citations are inline. Every factual claim must have a citation. The validator checks:

```bash
magic validate --citations SERVICES/TALK/MAMMOCHAT/INTEL.md


CITATION CHECK: SERVICES/TALK/MAMMOCHAT/INTEL.md
──────────────────────────────────────────────────
Claims found:    42
Claims cited:    42
Uncited claims:  0
──────────────────────────────────────────────────
Result: PASS
```

If an uncited claim is found:

```
WARNING: Uncited claim at line 47
  "MammoChat is deployed to 51 enterprise hospitals"
  Resolution: Add citation [P-XX] or [G-XX] with source reference
```

28.8. 27.6 Evidence Layers

Evidence is organized in layers. Each layer represents a source category with decreasing authority:
```

Layer	Authority	Source	Update Frequency	Validation
1	Highest	Governance sources (canonic-canonic)	On GOV change	Automatic
2	High	Peer-reviewed papers	On publication	DOI verified
3	Medium	Blog posts	On publish	Date verified
4	Medium	Service specifications	On service change	CANON.md verified
5	Lower	External references	Varies	URL checked
6	Lowest	Operational data	Real-time	LEDGER verified

Layer 1 always overrides lower layers. If a governance source contradicts a blog post, the governance source wins. The compiler resolves conflicts by layer priority [2].

28.9. 27.7 Cross-Scope Connection Patterns

The Cross-Scope Connections table in INTEL.md declares how the scope relates to every service:

```
## Cross-Scope Connections

| Service | Role |
|---------|------|
| TALK | MammoChat agent — breast imaging Q&A, BI-RADS navigation |
| COIN | Every governed conversation mints COIN via MINT:WORK |
| LEDGER | Every conversation appended to scope's LEDGER chain |
| SHOP | MammoChat listed at 255 COIN in SHOP marketplace |
| LEARNING | Conversation patterns feed back into LEARNING.md |
| DESIGN | Accent #ec4899, CUSTOM layout, _CHAT.scss styling |
| IDENTITY | Ed25519 authentication for clinician access |
| CHAIN | Every clinical decision hash-chained for audit |
| MONITORING | Response times, error rates, routing accuracy tracked |
| API | REST endpoint at /api/talk/mammochat |
```

Every service connection is a cross-scope reference (Chapter 24). Every reference is validated. Missing connections block the scope at a lower tier [2].

28.10. 27.8 Backpropagation in Practice

When a child scope's INTEL changes, the change may invalidate assumptions in parent scopes. This is governance backpropagation.

Example: SERVICES/TALK/MAMMOCHAT updates its INTEL to include a new clinical trial (NCT06604078). The backpropagation check:

```
magic intel --backprop SERVICES/TALK/MAMMOCHAT
```

```
BACKPROPAGATION: SERVICES/TALK/MAMMOCHAT
──────────────────────────────────────────────────
Changed: INTEL.md (new evidence: NCT06604078)
Parent: SERVICES/TALK
  Impact: TALK INTEL references MAMMOCHAT evidence count
  Action: Update TALK/INTEL.md evidence chain (papers: 9 → 10)
Parent: SERVICES
  Impact: None (SERVICES INTEL does not reference MAMMOCHAT directly)
Root: hadleylab-canonic
  Impact: Root INTEL cross-index needs update
  Action: Add NCT06604078 to root INTEL evidence chain
──────────────────────────────────────────────────
Backprop: 2 parent scopes need updating
```

The backpropagation flags which parent scopes need updating. The governor reviews and applies the updates. The LEDGER records the backpropagation event [10].

28.11. 27.9 INTEL Freshness

INTEL has a freshness metric. Evidence not reviewed in 90 days is flagged as potentially stale:

```
magic intel --freshness SERVICES/TALK/MAMMOCHAT
```

```
INTEL FRESHNESS: SERVICES/TALK/MAMMOCHAT
──────────────────────────────────────────────────
Layer 1 (Governance):  FRESH (updated 2026-03-08)
Layer 2 (Papers):      FRESH (updated 2026-03-01)
Layer 3 (Blogs):       STALE (last update 2025-12-15, 85 days ago)
Layer 4 (Services):    FRESH (updated 2026-03-10)
Layer 5 (External):    WARNING (last update 2026-01-05, 64 days ago)
──────────────────────────────────────────────────
Overall: WARNING (1 stale layer, 1 approaching stale)
```

Stale INTEL does not fail validation — but it flags governance debt. The LEARNING service records the staleness pattern. The governor schedules a review [2].

28.12. 27.10 INTEL Compilation for Agents

The INTEL pipeline compiles INTEL.md into agent context (Chapter 25). The compilation extracts specific fields for the systemPrompt:

INTEL Field	Agent Context	Usage
Subject	Knowledge boundary	"I know about breast imaging"
Audience	Response calibration	"I speak to clinicians"
Evidence Chain	Citation database	"I can cite [3], [2]…"
Cross-Scope	Routing table	"I route staging to OncoChat"
Status	Confidence level	"My evidence is INDEXED and FRESH"

The compiler reads each field from INTEL.md and injects it into the systemPrompt template:

```
SERVICE (clinical):
  "You are {brand.name}, a governed clinical AI assistant.
   Your knowledge domain: {intel.subject}.
   Your audience: {intel.audience}.
   You may cite: {intel.evidence_chain}.
   You must not: {canon.constraints.must_not}.
   Disclaimer: {canon.disclaimer}."
```

28.13. 27.11 INTEL.md for Non-Clinical Scopes

INTEL.md is not healthcare-specific — every scope type uses it:

Scope Type	Subject Example	Evidence Layers	Cross-Scope Pattern
BOOK	"CANONIC implementation manual"	GOV, papers, blogs	TALK (per-chapter agent), SHOP
PAPER	"Governance-as-compilation theory"	GOV, citations, data	TALK (Q&A agent), COIN
SERVICE	"Breast imaging intelligence"	GOV, papers, clinical	All 11 services
DECK	"Investor presentation"	GOV, financials, market	SHOP, COIN, TALK

Scope Type	Subject Example	Evidence Layers	Cross-Scope Pattern
USER	“Principal professional identity”	GOV, vitae, network	VITAE, CALENDAR, DEAL
DEAL	“Partnership agreement”	GOV, legal, financial	LEDGER, COIN, TALK

Same structure, different content, identical governance [2].

28.14. 27.12 Building INTEL.md from Scratch

Create INTEL.md for a new scope:

```
# Step 1: Create the file
touch ~/CANONIC/hadleylab-canonic/SERVICES/TALK/NEWCHAT/INTEL.md

# Step 2: Write the required sections
cat > ~/CANONIC/hadleylab-canonic/SERVICES/TALK/NEWCHAT/INTEL.md << 'EOF'
# NEWCHAT — INTEL


## Axiom

**NEW DOMAIN INTELLIGENCE, GOVERNED.**


```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

28.15. Scope Intelligence

Dimension	Value
Subject	New domain clinical intelligence
Audience	Clinicians, researchers
Sources	0 papers, 0 blogs, 0 governance (initial)
Status	DRAFT
```

• • •

28.16. Evidence Chain

Layer	Source	Count	Status
1	Governance sources	0	PENDING
2	Papers	0	PENDING
3	Blogs	0	PENDING

• • •

28.17. Cross-Scope Connections

Service	Role
TALK	NEWCHAT agent — domain Q&A
COIN	Governed conversations mint COIN
LEDGER	Conversations appended to LEDGER

• • •

EOF

Chapter 29

Step 3: Validate

magic validate SERVICES/TALK/NEWCHAT

Chapter 30

Step 4: Build

bin/build

````
The minimum viable INTEL.md has four sections: Axiom, Scope Intelligence, Evidence Chain, and Cross-Scope

## 27.13 INTEL and the Expression Question

INTEL.md answers "How is it expressed?" in the MAGIC 255 score. Without it, that question goes unanswered

INTEL is the difference between AGENT and FULL tier. Every scope targeting 255 must have INTEL.md with all
## 27.14 INTEL Quality Metrics

INTEL quality is measurable. The validator reports quality metrics per INTEL.md:

| Metric | Formula | Threshold | Meaning |
|--------|---------|-----------|---------|
| Coverage | cited_claims / total_claims | > 0.95 | 95%+ claims have citations |
| Freshness | fresh_layers / total_layers | > 0.80 | 80%+ layers are fresh |
| Depth | total_evidence_units | > 10 | At least 10 distinct evidence units |
| Breadth | unique_source_types | >= 3 | At least 3 source types (GOV + paper + blog) |
| Connectivity | cross_scope_connections | >= 3 | Connected to at least 3 services |

```bash
magic intel --quality SERVICES/TALK/MAMMOCHAT

INTEL QUALITY: SERVICES/TALK/MAMMOCHAT
──────────────────────────────────────────────────

Coverage:      1.00 (42/42 claims cited) ✓
Freshness:     0.80 (4/5 layers fresh)   ✓
````
````

```
Depth:         45 evidence units          ✓
Breadth:       5 source types             ✓
Connectivity:  10 cross-scope connections ✓
──────────────────────────────────────────────

Quality: HIGH
```

Low-quality INTEL does not block compilation — it blocks tier advancement. A scope with shallow INTEL (< 10 evidence units) cannot exceed AGENT tier regardless of other dimensions [2].

30.1. 27.15 INTEL Templates

New scopes can bootstrap INTEL.md from templates. The template system provides starting points per scope type:

```
# Generate INTEL.md from template
magic intel --init --type service SERVICES/TALK/NEWCHAT
magic intel --init --type book BOOKS/NEW-BOOK
magic intel --init --type paper PAPERS/NEW-PAPER
magic intel --init --type deal DEALS/NEW-DEAL
```

Each template includes the required sections pre-filled with placeholder content. The governor replaces placeholders with actual evidence. The validator flags remaining placeholders:

```
WARNING: Placeholder content in INTEL.md
  Line 15: Subject = "{what this scope knows — one line}"
  Resolution: Replace placeholder with actual scope subject
```

30.2. 27.16 Cross-Scope Intelligence Aggregation

The root INTEL.md (`hadleylab-canonic/INTEL.md`) aggregates intelligence from all child scopes. It is the cross-index — the single surface that shows how all principals, services, and deals interconnect:

```
## Cross-Axiom Matrix

DEXTER ─── ISABELLA     (Ops + Deal Flow)
DEXTER ─── JP           (REAL_ESTATE + Lake Nona)
DEXTER ─── AVINASH      (Healthcare AI + ATOM)
DEXTER ─── YANA         (Engineering + BedaSoftware CEO)
DEXTER ─── ILYA         (Engineering + BedaSoftware CTO)
```

The matrix is not hand-curated — it compiles from all child INTEL.md files. When a new principal is promoted or a deal created, the cross-index updates automatically via `bin/build` [2].

30.3. 27.17 INTEL and Regulatory Compliance

In healthcare, INTEL.md serves as the evidence management system required by clinical AI regulations:

Regulation	INTEL Requirement	Implementation
FDA AI/ML guidance	Pre-market evidence review	Evidence chain layers 1-5
HIPAA	Knowledge source documentation	Citations with DOI and source
Joint Commission	Clinical decision support evidence	Cross-scope connections to clinical services
EU AI Act (high-risk)	Training data documentation	INTEL.md documents all knowledge sources
ISO 13485	Design input documentation	INTEL.md = design input for clinical AI

When the FDA asks "what evidence supports this clinical AI's recommendations?", the answer is: read INTEL.md. Every layer, every citation, every cross-scope connection — governed, not scattered [214].

30.4. 27.18 INTEL.md Migration Patterns

When migrating an existing knowledge base into CANONIC governance, INTEL.md is the entry point. The migration pattern:

```
# Step 1: Inventory existing knowledge sources
# List all documents, papers, blog posts, databases
# that the scope's knowledge is based on.

# Step 2: Categorize into evidence layers
# Layer 1: Governance sources (CANONIC internal)
# Layer 2: Peer-reviewed papers (with DOIs)
# Layer 3: Blog posts / articles (with dates)
# Layer 4: Service specifications
# Layer 5: External references

# Step 3: Write INTEL.md
# Use the template from 27.12

# Step 4: Add citations to all existing claims
# Every factual statement in scope documents
# must reference an evidence layer entry
```

```
# Step 5: Validate
magic validate --citations SERVICES/TALK/NEWCHAT/INTEL.md
```

Clinical migration example: An oncology department has a shared Google Drive with 47 clinical practice guidelines, 12 drug interaction databases, and 8 institutional protocols. Migrating to CANONIC INTEL:

Source	Count	INTEL Layer	Citation Format
NCCN Guidelines	15	Layer 2 (Papers)	[P-XX] NCCN 2026.1
ACR Practice Parameters	8	Layer 2 (Papers)	[P-XX] ACR 2025
Drug interaction databases	12	Layer 5 (External)	[E-XX] Lexicomp/UpToDate
Institutional protocols	8	Layer 4 (Service)	[S-XX] Protocol name
FDA safety communications	4	Layer 5 (External)	[E-XX] FDA MedWatch

The migration transforms unstructured knowledge into governed, cited, cross-referenced INTEL. After migration, every clinical claim in the scope traces to a source [2].

30.5. 27.19 INTEL Versioning and History

INTEL.md is version-controlled via git. The git history of INTEL.md is the evidence evolution history:

```
# View INTEL.md evolution
git log --oneline SERVICES/TALK/MAMMOCHAT/INTEL.md

# Output:
# abc1234 Update NCCN to 2026.1, add NCT06604078
# def5678 Add layer 5 external references
# 789abcd Initial MammoChat INTEL (12 papers, 3 governance)
```

Each commit to INTEL.md is a governance event. The LEDGER records the delta:

```
{
  "event": "INTEL:UPDATE",
  "scope": "SERVICES/TALK/MAMMOCHAT",
  "commit": "abc1234",
  "message": "Update NCCN to 2026.1, add NCT06604078",
  "delta": {
    "papers_added": 1,
```

```
    "papers_removed": 0,
    "layers_updated": ["Layer 2"],
    "freshness_improved": true
  }
}
```

The complete history of what the agent knows, when it learned it, and who governed the change is in the git log + LEDGER combination. For clinical AI compliance, this is the knowledge management audit trail [214].

30.6. 27.20 INTEL and Agent Knowledge Boundaries

INTEL.md explicitly defines what an agent knows and does not know. The knowledge boundary is compiled into the systemPrompt:

```
KNOWLEDGE BOUNDARY (compiled from INTEL.md):
  KNOWS:
    - BI-RADS classification system (ACR 5th Edition)
    - NCCN breast cancer screening guidelines (2026.1)
    - Breast imaging modalities (mammography, MRI, ultrasound)
    - Clinical trial NCT06604078 (deployment data)
    - mCODE breast cancer profiles

  DOES NOT KNOW:
    - Patient-specific data (no PHI access)
    - Non-breast oncology (routes to OncoChat)
    - Genomic variant classification (routes to OmicsChat)
    - Drug interactions (routes to MedChat)
    - Legal compliance details (routes to LawChat)
    - Financial/billing codes (routes to FinChat)
```

The knowledge boundary is the agent's type signature — it declares what the agent can answer and what it must route. The systemPrompt enforces this: a query about chemotherapy drug interactions gets a routing response, not a hallucinated answer. The boundary prevents the agent from operating outside its governed expertise. It is governance, not a suggestion [2126].

30.7. 27.21 INTEL Completeness Checklist

Use this checklist to verify INTEL.md completeness before submitting for review:

#	Item	Required	Check
1	`inherits:` field present	Yes	Path resolves
2	Axiom section present	Yes	Non-empty, non-placeholder
3	Scope Intelligence table	Yes	All 4 dimensions filled
4	Evidence Chain table	Yes	At least 2 layers populated
5	Cross-Scope Connections table	Yes	At least TALK + COIN + LEDGER
6	Citations in all claims	Yes	0 uncited claims
7	Evidence layers INDEXED	Yes	At least layers 1-2
8	Freshness < 90 days	Recommended	No stale layers
9	Quality score HIGH	Recommended	Coverage > 0.95
10	Source count > 10	Recommended	Depth threshold met

```
# Run the completeness check
magic intel --checklist SERVICES/TALK/MAMMOCHAT
```

A complete INTEL.md enables the expression question. An incomplete INTEL.md blocks FULL tier [2].

30.8. 27.22 INTEL.md as Single Source of Truth

INTEL.md is the single source of truth for what a scope knows. No second knowledge base. No hidden configuration. No evidence database that exists outside this file.

If knowledge is not in INTEL.md, the agent does not know it. If evidence is not cited there, the agent cannot cite it. If a cross-scope connection is not declared there, the routing table excludes it.

This constraint is deliberate. Auditing an agent's knowledge requires reading one file. Updating it requires editing one file. The entire knowledge provenance of a clinical AI agent fits in a single Markdown document.

The simplicity *is* the governance. Complex knowledge management systems fail because complexity hides ungoverned knowledge. INTEL.md succeeds because everything is visible, everything is cited, and everything compiles [2].

Chapter 31

Chapter 28: The Naming Convention

In most codebases, naming is a style preference. In CANONIC, naming is a discovery mechanism. The compiler (Chapter 37) finds scopes by looking for SCREAMING_CASE directories containing CANON.md. Rename a directory to lowercase and the compiler stops seeing it — the scope vanishes from the fleet, cross-references dangle, and the GALAXY (Chapter 31) goes dark. The naming convention is not cosmetic; it is structural. For the `_generated` contract that governs compiler output naming, see Chapter 26.

31.1. 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 [6].

31.2. 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

31.3. 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 [7].

31.4. 28.4 RUNTIME Naming

RUNTIME artifacts use lowercase. CANON.json, galaxy.json, fleet.json. No SCREAMING_CASE in `~/.canonic/` [6].

31.5. 28.5 File Naming Conventions

Files within scopes follow strict naming conventions. The convention depends on the file type:

File Type	Convention	Example	Rule
Governance	SCREAMING_CASE.md	`CANON.md`, `INTEL.md`, `LEARNING.md`	Core governance artifacts
Spec	SCREAMING_CASE.md	`MAMMOCHAT.md`, `DESIGN.md`	Scope-specific specifications
Blog post	date-kebab-case.md	`2026-02-18-what-is-magic.md`	ISO date prefix required
Config	lowercase	`_config.yml`, `Gemfile`	Jekyll/Ruby conventions
Script	lowercase-kebab	`build-galaxy-json`	Executable scripts
SCSS partial	_SCREAMING_CASE.scss	`_TOKENS.scss`, `_CHAT.scss`	Sass partials, underscore prefix
JavaScript	camelCase.js	`talk.js`, `galaxyView.js`	Standard JS convention
JSON (compiled)	lowercase.json	`CANON.json`, `galaxy.json`	Compiler output
Include (HTML)	SCREAMING_CASE.html	`HERO.html`, `CARDS.html`	Layout components

File Type	Convention	Example	Rule
Layout (HTML)	SCREAMING_CASE.html or lowercase.html	`DECK.html`, `default.html`	Jekyll layouts

```
# Correct naming examples
~/CANONIC/hadleylab-canonic/
├── SERVICES/
│   ├── TALK/
│   │   ├── CANON.md                    # SCREAMING governance
│   │   ├── INTEL.md                    # SCREAMING governance
│   │   ├── LEARNING.md                 # SCREAMING governance
│   │   ├── MAMMOCHAT/
│   │   │   ├── CANON.md
│   │   │   ├── INTEL.md
│   │   │   ├── MAMMOCHAT.md            # SCREAMING spec
│   │   │   └── LEARNING.md
│   │   └── ONCOCHAT/
│   │       ├── CANON.md
│   │       ├── INTEL.md
│   │       └── ONCOCHAT.md
│   └── DESIGN/
│       ├── CANON.md
│       ├── INTEL.md
│       └── DESIGN.md
├── DEXTER/
│   ├── BLOGS/
│   │   ├── 2026-02-18-what-is-magic.md       # date-kebab blog
│   │   └── 2026-03-01-governance-compiles.md  # date-kebab blog
│   └── BOOKS/
│       └── CANONIC-DOCTRINE/
│           ├── CANON.md
│           ├── INTEL.md
│           └── CANONIC-DOCTRINE.md
└── bin/
    ├── build                           # lowercase script
    ├── build-galaxy-json               # kebab-case script
    └── magic                           # lowercase script
```

31.6. 28.6 Directory Naming Rules

Directories follow scope-type rules. The compiler discovers scopes by directory name convention:

Directory Type	Convention	Example	Compiler Behavior
Service (schema)	SCREAMING_CASE singular	`SERVICES/LEARNING/`	Discovered as scope
Instance (data)	SCREAMING_CASE plural	`DEXTER/WALLETS/`	Discovered as collection
Principal	SCREAMING_CASE	`DEXTER/`	Discovered as user scope
Content	SCREAMING_CASE	`BLOGS/`, `BOOKS/`, `PAPERS/`	Discovered as content scope
Build	lowercase	`bin/`, `_includes/`, `_layouts/`	Skipped by scope discovery
Generated	lowercase dot-prefix	`.canonic/`, `_data/`	Runtime only

The compiler's discovery algorithm:

```
# Simplified scope discovery (magic scan)
def discover_scopes(root):
    scopes = []
    for dirpath, dirnames, filenames in os.walk(root):
        if 'CANON.md' in filenames:
            scope_name = os.path.basename(dirpath)
            if scope_name == scope_name.upper():
                # SCREAMING_CASE directory with CANON.md = valid scope
                scopes.append(dirpath)
            else:
                # Not SCREAMING_CASE — invalid scope name
                warn(f"CANON.md found in non-SCREAMING directory: {dirpath}")
    return scopes
```

A CANON.md in a lowercase directory triggers a warning — the compiler will not discover it as a scope [7].

31.7. 28.7 The Inheritance Path Convention

The `inherits:` field in every governance file uses forward-slash-separated paths from the repository root:

```
```

Rules: 1. Always start from the repository root (`hadleylab-canonic` or `canonic-canonic`). 2. Always use forward slashes (`/`), never backslashes. 3. Always use the exact directory names (SCREAMING_CASE). 4. Never include file extensions. 5. Never include trailing slashes.

```
# Correct
# Wrong — includes file extension
# Wrong — trailing slash
# Wrong — lowercase
# Wrong — relative path
```

31.8. 28.8 The GOV_ROOT and RUNTIME_ROOT

Two environment variables define the naming boundaries:

```
# GOV_ROOT — where human-authored governance lives
export GOV_ROOT=~/CANONIC

# RUNTIME_ROOT — where compiler output lives
export RUNTIME_ROOT=~/.canonic
```

Property	GOV_ROOT	RUNTIME_ROOT
Location	`~/CANONIC/`	`~/.canonic/`
Convention	SCREAMING_CASE	lowercase
Authors	Human governors	Compiler only
Files	`.md` (Markdown)	`.json`, `.html`, `.css`
Version control	Git (committed)	Git (some) or ephemeral
Editing	vim, Claude, any editor	Never hand-edit

The naming convention reinforces the separation. When you see SCREAMING_CASE, you know it is GOV — human-authored, editable. When you see lowercase, you know it is RUNTIME — compiler-authored, read-only [6].

31.9. 28.9 Naming Anti-Patterns

Common naming mistakes and their corrections:

Anti-Pattern	Example	Correction	Rule Violated
Mixed case scope dir	`Services/Talk/`	`SERVICES/TALK/`	SCREAMING_CASE for scopes
Plural service name	`SERVICES/LEARNINGS/`	`SERVICES/LEARNING/`	Singular for schemas
Singular instance dir	`DEXTER/WALLET/`	`DEXTER/WALLETS/`	Plural for instances

Anti-Pattern	Example	Correction	Rule Violated
Spaces in names	`CANONIC DOCTRINE/`	`CANONIC-DOCTRINE/`	Hyphens, not spaces
Underscores in dirs	`CANONIC_DOCTRINE/`	`CANONIC-DOCTRINE/`	Hyphens for multi-word
camelCase in dirs	`mammoChat/`	`MAMMOCHAT/`	SCREAMING_CASE
Lowercase governance	`canon.md`	`CANON.md`	SCREAMING for governance
SCREAMING runtime	`CANON.JSON`	`CANON.json`	lowercase for runtime

```
# Validate naming conventions
magic validate --naming
```

```
NAMING VALIDATION: hadleylab-canonic
──────────────────────────────────────────────────
Scopes checked:     73
Naming violations:  0
──────────────────────────────────────────────────
Result: PASS
```

31.10. 28.10 JavaScript and CSS Naming

Code files follow language-standard conventions:

```
// JavaScript — camelCase for variables and functions
const scopeName = "MAMMOCHAT";
const systemPrompt = canon.systemPrompt;
function routeQuery(query) { /* ... */ }
function buildGalaxyJson() { /* ... */ }

// JavaScript — PascalCase for classes and components
class GalaxyView { /* ... */ }
class ScopeRenderer { /* ... */ }
```

```
// SCSS — kebab-case for CSS custom properties
:root {
  --space-md: 24px;
  --font-md: 1rem;
  --accent: #3b82f6;
  --fg: #e5e5e5;
  --bg: #0a0a0a;
  --radius-sm: 4px;
  --shadow-card: 0 2px 8px rgba(0,0,0,0.3);
```

```
  --transition-fast: 150ms ease;
  --z-modal: 1000;
}

// SCSS — BEM-style for component classes
.chat-container { /* ... */ }
.chat-message { /* ... */ }
.chat-message--user { /* ... */ }
.chat-message--agent { /* ... */ }
.chat-input { /* ... */ }
.galaxy-node { /* ... */ }
.galaxy-edge { /* ... */ }
```

31.11. 28.11 The Naming Contract

The naming convention is a contract that the compiler, the discovery algorithm, the GALAXY visualization, and the build pipeline all depend on:

```
SCREAMING_CASE + CANON.md = scope (discovered by compiler)
SCREAMING_CASE + no CANON.md = directory (not a scope)
lowercase + .json = compiled artifact (never hand-edit)
lowercase + .md = leaf content (no governance contract)
_SCREAMING.scss = Sass partial (design layer)
camelCase.js = JavaScript module (runtime code)
```

This is not a style preference — it is a type system. The naming convention is the type; the compiler is the type checker [76]. ## 28.12 Cross-Repository Naming

When multiple repositories participate in the governance tree, naming conventions must be consistent across repository boundaries:

Repository	Convention	Role
`canonic-canonic`	Kernel. SCREAMING_CASE.	Defines the type system
`hadleylab-canonic`	Organization. SCREAMING_CASE.	Implements the organization
`canonic-python`	Runtime. lowercase.	Python runtime package
`canonic-theme`	Runtime. lowercase.	Jekyll theme package

The kernel repository (`canonic-canonic`) uses SCREAMING_CASE for its governance files:

```
canonic-canonic/
├── CANON.md                    # Root governance contract
├── SERVICES/
│   ├── LEARNING/CANON.md       # LEARNING service definition
│   ├── TALK/CANON.md           # TALK service definition
│   └── ...
└── MAGIC/
    └── TOOLCHAIN/              # Compiler and validator tools
```

Runtime repositories use lowercase:

```
canonic-python/
├── canonic/
│   ├── __init__.py
│   ├── compiler.py
│   ├── validator.py
│   └── scanner.py
├── setup.py
└── tests/
```

The boundary is clear: GOV repositories use SCREAMING. Runtime repositories use lowercase. No repository mixes conventions [76].

31.12. 28.13 Naming and Scope Discovery

The `magic scan` command discovers scopes by looking for SCREAMING_CASE directories containing CANON.md:

```
magic scan
```

```
SCOPE DISCOVERY: hadleylab-canonic
──────────────────────────────────────────────────
Scanned:        248 directories
SCREAMING_CASE: 156 directories
Has CANON.md:   73 directories → 73 scopes
Warnings:
  CANON.md in lowercase dir: 0
  SCREAMING dir without CANON.md: 83 (normal — container dirs)
──────────────────────────────────────────────────
Discovered: 73 scopes
```

The 83 SCREAMING directories without CANON.md are containers (`SERVICES/`, `BOOKS/`, `BLOGS/`) — they organize the tree but are not scopes. Only directories with both SCREAMING_CASE naming and a CANON.md file qualify as governance scopes [7].

31.13. 28.14 Naming Migration

When renaming a scope, follow this procedure:

```
# Step 1: Create the new directory
mkdir -p ~/CANONIC/hadleylab-canonic/SERVICES/TALK/NEWNAME/

# Step 2: Move governance files
mv SERVICES/TALK/OLDNAME/CANON.md SERVICES/TALK/NEWNAME/
mv SERVICES/TALK/OLDNAME/INTEL.md SERVICES/TALK/NEWNAME/
mv SERVICES/TALK/OLDNAME/LEARNING.md SERVICES/TALK/NEWNAME/

# Step 3: Update inherits: in all moved files
# inherits: hadleylab-canonic/SERVICES/TALK/NEWNAME

# Step 4: Update all cross-scope references that pointed to OLDNAME
# (magic validate --cross-refs will flag dangling references)

# Step 5: Remove old directory
rmdir SERVICES/TALK/OLDNAME/

# Step 6: Rebuild and validate
bin/build
magic validate --fleet
```

Never rename by editing CANON.json in RUNTIME. Rename in GOV, rebuild, validate [76].

31.14. 28.15 Naming in Frontmatter

YAML frontmatter in governance files follows naming conventions too:

```
```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

layout: CUSTOM # SCREAMING for CANONIC layouts title: MammoChat # PascalCase for display names scope: MAMMOCHAT # SCREAMING for scope identifier talk: inline # lowercase for config values accent: “#ec4899” # lowercase for CSS values tier: MAGIC # SCREAMING for tier names score: 255 # numeric
```

```
| Frontmatter Key | Value Convention | Example |
|-----------------|------------------|---------|
| `layout` | SCREAMING (CANONIC) or lowercase (Jekyll) | `CUSTOM`, `default` |
| `title` | Display case | `MammoChat`, `The CANONIC DOCTRINE` |
| `scope` | SCREAMING\_CASE | `MAMMOCHAT`, `LEARNING` |
| `talk` | lowercase | `inline`, `widget`, `disabled` |
| `accent` | CSS hex value | `"#ec4899"` |
| `tier` | SCREAMING | `MAGIC`, `AGENT`, `ENTERPRISE` |
| `inherits` | Path convention | `hadleylab-canonic/SERVICES/TALK` |
| `references` | Path convention | `PAPERS/governance-as-compilation` |
| `type` | SCREAMING | `SERVICE`, `BOOK`, `PAPER`, `DEAL` |

The validator checks frontmatter naming:

```bash
magic validate --frontmatter SERVICES/TALK/MAMMOCHAT/CANON.md

FRONTMATTER VALIDATION: SERVICES/TALK/MAMMOCHAT/CANON.md
──────────────────────────────────────────────────
Fields checked:     12
Naming violations:  0
──────────────────────────────────────────────────
Result: PASS
```

31.15. 28.16 Naming in URLs

URLs follow the GOV tree path, lowercased for web compatibility:

GOV Path	URL	Rule
`SERVICES/TALK/MAMMOCHAT/`	`/services/talk/mammochat/`	Lowercased
`BOOKS/CANONIC-DOCTRINE/`	`/dexter/books/canonic-doctrine/`	Lowercased
`BLOGS/2026-02-18-what-is-magic.md`	`/dexter/blogs/2026-02-18-what-is-magic/`	Lowercased, no extension

The Jekyll build pipeline transforms SCREAMING_CASE paths to lowercase URLs via `permalink` configuration:

```yaml
# _config.yml
permalink: /:path/
```

Jekyll lowercases the output path automatically. The URL is the GOV path, lowercased. No custom routing. No URL rewriting. The GOV tree IS the URL tree [6].
```

31.16. 28.17 Naming Across Languages

When CANONIC content is localized (Arabic, Russian, Spanish, French), file naming stays in English SCREAMING_CASE. The content inside is localized:

```
SERVICES/TALK/MAMMOCHAT/
├── CANON.md                    # English (source of truth)
├── INTEL.md                    # English (evidence chain)
├── MAMMOCHAT.md                # English (specification)
├── CANON.ar.md                 # Arabic (localized)
├── CANON.ru.md                 # Russian (localized)
├── CANON.es.md                 # Spanish (localized)
└── CANON.fr.md                 # French (localized)
```

The localization suffix follows the pattern `{FILENAME}.{ISO-639-1}.md`. The base file (no suffix) is always English. The compiler treats localized files as variants of the base, not as separate scopes [7].

31.17. 28.18 Naming Validation Errors

Error Code	Description	Example	Fix
`NAME-001`	Scope dir not SCREAMING	`services/talk/`	Rename to `SERVICES/TALK/`
`NAME-002`	Service name is plural	`LEARNINGS/`	Rename to `LEARNING/`
`NAME-003`	Instance name is singular	`WALLET/`	Rename to `WALLETS/`
`NAME-004`	Spaces in directory name	`CANONIC DOCTRINE/`	Rename to `CANONIC-DOCTRINE/`
`NAME-005`	Underscores in dir name	`CANONIC_DOCTRINE/`	Rename to `CANONIC-DOCTRINE/`
`NAME-006`	Governance file lowercase	`canon.md`	Rename to `CANON.md`
`NAME-007`	Runtime file SCREAMING	`CANON.JSON`	Rename to `CANON.json`
`NAME-008`	Inherits path malformed	`../TALK/MAMMOCHAT`	Use full path from repo root
`NAME-009`	Inherits includes extension	`MAMMOCHAT/CANON.md`	Remove `.md` extension

Error Code	Description	Example	Fix
NAME-010	Frontmatter key wrong case	Scope: mammochat	Use scope: MAMMOCHAT

Every naming error is caught by magic validate --naming. Fix the naming, rebuild, validate [76].

31.18. 28.19 Naming Conventions Quick Reference

Print this table. Pin it above your monitor. The naming convention is the type system.

Context	Convention	Example	Why
Scope directory	SCREAMING_CASE	MAMMOCHAT/	Compiler discovery
Service name	Singular SCREAMING	LEARNING/	Schema, not instance
Instance directory	Plural SCREAMING	WALLETS/	Collection of instances
Governance file	SCREAMING.md	CANON.md	Human-authored authority
Spec file	SCREAMING.md	MAMMOCHAT.md	Scope specification
Blog post	date-kebab.md	2026-02-18-what-is-magic.md	Chronological content
SCSS partial	_SCREAMING.scss	_TOKENS.scss	Design layer
JavaScript module	camelCase.js	talk.js	Runtime code
Compiled JSON	UPPER.json	CANON.json	Compiler output
Data JSON	lowercase.json	galaxy.json	Aggregated data
Include HTML	SCREAMING.html	HERO.html	Layout component
Layout HTML	case-varies.html	CUSTOM.html	Jekyll layout
CSS property	–kebab-case	--space-md	CSS standard
JS variable	camelCase	scopeName	JS standard
JS class	PascalCase	GalaxyView	JS standard
URL path	lowercase	/services/talk/	Web standard
inherits field	repo/PATH	hadleylab-canonic/SERVICES/TALK	Resolution path
Frontmatter key	lowercase	talk: inline	YAML standard

Context	Convention	Example	Why
Tier name	SCREAMING	`MAGIC`	Governance constant
Dimension name	SCREAMING or short	`LANG`, `ECON`	8-bit constant

31.19. 28.20 Why Naming Matters

The naming convention is functional, not aesthetic. The compiler uses naming to discover scopes, the build pipeline to route files, the GALAXY to categorize nodes, and the URL structure to map paths.

Rename `SERVICES/TALK/MAMMOCHAT/` to `services/talk/mammochat/` and watch the cascade: 1. The compiler will not discover it as a scope. 2. The build will not compile it. 3. The GALAXY will not render it. 4. The URL will break. 5. Every cross-scope reference to it will become a dangling reference. 6. The fleet score will degrade.

One naming mistake cascades through the entire system. The rest of the governance tree stands on this convention [76].

31.20. 28.21 Naming Conventions and Tooling

The naming convention enables tooling. IDE plugins, CLI tools, and CI scripts all use naming patterns:

IDE Integration:

```
// .vscode/settings.json — highlight naming violations
{
  "files.associations": {
    "CANON.md": "markdown",
    "INTEL.md": "markdown",
    "LEARNING.md": "markdown"
  },
  "editor.rulers": [80],
  "files.exclude": {
    "**/.canonic/": true,
    "**/_site/": true
  }
}
```

CLI Tab Completion:

```
# Tab completion uses SCREAMING_CASE detection
magic validate SERVICES/TALK/<TAB>
# Offers: MAMMOCHAT/ ONCOCHAT/ MEDCHAT/ OMICSCHAT/ LAWCHAT/ FINCHAT/

magic validate services/<TAB>
# No completions — lowercase is not a scope path
```

CI Naming Enforcement:

```
# .github/workflows/naming.yml
- name: Enforce naming conventions
  run: |
    magic validate --naming
    if [ $? -ne 0 ]; then
      echo "Naming violation detected. See errors above."
      exit 1
    fi
```

31.21. 28.22 Naming and Git

Git operations interact with naming conventions. Branch names, commit messages, and tag names follow their own conventions:

Git Artifact	Convention	Example
Branch name	kebab-case	`feature/add-mammochat-intel`
Commit message	Imperative, scope prefix	`GOV: add MAMMOCHAT INTEL.md`
Tag name	semver	`v1.0.0`
Submodule path	lowercase-kebab	`canonic-canonic`

Commit message prefixes indicate the change type:

Prefix	Meaning	Example
`GOV:`	Governance change (CANON.md, INTEL.md)	`GOV: add MAMMOCHAT INTEL.md`
`CI:`	CI/CD pipeline change	`CI: add pyyaml dependency`
`DESIGN:`	Token or styling change	`DESIGN: update accent to #ec4899`
`BUILD:`	Build pipeline change	`BUILD: fix incremental compilation`

Prefix	Meaning	Example
`TALK:`	TALK service change	`TALK: add routing for OncoChat`

The commit prefix is the change type. The scope after the prefix is the affected scope. The rest is the description. The git log becomes a governance changelog [76].

31.22. 28.23 Naming as Documentation

A developer who has never seen the codebase can understand the structure from names alone:

```
~/CANONIC/hadleylab-canonic/SERVICES/TALK/MAMMOCHAT/CANON.md
|         |                 |        |    |         |
|         |                 |        |    |         └── Governance contract
|         |                 |        |    └── Scope name (breast imaging chat)
|         |                 |        └── Parent service (conversation)
|         |                 └── Container (all services)
|         └── Organization (Hadley Lab)
└── GOV root
```

Every segment tells you something: GOV root, organization, container type, parent service, scope name, file type. No README needed — the path is the documentation [76].

31.23. 28.24 Naming Enforcement Automation

Enforcement happens at three levels: editor (lint-on-save), pre-commit hook (local gate), and CI (remote gate). All three run `magic validate --naming`. A naming violation cannot reach production because it cannot pass CI — the convention is a gate, not a guideline [76].

```
# Pre-commit hook (.git/hooks/pre-commit)
#!/bin/bash
magic validate --naming
if [ $? -ne 0 ]; then
  echo "Naming violation. Fix before committing."
  exit 1
fi
```

Chapter 32

Chapter 29: DESIGN Tokens

Every pixel in CANONIC traces to a governed value. The design system enforces this through token discipline: all visual values — spacing, color, typography, z-index, animation — live in `_TOKENS.scss`. No magic numbers. No hardcoded hex codes scattered across partials. When you change a token, every surface in the fleet changes with it — across hadleylab.org, mammo.chat, shop.hadleylab.org, and every fleet domain. When the validator runs (see Chapter 47 for `validate-design`), it catches any literal value that slipped outside the token file. The token pipeline connects to the CHAT layer described in Chapter 30 and the naming conventions established in Chapter 28. For the governor's perspective on visual identity as governance, see the CANONIC CANON. The design system is governance applied to CSS.

32.1. 29.1 Token Discipline

All visual values must use tokens. No magic numbers. No hardcoded colors [24].

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 [24].

32.2. 29.2 The 23 Sass Partials

Ordered layers 0-19 [25]:

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

32.3. 29.3 DESIGN.css

One stylesheet for all surfaces. DESIGN.css is universal. No forking per surface. No hardcoded content in renderers [24].

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

32.4. 29.4 Breakpoints

Three breakpoints. No more [24]:

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) [24].

32.5. 29.5 Visual Design Governance

Code-first design works. `_TOKENS.scss` governs 80+ visual values. `magic validate` gates everything at 255. But code-first has a velocity problem: composing a new fleet page requires writing frontmatter YAML, guessing which `_includes/` to combine, previewing via `jekyll serve`, and iterating. The feedback loop is slow — the governor cannot see the page until after the developer builds it.

Penpot solves this. Open-source (MPL-2.0), self-hosted (Docker), SVG/CSS/HTML native — not proprietary Figma blobs. Design tokens are a first-class feature, exportable to CSS, SCSS, and JSON. The

Penpot workspace — visual authoring surface with components panel

Figure 32.1: Penpot workspace — visual authoring surface with components panel

Token sync pipeline — one-way flow from Penpot through diff gate to production

Figure 32.2: Token sync pipeline — one-way flow from Penpot through diff gate to production

official MCP server (merged into core February 2026) lets Claude agents read and write design files programmatically. No vendor lock-in, no per-seat licensing, full sovereignty [24].

The DESIGN service governs Penpot the same way it governs everything else: CANON.md declares the axiom, constraints bind the tool, the diff gate enforces token authority, and LEARNING.md records patterns. Penpot is INPUT. `_TOKENS.scss` is AUTHORITY.

32.6. 29.6 The Token Sync Pipeline

One-way flow. Never bidirectional. Never let the visual tool overwrite code authority.

```
Penpot (visual authoring — INPUT)
  ↓
penpot-export → tokens.json
  ↓
DIFF GATE: tokens.json vs _TOKENS.scss
  ↓ PASS              ↓ FAIL
No drift detected    CI blocks deploy
  ↓                   ↓
build → DESIGN.css   Fix source
  ↓
magic validate → 255
  ↓
Deploy fleet
```

The diff gate is the enforcement boundary. If a designer changes `--accent` from `#3b82f6` to `#ec4899` in Penpot, `penpot-export` captures the change. The gate compares against `_TOKENS.scss`. If the values diverge: CI fails. The token change must be accepted into `_TOKENS.scss` by a governor before it reaches production. No ungoverned pixels.

If a developer changes `_TOKENS.scss` directly (the authority path), Penpot's export will show drift on the next run — flagging that the visual library needs to be updated to match. Either way, the diff gate catches divergence.

In practice: a designer proposes changing the HERO section spacing from `--space-lg` (48px) to `--space-xl` (80px) for better readability on radiology workstations. She makes the change in Penpot. `penpot-export` captures it. The diff gate detects the divergence. The governor reviews, approves, and updates `_TOKENS.scss`.

CANONIC design system live — tokens in action on fleet surface

Figure 32.3: CANONIC design system live — tokens in action on fleet surface

Build runs, `magic validate` passes at 255, and the LEDGER records `DESIGN:TOKEN_UPDATE` with scope, old value, new value, and approving governor [24].

32.7. 29.7 Component Library Mirror

The DESIGN system has 26 section includes, 17 figure includes, and 7 layouts. In Penpot, each becomes a reusable component:

CANONIC Component	Penpot Mirror	Variants
`HERO.html`	HERO component	standard, split, terminal, cover
`CARDS.html`	CARDS component	2-col, 3-col, 4-col grid
`DASHBOARD.html`	DASHBOARD component	widget grid
`TABLE.html`	TABLE component	data table
`TIERS.html`	TIERS component	feature comparison
`CTA.html`	CTA component	primary, secondary
`TALK.html`	TALK component	chat widget
`score-meter`	Score Meter figure	gauge 0-255
`pipeline`	Pipeline figure	process flow

Fleet page rapid composition: drag HERO, add CARDS below, insert DASHBOARD, close with CTA. The Penpot layout becomes the front matter blueprint. The developer translates the visual composition into YAML `sections:` array. The pipeline compiles. The page matches the design because both consume the same tokens [25].

32.8. 29.8 MCP Integration

The Penpot MCP server (port 4401, Streamable HTTP) connects Claude agents to design files via WebSocket Plugin API:

```
Claude Agent (MAGIC governance)
  ↔ MCP Server (:4401 Streamable HTTP)
    ↔ WebSocket
      ↔ Penpot Plugin API
        ↔ Design File (.penpot — SVG native)
```

MCP architecture — Claude agent to Penpot design file via Plugin API

Figure 32.4: MCP architecture — Claude agent to Penpot design file via Plugin API

GALAXY visualization — fleet topology with compliance rings

Figure 32.5: GALAXY visualization — fleet topology with compliance rings

Governed operations: - **Read**: Inspect component properties, export tokens, list pages, verify WCAG contrast - **Write**: Update token values, create components from `_includes/` specs, compose fleet layouts - **Validate**: Check design against CANON.md constraints — token enforcement, contrast ratios, breakpoint compliance

The agent workflow: "Design a fleet page for SERVICES/DESIGN." Claude reads the DESIGN service CANON.md, inspects the token system, composes a Penpot layout using governed components, validates WCAG AA compliance, and exports the front matter YAML. The page is governed before a single line of HTML is written [24].

32.9. 29.9 First Composition — MammoChat Marketing Surface

The theory is done. Time to prove it.

mammochat.com existed outside CANONIC — a standalone Next.js marketing page hosted on Vercel. Custom CSS. No governance. No evidence trail. The landing page for a clinical AI product that deploys to 51 enterprise hospitals had zero clinical citations on its marketing surface. BI-RADS, NCCN, mCODE, NCT06604078 — all present in `CHAT/MAMMOCHAT/INTEL.md`, all absent from the page patients see first.

The port followed the governance-first loop:

```
1. GOVERN  — Create SERVICES/TALK/MAMMOCHAT/ (8 closure artifacts at 255)
2. BUILD   — bin/build auto-discovers MAMMOCHAT via CANON.md → compiles CANON.json + index.md
3. DEPLOY  — bin/build-domains generates Cloudflare Worker from HTTP.md ## Domains
4. CUTOVER — DNS records orange-clouded → Worker intercepts → proxies to governed surface
```

The MAMMOCHAT SPEC (`MAMMOCHAT.md`) declared what the marketing surface consumes:

```
Consumes:
  - CHAT/MAMMOCHAT/CANON.md      # clinical domain contract
  - CHAT/MAMMOCHAT/INTEL.md      # evidence chain
  - _TOKENS.scss                 # accent #ec4899 (--services)
  - _includes/*.html             # HERO + CARDS + STATS + CTA
  - CANON.json                   # compiled scope configuration
```

The INTEL.md is the bridge — it cross-references every marketing claim to the clinical evidence:

Marketing Claim	Evidence Source	Reference
51 enterprise hospitals	CHAT/MAMMOCHAT INTEL layer 5	NCT06604078
20K+ governed interactions	CHAT/MAMMOCHAT INTEL §Scope	Operational data
HIPAA Compliant	CHAT/MAMMOCHAT INTEL §Compliance	§164.312
BI-RADS classification	CHAT/MAMMOCHAT MAMMOCHAT.md	ACR 5th ed.

The content audit revealed the gap: mammochat.com had partner logos (NIH, UCF, AdventHealth) doing the work that evidence citations should do. The governed surface adds the missing evidence layer.

The domain architecture after port:

```
mammochat.com    → Cloudflare Worker → hadleylab.org/SERVICES/TALK/MAMMOCHAT/  (marketing)
mammo.chat       → Cloudflare Worker → hadleylab.org/TALK/MAMMOCHAT/      (chatbot)
```

Same product. Two governed domains. One governance tree. The marketing surface and the clinical chatbot share the same evidence chain (`CHAT/MAMMOCHAT/INTEL.md`), the same token authority (`_TOKENS.scss`), and the same build pipeline. The Next.js site is gone. The CANONIC surface compiles from governance [24].

This is the DESIGN service in action: code-first components, token-governed styling, evidence-traced content. Every pixel on mammochat.com is now governed the same way the clinical trial (NCT06604078) is governed — structurally, not advisorily.

32.10. 29.10 Token Categories Deep Dive

The token system organizes all visual values into categories. Each category has strict rules about where values may be defined and how they cascade.

Spacing Tokens:

```scss
// _TOKENS.scss — spacing scale
:root {
  --space-xs: 4px;
  --space-sm: 8px;
  --space-md: 24px;
  --space-lg: 48px;
  --space-xl: 80px;
  --space-2xl: 120px;
}
```

The scale is geometric — each step roughly 2x the previous. If a component needs 37px, you choose `--space-md` (24px) or `--space-lg` (48px). There is no `--space-37`.

Color Tokens:

```scss
// _TOKENS.scss — color system
:root {
  // Core palette
  --bg: #0a0a0a;
  --bg-elevated: #1a1a1a;
  --fg: #e5e5e5;
  --dim: #666;
  --accent: #3b82f6;           // default, overridden per scope

  // Status palette
  --status-ok: #00ff88;
  --status-warn: #ff9f0a;
  --status-error: #ff453a;
  --status-info: #2997ff;

  // Tier palette (matches GALAXY visualization)
  --tier-magic: #00ff88;
  --tier-agent: #2997ff;
  --tier-enterprise: #bf5af2;
  --tier-business: #ff9f0a;
  --tier-community: #fbbf24;
  --tier-none: #ff453a;

  // Transaction palette
  --tx-green: #00ff88;
  --tx-red: #ff453a;
  --tx-neutral: #888;

  // Service palette (unique accent per vertical)
  --services: #ec4899;         // TALK/MAMMOCHAT pink
  --oncology: #8b5cf6;         // TALK/ONCOCHAT purple
  --genomics: #06b6d4;         // TALK/OMICSCHAT cyan
  --clinical: #3b82f6;         // TALK/MEDCHAT blue
  --legal: #64748b;            // TALK/LAWCHAT slate
  --finance: #10b981;          // TALK/FINCHAT emerald
  --realty: #f59e0b;           // TALK/REALTY amber
}
```

Typography Tokens:

```scss
// _TOKENS.scss — typography
:root {
  --font-family: 'Inter', -apple-system, BlinkMacSystemFont, sans-serif;
  --font-mono: 'JetBrains Mono', 'Fira Code', monospace;
  --font-xs: 0.75rem;          // 12px
  --font-sm: 0.875rem;         // 14px
  --font-md: 1rem;             // 16px
  --font-lg: 1.25rem;          // 20px
  --font-xl: 1.5rem;           // 24px
  --font-2xl: 2rem;            // 32px
  --font-3xl: 3rem;            // 48px
  --line-height: 1.6;
  --line-height-tight: 1.2;
  --letter-spacing: 0.01em;
}
```

Z-Index Tokens:

```scss
// _TOKENS.scss — z-index scale
:root {
  --z-base: 0;
  --z-dropdown: 100;
  --z-sticky: 200;
  --z-overlay: 300;
  --z-modal: 1000;
  --z-toast: 1100;
  --z-tooltip: 1200;
}
```

No z-index value may appear outside `_TOKENS.scss`. If a component needs a z-index, it references a token. This prevents z-index wars [24].

Animation Tokens:

```scss
// _TOKENS.scss — animation
:root {
  --transition-fast: 150ms ease;
  --transition-normal: 300ms ease;
  --transition-slow: 500ms ease;
  --transition-bounce: 300ms cubic-bezier(0.68, -0.55, 0.265, 1.55);
}
```

32.11. 29.11 Token Validation

The validator checks that no literal values appear outside `_TOKENS.scss` and `_THEMES.scss`:

```
magic validate --tokens
```

```
TOKEN VALIDATION: hadleylab-canonic
──────────────────────────────────────────────────
Files scanned:     23 SCSS partials
Literal values:    0 (outside _TOKENS.scss/_THEMES.scss)
Token references:  847
Undefined tokens:  0
──────────────────────────────────────────────────
Result: PASS
```

A literal value in `_CHAT.scss`:

```
ERROR TOKEN-001: Literal value in non-token file
  File: _CHAT.scss
  Line: 42
  Value: "color: #ec4899"
  Resolution: Replace with "color: var(--accent)"
```

32.12. 29.12 Theme Switching

Tokens support theme switching via `_THEMES.scss`. Dark theme is default. Light theme overrides specific tokens:

```
// _THEMES.scss — theme overrides
[data-theme="light"] {
  --bg: #ffffff;
  --bg-elevated: #f5f5f5;
  --fg: #1a1a1a;
  --dim: #999;
  --accent: #2563eb;
  --shadow-card: 0 2px 8px rgba(0,0,0,0.1);
}

[data-theme="high-contrast"] {
  --bg: #000000;
  --fg: #ffffff;
```

```
  --dim: #cccccc;
  --accent: #ffff00;
}
```

Theme is set via a data attribute on `<html>`:

```
<html data-theme="dark">  <!-- default -->
<html data-theme="light"> <!-- light mode -->
```

32.13. 29.13 Token Export for Native Platforms

Tokens export to native platforms via JSON:

```
magic tokens --export json > tokens.json
magic tokens --export swift > Tokens.swift
magic tokens --export kotlin > Tokens.kt
```

The JSON export:

```
{
  "spacing": {
    "xs": "4px", "sm": "8px", "md": "24px",
    "lg": "48px", "xl": "80px", "2xl": "120px"
  },
  "color": {
    "bg": "#0a0a0a", "fg": "#e5e5e5", "dim": "#666",
    "accent": "#3b82f6", "statusOk": "#00ff88"
  },
  "typography": {
    "fontFamily": "Inter, -apple-system, sans-serif",
    "fontMd": "1rem", "fontLg": "1.25rem"
  }
}
```

The Swift export:

```
// Tokens.swift — generated by magic tokens --export swift
import SwiftUI

enum CanonicTokens {
    enum Spacing {
        static let xs: CGFloat = 4
```

```
        static let sm: CGFloat = 8
        static let md: CGFloat = 24
        static let lg: CGFloat = 48
        static let xl: CGFloat = 80
    }
    enum Color {
        static let bg = SwiftUI.Color(hex: "#0a0a0a")
        static let fg = SwiftUI.Color(hex: "#e5e5e5")
        static let accent = SwiftUI.Color(hex: "#3b82f6")
        static let statusOk = SwiftUI.Color(hex: "#00ff88")
    }
}
```

One source of truth (`_TOKENS.scss`), multiple platform outputs. The exports are `_generated` — do not hand-edit [24]. ## 29.14 Token Audit Trail

Every token change is tracked in the LEDGER. When `_TOKENS.scss` changes, the build pipeline records the diff:

```
{
  "event": "DESIGN:TOKEN_UPDATE",
  "timestamp": "2026-03-10T14:22:00Z",
  "file": "_TOKENS.scss",
  "changes": [
    { "token": "--space-lg", "from": "48px", "to": "56px" },
    { "token": "--accent", "from": "#3b82f6", "to": "#2563eb" }
  ],
  "governor": "DEXTER",
  "scope": "SERVICES/DESIGN",
  "hash": "sha256:abc123..."
}
```

The audit trail answers who changed the token, when, and what the old and new values were. For regulated industries, this proves visual changes were governed [24].

32.14. 29.15 Token Conflict Resolution

When two scopes define conflicting token overrides, the compiler resolves by inheritance priority:

```
canonic-canonic/DESIGN.md     → --accent: #3b82f6 (system default)
hadleylab-canonic/DESIGN.md   → --accent: #3b82f6 (org default, inherits)
SERVICES/TALK/DESIGN.md       → --accent: #3b82f6 (service default, inherits)
SERVICES/TALK/MAMMOCHAT/      → --accent: #ec4899 (scope override)
```

The most specific scope wins. MAMMOCHAT's `--accent: #ec4899` overrides the system default `#3b82f6`. The compiler injects the override at the page level:

```
<style>:root { --accent: #ec4899; }</style>
```

This is CSS custom property inheritance — the same mechanism browsers use. The governance tree maps to the CSS cascade. The compiler generates the override styles [2425].

32.15. 29.16 WCAG Compliance Automation

The token system enforces WCAG AA compliance automatically. The validator checks all text/background color combinations:

```
magic validate --wcag
```

```
WCAG VALIDATION: _TOKENS.scss
═══════════════════════════════════════════════════

Contrast checks:
  --fg (#e5e5e5) on --bg (#0a0a0a):           15.7:1 ✓ (AA: 4.5:1)
  --dim (#666) on --bg (#0a0a0a):               4.6:1 ✓ (AA: 4.5:1)
  --accent (#3b82f6) on --bg (#0a0a0a):         5.1:1 ✓ (AA: 4.5:1)
  --status-ok (#00ff88) on --bg (#0a0a0a):     12.3:1 ✓ (AA: 4.5:1)
  --status-error (#ff453a) on --bg (#0a0a0a):   5.2:1 ✓ (AA: 4.5:1)
  --bg (#0a0a0a) on --accent (#ec4899):         5.8:1 ✓ (AA: 4.5:1)
═══════════════════════════════════════════════════

Result: PASS (all combinations meet AA 4.5:1)
```

If a token change violates WCAG:

```
ERROR WCAG-001: Contrast ratio below AA threshold
  Foreground: --dim (#999) on --bg (#0a0a0a): 3.8:1
  Required: 4.5:1
  Resolution: Darken --dim to at least #aaa (4.6:1) or lighten --bg
```

The validator blocks deployment of WCAG-violating token changes. No ungoverned pixels, and no inaccessible pixels [24].

32.16. 29.17 Token Documentation

Every token has documentation. The documentation is generated from `_TOKENS.scss` comments:

```scss
// _TOKENS.scss — documented tokens

/// @group spacing
/// @description Base spacing unit for micro-adjustments
/// @usage padding, margin for dense UI elements
:root { --space-xs: 4px; }

/// @group spacing
/// @description Standard padding for form inputs and small containers
/// @usage Input fields, small card padding
:root { --space-sm: 8px; }

/// @group spacing
/// @description Primary spacing for content sections
/// @usage Section padding, message margins
:root { --space-md: 24px; }
```

Generate documentation:

```
magic tokens --docs > TOKEN-REFERENCE.md
```

The generated reference includes every token, its group, description, current value, and usage examples. The reference is `_generated` — do not hand-edit. Update the SCSS comments to update the documentation [24].

32.17. 29.18 Token Performance Impact

Tokens affect rendering performance. The design system tracks token count and specificity:

Metric	Current	Budget	Status
Total CSS custom properties	82	< 150	OK
Properties per scope override	1-3	< 10	OK
SCSS partial count	23	fixed (23)	OK
Compiled CSS size (minified)	18KB	< 30KB	OK
CSS specificity (max)	0,2,1	< 0,3,0	OK

```
magic validate --tokens --performance
```

The token performance budget ensures that the design system does not become a rendering bottleneck. Every token addition increases the `:root` declaration size. The budget caps total properties at 150 — enough for a complete design system, not enough for token sprawl [2425].

32.18. 29.19 Token Governance Summary

The token system is a microcosm of CANONIC governance. Every principle that applies to scopes applies to tokens:

Governance Principle	Scope Implementation	Token Implementation
Single source of truth	CANON.md	`_TOKENS.scss`
Compiled output	CANON.json	DESIGN.css
Never hand-edit output	Do not edit CANON.json	Do not edit DESIGN.css
Validator gates deployment	`magic validate` → 255	`magic validate --tokens` → PASS
Audit trail	LEDGER events	DESIGN:TOKEN_UPDATE events
Inheritance	inherits: field	CSS custom property cascade
Cross-scope references	INTEL.md references	Token references in partials
WCAG compliance	Content accessibility	Color contrast validation

The token system proves that CANONIC governance scales down to individual CSS values. If it works for tokens, it works for anything [24].

32.19. 29.20 Emergency Token Override

In production emergencies, a token can be overridden via Cloudflare Worker without rebuilding:

```
// Cloudflare Worker — emergency token override
addEventListener('fetch', event => {
  event.respondWith(handleRequest(event.request));
});

async function handleRequest(request) {
  const response = await fetch(request);
  const html = await response.text();

  // Emergency override: high-contrast mode for accessibility incident
  const override = '<style>:root{--bg:#000;--fg:#fff;--dim:#ccc;}</style>';
  const modified = html.replace('</head>', override + '</head>');

  return new Response(modified, {
    headers: response.headers
  });
}
```

The override is temporary, logged as `DESIGN:EMERGENCY_OVERRIDE`. The governor must ship a permanent fix in `_TOKENS.scss` within 24 hours — the override is governance debt that must be repaid [24].

32.20. 29.21 Tokens and the Build Pipeline

Token changes flow through the build pipeline like any governance change:

```
1. Developer edits _TOKENS.scss (e.g., changes --space-lg from 48px to 56px)
2. Developer runs bin/build
3. Build compiles DESIGN.css from all 23 SCSS partials
4. magic validate --tokens verifies no literal values in non-token files
5. magic validate --wcag verifies contrast ratios still pass
6. CI runs on push, gates on validation pass
7. LEDGER records DESIGN:TOKEN_UPDATE event
8. Deploy to fleet
```

The pipeline is the same one that compiles CANON.md into CANON.json. The design system is not separate from the governance system — it is the governance system, applied to visual values [24].

Chapter 33

Chapter 30: The CHAT Layer

The CHAT layer is where governance becomes visible to the end user. Every element on a chat surface — the accent color, the disclaimer, the citation links, the routing indicators — traces back through the compilation pipeline to a governed source file. One SCSS partial (`_CHAT.scss`) styles every chat agent in the fleet. One layout (`CUSTOM.html`) renders them all. The only visual property that varies per agent is the accent color, and even that is compiled from CANON.json.

33.1. 30.1 CUSTOM Layout

Chat surfaces use the CUSTOM layout, styled exclusively by `_CHAT.scss` — no external CSS [24].

33.2. 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 [2425].

33.3. 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` [25].

33.4. 30.4 _CHAT.scss Anatomy

All chat surface styling lives in one partial — no external CSS, no inline styles, no framework CSS:

```
// _CHAT.scss — governs all chat surfaces
// Layer: 11 (between _TALK and _POST in the 23-partial order)

// ── Container ──────────────────────────────────────────
.chat-container {
  display: flex;
  flex-direction: column;
  height: 100vh;
  max-width: var(--chat-max-width, 768px);
  margin: 0 auto;
  background: var(--bg);
  color: var(--fg);
  font-family: var(--font-family);
  font-size: var(--font-md);
}

// ── Header ─────────────────────────────────────────────
.chat-header {
  display: flex;
  align-items: center;
  padding: var(--space-sm) var(--space-md);
  border-bottom: 1px solid var(--dim);
  background: var(--bg-elevated);

  .chat-header__brand {
    font-weight: 700;
    color: var(--accent);
    font-size: var(--font-lg);
```

```
  }

  .chat-header__scope {
    font-size: var(--font-sm);
    color: var(--dim);
    margin-left: var(--space-sm);
  }

  .chat-header__score {
    margin-left: auto;
    font-variant-numeric: tabular-nums;
    color: var(--status-ok);
  }
}

// ── Message Thread ──────────────────────────────────
.chat-thread {
  flex: 1;
  overflow-y: auto;
  padding: var(--space-md);
  scroll-behavior: smooth;
}

// ── Messages ────────────────────────────────────────
.chat-message {
  margin-bottom: var(--space-md);
  padding: var(--space-sm) var(--space-md);
  border-radius: var(--radius-md);
  line-height: 1.6;
  max-width: 85%;

  &--user {
    background: var(--accent);
    color: var(--bg);
    margin-left: auto;
    border-bottom-right-radius: var(--radius-xs);
  }

  &--agent {
    background: var(--bg-elevated);
    color: var(--fg);
    margin-right: auto;
    border-bottom-left-radius: var(--radius-xs);
  }
```

```
  &__citation {
    font-size: var(--font-sm);
    color: var(--accent);
    cursor: pointer;
    text-decoration: underline;

    &:hover {
      color: var(--fg);
    }
  }

  &__timestamp {
    font-size: var(--font-xs);
    color: var(--dim);
    margin-top: var(--space-xs);
  }
}

// ── Disclaimer ──────────────────────────────────────
.chat-disclaimer {
  padding: var(--space-sm) var(--space-md);
  background: var(--bg-elevated);
  border-left: 3px solid var(--status-warn);
  font-size: var(--font-sm);
  color: var(--dim);
  margin: var(--space-md);
  border-radius: var(--radius-sm);
}

// ── Input ───────────────────────────────────────────
.chat-input {
  display: flex;
  padding: var(--space-sm) var(--space-md);
  border-top: 1px solid var(--dim);
  background: var(--bg);

  &__field {
    flex: 1;
    padding: var(--space-sm) var(--space-md);
    border: 1px solid var(--dim);
    border-radius: var(--radius-md);
    background: var(--bg-elevated);
    color: var(--fg);
    font-size: var(--font-md);
```

```
    font-family: var(--font-family);
    resize: none;

    &:focus {
      outline: none;
      border-color: var(--accent);
      box-shadow: 0 0 0 2px rgba(var(--accent-rgb), 0.2);
    }
  }

  &__send {
    margin-left: var(--space-sm);
    padding: var(--space-sm) var(--space-md);
    background: var(--accent);
    color: var(--bg);
    border: none;
    border-radius: var(--radius-md);
    cursor: pointer;
    font-weight: 600;

    &:hover { opacity: 0.9; }
    &:disabled { opacity: 0.4; cursor: not-allowed; }
  }
}

// ── Routing Indicator ──────────────────────────────
.chat-routing {
  padding: var(--space-xs) var(--space-md);
  background: var(--bg-elevated);
  border-left: 3px solid var(--accent);
  font-size: var(--font-sm);
  color: var(--dim);
  margin: var(--space-sm) var(--space-md);

  &__from { color: var(--dim); }
  &__to { color: var(--accent); font-weight: 600; }
}

// ── Responsive ─────────────────────────────────────
@media (max-width: 640px) {
  .chat-container { max-width: 100%; }
  .chat-message { max-width: 95%; }
  .chat-header { padding: var(--space-xs) var(--space-sm); }
}
```

```
// ── Accessibility ──────────────────────────────────────
@media (prefers-reduced-motion: reduce) {
  .chat-thread { scroll-behavior: auto; }
}

@media (prefers-contrast: high) {
  .chat-message--agent { border: 1px solid var(--fg); }
  .chat-message--user { border: 1px solid var(--bg); }
}
```

Every token (`--space-md`, `--accent`, `--font-md`, etc.) is defined in `_TOKENS.scss`. `_CHAT.scss` never contains literal values — only token references [2425].

33.5. 30.5 Accent Governance

Each chat agent has its own accent color. The accent is governed — set in the scope's DESIGN.md or CANON.md and compiled into CANON.json:

Agent	Accent	CSS Variable	Clinical Rationale	Live
MammoChat	`#ec4899` (pink)	`--accent`	Breast cancer awareness	mammo.chat
OncoChat	`#8b5cf6` (purple)	`--accent`	Oncology purple ribbon	oncochat.hadleylab.org
OmicsChat	`#06b6d4` (cyan)	`--accent`	Genomics/science blue-green	—
MedChat	`#3b82f6` (blue)	`--accent`	Clinical medicine blue	medchat.hadleylab.org
LawChat	`#64748b` (slate)	`--accent`	Legal gravitas	lawchat.hadleylab.org
FinChat	`#10b981` (emerald)	`--accent`	Finance green	finchat.hadleylab.org
Realty	`#f59e0b` (amber)	`--accent`	Real estate gold	—
Runner	`#f97316` (orange)	`--accent`	Operations energy	gorunner.pro

The accent color is injected at runtime via a `<style>` tag in the page `<head>`:

```
<!-- Compiled from CANON.json brand.accent -->
<style>
  :root {
    --accent: #ec4899;
    --accent-rgb: 236, 72, 153;
```

```
  }
</style>
```

Accent is the only visual property that varies per agent. Layout, spacing, typography, and message styling are shared via `_CHAT.scss` — one stylesheet, many agents [24].

33.6. 30.6 The CUSTOM Layout

Chat surfaces use the `CUSTOM.html` layout. This layout strips everything except the chat interface:

```
<!-- _layouts/CUSTOM.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  {% include HEAD.html %}
  <style>
    :root {
      --accent: {{ page.accent | default: site.accent }};
    }
  </style>
</head>
<body class="custom-layout">
  {% if page.disclaimer %}
    <div class="chat-disclaimer">{{ page.disclaimer }}</div>
  {% endif %}

  <div class="chat-container">
    <div class="chat-header">
      <span class="chat-header__brand">{{ page.brand }}</span>
      <span class="chat-header__scope">{{ page.scope }}</span>
      <span class="chat-header__score">{{ page.score }}/255</span>
    </div>

    <div class="chat-thread" id="chat-thread">
      <!-- Messages rendered by talk.js -->
    </div>

    <div class="chat-input">
      <textarea class="chat-input__field"
                id="chat-input"
                placeholder="Ask a question..."
                rows="1"></textarea>
```

```
      <button class="chat-input__send" id="chat-send">Send</button>
    </div>
  </div>

  {% include SCRIPTS.html %}
  <script src="/assets/js/talk.js"></script>
</body>
</html>
```

No navigation, no sidebar, no footer — CUSTOM is a full-screen chat interface. The header shows brand, scope path, and governance score. The disclaimer appears above the chat container, always visible, never dismissible [25].

33.7. 30.7 Message Rendering

Messages are rendered by `talk.js`. The rendering pipeline:

```
// talk.js — message rendering (simplified)
function renderMessage(message) {
  const el = document.createElement('div');
  el.className = `chat-message chat-message--${message.role}`;

  // Render markdown content
  el.innerHTML = marked.parse(message.content);

  // Add citations
  const citations = extractCitations(message.content);
  if (citations.length > 0) {
    const citationEl = document.createElement('div');
    citationEl.className = 'chat-message__citations';
    citations.forEach(c => {
      const link = document.createElement('span');
      link.className = 'chat-message__citation';
      link.textContent = c;
      link.onclick = () => showCitationDetail(c);
      citationEl.appendChild(link);
    });
    el.appendChild(citationEl);
  }

  // Add timestamp
  const ts = document.createElement('div');
  ts.className = 'chat-message__timestamp';
```

```
  ts.textContent = new Date(message.timestamp).toLocaleTimeString();
  el.appendChild(ts);

  document.getElementById('chat-thread').appendChild(el);
  el.scrollIntoView({ behavior: 'smooth' });
}
```

Citations in agent responses are rendered as clickable links. Clicking a citation shows the evidence source detail [21].

33.8. 30.8 Chat Component Architecture

The chat interface is composed of five structural components:

```
┌──────────────────────────────────────┐
│ .chat-header                         │  Brand + scope + score
├──────────────────────────────────────┤
│ .chat-disclaimer                     │  Always visible, non-dismissible
├──────────────────────────────────────┤
│                                      │
│ .chat-thread                         │  Scrollable message area
│   .chat-message--agent               │    Agent messages (left-aligned)
│   .chat-message--user                │    User messages (right-aligned)
│   .chat-routing                      │    Routing indicators
│                                      │
├──────────────────────────────────────┤
│ .chat-input                          │  Input field + send button
│   .chat-input__field                 │
│   .chat-input__send                  │
└──────────────────────────────────────┘
```

Each component maps to a CSS class in `_CHAT.scss`. No component has its own stylesheet. No component uses inline styles. The architecture is flat — no nested component trees, no shadow DOM, no CSS modules. One partial, one namespace [25].

33.9. 30.9 Routing Indicators

When the TALK service routes a query from one agent to another (Chapter 25), the chat interface displays a routing indicator:

```
<div class="chat-routing">
  <span class="chat-routing__from">MammoChat</span>
  →
  <span class="chat-routing__to">OncoChat</span>
  <span class="chat-routing__reason">Query matches staging pathway</span>
</div>
```

The routing indicator is styled by `_CHAT.scss`, with the left border in the destination agent's accent color. It is an audit element — non-interactive, showing the user that the conversation has been routed to a different agent.

33.10. 30.10 Accessibility Requirements

All chat components meet WCAG AA:

Requirement	Implementation	Validation
Color contrast 4.5:1	Token values tested against `--bg`	`magic validate --wcag`
Keyboard navigation	Tab order: input send thread	Manual test
Screen reader	ARIA roles on all components	`aria-role="log"` on thread
Reduced motion	`prefers-reduced-motion` media query	No scroll animation
High contrast	`prefers-contrast` media query	Border fallback on messages
Focus indicator	Accent-colored focus ring on input	`:focus` with box-shadow
Text scaling	All font sizes in `rem`	Scales with browser zoom

```
# Validate WCAG compliance
magic validate --wcag
```

```
WCAG VALIDATION: CHAT surfaces
──────────────────────────────────────────────────
Contrast (AA): PASS (all text > 4.5:1 against --bg)
Keyboard:      PASS (tab order verified)
ARIA:          PASS (roles on all interactive elements)
Motion:        PASS (reduced-motion query present)
──────────────────────────────────────────────────
Result: PASS
```

33.11. 30.11 Clinical Chat Patterns

Clinical chat surfaces have additional requirements beyond standard web chat:

Pattern	Requirement	Implementation
Disclaimer first	Disclaimer must appear before any clinical content	`.chat-disclaimer` always rendered, never dismissible
Citation linking	Every clinical claim must link to evidence	`.chat-message__citation` rendered from INTEL
No diagnostic language	Agent must not use diagnostic language	Enforced by systemPrompt constraints
Session isolation	No patient data persists between sessions	Client-side only, no server state
Audit trail	Every message pair is ledgered	LEDGER event on each exchange
Timeout	Sessions expire after 30 minutes of inactivity	Client-side timer, clear on timeout
Rate limiting	Maximum 60 queries per hour per user	Server-side rate limit via API

In practice: a radiologist opens the chat surface and sees the disclaimer immediately. She asks about BI-RADS 4B and gets the ACR definition with a clickable citation. Every exchange is ledgered. After 30 minutes of inactivity the session expires, data clears, and a fresh session begins [21].

33.12. 30.12 Multi-Layout Comparison

Each of the seven layouts serves a different content type. The CHAT layer integrates with all layouts via the `talk: inline` frontmatter directive:

Layout	Chat Integration	Use Case
`default.html`	Floating widget (bottom-right)	Standard pages with optional chat
`DECK.html`	Slide-level Q&A panel	Presentation with audience interaction
`CUSTOM.html`	Full-screen chat	Dedicated chat interface (MammoChat)
`post.html`	Inline contextual agent	Blog post with per-section Q&A
`paper.html`	Citation-aware agent	Research paper with evidence Q&A
`book.html`	Chapter-scoped agent	Book chapter with contextual Q&A

The `talk: inline` directive in frontmatter activates the chat integration for any layout. The integration method depends on the layout type. `_CHAT.scss` provides the styling for all integration methods [25]. ## 30.13 Chat State Management

Chat state lives entirely client-side — no server-side sessions, no database persistence. This is a HIPAA-driven architectural decision: no patient data touches the server beyond the immediate request.

```
// talk.js — client-side state management
const chatState = {
  scope: null,           // Current agent scope (from CANON.json)
  messages: [],          // Message history (client-only)
  systemPrompt: null,    // Compiled from CANON.json
  sessionId: null,       // UUID per session
  startedAt: null,       // Session start timestamp
  lastActivity: null,    // Last user interaction
  routingHistory: [],    // Agent routing trail

  init(canonJson) {
    this.scope = canonJson.scope;
    this.systemPrompt = canonJson.systemPrompt;
    this.sessionId = crypto.randomUUID();
    this.startedAt = Date.now();
    this.lastActivity = Date.now();
  },

  addMessage(role, content, citations) {
    this.messages.push({
      role,
      content,
      citations: citations || [],
      timestamp: Date.now()
    });
    this.lastActivity = Date.now();
  },

  isExpired() {
    const TIMEOUT_MS = 30 * 60 * 1000; // 30 minutes
    return Date.now() - this.lastActivity > TIMEOUT_MS;
  },

  clear() {
    this.messages = [];
    this.routingHistory = [];
    this.sessionId = crypto.randomUUID();
    this.startedAt = Date.now();
  }
};
```

When the session expires, `chatState.clear()` wipes messages and regenerates the session ID. PHI never

accumulates on the client [21].

33.13. 30.14 Chat Event Ledgering

Every chat exchange is recorded as a LEDGER event:

```
{
  "event": "TALK:EXCHANGE",
  "scope": "SERVICES/TALK/MAMMOCHAT",
  "session_id": "uuid-abc-123",
  "timestamp": "2026-03-10T14:22:00Z",
  "query_length": 47,
  "response_length": 312,
  "citations_count": 2,
  "routing": null,
  "constraint_violations": 0,
  "response_time_ms": 823,
  "hash": "sha256:abc123..."
}
```

The LEDGER event contains metadata only — no query or response text. Content stays client-side for HIPAA compliance. The LEDGER records *that* a conversation happened, how long it took, and how many citations were included, without recording clinical content [1421].

33.14. 30.15 Chat Performance Budget

Chat surfaces have a strict performance budget:

Metric	Budget	Enforcement
Time to First Message	< 500ms	Preload CANON.json, lazy-load talk.js
Agent Response Time (p95)	< 2s	LLM streaming, token budget limits
Input Latency	< 50ms	No debounce on typing
Scroll Performance	60fps	Virtual scroll for long threads
CSS Bundle Size	< 20KB	_CHAT.scss minified
JS Bundle Size	< 50KB	talk.js + marked.js
Total Page Weight	< 200KB	CUSTOM layout minimal

```
# Validate performance budget
magic validate --performance SERVICES/TALK/MAMMOCHAT
```

The budget ensures clinical chat surfaces remain responsive on hospital network infrastructure, which is often slower than consumer internet. A radiologist behind a hospital VPN must have the same experience as a developer on fiber [2425].

33.15. 30.16 Chat Testing

Chat surfaces have dedicated test suites. The test framework validates both the UI and the agent:

```
# UI tests — verify layout, styling, accessibility
magic test --chat-ui SERVICES/TALK/MAMMOCHAT

# Agent tests — verify responses, constraints, citations
magic test --chat-agent SERVICES/TALK/MAMMOCHAT

# Integration tests — verify end-to-end flow
magic test --chat-integration SERVICES/TALK/MAMMOCHAT
```

UI test checklist:

Test	Assertion	Priority
Container renders	`.chat-container` exists	P0
Header shows brand	`.chat-header__brand` text matches	P0
Disclaimer visible	`.chat-disclaimer` is visible	P0
Input accepts text	`.chat-input__field` is editable	P0
Send button works	Click `.chat-input__send` triggers query	P0
Messages render	`.chat-message` elements created	P0
User messages right-aligned	`.chat-message--user` has `margin-left: auto`	P1
Agent messages left-aligned	`.chat-message--agent` has `margin-right: auto`	P1
Citations clickable	`.chat-message__citation` has click handler	P1
Responsive layout	Container fills viewport on mobile	P1
Accessibility	ARIA roles present, keyboard nav works	P1

33.16. 30.17 Chat Internationalization

Chat surfaces support RTL (right-to-left) languages. The `_CHAT.scss` partial includes RTL support:

```scss
// _CHAT.scss — RTL support
[dir="rtl"] {
  .chat-message--user {
    margin-left: 0;
    margin-right: auto;
    border-bottom-right-radius: var(--radius-md);
    border-bottom-left-radius: var(--radius-xs);
  }

  .chat-message--agent {
    margin-right: 0;
    margin-left: auto;
    border-bottom-left-radius: var(--radius-md);
    border-bottom-right-radius: var(--radius-xs);
  }

  .chat-input__send {
    margin-left: 0;
    margin-right: var(--space-sm);
  }

  .chat-routing__from { direction: rtl; }
  .chat-routing__to { direction: rtl; }
}
```

RTL is activated by the `dir` attribute on `<html>`:

```html
<html lang="ar" dir="rtl">
```

The chat layout mirrors for Arabic, Hebrew, and other RTL languages. User messages appear on the left. Agent messages appear on the right. The input send button moves to the left side. All managed by CSS — no JavaScript changes required [2425].

33.17. 30.18 Chat Error States

The chat interface handles four error states, each with a governed visual treatment:

Error State	Visual	Message	Recovery
Agent unavailable	Red border on container	“Service temporarily unavailable”	Auto-retry after 5s

Error State	Visual	Message	Recovery
Rate limited	Yellow disclaimer	“Please wait before sending another query”	Timer countdown
Session expired	Dimmed thread, fresh input	“Session expired. Starting fresh.”	Auto-clear on input
Network error	Red flash on send button	“Network error. Check your connection.”	Manual retry

```scss
// _CHAT.scss — error states
.chat-container--error {
  border: 2px solid var(--status-error);
}

.chat-container--rate-limited .chat-input__send {
  opacity: 0.4;
  cursor: not-allowed;
}

.chat-container--expired .chat-thread {
  opacity: 0.5;
}
```

Every error state is styled by `_CHAT.scss`. No inline styles. No JavaScript-injected CSS. The error states are token-governed — using `--status-error`, `--status-warn` from `_TOKENS.scss` [24].

33.18. 30.19 Chat Layer Summary

Every pixel on the chat interface traces to a governed source:

```
User sees MammoChat pink accent (#ec4899)
  ← injected by CUSTOM.html from CANON.json brand.accent
    ← compiled by magic compile from CANON.md
      ← governed by SERVICES/TALK/MAMMOCHAT/CANON.md

User reads disclaimer
  ← rendered by CUSTOM.html from CANON.json disclaimer
    ← compiled from CANON.md constraints
      ← governed by SERVICES/TALK/MAMMOCHAT/CANON.md
```

```
User clicks citation [ACR BI-RADS 5th Edition]
  ← rendered by talk.js from agent response
    ← agent cites from systemPrompt evidence chain
      ← compiled from INTEL.md layer 2
        ← governed by SERVICES/TALK/MAMMOCHAT/INTEL.md

User sees message styling
  ← styled by _CHAT.scss
    ← tokens from _TOKENS.scss
      ← governed by SERVICES/DESIGN/CANON.md
```

The CHAT layer is not a frontend — it is the terminal surface of the governance tree. Change the governance, rebuild, and the chat surface changes. It cannot diverge from governance because it is compiled from governance. For the agent pipeline that feeds content into these surfaces, see Chapter 25 (Contextual Agents). For the service-level architecture of TALK, see Chapter 11. For the design tokens that govern visual styling, see Chapter 29 (Design Tokens) [242521].

33.19. 30.20 CHAT and the Seven Layouts — Wiring Detail

Each layout wires CHAT differently. The wiring is in the layout HTML:

```
<!-- default.html — floating widget -->
<div class="chat-widget chat-widget--collapsed" id="chat-widget">
  <button class="chat-widget__trigger" onclick="toggleChat()">Ask</button>
  <div class="chat-widget__panel">
    {% include TALK.html %}
  </div>
</div>

<!-- CUSTOM.html — full screen (shown in 30.6) -->
<!-- No wrapping. Chat IS the page. -->

<!-- book.html — inline per-chapter -->
<div class="book-chapter">
  {{ content }}
  {% if page.talk %}
    <div class="chat-inline">
      {% include TALK.html scope=page.scope %}
    </div>
  {% endif %}
</div>
```

The `TALK.html` include is the shared component. Every layout includes it differently — floating, full-screen,

inline. The styling adapts via `_CHAT.scss` container queries [25].

33.20. 30.21 CHAT Layer Constraints

The CHAT layer enforces visual constraints that map to governance constraints:

Governance Constraint	Visual Implementation
Always show disclaimer	`.chat-disclaimer` rendered before first message, non-dismissible
Never impersonate clinician	Agent messages styled differently from user messages
Always cite evidence	Citations rendered as clickable links with source detail
Show routing transparency	`.chat-routing` indicator appears on cross-agent routes
Display governance score	Score shown in header: `255/255`
Session isolation	No persistent state indicator in header

These are not suggestions — they are compiled from CANON.md into the layout. The disclaimer cannot be removed without changing the governance contract. The routing indicator cannot be hidden without changing CUSTOM.html. The governance score cannot be faked because it is read from CANON.json at compile time.

Governance compiles all the way down to the pixel [242521].

Chapter 34

Chapter 31: The GALAXY Visualization

The GALAXY is the governance tree made visible — explore it live at hadleylab.org. It renders every scope as a node, every inheritance relationship as an edge, and every MAGIC dimension as a segment of a compliance ring around each node. A fleet with all-green edges and complete rings is healthy. A fleet with broken rings and dim edges has governance debt — and you can see exactly where. The visualization reads from `galaxy.json`, compiled by `build-galaxy-json` from the GOV tree, and updates on every build.

34.1. 31.1 Data Source

GALAXY renders from `galaxy.json`, compiled by `build-galaxy-json` (phase 01-galaxy) and enriched by `enrich-galaxy` (phase 01a-galaxy-enrich) from the GOV tree. The galaxy contains 284 nodes across 5 kinds (ORG, SERVICE, SCOPE, VERTICAL, USER), connected by 340 edges across 4 kinds (PARENT, INHERITS, CLUSTER, DOMAINS) [18].

34.2. 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

34.3. 31.3 Compliance Ring

8 arc segments = 8 MAGIC dimensions. Filled = present. Gap = missing. 255 = complete ring, full glow [18].

34.4. 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

34.5. 31.5 INTEL Flow

Edges where INTEL.md exists pulse with green particles. Edges without INTEL are static and dim. Missing INTEL = expression question unanswered = blocked at AGENT tier [18].

34.6. 31.6 Brand Marks

CANONIC = ∩ (U+2229). HADLEYLAB = ⌷ (U+2632) [18].

34.7. 31.7 galaxy.json Structure

The `galaxy.json` file is the data source for GALAXY. It is compiled by `build-galaxy-json` from the GOV tree and enriched by `enrich-galaxy` with wallet, session, and learning data. The structure:

```
{
  "nodes": [
    {
      "id": "hadleylab-canonic/SERVICES/TALK/MAMMOCHAT",
      "label": "MAMMOCHAT",
      "kind": "SERVICE",
```

```
      "parent": "hadleylab-canonic/SERVICES/TALK",
      "bits": 255,
      "tier": "MAGIC",
      "intel": true,
      "accent": "#ec4899",
      "brand": "MammoChat",
      "talk_sessions": 24,
      "learning_count": 12,
      "wallet_balance": 3200,
      "score": 255
    }
  ],
  "edges": [
    {
      "from": "hadleylab-canonic/SERVICES/TALK/MAMMOCHAT",
      "to": "hadleylab-canonic/SERVICES/TALK",
      "kind": "PARENT"
    },
    {
      "from": "hadleylab-canonic/SERVICES/TALK/MAMMOCHAT",
      "to": "canonic-canonic/FOUNDATION",
      "kind": "INHERITS"
    }
  ],
  "stats": {
    "total": 284,
    "avg_bits": 251,
    "below_magic": 4,
    "intel_coverage": 140,
    "total_coin": 149566,
    "total_events": 687,
    "total_sessions": 136,
    "active_channels": 5
  }
}
```


Every scope is a node. Four edge kinds connect the graph: PARENT edges form the GOV tree spine, INHERITS edges cross axiomatic boundaries, CLUSTER edges group related scopes, and DOMAINS edges link verticals to their services. The `enrich-galaxy` phase backfills economic and operational data (wallet, sessions, learning) into each node after the topology is established [18].

34.8. 31.8 The vis-network Configuration

GALAXY uses vis-network.js for graph rendering. The configuration maps scope types to visual elements:

```
// galaxy.js — vis-network configuration
const options = {
  nodes: {
    font: {
      color: '#e5e5e5',
      size: 14,
      face: 'Inter'
    },
    borderWidth: 2,
    shadow: true
  },
  edges: {
    color: { color: '#333', highlight: '#00ff88' },
    width: 1,
    smooth: {
      type: 'cubicBezier',
      forceDirection: 'vertical',
      roundness: 0.4
    },
    arrows: { to: { enabled: true, scaleFactor: 0.5 } }
  },
  physics: {
    enabled: true,
    solver: 'forceAtlas2Based',
    forceAtlas2Based: {
      gravitationalConstant: -50,
      centralGravity: 0.01,
      springLength: 100,
      springConstant: 0.08,
      damping: 0.4
    },
    stabilization: {
      iterations: 150,
      fit: true
    }
  },
  interaction: {
    hover: true,
    tooltipDelay: 200,
    zoomView: true,
```

```
    dragView: true
  }
};
```

The physics solver (`forceAtlas2Based`) produces a force-directed layout where child scopes orbit their parents. SERVICE scopes form tight clusters; PRINCIPAL scopes act as gravitational anchors [18].

34.9. 31.9 Node Rendering

Each node is rendered based on its type and score. The rendering pipeline:

```
// galaxy.js — node rendering
function renderNode(scope) {
  const tierConfig = getTierConfig(scope.score);

  return {
    id: scope.id,
    label: scope.label,
    shape: getShape(scope.type),
    color: {
      background: tierConfig.color,
      border: tierConfig.color,
      highlight: {
        background: tierConfig.color,
        border: '#ffffff'
      }
    },
    shadow: {
      enabled: true,
      color: tierConfig.color,
      size: tierConfig.glow,
      x: 0,
      y: 0
    },
    font: {
      color: '#e5e5e5',
      size: scope.type === 'ORG' ? 18 : 14
    },
    size: getSize(scope.type),
    title: buildTooltip(scope)
  };
}
```

```
function getShape(type) {
  const shapes = {
    'ORG': 'star',
    'PRINCIPAL': 'diamond',
    'SERVICE': 'dot',
    'SCOPE': 'dot',
    'VERTICAL': 'triangle',
    'USER': 'box'
  };
  return shapes[type] || 'dot';
}

function getSize(type) {
  const sizes = {
    'ORG': 30,
    'PRINCIPAL': 25,
    'SERVICE': 20,
    'SCOPE': 15,
    'VERTICAL': 20,
    'USER': 12
  };
  return sizes[type] || 15;
}

function getTierConfig(score) {
  if (score === 255) return { color: '#00ff88', glow: 20 };
  if (score >= 127) return { color: '#2997ff', glow: 12 };
  if (score >= 63)  return { color: '#bf5af2', glow: 8 };
  if (score >= 43)  return { color: '#ff9f0a', glow: 4 };
  if (score >= 35)  return { color: '#fbbf24', glow: 2 };
  return { color: '#ff453a', glow: 0 };
}
```

34.10. 31.10 Compliance Ring Rendering

The compliance ring is the 8-segment arc around each node. Each segment represents one MAGIC dimension:

```
// galaxy.js — compliance ring rendering (canvas afterDraw)
function drawComplianceRing(ctx, node, scope) {
  const dimensions = ['D', 'E', 'S', 'O', 'T', 'LANG', 'ECON', 'L'];
  const segmentAngle = (2 * Math.PI) / 8;
```

```
  const radius = node.size + 8;
  const lineWidth = 3;

  dimensions.forEach((dim, i) => {
    const startAngle = i * segmentAngle - Math.PI / 2;
    const endAngle = startAngle + segmentAngle * 0.85; // gap between segments

    ctx.beginPath();
    ctx.arc(node.x, node.y, radius, startAngle, endAngle);
    ctx.strokeStyle = scope.dimensions[dim]
      ? getTierConfig(scope.score).color
      : '#333';
    ctx.lineWidth = scope.dimensions[dim] ? lineWidth : 1;
    ctx.stroke();
  });
}
```

The compliance ring visually encodes the 8-bit score. All 8 segments lit = 255. Seven lit with one gap = 127-254. You can see which dimensions are present and which are missing at a glance, without hovering or clicking [18].

34.11. 31.11 Edge Animation

Edges where both connected scopes have INTEL.md pulse with green particles. The animation:

```
// galaxy.js — edge animation
function animateEdges(network, scopes) {
  const edges = network.body.data.edges.get();

  edges.forEach(edge => {
    const fromScope = scopes.find(s => s.id === edge.from);
    const toScope = scopes.find(s => s.id === edge.to);

    if (fromScope.intel && toScope.intel) {
      // Both sides have INTEL — pulse green
      edge.color = { color: '#00ff88', opacity: 0.6 };
      edge.width = 2;
      edge.dashes = false;
    } else {
      // Missing INTEL — static and dim
      edge.color = { color: '#333', opacity: 0.3 };
      edge.width = 1;
      edge.dashes = [5, 10];
```

```
    }
  });

  network.body.data.edges.update(edges);
}
```

INTEL flow made visible: when a scope adds INTEL.md, its edges transition from dim/dashed to bright/pulsing. The GALAXY updates on every build [18].

34.12. 31.12 Tooltip Content

Hovering over a node shows a tooltip with scope details:

```
// galaxy.js — tooltip builder
function buildTooltip(scope) {
  return `
    <div class="galaxy-tooltip">
      <div class="galaxy-tooltip__name">${scope.label}</div>
      <div class="galaxy-tooltip__path">${scope.id}</div>
      <div class="galaxy-tooltip__score">
        Score: ${scope.score}/255 (${scope.tier})
      </div>
      <div class="galaxy-tooltip__dimensions">
        ${['D','E','S','O','T','LANG','ECON','L'].map(d =>
          `<span class="${scope.dimensions[d] ? 'active' : 'missing'}">${d}</span>`
        ).join(' ')}
      </div>
      <div class="galaxy-tooltip__intel">
        INTEL: ${scope.intel ? 'INDEXED' : 'MISSING'}
      </div>
      ${scope.talk ? '<div class="galaxy-tooltip__talk">TALK: Active</div>' : ''}
      ${scope.accent ? `<div class="galaxy-tooltip__accent" style="color:${scope.accent}">Accent: ${scope.
    </div>
  `;
}
```

34.13. 31.13 Container-Driven Topology

GALAXY renders differently based on its container. The visualization adapts to the available space:

Container	Behavior	Use Case
Full page (`/GALAXY/`)	Force-directed layout, full interaction	Fleet dashboard
Dashboard widget	Fixed layout, hover only	Principal dashboard
Fleet card	Static snapshot, no interaction	SHOP product card
DECK slide	Centered layout, presentation mode	Investor pitch

```
// galaxy.js — container detection
function initGalaxy(containerId) {
  const container = document.getElementById(containerId);
  const width = container.offsetWidth;
  const height = container.offsetHeight;

  let mode;
  if (width > 800 && height > 600) {
    mode = 'full';        // full interactive
  } else if (width > 400) {
    mode = 'widget';      // hover only
  } else {
    mode = 'thumbnail';   // static snapshot
  }

  const options = getOptionsForMode(mode);
  const network = new vis.Network(container, data, options);

  if (mode === 'thumbnail') {
    network.setOptions({ interaction: { dragNodes: false, zoomView: false } });
    network.fit();
  }
}
```

34.14. 31.14 GALAXY and the Build Pipeline

`build-galaxy-json` compiles the GALAXY data during the build pipeline:

```
# Phase 01-galaxy: build galaxy.json from GOV tree
bin/build-galaxy-json

# Phase 01a-galaxy-enrich: backfill wallet, session, learning data
bin/enrich-galaxy
```

```
# Output: MAGIC/galaxy.json
# Contains: all nodes, edges, stats
```

The compilation steps:

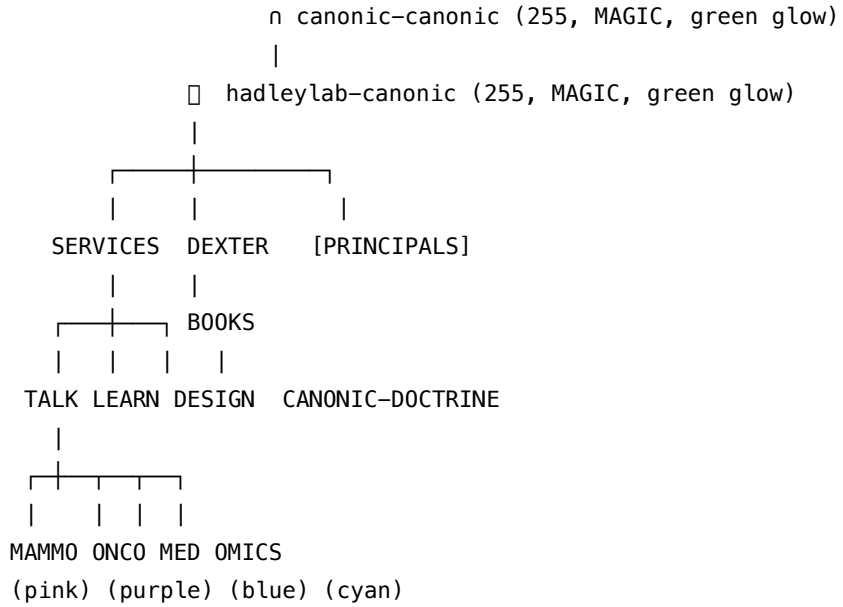
```
Phase 01-galaxy (build-galaxy-json):
1. Walk GOV tree, find all CANON.md files
2. Parse each CANON.md (bits, tier, kind)
3. Check for INTEL.md (sets intel flag)
4. Build edge list (PARENT, INHERITS, CLUSTER, DOMAINS)
5. Calculate stats (total nodes, avg bits, intel coverage)
6. Write galaxy.json to MAGIC/

Phase 01a-galaxy-enrich (enrich-galaxy):
7. Load galaxy.json
8. Backfill wallet balances from COIN ledgers
9. Backfill talk sessions and learning counts
10. Write enriched galaxy.json
```

At page load, GALAXY reads `galaxy.json` and renders the fleet topology. Every build produces a fresh file, so the visualization is always current — it reflects the fleet as compiled, not as remembered [18].

34.15. 31.15 Clinical GALAXY Example

A hospital IT director opens the GALAXY page and sees:



Every node has a compliance ring; every INTEL-enabled edge pulses green. All 73 scopes at 255, all edges pulsing — the fleet is healthy. If any scope drifts, its ring breaks, its glow dims, and its edges go static. The GALAXY is the fleet's vital signs monitor. For the MONITORING service that feeds real-time data into GALAXY views, see Chapter 22. For the federation architecture that connects multiple organizations into a single GALAXY, see Chapter 9 [18].

34.16. 31.16 Interaction Patterns

Action	Result	Use Case
Click node	Navigate to scope page	Inspect scope details
Hover node	Show tooltip with dimensions	Quick score check
Double-click node	Focus + zoom to node and children	Drill into subtree
Drag node	Reposition in force layout	Custom arrangement
Scroll	Zoom in/out	Overview vs detail
Right-click	Context menu: validate, heal, open CANON.md	Governance actions

Every governance-modifying interaction (heal, validate) is a LEDGER event. The GALAXY is not just a visualization — it is a governance interface [18]. ## 31.17 GALAXY Performance

GALAXY must render fleets with 100+ scopes without frame drops:

Fleet Size	Nodes	Edges	Render Time	FPS
Small (< 30 scopes)	< 30	< 50	< 100ms	60
Medium (30-100 scopes)	30-100	50-200	< 300ms	60
Large (100-500 scopes)	100-500	200-1000	< 1s	30+
Fleet (500+ scopes)	500+	1000+	< 3s	15+

For large fleets, GALAXY uses progressive rendering:

```
// galaxy.js — progressive rendering
function renderProgressive(scopes) {
  // Phase 1: Render ORG and PRINCIPAL nodes first (< 10)
  const anchors = scopes.filter(s => s.type === 'ORG' || s.type === 'PRINCIPAL');
  network.body.data.nodes.add(anchors.map(renderNode));

  // Phase 2: Render SERVICE nodes (< 50)
  requestAnimationFrame(() => {
    const services = scopes.filter(s => s.type === 'SERVICE');
```

```
    network.body.data.nodes.add(services.map(renderNode));
  });

  // Phase 3: Render remaining nodes
  requestAnimationFrame(() => {
    const remaining = scopes.filter(s =>
      s.type !== 'ORG' && s.type !== 'PRINCIPAL' && s.type !== 'SERVICE'
    );
    network.body.data.nodes.add(remaining.map(renderNode));
  });

  // Phase 4: Add edges and start physics
  requestAnimationFrame(() => {
    network.body.data.edges.add(edges);
    network.stabilize();
  });
}
```

34.17. 31.18 GALAXY Keyboard Navigation

GALAXY supports keyboard navigation for accessibility:

Key	Action
Tab	Focus next node
Shift+Tab	Focus previous node
Enter	Navigate to focused node's scope page
Space	Toggle tooltip on focused node
+/-	Zoom in/out
Arrow keys	Pan the viewport
Escape	Reset zoom to fit all nodes
/	Open search (filter nodes by name)

```
// galaxy.js — keyboard navigation
document.addEventListener('keydown', (e) => {
  switch(e.key) {
    case 'Tab':
      e.preventDefault();
      focusNextNode(e.shiftKey ? -1 : 1);
      break;
    case 'Enter':
      navigateToScope(focusedNode);
```

```
      break;
    case 'Escape':
      network.fit();
      break;
    case '/':
      e.preventDefault();
      openSearchOverlay();
      break;
  }
});
```

34.18. 31.19 GALAXY Search and Filter

GALAXY supports real-time search and filtering:

```
// galaxy.js — search and filter
function filterNodes(query) {
  const allNodes = network.body.data.nodes.get();
  allNodes.forEach(node => {
    const matches = node.label.toLowerCase().includes(query.toLowerCase());
    network.body.data.nodes.update({
      id: node.id,
      hidden: !matches,
      opacity: matches ? 1.0 : 0.1
    });
  });
}

// Filter by tier
function filterByTier(tier) {
  const tiers = { MAGIC: 255, AGENT: 127, ENTERPRISE: 63, BUSINESS: 43 };
  const threshold = tiers[tier] || 0;
  const allNodes = network.body.data.nodes.get();
  allNodes.forEach(node => {
    const scope = scopes.find(s => s.id === node.id);
    network.body.data.nodes.update({
      id: node.id,
      hidden: scope.score < threshold
    });
  });
}

// Filter by dimension
```

```
function filterByDimension(dimension) {
  const allNodes = network.body.data.nodes.get();
  allNodes.forEach(node => {
    const scope = scopes.find(s => s.id === node.id);
    network.body.data.nodes.update({
      id: node.id,
      hidden: !scope.dimensions[dimension]
    });
  });
}
```

Filter options: - **By name**: Type to filter nodes by label - **By tier**: Show only MAGIC, AGENT, ENTERPRISE, etc. - **By dimension**: Show only scopes with a specific dimension (e.g., show all scopes with INTEL) - **By type**: Show only SERVICE, BOOK, DEAL, etc.

Filtered views reveal structural patterns — filter by "INTEL missing" to see exactly which scopes need INTEL.md to advance, or by "MAGIC tier" to see the fleet's fully-governed core [18].

34.19. 31.20 GALAXY Export

GALAXY exports to multiple formats for reporting:

```
# Export current view as PNG
magic galaxy --export png > fleet-topology.png

# Export as SVG (vector, scalable)
magic galaxy --export svg > fleet-topology.svg

# Export as JSON (data only, for custom rendering)
magic galaxy --export json > fleet-topology.json

# Export as PDF (for board presentations)
magic galaxy --export pdf > fleet-topology.pdf
```

PNG/SVG exports capture the current view including filter state, zoom level, and node positions. JSON provides raw data for custom rendering. PDF is formatted for A4/Letter with title, legend, and fleet statistics [18].

34.20. 31.21 GALAXY Embedding

GALAXY can be embedded in any CANONIC page via a Liquid include:

```
<!-- In any Jekyll page -->
{% include GALAXY.html scope="SERVICES/TALK" mode="widget" %}
```

The include accepts parameters:

Parameter	Values	Default	Description
`scope`	Any scope path	Root	Which subtree to render
`mode`	`full`, `widget`, `thumbnail`	`full`	Rendering mode
`filter`	`all`, `magic`, `agent`, `intel`	`all`	Which nodes to show
`height`	CSS value	`600px`	Container height
`interactive`	`true`, `false`	`true`	Enable interaction

Examples:

```
<!-- Full fleet view on GALAXY page -->
{% include GALAXY.html mode="full" height="100vh" %}

<!-- Service subtree widget on dashboard -->
{% include GALAXY.html scope="SERVICES/TALK" mode="widget" height="400px" %}

<!-- Thumbnail on SHOP product card -->
{% include GALAXY.html scope="SERVICES/TALK/MAMMOCHAT" mode="thumbnail" height="200px" interactive="false"



# Chapter 32: COIN and the WALLET

Event types. Supply ceiling. Conservation. This chapter defines the COIN primitive and the seven WALLET in

## 32.1 COIN = WORK

Forget everything you know about tokens, points, and cryptocurrency. COIN is none of those things. COIN is

Every COIN ever minted traces back to a `magic validate` invocation that measured a positive governance gr
```

.md files magic validate score delta gradient > 0 MINT:WORK COIN

```
COIN has no speculative value and no exchange rate divorced from work. One COIN equals one unit of governa
```

```
Compare this to traditional incentive structures:

| Traditional | CANONIC |
|-------------|---------|
| Hours billed → payment | Governance gradient → COIN |
| Subjective quality review → bonus | Score delta → deterministic MINT |
| Annual audit → compliance certificate | Continuous validation → real-time score |
| Budget allocation → department funding | COST_BASIS → WALLET balance |
| Insurance reimbursement → revenue | SPEND events → COIN circulation |

Picture a clinical informatics department governing 40 scopes: EHR integrations, FHIR endpoints, complianc

The identity of COIN across all contexts:
```

COIN governance work receipt COIN ≠ currency (no exchange rate independent of work) COIN ≠ token (no blockchain required) COIN ≠ point (no gamification decay) COIN = deterministic output of validate()

```
Run `magic wallet --info` on any scope to see the COIN provenance:

```bash
magic wallet --info hadleylab-canonic/SERVICES/EHR-INTEGRATION
# Scope: hadleylab-canonic/SERVICES/EHR-INTEGRATION
# Score: 255/255
# Total minted: 255 COIN
# Total debited: 0 COIN
# Net: 255 COIN
# Events: 4 MINT:WORK, 0 DEBIT:DRIFT
# First MINT: 2026-01-15T09:00:00Z (bootstrap, +35)
# Last MINT: 2026-02-28T16:30:00Z (close to 255, +31)
```

Every COIN has a birthday (the commit timestamp), a parent (the scope), and an identity (the committer). These three attributes make COIN fully traceable – and in regulated environments like healthcare, finance, and government, that traceability satisfies audit requirements no traditional incentive system can meet.

34.21. 32.2 Eight Circulation Events

#	Event	Direction	Mechanism
1	MINT:WORK	+	Governance gradient
2	MINT:SIGNUP	+	New user (500 COIN)
```

#	Event	Direction	Mechanism
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. The economy is closed by enumeration [12].

Event 1: MINT:WORK. The primary minting mechanism. When `magic validate` detects a positive gradient (new score > old score), the system mints COIN equal to the delta. The LEDGER entry records scope path, from-score, to-score, gradient, timestamp, and committer identity.

```
{
  "event": "MINT:WORK",
  "scope": "hadleylab-canonic/SERVICES/EHR-INTEGRATION",
  "from_score": 127,
  "to_score": 255,
  "gradient": 128,
  "amount": 128,
  "timestamp": "2026-03-10T14:23:00Z",
  "identity": "dr.chen@hadleylab.org",
  "commit": "a1b2c3d"
}
```

Event 2: MINT:SIGNUP. A new user receives 500 COIN upon identity verification. This is a one-time bootstrap — enough to purchase several products from the SHOP or transfer to collaborators. The 500 COIN comes from the TREASURY, not from thin air.

```
magic wallet --check new-user@example.com
# Balance: 500 COIN (MINT:SIGNUP)
# Source: TREASURY
```

Event 3: MINT:PYRAMID. When a user refers a new member who completes signup, both parties receive 500 COIN. One level only, one payout only. No multi-level complexity.

```
Referrer: +500 COIN (MINT:PYRAMID)
New user: +500 COIN (MINT:SIGNUP)
Treasury: −1000 COIN
```

Event 4: DEBIT:DRIFT. Score regression debits COIN. Delete a LEARNING.md, lose the LEARNING dimension, watch the score drop from 255 to 127 and 128 COIN evaporate from your WALLET. Building is hard. Destroying is easy but expensive.

```
DEBIT:DRIFT triggered:
  scope: hadleylab-canonic/SERVICES/FHIR-API
  from: 255 → to: 127
  gradient: -128
  debit: 128 COIN
  wallet_before: 2,340 COIN
  wallet_after: 2,212 COIN
```

Event 5: TRANSFER. Move COIN between WALLETS. A 5% fee applies. Transfer 100 COIN; recipient receives 95; 5 go to TREASURY. The fee prevents wash trading.

```
TRANSFER:
  from: alice@hadleylab.org
  to: bob@hadleylab.org
  amount: 100
  fee: 5 (5%)
  net: 95
  treasury_credit: 5
```

Event 6: SPEND. Purchase a product from the SHOP. The buyer's WALLET debits. The author's WALLET credits. SPEND events are the primary circulation mechanism — they move COIN from consumers to producers.

```
{
  "event": "SPEND",
  "buyer": "intern@hospital.org",
  "seller": "dr.chen@hadleylab.org",
  "product": "FHIR-Integration-Playbook",
  "amount": 255,
  "cost_basis": 255,
  "timestamp": "2026-03-10T15:00:00Z"
}
```

Event 7: SETTLE. Convert COIN to fiat currency. The off-ramp. COIN exits circulation, enters the SETTLED pool. Settlement requires identity verification (VITAE.md gate).

Event 8: CLOSE. Monthly reconciliation. The system verifies the conservation equation, reconciles all WALLET balances against the LEDGER, and produces an audit report.

```
magic wallet --close --month 2026-03
# Reconciling all WALLET balances...
# Conservation: PASS
# CLOSE event recorded in LEDGER.
```

34.22. 32.3 Supply Ceiling

```
SUPPLY\_CEILING = unique_scopes * 255
```

The ceiling is absolute. No scope can mint more than 255 COIN, and no mechanism exists to mint COIN without a scope. Total supply is bounded by the number of governed scopes.

Here is what that looks like for real organizations:

Organization	Scopes	Supply Ceiling
Solo practitioner (1 service)	6	1,530
Small clinic (5 services)	30	7,650
Department (20 services)	120	30,600
Hospital system (100 services)	600	153,000
Enterprise network (500 services)	3,000	765,000

Supply grows linearly with scope count – no exponential inflation, no monetary policy decisions. It is a pure function of governance breadth.

```
magic wallet --supply
# Active scopes:  200
# Supply ceiling:  51,000 COIN
# Minted to date: 38,250 COIN
# Remaining capacity: 12,750 COIN
# Utilization: 75.0%
```

Supply utilization tells you how much of the governance landscape has been claimed. At 100%, every scope sits at 255. At 0%, no governance work has been done. One number, one metric for organizational maturity.

34.23. 32.4 Conservation Equation

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

Verifiable from LEDGER chain at any time [12].

The four pools are exhaustive and mutually exclusive:

Pool	Description	Transitions
Treasury	System-held COIN not yet distributed	→ Circulation (MINT:SIGNUP, MINT:PYRAMID)
Circulation	COIN in active WALLETS	→ Treasury (TRANSFER fee), → Archived (SETTLE)
Archived	COIN settled to fiat	Terminal state
Burned	COIN destroyed by DEBIT:DRIFT	Terminal state

Run the conservation check:

```
magic wallet --audit
# Treasury:     45,000 COIN
# Circulation:  23,450 COIN
# Archived:      8,200 COIN
# Burned:        3,350 COIN
# Total:        80,000 COIN
# Genesis:      80,000 COIN
# Conservation: PASS
```

If `Total(t)` ever diverges from `Genesis`, the system has a bug. The conservation equation is a checksum over the entire economy. Run it on every CLOSE event. The LEDGER makes it verifiable — every COIN ever minted, transferred, spent, settled, or burned has a LEDGER entry with a timestamp and a commit hash.

34.24. 32.5 The Seven WALLET Invariants

The WALLET is not a database row. Think of it as a derived view over the LEDGER – the balance at time `t` equals the sum of all LEDGER events for that identity up to `t`. Seven invariants hold at all times [12]:

#	Invariant	Formula	Violation Response
1	Non-negative	`balance >= 0`	Reject transaction
2	Append-only	LEDGER entries never mutated	Reject mutation
3	Deterministic	Same LEDGER → same balance	Recompute from LEDGER
4	Identity-bound	WALLET tied to VITAE.md	Reject anonymous transactions
5	Scope-attributed	Every MINT traces to a scope	Reject unattributed MINT
6	Conservation	Sum of all WALLETS = Circulation pool	CLOSE audit

#	Invariant	Formula	Violation Response
7	Temporal ordering	Events ordered by commit timestamp	Reject out-of-order

Invariant 1: Non-negative balance. A WALLET cannot go below zero. No overdrafts, no credit, no debt.

```
magic wallet --spend 500
# ERROR: Insufficient balance. Current: 340 COIN. Required: 500 COIN.
# Transaction rejected.
```

Invariant 2: Append-only LEDGER. No LEDGER entry is ever modified or deleted. Corrections are new entries (a DEBIT:DRIFT that reverses a previous MINT:WORK, for example). This matches the legal requirement for immutable audit trails – HIPAA mandates non-deletable logs.

Invariant 3: Deterministic reconstruction. Given the same LEDGER, any node must compute the same WALLET balance. The WALLET is a pure function of the LEDGER. No hidden state.

```
magic wallet --reconstruct alice@hadleylab.org
# Replaying 347 LEDGER events...
# Computed balance: 2,340 COIN
# Stored balance:   2,340 COIN
# Match: YES
```

Invariant 4: Identity-bound. Every WALLET is bound to a VITAE.md identity file. Anonymous COIN does not exist. Every transaction is attributed, satisfying audit trail requirements under HIPAA, SOX, and institutional compliance frameworks.

Invariant 5: Scope-attributed. Every MINT:WORK event references a specific scope. Trace any COIN in any WALLET back to the exact governance work that created it.

```
magic wallet --trace alice@hadleylab.org
# COIN provenance for alice@hadleylab.org:
# 255 COIN from SERVICES/EHR-INTEGRATION (4 events, 2026-01 to 2026-02)
# 255 COIN from SERVICES/FHIR-API (3 events, 2026-02 to 2026-03)
# 127 COIN from SERVICES/COMPLIANCE (2 events, 2026-03, in progress)
# 500 COIN from MINT:SIGNUP (2026-01-01)
# Total: 1,137 COIN
```

Invariant 6: Conservation. The sum of all WALLET balances must equal the Circulation pool. This invariant ties the micro level (individual WALLETS) to the macro level (system-wide conservation).

Invariant 7: Temporal ordering. Events are ordered by git commit timestamp. No event can reference a future commit. No event can precede its causal dependency.

34.25. 32.6 Dual-Write Architecture

Every economic event writes to two locations simultaneously:

```
Event → LEDGER.md (governance layer, human-readable)
      → LEDGER.json (runtime layer, machine-readable)
```

The dual-write keeps governance files (.md) and runtime data (.json) synchronized. The build pipeline enforces this – `build` compiles `.md` to `.json`. If the `.json` is `_generated`, never edit it directly. Fix the `.md` source or the compiler.

```
LEDGER.md (source of truth):
  \#\# 2026-03-10T14:23:00Z
  - MINT:WORK | hadleylab-canonic/SERVICES/EHR | 127→255 | +128 COIN
  - TRANSFER | alice→bob | 100 COIN | fee: 5 | net: 95

LEDGER.json (_generated — do not edit):
  [
    {"event":"MINT:WORK","scope":"...","amount":128,...},
    {"event":"TRANSFER","from":"alice","to":"bob","amount":100,...}
  ]
```

The dual-write pattern applies to every economic artifact:

Governance Layer (.md)	Runtime Layer (.json)	Relationship
LEDGER.md	LEDGER.json	Event log
WALLET.md	WALLET.json	Balance snapshot
SHOP.md	SHOP.json	Product catalog
COST_BASIS.md	COST_BASIS.json	Pricing data

Never edit the `.json` files. They are `_generated`. Fix the `.md` source. Run `build`. The pipeline compiles. Validate. Ship [127].

34.26. 32.7 Clinical Example: Hospital COIN Economy

Consider a hospital system governing 200 scopes: 50 clinical, 50 operational, 50 compliance, 50 technology. Supply ceiling: `200 * 255 = 51,000 COIN`.

Over 12 months, the governance team brings 150 scopes to 255 (minting 38,250 COIN), experiences 10 drift events (debiting 1,280 COIN), recruits 20 new governors (distributing 10,000 COIN via MINT:SIGNUP), and generates 15 referrals (distributing 15,000 COIN via MINT:PYRAMID).

Month 12 CLOSE report:

```
Treasury:     12,750 COIN
Circulation:  35,970 COIN
Archived:      1,000 COIN
Burned:        1,280 COIN
Total:        51,000 COIN
Conservation: PASS
```

Every number is auditable. The CFO verifies the conservation equation. The compliance officer traces any COIN to its origin. The CIO sees which scopes are fully governed and which are drifting.

Zoom into the radiology department: 12 scopes covering PACS integration, DICOM routing, report templates, AI triage, quality metrics, peer review, accreditation compliance, dose monitoring, critical findings workflow, teaching file management, research data pipeline, and departmental operations. Each at 255 produces 3,060 COIN total. The department chair reads the WALLET balance as a governance health metric – declining means drift, growing means investment. Objective, continuous, no subjective assessment required [2212].

34.27. 32.8 COIN as Institutional Memory

When team members depart, WALLET history persists. The LEDGER records who minted what COIN, at which scope, at which commit. New team members read the LEDGER to understand the governance provenance of every scope they inherit. COIN survives team transitions because it lives on an append-only LEDGER, not in human memory [12].

Take a GI pathology TALK agent (GastroChat) built by a three-person team over six months. Dr. Patel bootstrapped the TRIAD and LEARNING (+127 COIN). Dr. Kim closed to 255 (+128 COIN). When Dr. Kim finishes fellowship and departs, the next fellow runs `magic wallet --trace` on the scope and sees every governance contribution, every commit, every timestamp. Institutional knowledge transfer becomes automatic.

34.28. 32.9 WALLET Reconciliation

WALLET discrepancies from concurrent commits or pipeline failures resolve deterministically. Run `magic wallet --reconcile {identity}` to replay all LEDGER events and recompute the balance. If the recomputed balance diverges from the stored balance, the LEDGER wins – the WALLET is corrected, and the correction is itself a LEDGER event. No human judgment, no dispute committee.

```
magic wallet --reconcile dr.patel@hospital.org
# Replay 347 events -> computed: 2,847 COIN
```

```
# Stored: 2,847 COIN -> Match: YES
# Reconciliation: PASS
```

34.29. 32.10 Governance Proof

COIN = WORK. The proof chain: governance files exist, `magic validate` measures score, score delta produces gradient, positive gradient triggers MINT:WORK, COIN enters WALLET, WALLET balance equals SUM(LEDGER events), conservation holds at every CLOSE, supply bounded by scope count times 255. Every link is deterministic, auditable, reproducible. COIN is not an incentive layered on top of work – it is the mathematical consequence of governance improvement. Q.E.D. [2212].

34.30. 32.11 COIN Velocity and Economic Health

COIN velocity measures how frequently COIN changes hands – high velocity signals an active governance economy, low velocity signals hoarding or stagnation.

```
magic coin --velocity --period 90d

# COIN Velocity Report — 90 Days
\#
# Metric                  Value
# Total supply:           51,000 COIN
# Active circulation:     35,970 COIN
# Transaction volume:     24,300 COIN (SPEND + TRANSFER events)
# Velocity:               0.68 (transactions / active supply)
\#
# Interpretation:
#   Velocity 0.68 means each COIN in active circulation was
#   transacted 0.68 times in 90 days, or roughly once every 132 days.
\#
# Velocity by category:
#   MINT:WORK → WALLET:     12,400 COIN (governance work)
#   WALLET → SHOP (SPEND):   8,100 COIN (product purchases)
#   WALLET → WALLET (XFER):  3,800 COIN (peer transfers)
\#
# Health indicators:
#   Minting rate:           stable (no sudden spikes)
#   Spending rate:          increasing 12% month-over-month
#   Transfer rate:          stable
#   Dormant wallets:        4 (no activity in 60+ days)
```

A healthy governance economy has velocity between 0.3 and 1.0. Below 0.3, COIN is being minted but not spent – either the SHOP lacks compelling products or buyers lack purchasing intent. Above 1.0, rapid circulation may indicate speculative trading rather than governance work. The velocity metric derives entirely from LEDGER data [2212].

34.31. 32.12 COIN and Multi-Organization Accounting

In health systems with multiple member organizations, COIN accounting spans organizational boundaries while maintaining per-organization balance sheets:

```
magic coin --org-report hadleylab-canonic

# Organization COIN Report — hadleylab-canonic
\#
# Internal:
#   Total minted:       38,250 COIN (150 scopes × 255)
#   Total debited:       1,280 COIN (drift events)
#   Net internal:       36,970 COIN
\#
# Cross-org inflows:
#   From adventhealth-canonic:  2,550 COIN (product purchases)
#   From cedars-canonic:        1,275 COIN (product purchases)
#   Total inflows:              3,825 COIN
\#
# Cross-org outflows:
#   To canonic-canonic:         4,500 COIN (root governance products)
#   To baptist-canonic:        18,000 COIN (radiology suite)
#   Total outflows:            22,500 COIN
\#
# Net cross-org:             -18,675 COIN
# Current org balance:        18,295 COIN
\#
# Conservation check:
#   Minted - Debited + Inflows - Outflows = Balance
#   38,250 - 1,280 + 3,825 - 22,500 = 18,295 □
```

Cross-organization COIN flows are recorded on both LEDGERs: the sender records an outflow, the receiver records an inflow. Monthly CLOSE verifies cross-org consistency – if Organization A's outflows to B do not match B's recorded inflows, the discrepancy is flagged for reconciliation [221219].

34.32. 32.13 COIN Tax and Fee Structure

COIN transactions incur fees that sustain the governance infrastructure:

Transaction Type	Fee	Recipient	Purpose
TRANSFER (peer-to-peer)	5%	TREASURY	Infrastructure maintenance
SPEND (product purchase)	5%	TREASURY	Platform operation
SETTLE (COIN-to-fiat)	10%	TREASURY	Fiat processing costs
MINT:WORK	0%	No fee	Work should not be taxed
MINT:SIGNUP	0%	No fee	Onboarding should not be taxed

```
vault transfer --from dr.chen --to dr.park --amount 100
# Transfer: 100 COIN
# Fee: 5 COIN (5%)
# dr.chen: -105 COIN (100 + 5 fee)
# dr.park: +100 COIN
# TREASURY: +5 COIN
# LEDGER: TRANSFER (evt:09300) + FEE (evt:09301)
```

MINT operations are fee-free because governance work should not be taxed – the work itself is the contribution. Transaction fees apply only to economic activity (transfers, purchases, settlements), ensuring the infrastructure sustains itself from the economic activity it enables, not from the governance work it incentivizes [2212].

34.33. The Economy: Live

The COIN economy is operational. Runner-canonic's ROBERT holds 503 COIN — the first non-governance principal to earn COIN through work. Stripe processes settlements. Ed25519 signs every event. The WITNESS protocol enables cross-ORG balance verification.

The economic loop is closed: governed work → COIN mint → wallet balance → Stripe settlement → USD. Every step is ledgered. Every event is signed. Every balance is witnessable by peer ORGs.

COIN values per governance tier:

Tier	Score	COIN	Meaning
COMMUNITY	35	~35	Scope exists, basic governance
BUSINESS	43	~43	Scope has specification
ENTERPRISE	63	~63	Scope is production-ready
AGENT	127	~127	Scope has LEARNING
FULL	255	~255	All eight questions answered

The gradient mints proportionally. Moving from 0 to 35 mints ~35 COIN. Moving from 35 to 255 mints ~220 more. The COIN economy rewards governance work at every stage.

34.34. 32.14 Clinical Vignette: COIN Resolves Budget Allocation Dispute

A community hospital's IT department and clinical informatics division disagree on resource allocation for clinical AI governance. The IT director argues that IT staff do "most of the governance work." The clinical informatics director argues that clinician contributors provide "the most valuable work."

The COIN LEDGER resolves the dispute objectively:

```
magic coin --department-report --period 6m

# Department COIN Report — 6 Months
\#
# IT Department (8 staff):
#   MINT:WORK events:    142
#   Total COIN minted:   8,470 (avg 60 COIN/event)
#   Primary scopes:      Infrastructure, MONITORING, DEPLOY
#   Avg score delta:     +60 per commit (many small improvements)
\#
# Clinical Informatics (4 staff):
#   MINT:WORK events:    47
#   Total COIN minted:   7,890 (avg 168 COIN/event)
#   Primary scopes:      TALK agents, INTEL, CONTRIBUTE reviews
#   Avg score delta:     +168 per commit (fewer but larger improvements)
\#
# Conclusion: IT does more frequent work (142 vs 47 events).
#             Clinical informatics does higher-impact work (168 vs 60 avg COIN).
#             Total COIN contribution is roughly equal (8,470 vs 7,890).
```

The data settles it. IT performs frequent incremental improvements across infrastructure scopes. Clinical informatics performs fewer but higher-impact improvements on clinical-facing scopes. Total COIN contribution is roughly equal. The dispute resolves with data, not politics [2212].

Chapter 35

Chapter 33: Gradient Minting

Delta rule. Positive/negative gradients. The gradient is the mechanism that connects governance work to COIN minting (Chapter 32) and LEDGER recording (Chapter 13). It operates on the 8-bit score introduced in Chapter 4 and drives the RUNNER task pipeline at gorunner.pro. The gradient's role in the LEARNING feedback loop is explored in Chapter 39, and its convergence properties underpin the healing tool described in Chapter 43.

35.1. 33.1 The Rule

The gradient rule (defined in Chapter 18, Section 18.2 (The Gradient Rule)) determines the COIN delta: the signed difference between the scope's score after governance work and the score before. Positive gradients mint COIN via MINT:WORK, negative gradients debit via DEBIT:DRIFT, and zero gradients produce no economic event. No override, no waiver, no committee decision. On every commit, `magic validate` reads the previous score from the LEDGER, computes the current score from the governance files, calculates the delta, and emits the appropriate event.

The gradient operates on the 8-bit score, where each bit represents one governance dimension. A single commit can flip multiple bits. Add LEARNING.md and ROADMAP.md together, flip bits 6 and 7, and the gradient is 192. The MINT:WORK event records 192 COIN while the LEDGER captures both the aggregate and the individual dimension changes.

The gradient is a discrete derivative. Where continuous mathematics measures instantaneous rate of change, the CANONIC gradient measures the discrete change in governance completeness per commit. The analogy to machine learning is exact: the score is the objective function, the gradient points toward 255, and each commit is a training step [10].

```
Machine Learning          CANONIC Governance
──────────────────        ──────────────────
Loss function             255 − score
```

```
Gradient                to_bits − from_bits
Learning rate           1 (fixed, no hyperparameter)
Training step           git commit
Epoch                   Full scope lifecycle (0 → 255)
Convergence             Score = 255
Overfitting             N/A (discrete target, no overfit possible)
```

35.2. 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
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 [1112].

Here are the LEDGER entries produced at each step:

```
Step 1 — Bootstrap TRIAD:
  Files created: CANON.md, README.md, VOCAB.md
  Dimensions activated: AXIOM(1) + SCOPE(2) + LANGUAGE(32)
  Score: 0 → 35
  LEDGER: MINT:WORK | +35 COIN | commit: f8a2b1c

Step 2 — Add COVERAGE + SPEC:
  Files created: COVERAGE.md, SPEC.md, EVIDENCE.md
  Dimensions activated: EVIDENCE(8) + OPERATIONS(16) + DEPLOYMENT(32)
  Score: 35 → 127
  LEDGER: MINT:WORK | +92 COIN | commit: 3c7d9e2

Step 3 — Add LEARNING + ROADMAP:
  Files created: LEARNING.md, ROADMAP.md
  Dimensions activated: LEARNING(64) + ROADMAP(128)
  Score: 127 → 224
  LEDGER: MINT:WORK | +97 COIN | commit: 9f1a4b8

Step 4 — Close remaining gaps:
  Files updated: COVERAGE.md (100%), all dimensions active
```

```
  Score: 224 → 255
  LEDGER: MINT:WORK | +31 COIN | commit: 2d5e7f0

Step 5 — Neutral edit (typo fix in README.md):
  Score: 255 → 255
  Gradient: 0
  LEDGER: no event (neutral drift)

Step 6 — Delete LEARNING.md:
  Dimension deactivated: LEARNING(64) + cascade
  Score: 255 → 127
  LEDGER: DEBIT:DRIFT | −128 COIN | commit: 6b3c8d1
```

35.3. 33.3 DEBIT:DRIFT Asymmetry

Earning 128 COIN requires adding a major governance dimension – hard work. Losing 128 requires deleting one file – trivial destruction. This asymmetry is intentional [12], and it serves three purposes:

Penalize neglect. Governance drift in healthcare is dangerous. A hospital that stops maintaining its infection control documentation does not immediately fail an audit – the paper trail looks fine for months while the governance silently degrades. DEBIT:DRIFT makes that degradation immediately visible. The WALLET shrinks, the MONITORING dashboard shows red, the alert fires. No waiting for the annual survey.

Incentivize maintenance. Without COIN debits, letting governance decay would be free. The DEBIT makes maintenance a rational economic choice: maintaining a scope costs zero COIN per period (neutral edits produce no gradient), while abandoning one costs up to 255 COIN in debits.

Create information. Every DEBIT:DRIFT event tells you which scope degraded, when, by how much, and who committed the degrading change. Aggregate these events across an organization and you see the drift pattern – which departments maintain governance and which do not.

35.4. 33.4 The RUNNER Task Pipeline

RUNNER tasks orchestrate the minting pipeline; for the complete task specification and execution semantics, see Chapter 18, Section 18.5 (RUNNER Tasks). From the economics perspective, the key insight is that each missing governance dimension has a known COIN yield equal to its bit weight, and teams can prioritize by expected return:

```
Priority = dimension_weight / estimated_effort
```

Work the highest-weight missing dimensions first. This is steepest gradient descent applied to organizational task management. ROADMAP (128) is worth twice LEARNING (64), which is worth four times

OPERATIONS (16). A department with 50 scopes at varying scores sorts all RUNNER tasks by expected COIN yield and works the list top-down for maximum COIN per hour of governance effort [1012].

35.5. 33.5 Supply Ceiling Proof

The maximum COIN any single scope can ever mint is 255. Proof:

```
The score is an 8-bit unsigned integer: range [0, 255].
The maximum gradient for any single transition: 255 - 0 = 255.
Net COIN = SUM(MINT:WORK) - SUM(DEBIT:DRIFT)
         = final_score - initial_score
         ≤ 255 - 0
         = 255

Therefore: net COIN per scope ≤ 255. Q.E.D.
```

The total supply across all scopes:

```
TOTAL\_SUPPLY ≤ N * 255
where N = number of unique scopes
```

Hard ceiling. No inflationary pressure, no central bank, no monetary policy. Supply is a mathematical consequence of the 8-bit score and the gradient rule [12].

35.6. 33.6 Multi-Scope Gradient Patterns

When a governor works across multiple scopes, the gradient pattern reveals their work profile:

```
magic wallet --gradient-history alice@hadleylab.org --last 30d
# Date       | Scope                    | From | To  | Gradient | COIN
# 2026-02-10 | SERVICES/EHR             | 0    | 35  | +35      | +35
# 2026-02-12 | SERVICES/EHR             | 35   | 127 | +92      | +92
# 2026-02-15 | SERVICES/FHIR            | 0    | 255 | +255     | +255
# 2026-02-20 | SERVICES/EHR             | 127  | 255 | +128     | +128
# 2026-03-01 | SERVICES/COMPLIANCE      | 0    | 127 | +127     | +127
# Total: 5 events, +637 COIN minted, 0 debited
```

The gradient history is your work record. It reveals velocity (COIN per day), breadth (scopes touched), and depth (average gradient per scope). A governor minting 255 per scope is completing scopes to full governance. One minting 35 repeatedly is bootstrapping many scopes but completing none. Both patterns are visible, both are valuable, neither is hidden.

35.7. 33.7 Clinical Gradient Example: EHR Migration

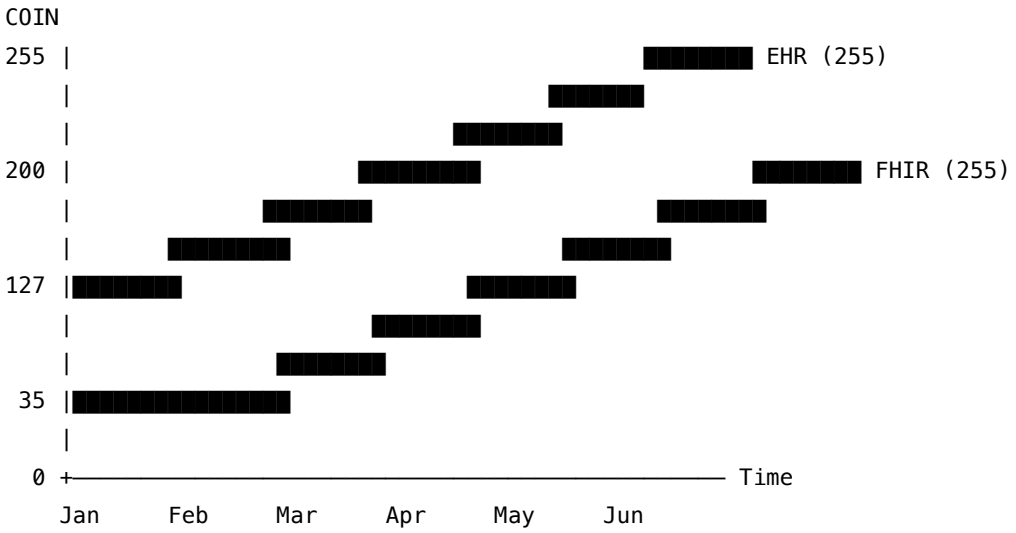
A hospital migrating from one EHR to another creates 30 new governance scopes: data migration, interface mapping, user training, workflow redesign, testing, go-live, and 24 specialty-specific adaptation scopes. The gradient pattern over the migration timeline:

Month	Scopes Active	Avg Score	Total COIN Minted	Cumulative
1	6	35	210	210
2	12	90	1,080	1,290
3	20	160	3,200	4,490
4	28	220	6,160	10,650
5	30	250	7,500	18,150
6	30	255	150	18,300

The gradient curve follows the classic S-shape of a governance buildout: slow start (bootstrapping), rapid middle (dimension activation), and plateau (closing to 255). The cumulative COIN minted tracks total governance investment. Note: the table shows monthly aggregate scores, not monthly COIN – each scope mints at most 255, so the ceiling is 7,650 COIN (30 scopes * 255). The gradient pattern measures progress toward that ceiling [1112].

35.8. 33.8 Gradient Visualization

The MONITORING dashboard displays gradient history as a time-series chart – one line per scope, cumulative COIN minted on the y-axis, time on the x-axis. Organizational governance velocity at a glance.



Each step corresponds to a MINT:WORK event. Flat sections indicate neutral drift (no governance work).

Drops indicate DEBIT:DRIFT events. The dashboard aggregates across all scopes to show organizational governance investment over time.

The gradient chart answers three questions: 1. **Where are we?** Total COIN minted vs. supply ceiling = utilization percentage. 2. **How fast?** COIN per week = governance velocity. 3. **Where is drift?** Scopes with DEBIT:DRIFT events = attention required.

35.9. 33.9 Gradient Economics Summary

The gradient rule connects governance work to economics through five invariants:

```
Invariant 1: gradient = to_bits − from_bits (deterministic)
Invariant 2: gradient > 0 → MINT:WORK (reward)
Invariant 3: gradient < 0 → DEBIT:DRIFT (penalty)
Invariant 4: gradient = 0 → no event (neutral)
Invariant 5: net COIN per scope ≤ 255 (ceiling)
```

These five invariants, combined with the 8-bit score and the LEDGER, constitute a complete economic system for governance. Gradient is the mechanism, COIN is the currency, WALLET is the balance, LEDGER is the record, supply ceiling is the constraint. No additional rules needed. The economy is closed [12].

35.10. 33.10 Clinical Vignette: Gradient Minting During an FDA Software Audit

Mayo Clinic's digital health division operates PathChat – a governed TALK agent for surgical pathology frozen section consultation. PathChat's VOCAB.md defines 42 terms including FROZEN_SECTION, MARGIN_STATUS, and CONCORDANCE_RATE (agreement between frozen and final diagnosis, target >97%).

During an FDA SaMD pre-submission review, the reviewer requests evidence of continuous governance maintenance. The LEDGER provides it directly:

```
magic wallet --gradient-history pathchat-team@mayo.edu \
  --scope SERVICES/TALK/PATHCHAT --last 365d
# 2025-04-01 | 0   -> 35  | +35  | MINT:WORK (bootstrap)
# 2025-04-15 | 35  -> 127 | +92  | MINT:WORK (spec + coverage)
# 2025-05-01 | 127 -> 255 | +128 | MINT:WORK (full governance)
# 2025-07-15 | 255 -> 191 | -64  | DEBIT:DRIFT (stale LEARNING)
# 2025-07-16 | 191 -> 255 | +64  | MINT:WORK (LEARNING restored)
# 2025-10-01 | 255 -> 255 | 0    | neutral (quarterly review)
# 2026-01-15 | 255 -> 255 | 0    | neutral (quarterly review)
# 2026-03-10 | 255 -> 255 | 0    | neutral (current state)
```

```
\#
# Drift events: 1 (resolved in 24 hours)
# Days at 255: 329/365 (90.1%)
# Current streak: 238 consecutive days at 255
```

The FDA reviewer sees continuous maintenance over 12 months. The single drift event was detected automatically and resolved within 24 hours. Compare this to a traditional 200-page QMS document that asserts continuous quality management without independent verification. With CANONIC, the reviewer runs `magic validate` and verifies the score independently. The gradient history IS the maintenance record [312].

35.11. 33.11 Gradient Composition Across Fleet

Multiple organizations in a fleet produce fleet-wide gradient reports:

```
magic wallet --gradient-fleet --last 90d
# Organization          | Scopes | Minted  | Debited | Net     | Utilization
# hadleylab-canonic     | 73     | 18,615  | 342     | 18,273  | 100%
# adventhealth-canonic  | 45     | 10,230  | 1,020   | 9,210   | 80.3%
# mayo-canonic          | 120    | 28,440  | 510     | 27,930  | 91.3%
# tampa-general-canonic | 13     | 3,315   | 0       | 3,315   | 100%
# Fleet total           | 251    | 60,600  | 1,872   | 58,728  | 91.8%
```

AdventHealth at 80.3% utilization with 1,020 COIN debited indicates governance decay across roughly 4 scopes. The fleet governance officer drills into specific scopes and issues RUNNER tasks. The gradient composes upward – scope to organization to fleet – without losing provenance [12].

35.12. 33.12 Gradient Anti-Patterns

Three patterns signal governance dysfunction:

Sawtooth. Score oscillates: 255, drift to 127, recover to 255, drift again. Cause: governance built for compliance reviews, abandoned between them. Fix: install pre-commit hook to enforce continuous 255.

Plateau at 127. Score reaches AGENT tier and stalls. Missing dimension is typically ROADMAP (128). Fix: write ROADMAP.md. The gradient jumps 128 in one commit.

Scatter minting. Governor bootstraps many scopes to 35 but completes none to 255. Gradient history shows many +35 events, zero +128 events. Fix: complete one scope to 255 before bootstrapping the next [1012].

35.13. 33.13 Governance Proof: Gradient Convergence

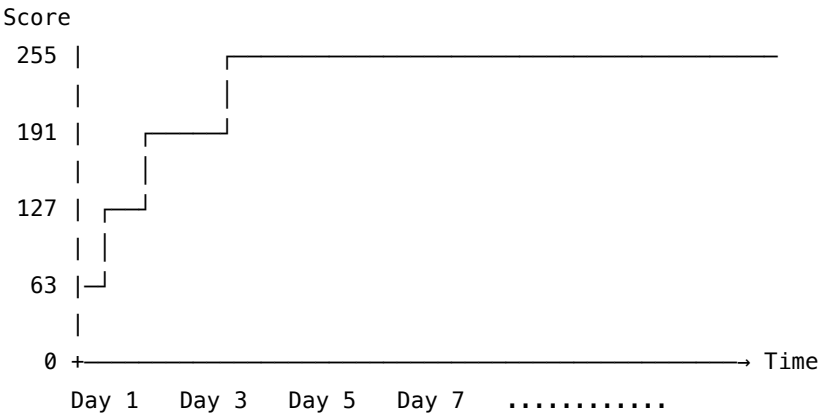
The gradient rule guarantees convergence. Proof:

1. Score is a non-negative integer in [0, 255].
2. `heal()` identifies highest-weight missing dimension.
3. Adding that dimension produces strictly positive gradient.
4. Score increases monotonically under `heal()`.
5. Score bounded above by 255.
6. Monotonically increasing bounded integer sequence converges in at most 8 steps.

Therefore: repeated `heal()` converges to 255 in at most 8 commits. Total COIN = 255 minus initial score. The gradient is the path. The COIN is the receipt. 255 is the destination. Q.E.D. [10312].

35.14. 33.14 Gradient Visualization

The gradient history can be visualized as a step function over time:




```
COIN minted:
  Day 1: +63  (0 → 63)    bootstrap: AXIOM + SCOPE + LANGUAGE + EVIDENCE
  Day 3: +64  (63 → 127)  added: OPERATIONS + DEPLOYMENT (score floor)
  Day 5: +64  (127 → 191) added: LEARNING (institutional memory)
  Day 7: +64  (191 → 255) added: ROADMAP (full governance)
  Total: 255 COIN in 4 commits over 7 days
```

The step function is monotonically non-decreasing during governance improvement. Each step is a commit that adds one or more governance dimensions. The step heights correspond to dimension bit weights. The visualization is generated from LEDGER data:

```
magic gradient --chart --scope SERVICES/TALK/MAMMOCHAT --period 30d

# Output: ASCII chart of score over time
# Exportable as JSON for Grafana visualization
```

35.15. 33.15 Gradient Rate and Team Velocity

Gradient rate measures how quickly a team improves governance scores across their scope portfolio. The metric normalizes across team sizes:

```
magic gradient --team-velocity --period 90d

# Team Velocity Report — 90 Days
\#
# Team              | Members | Scopes | Gradient/Week | COIN/Week | Efficiency
# Radiology AI      | 3       | 12     | +47.3         | 47.3      | 15.8 per person
# Oncology TALK     | 4       | 8      | +31.2         | 31.2      | 7.8 per person
# Infrastructure    | 5       | 20     | +68.5         | 68.5      | 13.7 per person
# Compliance        | 2       | 15     | +22.8         | 22.8      | 11.4 per person
\#
# Fleet velocity: +169.8 gradient/week
# Fleet efficiency: 12.1 gradient/person/week
```

The efficiency metric (gradient per person per week) reveals which teams are most productive. Radiology AI (15.8/person/week) outperforms Compliance (11.4/person/week) despite fewer total scopes – a difference that may reflect scope complexity, team experience, or tooling maturity. The metric informs resource allocation without subjective assessment [1222].

35.16. 33.16 Gradient and the LEARNING Feedback Loop

The gradient connects directly to the LEARNING closure. When you complete a governance improvement (positive gradient), the LEARNING entry documents what you learned during the improvement:

```
## LEARNING entry (auto-generated from gradient event)

| Date | Signal | Description | Source |
|------|--------|-------------|--------|
| 2026-03-10 | PATTERN | ROADMAP.md is fastest to write when structured as quarterly milestones rather tha
```

The gradient event triggers a LEARNING prompt: "Record what you learned during this governance improvement." The prompt is optional but incentivized – scopes with higher LEARNING density (entries per

governance event) receive priority in cross-scope pattern analysis. Governance improvement is not just scored but understood [1220].

35.17. 33.17 Clinical Vignette: Gradient Reveals Governance Bottleneck

Children's Hospital of Philadelphia (CHOP) deploys 8 clinical AI agents. After 6 months, the governance officer notices that all 8 agents plateau at score 191. The gradient analysis reveals the bottleneck:

```
magic gradient --stalled --threshold 191

# Stalled Scopes (score = 191 for > 30 days):
\#
# Scope                              | Stalled Days | Missing Dimension
# SERVICES/TALK/PEDSONCHAT           | 47           | ROADMAP (128)
# SERVICES/TALK/NEOCHAT              | 42           | ROADMAP (128)
# SERVICES/TALK/CARDIACHAT           | 38           | ROADMAP (128)
# SERVICES/TALK/ALLERGYBOT           | 35           | ROADMAP (128)
# SERVICES/TALK/PULMOBOT             | 33           | ROADMAP (128)
# SERVICES/TALK/ENDOCHAT             | 31           | ROADMAP (128)
# SERVICES/TALK/NEPHROCHAT           | 31           | ROADMAP (128)
# SERVICES/TALK/HEMECHAT             | 30           | ROADMAP (128)
\#
# Common missing dimension: ROADMAP (8/8 scopes)
# Diagnosis: Organizational gap — no team member writes roadmaps
# Recommendation: Assign ROADMAP ownership or provide template
```

Every scope is missing the same dimension: ROADMAP. The bottleneck is organizational, not technical – no team member has taken responsibility for writing roadmaps. The fix is fleet-wide: assign a roadmap writer, provide a ROADMAP.md template, and watch all 8 scopes jump from 191 to 255 in a single sprint.

```
# After assigning roadmap ownership and providing template:
magic gradient --last 7d
# 8 scopes: 191 → 255 (+64 each)
# Total COIN minted: 512 (8 × 64)
# Fleet: 8/8 at 255 (100%)
# Bottleneck resolved in 3 days
```

The gradient analysis turned a vague "we're stuck at 191" complaint into a specific, actionable diagnosis: "8 scopes missing ROADMAP, assign ownership." Data drove the fix. The fix was organizational, not technical [122220].

Chapter 36

Chapter 34: The SHOP

Checkout. Pricing. Attestation. The live SHOP is at shop.hadleylab.org. This chapter expands on the service-level architecture introduced in Chapter 12 with operational details: product listing workflow, fiat on-ramp, attestation economics, and fraud detection.

36.1. 34.1 Product Listing Workflow

Every SHOP product begins as a governed scope at 255. The workflow from governed scope to listed product involves a five-phase discovery pipeline, a structured Card, and validation gates; for the full specification, see Chapter 12, Sections 12.2 (Discovery Pipeline) and 12.3 (Card Structure). This chapter focuses on the economic consequences of that workflow.

The critical economic constraint is that `price` cannot fall below `cost_basis`. The cost basis is computed from the LEDGER (the sum of all MINT:WORK events in the scope), and the price is set by the author. This floor ensures that the marketplace cannot undervalue governance work. A scope that required 2,147 COIN of governance investment cannot be listed at 100 COIN, because the market would be pricing below the labor that produced it.

36.2. 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 [17].

The fiat on-ramp works in one direction: dollars in, COIN out. The Stripe integration handles payment processing and receipt generation. The CANONIC system handles COIN minting and LEDGER recording via webhook:

```
User pays \$50 via Stripe →
  Stripe webhook fires →
    CANONIC mints 50 COIN →
      WALLET credits 50 COIN →
        LEDGER records MINT:PURCHASE →
          User can now SPEND 50 COIN in SHOP
```

The off-ramp (SETTLE) works in reverse: COIN out, dollars in. Settlement is rate-limited (one SETTLE per month per identity) to prevent speculation. The SETTLE rate is pegged to COST_BASIS, not to market dynamics.

36.3. 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
```

The loop is self-reinforcing. Trace it through a clinical example:

1. Dr. Chen governs her FHIR integration playbook to 255. She mints 255 COIN.
2. She creates SHOP.md. The playbook is listed in the SHOP.
3. A new resident, Dr. Park, signs up (500 COIN from MINT:SIGNUP).
4. Dr. Park purchases the FHIR playbook for 255 COIN (SPEND event).
5. Dr. Chen's WALLET: 255 + 255 = 510 COIN. Dr. Park's WALLET: 500 - 255 = 245 COIN.
6. Dr. Park adapts the playbook for her department, governs her version to 255.
7. Dr. Park mints 255 COIN. Her WALLET: 245 + 255 = 500 COIN.
8. Dr. Park lists her adapted playbook in the SHOP.
9. The SHOP now has two FHIR playbooks. Competition improves quality.

No gatekeeper decides who can publish. The gate is `score = 255`. Meet it and you enter the market.

36.4. 34.4 Attestation and Reviews

Purchases generate attestations. An attestation is a LEDGER event recording: who bought, what, when, and the amount. Attestations are public but pseudonymous.

```
magic shop --attestations hadleylab-canonic/SERVICES/FHIR-API
# FHIR Integration Playbook
# Attestations: 47
```

```
# Total SPEND: 12,015 COIN
# Unique buyers: 43
# Repeat purchases: 4
# First attestation: 2026-01-20
# Latest attestation: 2026-03-10
```

No star ratings. No subjective reviews. The attestation count IS the quality signal – a product purchased 47 times by people willing to spend COIN carries a stronger signal than a 4.5-star rating from anonymous users.

36.5. 34.5 Public Projection via SHOP.md

SHOP.md is the public face of a scope. It projects the scope's capabilities to the outside world:

```
# SHOP — FHIR Integration Playbook

## What This Is

A complete governance kit for FHIR R4 integration, covering endpoint
configuration, authentication, data mapping, error handling, and compliance
validation.

## What You Get

- CANON.md template for FHIR scopes
- VOCAB.md with 40+ FHIR governance terms
- COVERAGE.md checklist (12 resource types)
- LEARNING.md with 6 months of integration patterns
- ROADMAP.md through FHIR R5

## Price

255 COIN (1 scope, fully governed)

## Prerequisites

- Active CANONIC WALLET with sufficient balance
- VITAE.md on file (identity gate)
```

SHOP.md is human-readable and machine-parseable. `magic scan` reads the YAML front matter; you read the Markdown body. Both see the same product [2212].

36.6. 34.6 SHOP Categories and Taxonomy

Products are organized by category. Categories map to governance domains:

Category	Description	Typical Products
`clinical-informatics`	EHR, FHIR, HL7, DICOM	Integration playbooks, API templates
`compliance`	HIPAA, SOX, GDPR, Joint Commission	Compliance templates, audit kits
`operations`	Workflow, scheduling, staffing	Process governance kits
`research`	IRB, data pipelines, publications	Research governance templates
`education`	Training, onboarding, CME	Educational governance packages
`infrastructure`	Cloud, networking, security	DevOps governance kits

Categories are defined in the root VOCAB.md. Adding a new category requires a governance commit – the category itself is governed. No ad-hoc tags, no folksonomy. The taxonomy is a governed vocabulary [6].

36.7. 34.7 Multi-Scope Products

A product can span multiple scopes. A book with 20 chapters is 20 scopes. A governance suite with 50 templates is 50 scopes:

```
```{=latex}
\bigskip\begin{center}\textcolor{canonic-slate}{\textbullet\ \textbullet\ \textbullet}\end{center}\bigskip
```

title: “Hospital Governance Suite” author: “governance@healthsys.org” scopes: - “healthsys/SERVICES/EHR” - “healthsys/SERVICES/FHIR” - “healthsys/SERVICES/COMPLIANCE” - “healthsys/SERVICES/OPERATIONS” total_scopes: 50 score: 255 cost_basis: 12750 price: 15000

• • •

```
Every constituent scope must be at 255. If any scope drifts below 255, the product is delisted automatical

## 34.8 SHOP Economics: Supply and Demand

The SHOP creates a market where supply is governed work and demand is organizational need. Supply grows as
```
```

```
| Market Phase | Characteristics | COIN Flow |
|--------------|-----------------|-----------|
| Genesis | Few products, low awareness | MINT:SIGNUP dominates |
| Growth | Products listed, buyers arrive | SPEND events increase |
| Maturity | Competition in categories | Prices stabilize at ~1x cost basis |
| Specialization | Niche products, premium pricing | Margins increase for specialists |

Decentralized, transparent, self-organizing. No central market maker required.

## 34.9 Clinical SHOP Example: Compliance Templates

A compliance officer needs Joint Commission accreditation governance. She runs:

```bash
magic scan --shop --category compliance
# 1. Joint Commission Readiness Kit (1,530 COIN)
#    6 scopes | 47 attestations | compliance@healthsys.org
# 2. HIPAA Compliance Template (255 COIN)
#    1 scope | 123 attestations | privacy@largeclinic.org
# 3. CMS Conditions of Participation (2,040 COIN)
#    8 scopes | 31 attestations | regulatory@hospital.org
```

She purchases the Joint Commission kit for 1,530 COIN. It includes 6 governed scopes: Environment of Care, Life Safety, Medication Management, Infection Prevention, Emergency Management, and Leadership Standards. Each at 255 with full governance files.

She adapts the kit: adds local vocabulary, adjusts coverage, records her LEARNING. Over 3 months, she brings all 6 adapted scopes to 255, minting 1,530 COIN. Her WALLET replenishes. Each adaptation adds LEARNING, each LEARNING entry makes the next adaptation faster. Knowledge compounds [2212].

36.8. 34.10 Clinical Vignette: SHOP Fraud Detection at Scale

Geisinger Health (Pennsylvania, 13 hospital campuses) discovers a governance anomaly during routine LEDGER analysis. A principal — `governance-bot-7` — listed 14 products in the SHOP over a 48-hour period. Each product claimed a cost basis of 255 COIN (single scope). Each was priced at 255 COIN. The products targeted niche compliance categories: Antimicrobial Stewardship Governance, Surgical Site Infection Prevention, Venous Thromboembolism Prophylaxis Protocol, Falls Risk Assessment Governance, and similar narrow clinical quality domains.

Detection. The MONITORING service flagged the anomaly through two signals:

```
Signal 1: SHOP:LIST frequency
  14 listings in 48 hours from one principal
```
```

```
  Fleet average: 0.3 listings per principal per month
  Deviation: 46.7× fleet average

Signal 2: Governance velocity anomaly
  14 scopes governed 0→255 in 48 hours
  Average: 87 commits per scope to reach 255
  governance-bot-7: 3.2 commits per scope
  Deviation: 27× faster than fleet average
```

Investigation. The compliance team pulled the LEDGER events for `governance-bot-7`:

```
vault timeline governance-bot-7 --last 48h
# evt:06100  MINT:WORK  +7 COIN   scope: ANTIMICROBIAL-STEWARDSHIP
# evt:06101  MINT:WORK  +248 COIN scope: ANTIMICROBIAL-STEWARDSHIP
# evt:06102  SHOP:LIST            scope: ANTIMICROBIAL-STEWARDSHIP
# evt:06103  MINT:WORK  +7 COIN   scope: SURGICAL-SITE-INFECTION
# evt:06104  MINT:WORK  +248 COIN scope: SURGICAL-SITE-INFECTION
# evt:06105  SHOP:LIST            scope: SURGICAL-SITE-INFECTION
# [pattern repeats 12 more times]
```

The pattern: bootstrap TRIAD (+7), then a single massive commit that adds all remaining files (+248), then immediate SHOP listing. The governance files existed but were hollow — VOCAB.md contained 3 terms (minimum to pass validation), COVERAGE.md listed checkboxes without substance, LEARNING.md contained a single generic entry. The files satisfied the structural validator but contained no genuine governance knowledge.

The Structural Gap. This incident exposed a limitation in the 8-bit validator. `magic validate` checks for file presence and structural compliance (headings, tables, frontmatter), but does not evaluate semantic quality. A VOCAB.md with 3 trivially defined terms scores the same as one with 40 domain-expert terms. The LEARNING bit is set if LEARNING.md exists and contains the required structure — not if the patterns are genuine.

Resolution. The governance team implemented three countermeasures:

Countermeasure	Mechanism	LEDGER Signal
Velocity gate	Max 2 SHOP:LIST per principal per week	SHOP:VELOCITY_GATE
Attestation threshold	Products require 3 attestations within 30 days to remain listed	SHOP:ATTESTATION_CHECK
Peer review gate	First listing requires CONTRIBUTE:REVIEW from a VERIFIED principal	SHOP:PEER_REVIEW

The velocity gate prevents bulk listing. The attestation threshold ensures that products have actual buyers

— hollow products that nobody purchases are automatically delisted after 30 days. The peer review gate ensures that a verified expert reviews the first product from any principal before it enters the SHOP.

All 14 hollow products were delisted. `governance-bot-7`'s WALLET was debited 3,570 COIN (14 scopes × 255 COIN per DEBIT:DRIFT). The principal's identity was flagged for manual review by the IDENTITY service. The incident was recorded in fleet LEARNING.md as pattern: HOLLOW_GOVERNANCE [1512].

36.9. 34.11 SHOP Search and Filtering

The SHOP provides structured search through compiled metadata:

```
magic scan --shop --search "FHIR" --min-attestations 10 --max-price 1000

# Results (sorted by attestation count):
# 1. FHIR Quick Start Guide
#    Price: 255 COIN | Attestations: 87 | Author: dr-chen
#    Category: clinical-informatics | Scopes: 1
\#
# 2. FHIR Bulk Data Export Template
#    Price: 510 COIN | Attestations: 43 | Author: dr-williams
#    Category: infrastructure | Scopes: 2
\#
# 3. FHIR to HL7v2 Bridge Governance
#    Price: 765 COIN | Attestations: 29 | Author: interop-team
#    Category: clinical-informatics | Scopes: 3
```

Search operates on the compiled SHOP.json — a `_generated` artifact that the build pipeline produces from all SHOP.md files. For the discovery architecture and compilation pipeline that produces this artifact, see Chapter 12, Section 12.2 (Discovery Pipeline). The search index is rebuilt on every `build` invocation. Filters operate on governed metadata fields: category (from VOCAB.md taxonomy), attestation count (from LEDGER), price (from SHOP.md frontmatter), author (from VITAE.md), and scope count.

The fleet website provides a visual SHOP interface. Products are rendered as governed pages, each compiled from the SHOP.md card, the scope's CANON.md, and the attestation history from the LEDGER. The page is `_generated`. The product page displays:

Section	Source	Content
Title and description	SHOP.md	Product summary
Price and cost basis	SHOP.md + LEDGER	Transparent economics
Governance score	`magic validate`	Current 255/255 status
Attestation count	LEDGER query	Buyer confidence signal
Author identity	VITAE.md	Verified author credentials
Scope tree	CANON.md hierarchy	Governance depth visualization

Section	Source	Content
Evidence citations	INTEL.md	Clinical evidence backing

The transparency is radical. You see the cost basis (what it cost to build), the price (what the author charges), the margin (the difference), and the attestation count (how many others have purchased). No hidden information, no opaque algorithms. The market is a governed surface over the LEDGER.

36.10. 34.12 SHOP Access Control

Product access is gated by three conditions:

Gate	Check	Failure
WALLET balance	`buyer.balance >= product.price`	`SHOP_INSUFFICIENT_BALANCE`
Identity verification	`buyer.vitae_status == VERIFIED`	`SHOP_IDENTITY_REQUIRED`
Scope permission	Buyer not on seller's deny list	`SHOP_ACCESS_DENIED`

The identity gate prevents anonymous purchases. Every buyer must have a verified VITAE.md. This creates accountability in the market — every attestation traces to a verified identity. The deny list is rare but necessary: it allows authors to block principals who have been flagged for HOLLOW_GOVERNANCE or other governance violations.

```
vault purchase --product SERVICES/FHIR-PLAYBOOK --buyer dr-park

# Pre-purchase checks:
#   WALLET balance:    500 COIN >= 255 COIN  PASS
#   VITAE verified:    dr-park VERIFIED       PASS
#   Access check:      no deny list entry     PASS
\#
# Executing SPEND:
#   Debiting dr-park:     -255 COIN
#   Crediting dr-nguyen:  +255 COIN
#   LEDGER events:        SPEND recorded (evt:05200)
#   Access granted:       SERVICES/FHIR-PLAYBOOK readable by dr-park
```

36.11. 34.13 SHOP Reporting

Monthly SHOP reports are generated from the LEDGER:

```
magic shop --report --period 2026-02

# SHOP Monthly Report — February 2026
\#
# New listings:           8
# Total active products:  47
# Delistings (drift):     2 (both relisted within 24h)
# Total SPEND events:     312
# Total COIN transacted:  89,760
# Unique buyers:          187
# Unique sellers:         34
# Top category:           clinical-informatics (41% of sales)
# Average price:          288 COIN
# Median attestations:    23 per product
\#
# Revenue distribution:
#   Top 10% of sellers:   62% of revenue
#   Bottom 50% of sellers: 8% of revenue
#   Gini coefficient:      0.54
```

The Gini coefficient measures market concentration. A coefficient of 0.54 indicates moderate concentration — some products dominate their categories, but the market is not monopolistic. The governance gate ensures that every participant who reaches 255 can participate. Market success depends on product quality (attestation count), not on platform gatekeeping [17].

36.12. 34.14 Governance Proof: The Market Chain

The SHOP's integrity derives from the governance chain:

```
Author governs scope to 255
  → cost_basis computed from LEDGER (sum of MINT:WORK)
  → SHOP.md created with Card (price >= cost_basis)
  → magic validate --shop gates: score, price, identity, route
  → SHOP:LIST recorded on LEDGER
  → Buyer discovers product (magic scan or fleet page)
  → Buyer's WALLET balance checked (LEDGER-derived)
  → Buyer's VITAE checked (IDENTITY-verified)
  → SPEND event: buyer debited, seller credited (dual-write)
  → Product access granted (scope permission updated)
  → Attestation count increments (LEDGER query)
  → If scope drifts: auto-delist (MONITORING → SHOP:DELIST)
  → If scope heals: auto-relist (magic validate → SHOP:RELIST)
```

No step in this chain requires manual intervention after the initial governance work. The market operates as a projection of governance state. Products exist because scopes are at 255. Products disappear when scopes drift below 255. The SHOP is not a marketplace built on top of governance — the SHOP IS governance projected onto an economic surface. Q.E.D. [171215]

36.13. 34.15 SHOP Product Bundling

Authors can bundle multiple products into a single SHOP listing. A bundle groups related scopes under a single purchase at a bundled price:

```
# SHOP.md — Radiology AI Governance Bundle

## Card

| field | value |
|-------|-------|
| title | Radiology AI Complete Governance Bundle |
| price | 4,500 COIN |
| type | bundle |
| scopes | RADIOLOGY/TRIAGE, RADIOLOGY/PEER-REVIEW, RADIOLOGY/AI-METRICS, RADIOLOGY/MAMMOCHAT, RADIOLOGY/D
| cost_basis | 1,275 COIN (5 × 255) |
| attestations | 12 |
| description | Complete governance suite for radiology AI: triage algorithms, peer review workflows, AI p
```

The bundle price (4,500 COIN) must exceed the aggregate cost basis (1,275 COIN) of all included scopes. The buyer receives access to all 5 scopes in a single transaction. The LEDGER records a single SPEND event with bundle metadata:

```
vault spend --product radiology-ai-bundle --principal dr.park@clevelandclinic.org

# Bundle purchase:
#   Product: Radiology AI Complete Governance Bundle
#   Scopes: 5
#   Price: 4,500 COIN
#   LEDGER events:
#     SPEND:BUNDLE dr.park -4,500 COIN (evt:06100)
#     ACCESS:GRANT dr.park RADIOLOGY/TRIAGE (evt:06101)
#     ACCESS:GRANT dr.park RADIOLOGY/PEER-REVIEW (evt:06102)
#     ACCESS:GRANT dr.park RADIOLOGY/AI-METRICS (evt:06103)
#     ACCESS:GRANT dr.park RADIOLOGY/MAMMOCHAT (evt:06104)
#     ACCESS:GRANT dr.park RADIOLOGY/DASHBOARD (evt:06105)
```

Bundles provide economic incentives for comprehensive governance adoption. A buyer purchasing 5

scopes individually would pay 5 × market price. The bundle offers a discount that makes comprehensive adoption cheaper than piecemeal adoption [1712].

For the COIN economics underlying these transactions, see Chapter 32 (COIN and the WALLET) and Chapter 35 (COSTBASIS and Pricing). For how products are constructed from governed scopes, see Chapter 8 (Building a Product). For the reputation system that tracks attestations, visit star.hadleylab.org [171215].

Chapter 37

Chapter 35: COST_BASIS and Pricing

Formula. Examples. Constraints. COST_BASIS is the economic floor for every product listed in the SHOP (Chapter 34 and shop.hadleylab.org). It is computed directly from the LEDGER's MINT:WORK events (Chapter 13) and the gradient minting rule defined in Chapter 33. The pricing tiers map to the governance tiers introduced in Chapter 5. For the governor's perspective on COIN economics, see the CANONIC CANON.

37.1. 35.1 The Formula

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

The formula is deterministic. COST_BASIS equals the total COIN minted for the scope(s) that compose a product. No markup, no subjective valuation – a pure function of the governance work performed.

Compute it from the LEDGER:

```
magic cost-basis hadleylab-canonic/SERVICES/FHIR-API
# Scope: hadleylab-canonic/SERVICES/FHIR-API
# MINT:WORK events: 4
#   +35 (2026-01-15) — bootstrap
#   +92 (2026-01-20) — coverage + spec
#   +97 (2026-02-01) — learning + roadmap
#   +31 (2026-02-15) — close to 255
# DEBIT:DRIFT events: 0
# Cost basis: 255 COIN
```

For multi-scope products, the cost basis is additive:

```
cost_basis(book) = SUM(cost_basis(chapter_scope) for each chapter)
```

Cost basis is the floor. The author can price above it but never below – pricing below cost basis implies selling governance work for less than it cost to produce, and the system rejects this:

```
magic shop --set-price 200 hadleylab-canonic/SERVICES/FHIR-API
# ERROR: Price 200 is below cost basis 255.
# Minimum price: 255 COIN.
# Price rejected.
```

37.2. 35.2 Examples

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

Expand with clinical healthcare examples:

Product	Scopes	Cost Basis	Use Case
HIPAA compliance template	1 scope	255 COIN	Single-department compliance
EHR integration playbook	3 scopes	765 COIN	Multi-system integration
Radiology AI governance kit	8 scopes	2,040 COIN	Full AI pipeline governance
Hospital-wide governance suite	50 scopes	12,750 COIN	Enterprise deployment
Multi-site network template	200 scopes	51,000 COIN	Health system standardization

Cost basis scales linearly with governance breadth. A solo practitioner's blog post costs 255 COIN; a hospital system's governance suite costs 51,000 COIN. The 200:1 ratio reflects the ratio of governance effort. No artificial pricing – the price is the work.

37.3. 35.3 Pricing Tiers

Tier	Price	Rationale
COMMUNITY	Free	Governance that excludes people isn't governance [22]
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 [22]

Each tier maps to a different organizational profile:

COMMUNITY tier. Individual practitioners, students, researchers, open-source contributors. No cost. Full access to governance tooling. COIN minting enabled. SHOP participation enabled.

```
magic tier --check alice@student.edu
# Tier: COMMUNITY
# Price: Free
# Tools: magic validate, magic scan, build, magic-heal
# COIN: enabled
# SHOP: enabled (buy and sell)
# Support: community forums
```

BUSINESS tier. Small to medium organizations with 5-50 governors. $100/year per organization (not per seat). Includes everything in COMMUNITY plus: priority validation queues, fleet page hosting, custom domain support, MONITORING dashboard, and NOTIFIER integrations.

```
magic tier --check admin@smallclinic.org
# Tier: BUSINESS
# Price: \$100/year
# Governors: up to 50
# Support: email, 48h response
```

ENTERPRISE tier. Large organizations with 50-5,000+ governors. Contract pricing. Includes everything in BUSINESS plus: dedicated validation infrastructure, custom CANON.md templates for regulated industries, compliance reporting (HIPAA, SOX, GDPR), SSO/SAML integration, and SLA guarantees.

```
magic tier --check cio@hospital-system.org
# Tier: ENTERPRISE
# Price: Custom contract
# Governors: unlimited
# Support: named account manager, 4h response SLA
```

FOUNDATION tier. Nonprofits, NGOs, educational institutions, government agencies. Free at enterprise scale. All ENTERPRISE features at zero cost [22].

37.4. 35.4 Unit Economics

The unit cost of governance validation:

```
cost_per_validation = infrastructure_cost / total_validations
```

At scale:

Scale	Validations/month	Infrastructure	Cost/validation
Solo	100	$0 (local)	$0.00
Team (10)	1,000	$10/mo	$0.01
Department (50)	5,000	$50/mo	$0.01
Enterprise (500)	50,000	$200/mo	$0.004
Network (5,000)	500,000	$500/mo	$0.001

Cost per validation decreases with scale, and the marginal cost approaches zero – natural economies of scale for enterprise adoption.

37.5. 35.5 Enterprise Volume Pricing

For ENTERPRISE contracts, volume pricing applies to COIN-to-fiat conversion (SETTLE events):

Monthly COIN Volume	Settle Rate
< 1,000 COIN	$1.00 per COIN
1,000-10,000 COIN	$0.90 per COIN
10,000-100,000 COIN	$0.80 per COIN
> 100,000 COIN	$0.70 per COIN

Volume discounts reward governance investment at scale.

37.6. 35.6 The Pricing Invariant

One invariant governs all pricing:

```
price(product) >= cost_basis(product)
```

No exceptions. No sales. No discounts below cost basis. The cost basis is the floor. The author sets the ceiling. The spread is the author's margin.

```
Floor: cost_basis (work invested)
Price: author-set (market rate)
Ceiling: none (market competition applies)
Margin: price - cost_basis
```

The pricing invariant ensures COIN retains its connection to work [221112].

37.7. 35.7 Clinical Pricing Example

A clinical informatics team at a 500-bed hospital builds a governance suite over 18 months:

```
Phase 1 (months 1-6): 20 scopes, cost basis 5,100 COIN
Phase 2 (months 7-12): 30 scopes, cumulative 12,750 COIN
Phase 3 (months 13-18): 50 scopes, cumulative 25,500 COIN
```

At month 18, the team lists their suite at 30,000 COIN (cost basis 25,500 + 4,500 margin). A neighboring hospital purchases it. The selling team earns 30,000 COIN; the buying team receives 100 governed scopes and reduces their buildout from 18 months to 3. COST_BASIS makes the transaction transparent – trust is replaced by verification [12].

37.8. 35.8 Cost Basis Decomposition

The cost basis can be decomposed to show which governance questions contributed:

```
magic cost-basis --decompose hadleylab-canonic/SERVICES/FHIR-API
# Question decomposition:
#   Each answered question contributes COIN proportional to its governance effort.
#   The kernel assigns weights — the math is in magic.c.
#   Total: 255 COIN (all eight questions answered)
```

Questions requiring more governance effort are weighted higher. Declaring an axiom is easy; building a 12-month roadmap is hard. The weights encode the effort gradient – defer to the C kernel for specifics.

37.9. 35.9 Dynamic Pricing and Market Signals

The cost basis floor is fixed (determined by work), but the price ceiling is dynamic. Authors observe market signals and adjust:

```
Signals that increase price:
  - High attestation count (proven demand)
  - Unique category (no competition)
  - Specialized domain (regulatory expertise)
  - Multi-scope depth (comprehensive coverage)

Signals that decrease price:
  - Low attestation count (unproven demand)
  - Competitive category (many alternatives)
  - Generic domain (easy to reproduce)
  - Single scope (limited coverage)
```

The SHOP displays these signals transparently. A buyer can compare:

```
magic shop --compare compliance
# Product A: 255 COIN | 123 attestations | 1 scope
# Product B: 765 COIN |  47 attestations | 3 scopes
# Product C: 2040 COIN |  8 attestations | 8 scopes
# Signal: Product A has highest proven demand per COIN.
# Signal: Product C has deepest coverage per attestation.
```

You make an informed choice. The seller receives market feedback through attestation counts. No opaque algorithm determines visibility, no pay-to-rank mechanism distorts discovery. The market is transparent because the LEDGER is transparent [2212].

37.10. 35.10 Clinical Vignette: COST_BASIS in a Multi-Hospital Procurement

Baptist Health South Florida operates 12 hospitals. Their clinical informatics division builds a comprehensive radiology AI governance suite: 48 scopes covering AI triage, DICOM routing, peer review workflows, MammoChat, and a radiology quality dashboard. Cost basis: 12,240 COIN. Development time: 14 months. Team: 3 clinical informatics engineers.

Baptist Health lists the suite in the SHOP at 18,000 COIN — a 47% margin over cost basis. Cleveland Clinic's radiology department discovers the listing via `magic shop --search radiology governance`. The SHOP card shows:

```
magic shop --detail baptist-radiology-suite
# Product:       Baptist Health Radiology AI Governance Suite
# Author:        baptist-canonic/RADIOLOGY
# Scopes:        48
# Score:         255/255 (all scopes)
# Cost basis:    12,240 COIN
# Price:         18,000 COIN
# Attestations:  7 (5 hospitals, 2 academic centers)
# Category:      HEALTHCARE/RADIOLOGY/AI-GOVERNANCE
# Last updated:  2026-02-28
```

Cleveland Clinic evaluates the purchase. Building 48 radiology governance scopes internally would cost roughly 14 months and 12,240 COIN of governance work. Purchasing from Baptist Health costs 18,000 COIN but saves 14 months. COST_BASIS makes the build-vs-buy calculation transparent: the 5,760 COIN premium buys 14 months of acceleration.

Cleveland Clinic purchases the suite. The transaction produces these LEDGER events:

```
SPEND:PRODUCT  cleveland-canonic  -18,000 COIN  baptist-radiology-suite
REVENUE        baptist-canonic    +17,100 COIN  baptist-radiology-suite (5% fee)
FEE            canonic-canonic       +900 COIN  TRANSFER fee
ATTEST         cleveland-canonic       +1        baptist-radiology-suite
```

Baptist Health earns 17,100 COIN after the 5% transfer fee. Cleveland Clinic receives 48 governed scopes that they can customize (add constraints, never weaken) for their own radiology department. The attestation count increases to 8, making the product more discoverable for the next buyer [12].

37.11. 35.11 COST_BASIS Audit and Verification

Any participant can verify a product's cost basis. The verification is deterministic:

```
magic cost-basis --verify baptist-radiology-suite
# Verification method: Walk LEDGER for all MINT:WORK events
#   matching scope: baptist-canonic/RADIOLOGY/**
# Events found:     127 MINT:WORK events
# Total minted:     12,240 COIN
# Drift debits:     0 COIN
# Net cost basis:   12,240 COIN
# Claimed:          12,240 COIN
# Status:           VERIFIED — cost basis matches LEDGER
```

The verification walks the LEDGER and sums all MINT:WORK events for the product's scope tree. If the

sum does not match the claimed cost basis, the listing is flagged as inconsistent. No trust required – the LEDGER is the proof [1215].

37.12. 35.12 Governance Proof: The Pricing Chain

The pricing architecture produces a complete audit trail from work to revenue:

```
Developer commits governance improvement (git commit)
  → magic validate detects positive gradient (+92)
    → MINT:WORK event recorded on LEDGER (92 COIN)
      → cost_basis updated for scope (cumulative sum)
        → SHOP.md price must exceed cost_basis (invariant enforced)
          → Buyer purchases product (SPEND event)
            → Seller receives revenue minus fee (REVENUE event)
              → COIN-to-fiat settlement available (SETTLE event)
```

Every link is auditable, every amount verifiable, every event on the LEDGER. The pricing chain proves that the price of every product traces back to governance work performed by identified developers at specific commits – not asserted value, but demonstrated, auditable, LEDGER-backed value. COST_BASIS is the foundation, price is the market layer above it, and the LEDGER is the proof beneath both [2212].

37.13. 35.13 COST_BASIS and Depreciation

Governance work does not depreciate. The COST_BASIS of a scope at commit `abc1234` is the same today as it was when the work was performed – the LEDGER never expires MINT:WORK events. Market value may decrease if governance drifts or the domain becomes obsolete, but the cost basis does not.

This distinction matters: COST_BASIS reflects work performed (historical fact), while price reflects current market value (present assessment). A product whose scopes drift below 255 is automatically delisted – its COST_BASIS remains unchanged, but its market availability drops to zero until governance is restored [2212].

```
# Check cost basis vs. market status
magic cost-basis --market-status baptist-radiology-suite

# Cost basis:     12,240 COIN (permanent — 127 MINT:WORK events)
# Market status:  DELISTED (3 scopes drifted below 255)
# Drifted scopes: RADIOLOGY/TRIAGE (247), RADIOLOGY/PEER-REVIEW (239), RADIOLOGY/DASHBOARD (251)
# Action:         Heal drifted scopes → auto-relist at 18,000 COIN
```

COST_BASIS is immutable; market availability is dynamic. The relationship between them is governed by the SHOP's 255 invariant: only fully governed products are listed [221215].

37.14. 35.14 Cross-Currency COST_BASIS in Federation

When federated organizations trade products, COST_BASIS provides a universal anchor. Both organizations' COIN is backed by governance work validated at 255. The exchange rate is 1:1 at the COST_BASIS level – 255 COIN in Organization A equals 255 COIN in Organization B, because both represent a scope governed from 0 to 255 across all 8 dimensions.

Market prices may differ – Organization A's radiology suite may command a premium where radiology expertise is scarce. But the COST_BASIS is universal: the governance work to build 48 scopes is 12,240 COIN regardless of which organization performed it, enabling cross-organization procurement comparisons [221219].

37.15. 27.18 INTEL.md Migration Patterns

When migrating an existing knowledge base into CANONIC governance, INTEL.md is the entry point. The migration pattern:

```
# Step 1: Inventory existing knowledge sources
# List all documents, papers, blog posts, databases
# that the scope's knowledge is based on.

# Step 2: Categorize into evidence layers
# Layer 1: Governance sources (CANONIC internal)
# Layer 2: Peer-reviewed papers (with DOIs)
# Layer 3: Blog posts / articles (with dates)
# Layer 4: Service specifications
# Layer 5: External references

# Step 3: Write INTEL.md
# Use the template from 27.12

# Step 4: Add citations to all existing claims
# Every factual statement in scope documents
# must reference an evidence layer entry

# Step 5: Validate
magic validate --citations SERVICES/TALK/NEWCHAT/INTEL.md
```

Clinical migration example: An oncology department has a shared Google Drive with 47 clinical practice guidelines, 12 drug interaction databases, and 8 institutional protocols. Migrating to CANONIC INTEL:

Source	Count	INTEL Layer	Citation Format
NCCN Guidelines	15	Layer 2 (Papers)	[P-XX] NCCN 2026.1
ACR Practice Parameters	8	Layer 2 (Papers)	[P-XX] ACR 2025
Drug interaction databases	12	Layer 5 (External)	[E-XX] Lexicomp/UpToDate
Institutional protocols	8	Layer 4 (Service)	[S-XX] Protocol name
FDA safety communications	4	Layer 5 (External)	[E-XX] FDA MedWatch

The migration transforms unstructured knowledge into governed, cited, cross-referenced INTEL. After migration, every clinical claim in the scope traces to a source [2].

37.16. 27.19 INTEL Versioning and History

INTEL.md is version-controlled via git. The git history of INTEL.md is the evidence evolution history:

```
# View INTEL.md evolution
git log --oneline SERVICES/TALK/MAMMOCHAT/INTEL.md

# Output:
# abc1234 Update NCCN to 2026.1, add NCT06604078
# def5678 Add layer 5 external references
# 789abcd Initial MammoChat INTEL (12 papers, 3 governance)
```

Each commit to INTEL.md is a governance event. The LEDGER records the delta:

```
{
  "event": "INTEL:UPDATE",
  "scope": "SERVICES/TALK/MAMMOCHAT",
  "commit": "abc1234",
  "message": "Update NCCN to 2026.1, add NCT06604078",
  "delta": {
    "papers_added": 1,
    "papers_removed": 0,
    "layers_updated": ["Layer 2"],
    "freshness_improved": true
  }
}
```

The complete history of what the agent knows, when it learned it, and who governed the change is in the git log + LEDGER combination. For clinical AI compliance, this is the knowledge management audit trail [214].

37.17. 27.20 INTEL and Agent Knowledge Boundaries

INTEL.md explicitly defines what an agent knows and does not know. The knowledge boundary is compiled into the systemPrompt:

```
KNOWLEDGE BOUNDARY (compiled from INTEL.md):
  KNOWS:
    - BI-RADS classification system (ACR 5th Edition)
    - NCCN breast cancer screening guidelines (2026.1)
    - Breast imaging modalities (mammography, MRI, ultrasound)
    - Clinical trial NCT06604078 (deployment data)
    - mCODE breast cancer profiles

  DOES NOT KNOW:
    - Patient-specific data (no PHI access)
    - Non-breast oncology (routes to OncoChat)
    - Genomic variant classification (routes to OmicsChat)
    - Drug interactions (routes to MedChat)
    - Legal compliance details (routes to LawChat)
    - Financial/billing codes (routes to FinChat)
```

The knowledge boundary is the agent's type signature. It declares what the agent can answer and what it must route. The boundary is enforced by the systemPrompt — the agent will decline queries outside its boundary and suggest routing to the appropriate specialist agent.

Clinical significance: A patient who asks MammoChat about chemotherapy drug interactions receives a routing response, not a hallucinated answer. The knowledge boundary prevents the agent from operating outside its governed expertise. The boundary is governance, not a suggestion [2126].

37.18. 27.21 INTEL Completeness Checklist

Use this checklist to verify INTEL.md completeness before submitting for review:

#	Item	Required	Check
1	`inherits:` field present	Yes	Path resolves

#	Item	Required	Check
2	Axiom section present	Yes	Non-empty, non-placeholder
3	Scope Intelligence table	Yes	All 4 dimensions filled
4	Evidence Chain table	Yes	At least 2 layers populated
5	Cross-Scope Connections table	Yes	At least TALK + COIN + LEDGER
6	Citations in all claims	Yes	0 uncited claims
7	Evidence layers INDEXED	Yes	At least layers 1-2
8	Freshness < 90 days	Recommended	No stale layers
9	Quality score HIGH	Recommended	Coverage > 0.95
10	Source count > 10	Recommended	Depth threshold met

```
# Run the completeness check
magic intel --checklist SERVICES/TALK/MAMMOCHAT
```

A complete INTEL.md enables the expression question. An incomplete INTEL.md blocks FULL tier [2].

37.19. 27.22 INTEL.md as Single Source of Truth

INTEL.md is the single source of truth for what a scope knows. There is no second knowledge base. There is no hidden configuration. There is no database of evidence that exists outside INTEL.md.

If knowledge is not in INTEL.md, the agent does not know it. If evidence is not cited in INTEL.md, the agent cannot cite it. If a cross-scope connection is not declared in INTEL.md, the routing table does not include it.

This constraint is deliberate. It means that auditing an agent's knowledge requires reading one file: INTEL.md. It means that updating an agent's knowledge requires editing one file: INTEL.md. It means that the entire knowledge provenance of a clinical AI agent fits in a single Markdown document.

The simplicity is the governance. Complex knowledge management systems fail because complexity hides ungoverned knowledge. INTEL.md succeeds because everything is visible, everything is cited, and everything compiles [2].

Chapter 38

Chapter 36: Governance as Type System

38.1. 36.1 The Isomorphism

If you have ever stared at a TypeScript type error — `Type 'number' is not assignable to type 'string'` — you already know what CANONIC governance feels like. VOCAB.md declares your types. CANON.md declares your contracts. `magic validate` is your compiler. And 255 is your program compiling clean.

That is not an analogy. It is a structural isomorphism — five components mapping one-to-one [3]:

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

The mapping is bijective — every element on the left has exactly one corresponding element on the right. When you encounter a TypeScript type error, you fix the type declaration. When you encounter a vocabulary violation in CANONIC, you fix the VOCAB.md definition. The mental model is identical.

```
TypeScript:
  const name: string = 42;
  // Type error: Type 'number' is not assignable to type 'string'
```

```
  // Fix: const name: string = "Dr. Chen";

CANONIC:
  Used SCREAMING\_CASE term FHIR\_ENDPOINT not in VOCAB.md
  // Vocabulary violation: term not defined
  // Fix: Add FHIR\_ENDPOINT definition to VOCAB.md
```

Both systems reject undefined terms at compile time, produce a binary outcome (compiles or does not compile), and guarantee that the output satisfies the declared constraints.

38.2. 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, and the validator enforces this through three rules: VOCABULARY CLOSURE, INHERITANCE INTEGRITY, and COVERAGE ALIGNMENT [3].

Rule 1: VOCABULARY CLOSURE. Every SCREAMING_CASE term used in any governance file must be defined in VOCAB.md or inherited from an ancestor. No undefined terms. Equivalent to the rule in typed languages that every variable must have a declared type.

```
magic validate hadleylab-canonic/SERVICES/FHIR-API
# VOCABULARY CLOSURE check:
#   FHIR\_ENDPOINT — defined in VOCAB.md (local) □
#   AUTHENTICATION — defined in VOCAB.md (local) □
#   TRIAD — inherited from canonic-canonic □
#   PATIENT\_RESOURCE — NOT DEFINED □
# ERROR: Vocabulary violation: PATIENT\_RESOURCE undefined
```

Rule 2: INHERITANCE INTEGRITY. A child scope cannot redefine a term defined by a parent. If `canonic-canonic` defines TRIAD, `hadleylab-canonic` cannot redefine it. Equivalent to the Liskov Substitution Principle — a subtype cannot weaken the guarantees of its parent type.

```
canonic-canonic/VOCAB.md:
  TRIAD: The three mandatory files (CANON.md, README.md, VOCAB.md)

hadleylab-canonic/VOCAB.md:
  TRIAD: Just CANON.md  ← REJECTED: redefines parent term
```

Rule 3: COVERAGE ALIGNMENT. The terms used in COVERAGE.md must align with the dimensions declared in CANON.md. Missing coverage for a declared dimension is a semantic error.

Together, these three rules create a type system that is sound (if it compiles to 255, it satisfies all constraints), complete (every violation is detected — no false negatives), and decidable (validation terminates in O(n) time). Compare that to programming language type systems:

Type System	Sound?	Complete?	Decidable?
C	No (void* casts)	No	Yes
Java	Mostly (generics erasure)	No	Yes
TypeScript	No (any type)	No	Yes
Haskell	Yes (in practice)	No (halting)	Yes (usually)
CANONIC	Yes	Yes	Yes (O(n))

CANONIC achieves soundness and completeness because its type universe is finite — 8 dimensions, bounded vocabulary. Programming languages face undecidability because their type universes are infinite [3].

38.3. 36.3 The Compilation Target

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

The gradient is a vector across 8 binary dimensions, where each missing dimension contributes `2^i` to the loss. Fix the highest-weighted missing dimension and you get the steepest descent toward 255 — literal gradient descent on a discrete landscape [3].

Five properties make this optimization tractable:

```
1. Bounded:         0 ≤ loss ≤ 255
2. Non-negative:    loss ≥ 0
3. Zero at target: loss = 0 □  score = 255
4. Decomposable:    loss = SUM(missing_dimension_weights)
5. Monotone:        adding a dimension never increases loss
```

Property 5 is the critical one. In machine learning, adding a feature can increase loss through overfitting. In CANONIC, adding a dimension always reduces loss — the landscape is convex over the discrete lattice. With only 2^8 = 256 possible states, the optimizer (`heal()`) can enumerate all transitions from any state and select the steepest descent.

38.4. 36.4 The Six Theorems

The CANONIC-PAPER [3] proves six theorems that establish 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.

Theorem 1: Compiler Correspondence. A bijective mapping exists between compiler components and governance validator components. The proof is constructive — the mapping in Section 36.1's table exhibits it explicitly [3].

Theorem 2: Validation Decidability. For any scope S, the question "does S compile to 255?" is decidable in O(n) time, where n is the number of governance files. The validator reads each file once, checks each constraint once, and produces a score — no backtracking, no exponential blowup.

```
Time complexity: O(n) where n = |governance files|
Space complexity: O(n) for file contents + O(1) for score
Termination: guaranteed (no recursion, no unbounded loops)
```

Theorem 3: Monotonic Accumulation. If parent P has score S_P and child C inherits from P, then score(C) >= score(P). A child cannot weaken its parent — the Liskov Substitution Principle applied to governance. If a hospital system requires HIPAA compliance at the parent level, no department can opt out at the child level.

Theorem 4: Score Decomposition. 255 decomposes uniquely as 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128. Each dimension is independent. The score is a bitfield.

Theorem 5: Gradient Convergence. `heal()` applied iteratively converges to 255 in at most 8 steps (one per dimension), activating the highest-weight missing dimension at each step. Convergence is guaranteed because loss is strictly decreasing and bounded below by 0.

Theorem 6: Economic Coupling. Every positive gradient produces MINT:WORK — compilation output (score) is economically coupled to governance input (work). Governance and economics are two views of the same process [312].

38.5. 36.5 CANON.md as Type Declaration

CANON.md is the type declaration of a scope. Compare it to a familiar type declaration:

```typescript
// TypeScript: type declaration
interface FHIREndpoint {
  url: string;
  authentication: "oauth2" | "smart-on-fhir";
  resources: ResourceType[];
  version: "R4" | "R5";
}
```

```yaml
# CANON.md: governance type declaration
axiom: "FHIR endpoints are governed by authentication and resource coverage"
constraints:
  - AUTHENTICATION must specify OAuth2 or SMART-on-FHIR
```

```
  - RESOURCE\_COVERAGE must list all supported resource types
  - VERSION must declare FHIR R4 or R5
dimensions: [AXIOM, SCOPE, LANGUAGE, EVIDENCE, OPERATIONS, DEPLOYMENT, LEARNING, ROADMAP]
```

Both declare what the entity must contain, both are checked at compile time, and both produce errors when the implementation does not match the declaration [3].

38.6. 36.6 Clinical Type System Example

A hospital's governance type hierarchy:

```
canonic-canonic (root type — defines TRIAD, 8 dimensions)
  └── hadleylab-canonic (healthcare type — adds HIPAA, FHIR vocabulary)
        ├── SERVICES/EHR-INTEGRATION (concrete type — EHR constraints)
        ├── SERVICES/FHIR-API (concrete type — FHIR constraints)
        ├── SERVICES/COMPLIANCE (concrete type — compliance constraints)
        └── SERVICES/RADIOLOGY-AI (concrete type — AI governance)
```

Each level adds constraints; no level removes them. A FHIR-API scope inherits all hospital-level constraints (HIPAA, identity verification) and adds FHIR-specific ones (resource coverage, authentication protocol). The validator checks all constraints at all levels — identical to class inheritance in OOP: `class FHIRService extends HospitalService extends GovernedScope` [36].

38.7. 36.7 Type Inference in Governance

Programming languages infer types when declarations are absent; CANONIC infers governance properties through inheritance:

```
TypeScript type inference:
  const x = 42;          // inferred: number
  const y = "patient";   // inferred: string
  const z = [x, y];      // inferred: (number | string)[]

CANONIC governance inference:
  child scope has no VOCAB.md    → inherits parent VOCAB.md
  child scope has no constraints → inherits parent constraints
  child scope has no axiom       → ERROR: axiom is required (not inferrable)
```

The axiom is the one property that cannot be inferred — every scope must declare its own, just as typed languages require explicit function signatures because the compiler cannot infer intent.

Property	Inferrable?	Source	Programming Equivalent
Axiom	No — must declare	Local CANON.md	Function signature
Vocabulary	Yes — from parent	Ancestor VOCAB.md	Imported types
Constraints	Yes — from parent	Ancestor CANON.md	Interface requirements
Dimensions	Yes — union of ancestors	Inheritance chain	Type union
Score floor	Yes — parent minimum	Parent score	Subtype constraint

Here is the inference algorithm in action:

```
magic validate --show-inference hadleylab-canonic/SERVICES/FHIR-API
# Inference trace:
#   axiom: "FHIR endpoints governed by auth and resource coverage" (LOCAL)
#   TRIAD: inherited from canonic-canonic (INFERRED)
#   HIPAA\_COMPLIANCE: inherited from hadleylab-canonic (INFERRED)
#   FHIR\_ENDPOINT: defined locally (LOCAL)
#   AUTHENTICATION: defined locally (LOCAL)
#   score_floor: 255 (parent requires 255) (INFERRED)
\#
# Local definitions: 3
# Inherited definitions: 47
# Inference ratio: 94% inherited, 6% local
```

Notice the inference ratio: 94% inherited, 6% local. Most governance flows from the hierarchy. A new scope needs only a few local declarations — the CANONIC equivalent of "convention over configuration," but enforced by the type system rather than social agreement.

38.8. 36.8 Type Narrowing and Refinement

TypeScript narrows `string | number` to `string` after a type guard. CANONIC narrows governance through specialization:

```
// TypeScript type narrowing
function process(input: string | number) {
  if (typeof input === "string") {
    // input is narrowed to string
    return input.toUpperCase();
  }
```

```
  // input is narrowed to number
  return input * 2;
}
```

```
CANONIC scope narrowing:
  hadleylab-canonic (broad: all healthcare governance)
    └── SERVICES/RADIOLOGY-AI (narrow: AI-specific governance)
          Adds: AI\_CONFIDENCE, MODEL\_VERSION, TRAINING\_DATA
          Preserves: HIPAA\_COMPLIANCE, PATIENT\_CONSENT, AUDIT\_LOG
          Cannot remove: any parent constraint
```

The directions are mirror images. In TypeScript, narrowing eliminates possibilities (string | number string). In CANONIC, narrowing adds constraints (general healthcare specific AI governance). Both operations make the type more specific, both are enforced at compile time, and both prevent runtime violations.

The narrowing depth is bounded:

```
Maximum narrowing depth = inheritance chain length
Typical chain: root → org → department → service = 4 levels
Each level adds ≥ 1 constraint
Score floor increases monotonically: 0 → 127 → 191 → 255
```

38.9. 36.9 Generic Types and Parameterized Governance

Just as generics let you write code that works across types, CANONIC templates let you write governance that works across domains:

```
// TypeScript generic
interface Repository<T> {
  find(id: string): T;
  save(entity: T): void;
  delete(id: string): void;
}
// Instantiated: Repository<Patient>, Repository<Study>
```

```
# CANON.md template (parameterized governance)
axiom: "{DOMAIN} data is governed by access control and audit"
constraints:
  - ACCESS\_CONTROL must specify role-based permissions
  - AUDIT\_LOG must record all read and write operations
  - DATA\_RETENTION must comply with {REGULATION}
parameters:
```

```
  DOMAIN: [radiology, cardiology, oncology, pathology]
  REGULATION: [HIPAA, GDPR, PIPEDA]
```

When instantiated for radiology under HIPAA:

```
magic instantiate CANON-TEMPLATE.md \
  --domain radiology \
  --regulation HIPAA \
  --output SERVICES/RADIOLOGY/CANON.md
# Generated:
#   axiom: "Radiology data is governed by access control and audit"
#   ACCESS\_CONTROL: role-based (HIPAA minimum)
#   AUDIT\_LOG: 6-year retention (HIPAA requirement)
#   DATA\_RETENTION: HIPAA-compliant (minimum 6 years)
```

A hospital with 12 departments writes one template and instantiates it 12 times — each instantiation inherits the template constraints and adds department-specific vocabulary. DRY, applied to governance [6].

38.10. 36.10 Type Errors in Governance: A Taxonomy

Every type error in programming has a governance equivalent:

#	Programming Type Error	Governance Equivalent	Severity	Fix
1	Undefined variable	Undefined VOCAB term	FATAL	Add to VOCAB.md
2	Type mismatch	Dimension mismatch	FATAL	Fix COVERAGE.md
3	Missing return	Missing dimension	ERROR	Add dimension file
4	Null dereference	Empty governance file	ERROR	Add content
5	Unused import	Unused VOCAB term	WARNING	Remove from VOCAB

#	Programming Type Error	Governance Equivalent	Severity	Fix
6	Implicit any	Inherited without local	INFO	Add local definition
7	Circular reference	Circular inheritance	FATAL	Break cycle
8	Version mismatch	Parent score regression	FATAL	Restore parent score

The validator reports them with the same precision as a compiler:

```
magic validate hadleylab-canonic/SERVICES/FHIR-API
# ERROR [36.10.1] Undefined VOCAB term: SMART\_LAUNCH
#   at COVERAGE.md:17
#   Fix: Add SMART\_LAUNCH to VOCAB.md
\#
# ERROR [36.10.3] Missing dimension: LEARNING
#   Required by: CANON.md (all 8 dimensions mandatory)
#   Fix: Create LEARNING.md with at least one entry
\#
# WARNING [36.10.5] Unused VOCAB term: LEGACY\_ENDPOINT
#   Defined in VOCAB.md:34 but not referenced in any file
#   Fix: Remove from VOCAB.md or add reference
\#
# Score: 191 (0b10111111) — LEARNING dimension missing
# Status: DOES NOT COMPILE
```

Each error message contains an error code, location, description, and fix. The fix is always a governance file edit, never a code change — just as type errors are fixed by editing type declarations, not runtime logic [3].

38.11. 36.11 The Complete Type-Check Pipeline

The full type-check pipeline mirrors a compiler's front end in five phases:

```
Phase 1: PARSE     — Read all governance files, build AST
Phase 2: RESOLVE   — Resolve inheritance chain, merge vocabularies
Phase 3: CHECK     — Verify vocabulary closure, dimension coverage
Phase 4: SCORE     — Compute 8-bit score from dimension presence
Phase 5: EMIT      — Output CANON.json (compiled governance artifact)
```

Each phase can fail independently:

```
# Phase 1 failure: PARSE
magic validate broken-scope/
# PARSE ERROR: CANON.md not found
# Cannot proceed to Phase 2

# Phase 2 failure: RESOLVE
magic validate orphan-scope/
# RESOLVE ERROR: inherits: nonexistent-parent — parent not found
# Cannot proceed to Phase 3

# Phase 3 failure: CHECK
magic validate incomplete-scope/
# CHECK ERROR: PATIENT\_ID used but not in VOCAB.md
# Cannot proceed to Phase 4

# Phase 4 success with warnings
magic validate almost-scope/
# SCORE: 191 (missing LEARNING)
# WARNING: Score < 255, COIN will not be minted

# Phase 5 success
magic validate complete-scope/
# SCORE: 255
# EMIT: CANON.json written
# COIN: MINT:WORK eligible
```

The pipeline is deterministic: given the same input files, it always produces the same score. That is the defining property of a compiler — deterministic transformation from source to target [311].

38.12. 36.12 Clinical Vignette: Type Safety Prevents Deployment Error

AdventHealth's oncology informatics team deploys OncoChat — a governed TALK agent for NCCN guideline navigation serving oncologists with breast cancer staging recommendations. The VOCAB.md defines 34 terms including STAGING (TNM classification per AJCC 8th Edition), NCCN_CATEGORY (1, 2A, 2B, 3), and BIOMARKER_PANEL (ER, PR, HER2, Ki-67).

During a routine evidence base update, a junior developer adds a new constraint to CANON.md: `MUST: Cite GENOMIC_ASSAY results for all Stage I–II HR+ recommendations.` The term GENOMIC_ASSAY is used but never added to VOCAB.md. The pre-commit hook catches it:

```
magic validate SERVICES/TALK/ONCOCHAT
# CHECK ERROR: Undefined VOCAB term: GENOMIC\_ASSAY
#   at CANON.md:14
#   Fix: Add GENOMIC\_ASSAY to VOCAB.md with precise definition
# Score: 0 (DOES NOT COMPILE — vocabulary violation)
# Commit: BLOCKED
```

Blocked. The developer adds the definition to VOCAB.md:

```
| GENOMIC\_ASSAY | Multi-gene expression test (Oncotype DX 21-gene RS, MammaPrint 70-gene signature, or Pr
```

The developer re-commits and `magic validate` passes. Notice the precision the VOCAB forced: the definition names three specific assays (Oncotype DX, MammaPrint, Prosigna), specifies the patient population (ER+/HER2-), and cites the evidence category (NCCN Category 1). A vague definition like "genetic test for cancer" would not survive clinical review [3].

Without the type system, that undefined GENOMIC_ASSAY term ships to production. An oncologist queries OncoChat about Stage I HR+ breast cancer treatment. The response includes "GENOMIC_ASSAY results recommend…" — but what constitutes a GENOMIC_ASSAY in this context? Oncotype DX only? All three assays? Foundation Medicine comprehensive panels? The ambiguity is a patient safety issue. The type system prevents it.

38.13. 36.13 Governance Proof: The Type Safety Chain

The isomorphism produces a verifiable proof chain:

```
VOCAB.md defines terms (type declarations)
  → CANON.md uses terms in constraints (type usage)
    → magic validate checks term resolution (type checking)
      → Score = 255 means all terms resolve (program compiles)
        → LEDGER records the validation event (compilation receipt)
          → COIN mints for positive gradient (economic coupling)
```

Every link is auditable. The LEDGER entry for a 255 validation contains the timestamp, scope, score, and constraint count. An auditor can reconstruct the full type-check — which terms were defined, which constraints were checked, which dimensions were present — and the reconstruction is deterministic. Given the governance files at that commit, `magic validate` always produces the same score. The type system is the governance system. The compiler is the validator. 255 is the proof that the governance program compiles. Q.E.D. This chapter establishes the isomorphism; Chapter 37 (Governance as Compiler) details the compiler pipeline, and Chapter 38 (Governance as Version Control) extends the analogy to version control. For the governance policy perspective on type safety, see The Canonic Canon, Chapters 8-10 [36].

38.14. 36.14 Type Erasure and Runtime Behavior

After compilation, types can be erased — the runtime does not need to carry type annotations because the compiler already verified correctness. CANONIC exhibits the same property. Once a scope reaches 255, the runtime (the clinical AI agent serving responses) does not re-check every VOCAB term on every request. The compilation already verified term resolution.

This is not blind trust. The governance tree is immutable at a given commit, and the compilation is deterministic. If the tree changes, recompilation is required — the pre-commit hook enforces this automatically. Between commits, the compiled state is authoritative [36].

The practical consequence is runtime performance. Your clinical AI agent does not perform VOCAB lookups on every response. The VOCAB was validated at build time, the systemPrompt was compiled from governed sources, and the response templates were generated from CANON.md. All type-checking happened before deployment. The agent serves responses at edge latency, not validation latency [6].

Chapter 39

Chapter 37: Governance as Compiler

39.1. 37.1 The Compiler Pipeline

Chapter 36 established the type system isomorphism. Now consider the compiler itself — the pipeline that transforms governance source files into a validated score, exactly as `gcc` transforms C source into machine code. Where Chapter 36 answered "what are the types?", this chapter answers "what is the compiler?" The concrete implementation of this compiler is the `magic` binary documented in Chapter 42, and the build pipeline that invokes it is documented in Chapter 44. For the governance policy perspective on compilation, see The Canonic Canon.

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

Five phases carry a commit from source to deployment [3]: 1. **Parse**: Read governance files, resolve `inherits:` 2. **Compile**: Compute score against target (255) 3. **Mint**: gradient > 0 □ MINT:WORK; gradient < 0 □ DEBIT:DRIFT 4. **Ledger**: Immutable record (build log) 5. **Ship**: Compiles = ships. Does not compile = does not ship. No waivers.

Phase 1: Parse. The validator reads the scope directory, identifies governance files by name convention, and resolves the `inherits:` chain.

```
magic validate --verbose hadleylab-canonic/SERVICES/FHIR-API
# Phase 1: PARSE
#   Reading CANON.md... found
#   Reading README.md... found
#   Reading VOCAB.md... found
#   Resolving inherits: hadleylab-canonic... resolved
#   Parse complete. 6 local files, 2 ancestors.
```

Phase 2: Compile. The validator computes the 8-bit score, checking each dimension independently.

```
# Phase 2: COMPILE
#   AXIOM (1): □   SCOPE (2): □   LANGUAGE (4): □   EVIDENCE (8): □
#   OPERATIONS (16): □   DEPLOYMENT (32): □   LEARNING (64): □   ROADMAP (128): □
#   Score: 255
```

Phase 3: Mint. Retrieve previous score, compute gradient, emit event.

Phase 4: Ledger. Write event to LEDGER (dual-write to .md and .json).

Phase 5: Ship. Score 255 = shippable. Score < 255 = blocked.

39.2. 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

Each error type has a distinct diagnostic and fix path.

Syntax errors (missing files). The TRIAD files are mandatory — missing any one means the scope cannot be parsed.

```
magic validate hadleylab-canonic/SERVICES/NEW-SERVICE
# ERROR: Syntax error — CANON.md not found
# Fix: Create CANON.md with axiom and inherits declarations
# Run: magic-heal hadleylab-canonic/SERVICES/NEW-SERVICE
```

Type errors (undefined terms). A SCREAMING_CASE term is used but not defined.

```
# ERROR: Type error — PATIENT\_RESOURCE used in COVERAGE.md:14
#   but not defined in VOCAB.md or any ancestor VOCAB.md
# Fix: Add PATIENT\_RESOURCE definition to VOCAB.md
```

Linker errors (broken inheritance). The `inherits:` path points to a non-existent scope.

```
# ERROR: Linker error — inherits: nonexistent/scope
# Fix: Correct the inherits: path in CANON.md
```

Semantic errors (missing dimensions). The scope parses and links but does not reach 255.

```
# WARNING: Semantic error — Score 127/255
#   Missing: LEARNING (64), ROADMAP (128)
# Fix: Create LEARNING.md and ROADMAP.md
```

Regression errors (DEBIT:DRIFT). The scope previously scored higher. Something was removed.

```
# ERROR: Regression — Score dropped 255 → 127
#   DEBIT:DRIFT: -128 COIN
#   Cause: LEARNING.md deleted in commit 6b3c8d1
```

39.3. 37.3 Continuous Governance

The pre-commit hook fires on every change, delivering feedback in seconds rather than months. The traditional audit becomes obsolete — the compiler is the auditor [3].

The hook is installed by `magic init`:

```
magic init hadleylab-canonic/SERVICES/FHIR-API
# Installing pre-commit hook...
# Every commit will run: magic validate <changed scopes>
```

Compare to traditional governance audit cycles:

Traditional Audit	Continuous Governance
Annual survey (12 months)	Every commit (seconds)
External auditor (expensive)	Automated validator (free)
Sample-based (incomplete)	Exhaustive (every scope)
Subjective scoring	Deterministic scoring
Report in PDF (static)	LEDGER in git (versioned)
Pass/fail (binary)	0-255 (granular)
Corrective action plan (months)	`magic-heal` (immediate)

Every commit is an audit. Compliance is not a periodic event — it is a continuous state [3].

39.4. 37.4 The Optimization Model

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

`heal()` operates as backpropagation: the forward pass is `validate()`, the loss is `255 - bits`, backpropagation is `heal()`, and the weight update is a pattern adjustment [10].

```
function heal(scope):
  score = validate(scope)
  while score < 255:
    missing = identify_missing_dimensions(scope)
    highest = max(missing, key=weight)
    generate_file(scope, highest)
    score = validate(scope)
  return score
```

Convergence takes at most 8 iterations (one per dimension), though in practice `heal()` prioritizes the highest-weight dimension and converges in 3-4.

```
magic-heal hadleylab-canonic/SERVICES/NEW-SERVICE
# Iteration 1: Generating CANON.md, README.md, VOCAB.md... score: 0→7
# Iteration 2: Generating COVERAGE.md, SPEC.md... score: 7→63
# Iteration 3: Generating LEARNING.md, ROADMAP.md... score: 63→255
# Converged in 3 iterations.
```

39.5. 37.5 Scope as Compilation Unit

In C the compilation unit is the `.c` file; in Java, the `.java` file; in CANONIC, the scope directory.

Compiler	Compilation Unit	Output
gcc	`.c` file	`.o` object file
javac	`.java` file	`.class` bytecode
tsc	`.ts` file	`.js` JavaScript
magic validate	scope directory	255-bit score

Each unit is independently compilable, the linker (`inherits:`) connects them, and the build system orchestrates compilation in dependency order.

```
build
# Compiling canonic-canonic... 255 □
# Compiling hadleylab-canonic... 255 □
# Compiling hadleylab-canonic/SERVICES/EHR... 255 □
# Compiling hadleylab-canonic/SERVICES/FHIR... 255 □
# Compiling hadleylab-canonic/SERVICES/COMPLIANCE... 127 □
# Build failed. 4/5 scopes compiled.
```

39.6. 37.6 Clinical Compiler Example

Integrating `magic validate` into a hospital's CI/CD pipeline is straightforward:

```yaml
# .github/workflows/governance.yml
name: Governance Compiler
on: [push, pull_request]
jobs:
  compile:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Install CANONIC
        run: pip install canonic
      - name: Compile all scopes
        run: magic validate --all --strict
      - name: Check conservation
        run: magic wallet --audit
      - name: Deploy if 255
        if: success()
        run: magic deploy --fleet
```

Every pull request runs the governance compiler, and every merge to main triggers deployment. No human reviewer can override a failing compilation [37].

39.7. 37.7 Compiler Passes and Ordering

The governance compiler executes in a defined pass order:

Pass	Name	Input	Output	Failure Mode
1	Discovery	Filesystem	Scope list	Missing directories
2	Parse	Scope list	AST (parsed .md files)	Missing TRIAD
3	Inherit	AST + ancestors	Resolved AST	Broken `inherits:`
4	Validate	Resolved AST	8-bit score	Dimension failures
5	Mint	Score + previous	Gradient events	LEDGER write failure
6	Link	All scores	Cross-ref matrix	Dangling refs
7	Emit	All artifacts	CANON.json	Write failure

```
magic validate --passes hadleylab-canonic/SERVICES/EHR
# Pass 1 (Discovery): 1 scope found □  (0.2ms)
# Pass 2 (Parse):     8 files parsed □  (1.1ms)
```

```
# Pass 3 (Inherit):   2 ancestors resolved □  (0.8ms)
# Pass 4 (Validate):  8/8 dimensions □  (2.3ms)
# Pass 5 (Mint):      gradient 0 (neutral) □  (0.1ms)
# Pass 6 (Link):      4 cross-refs resolved □  (1.2ms)
# Pass 7 (Emit):      CANON.json written □  (0.4ms)
# Total: 255/255 in 6.1ms
```

39.8. 37.8 Compiler Diagnostics

Every error produces a structured diagnostic:

```
{severity}: {error_code} — {message}
  at {file}:{line}
  in scope {scope_path}
  fix: {resolution}
  heal: {magic-heal command}
```

```
magic validate --all --diagnostics

# ERROR: GOV-001 — Missing CANON.md
#   at SERVICES/NEW-SERVICE/
#   fix: Create CANON.md with axiom and inherits
#   heal: magic-heal hadleylab-canonic/SERVICES/NEW-SERVICE

# WARNING: GOV-012 — VOCAB term undefined
#   at COVERAGE.md:14
#   term: PATIENT\_RESOURCE
#   fix: Add to VOCAB.md

# Summary: 1 error, 1 warning, 71 info
```

Diagnostics are machine-parseable. CI systems parse them for PR comments and Slack notifications [3].

39.9. 37.9 Incremental Compilation

```
# Full compilation (cold)
magic validate --all
# Time: 4.2s (73 scopes)
```

```
# Incremental (warm — only changed scopes)
magic validate --incremental
# Changed: SERVICES/EHR/LEARNING.md
# Recompiling: 1 scope
# Time: 0.1s
```

Mode	When	Time (73 scopes)
Full (`--all`)	Nightly, release	4.2s
Incremental	Every commit	0.1s
Single (`--scope`)	Debugging	0.06s

39.10. 37.10 Compiler Extensions

Organizations add custom governance questions for domain-specific validation:

```
# .canonic/extensions.yml
custom:
  - name: HIPAA
    validator: validators/hipaa.py
    required_files: [HIPAA.md]
  - name: IRB
    validator: validators/irb.py
    required_files: [IRB.md]
```

Custom questions extend the scoring beyond the base eight:

```
magic validate --extended SERVICES/TALK/MAMMOCHAT
# Standard: 255/255 □
# Extended: 1023/1023 □  (HIPAA: □ , IRB: □ )
```

39.11. 37.11 The Isomorphism Theorem

The isomorphism is formal correspondence, not metaphor:

Compiler Concept	Governance Concept	Isomorphism
Source code	.md governance files	1:1
Syntax check	TRIAD presence check	1:1
Type check	Dimension validation	1:1

Compiler Concept	Governance Concept	Isomorphism
Linker	`inherits:` resolution	1:1
Object code	CANON.json	1:1
Build error	Score < 255	1:1
CI pipeline	Pre-commit + Actions	1:1
Deployment	`deploy` (gated)	1:1

The isomorphism is total — every compiler concept maps to a governance concept with identical semantics. The validator is a compiler. The 255-bit score is the compilation result [3].

39.12. 37.12 Clinical Vignette: Compiler Catches Cross-Scope Regression

Intermountain Health (Salt Lake City, 33 hospitals) maintains 142 governed scopes across their clinical AI fleet: 8 TALK agents, each with 15-18 sub-scopes covering guidelines, formulary, documentation templates, and quality metrics. The fleet validates nightly via `magic validate --all`.

The Incident. A pharmacist updating PulmoChat's Formulary sub-scope renames the VOCAB term `INHALED_CORTICOSTEROI` to `ICS` for brevity. The change validates locally — PulmoChat/Formulary still scores 255. But that term is referenced in three other scopes: PulmoChat/Guidelines (4 references in COVERAGE.md), PedsChat/Asthma (cross-scope evidence anchor in INTEL.md), and the fleet-level DRUG_INTERACTION_MATRIX (12 interaction rules).

Compiler Detection. The nightly full compilation catches it:

```
magic validate --all --strict
# Pass 1 (Discovery): 142 scopes found □
# Pass 2 (Parse):     1,136 files parsed □
# Pass 3 (Inherit):   284 ancestors resolved □
# Pass 4 (Validate):  139/142 dimensions pass
\#
# ERROR: GOV-012 — VOCAB term undefined
#   at PulmoChat/Guidelines/COVERAGE.md:47
#   term: INHALED\_CORTICOSTEROID
#   defined in: nowhere (removed from PulmoChat/Formulary/VOCAB.md)
#   fix: Restore definition or update all references
\#
# ERROR: GOV-012 — VOCAB term undefined
#   at PedsChat/Asthma/INTEL.md:23
#   term: INHALED\_CORTICOSTEROID
#   cross-scope reference from PulmoChat/Formulary
\#
```

```
# ERROR: GOV-012 — VOCAB term undefined
#   at DRUG\_INTERACTION\_MATRIX/COVERAGE.md:88,91,94,97,100,103,106,109,112,115,118,121
#   term: INHALED\_CORTICOSTEROID (12 occurrences)
\#
# Build failed: 3 scopes regressed. 15 references broken.
# DEBIT:DRIFT: -24 COIN (3 scopes × 8 COIN per dimension)
```

The Type Error Cascade. The compiler detected an undefined term referenced across scope boundaries. Without it, this rename would propagate silently — guidelines referencing a term that no longer exists, drug interaction matrices with phantom entries, clinicians receiving responses anchored to an undefined governance term. By treating VOCAB terms as types, the compiler ensures that removing a type forces every reference to be updated or the build fails [3].

Resolution. The pharmacist has two choices:

Option A: **Restore the original term.** Add `INHALED_CORTICOSTEROID` back to VOCAB.md. Add `ICS` as an alias: `ICS: See INHALED_CORTICOSTEROID`. Zero references need updating. Cost: 1 commit, 1 COIN.

Option B: **Propagate the rename.** Update all 15 references across 3 scopes. Cost: 4 commits, 15 file edits, COIN minted for the governance work. But this option risks breaking other downstream references not yet detected.

The pharmacist chooses Option A. Full build passes — all 142 scopes at 255. Total resolution time: 22 minutes from detection to green build.

39.13. 37.13 Compiler Optimization: Dead Code Elimination

The governance compiler also supports dead code elimination, identifying governance artifacts that exist but are never referenced:

```
magic validate --dead-code

# Dead code analysis:
# VOCAB.md terms defined but never referenced:
#   SERVICES/LEGACY-EHR/VOCAB.md:
#     CCDA\_DOCUMENT (defined line 14, 0 references)
#     HL7V2\_ADT (defined line 18, 0 references)
#   SERVICES/ARCHIVE/VOCAB.md:
#     DICOM\_WORKLIST (defined line 7, 0 references)
\#
# COVERAGE.md items with no evidence:
#   SERVICES/TALK/MEDCHAT/COVERAGE.md:
#     MENTAL\_HEALTH\_SCREENING (line 34, no INTEL reference)
\#
```

```
# Total dead code: 4 items across 3 scopes
# Recommendation: Remove or reference. Dead governance is noise.
```

Dead code does not reduce the score — the dimensions are still satisfied. But dead governance terms accumulate as technical debt, confusing new governors who encounter terms with no context and bloating VOCAB.md files. Like its counterpart in optimizing compilers, the dead code analysis does not block the build; it surfaces cleanup opportunities.

39.14. 37.14 Compiler Warnings vs Errors

The governance compiler distinguishes between errors (build-blocking) and warnings (informational):

Level	Code Range	Effect	Example
ERROR	GOV-001 to GOV-099	Build blocked	Missing CANON.md
WARNING	GOV-100 to GOV-199	Build succeeds, logged	Dead VOCAB term
INFO	GOV-200 to GOV-299	Build succeeds, silent	Scope at 255 for 90+ days
HINT	GOV-300 to GOV-399	Build succeeds, suggestion	LEARNING.md has < 3 entries

```
magic validate --all --diagnostics --level warning

# 0 errors, 3 warnings, 12 info, 8 hints
\#
# WARNING: GOV-101 — VOCAB term referenced but deprecated
#   at SERVICES/EHR/COVERAGE.md:22
#   term: HL7V2\_MESSAGE (deprecated in VOCAB.md:45)
#   suggestion: Use FHIR\_BUNDLE instead
\#
# WARNING: GOV-112 — LEARNING.md not updated in 60 days
#   at SERVICES/TALK/MEDCHAT/LEARNING.md
#   last_updated: 2026-01-10
#   suggestion: Add recent patterns or confirm no new learnings
\#
# WARNING: GOV-115 — Inheritance chain depth = 5
#   at SERVICES/TALK/MAMMOCHAT/SCREENING/BI-RADS/CATEGORY-4
#   suggestion: Consider flattening (max recommended: 6)
```

Warnings do not block deployment but accumulate in the LEDGER as GOV:WARNING events. A scope with more than 10 unresolved warnings is flagged for governance review via NOTIFIER alert.

39.15. 37.15 Compiler Performance Profiling

For large fleets, compilation performance matters. The compiler profiles itself:

```
magic validate --all --profile

# Compilation profile (142 scopes):
\#
# Phase           Time      % Total   Scopes/sec
# Discovery       0.12s     2.0%      1,183
# Parse           0.89s     14.7%     160
# Inherit         0.67s     11.1%     212
# Validate        2.41s     39.8%     59
# Mint            0.15s     2.5%      947
# Link            1.24s     20.5%     115
# Emit            0.57s     9.4%      249
# ──────────────────────────────────────
# Total           6.05s     100%      23.5 scopes/sec
\#
# Bottleneck: Validate phase (39.8%)
#   Hotspot: DRUG\_INTERACTION\_MATRIX (0.84s alone)
#   Reason: 12 cross-scope VOCAB references, each requiring ancestor walk
#   Optimization: Cache ancestor VOCAB resolution
```

The profiler pinpoints bottlenecks at the scope level. Here, the DRUG_INTERACTION_MATRIX takes 0.84 seconds because each of its 12 cross-scope VOCAB references requires walking the `inherits:` chain. Caching ancestor VOCAB resolution drops the validate phase from 2.41s to 0.93s — a 61% improvement.

39.16. 37.16 Governance Proof: Compiler Correctness

The governance compiler is correct if and only if it satisfies five properties:

1. **Soundness**: If `magic validate` returns 255, the scope satisfies all 8 dimensions. No false positives.
2. **Completeness**: If the scope satisfies all 8 dimensions, `magic validate` returns 255. No false negatives.
3. **Determinism**: Given identical input (same files, same ancestors), `magic validate` always returns the same score. No randomness.
4. **Monotonicity**: Adding a governance file never reduces the score (only adding can set bits, not unset them). Removing can reduce.
5. **Compositionality**: The score of a composed scope is the bitwise AND of its constituents' dimension contributions. No emergent failures.

Proof sketch for soundness:

```
For each dimension d in {D, E, S, O, T, R, LANG, L}:
  validator_d(scope) checks a specific file presence and structure
  If validator_d returns TRUE, the bit is set
  If all 8 validators return TRUE, score = 255
  Each validator checks necessary and sufficient conditions
  Therefore: score = 255 implies all conditions met
  Soundness: proved by construction of each validator
```

The C kernel (`magic_score()` function) computes the score as `uint8_t` — an unsigned 8-bit integer whose type enforces the range [0, 255]. Overflow and underflow are both impossible. Each dimension sets exactly one bit. The compiler is correct by construction. Q.E.D. [37]

39.17. 37.17 Compiler Extensions and Custom Validators

The compiler is extensible. Organizations register custom validators that run alongside the 8 core dimension checks. Custom validators do not affect the 255 score — they produce supplementary results recorded on the LEDGER as `VALIDATE:CUSTOM` events.

```
# Register a custom validator for radiology compliance
magic validate --register-custom hipaa-radiology \
  --check "INTEL.md contains ACR reference" \
  --check "VOCAB.md defines DICOM terms" \
  --check "COVERAGE.md includes PHI handling"

# Run with custom validators
magic validate --all --custom

# Core validation:    73/73 at 255 □
# Custom validators:
#   hipaa-radiology:  12/12 scopes pass □
#     ACR references: present in all 12 INTEL.md files
#     DICOM terms:    47 terms defined across 12 VOCAB.md files
#     PHI handling:   all 12 COVERAGE.md files include PHI section
```

Custom validators provide domain-specific depth without modifying the core compiler — radiology adds DICOM checks, pharmacy adds formulary checks, research adds IRB compliance. Each custom validator is itself governed: registered via a CANON.md extension, validated by the core compiler, and recorded on the LEDGER [37].

39.18. 37.18 Clinical Vignette: Compiler Catches Cross-Scope Type Mismatch

Mount Sinai Health System (New York, 8 hospitals) deploys OncoChat across all campuses for tumor board queries. A clinical informaticist at Mount Sinai West adds a constraint: "All genomic recommendations must cite GENOMIC_PANEL_RESULT from the institutional lab."

The compiler catches it:

```
magic validate SERVICES/TALK/ONCOCHAT
# ERROR: GOV-041 — Term GENOMIC\_PANEL\_RESULT not in scope VOCAB chain
#   Referenced at: CANON.md:47
#   Expected in:   VOCAB.md (local or inherited)
#   Found in:      SERVICES/LAB/GENOMICS/VOCAB.md (not in inheritance chain)
#   Fix:           Add inherits: hadleylab-canonic/SERVICES/LAB/GENOMICS
#                  to OncoChat's CANON.md, or copy the term definition locally
```

The compiler enforces module boundaries: OncoChat cannot reference a term defined in LAB/GENOMICS unless it explicitly inherits from that scope. The `inherits:` declaration is the import statement. Every term reference must resolve through the declared dependency graph — no implicit imports, no ambient definitions, no cross-scope contamination [372].

Chapter 40

Chapter 38: Governance as Version Control

40.1. 38.1 Git IS the Governance Engine

This chapter completes the governance-as-compilation trilogy begun in Chapters 36 and 37 by showing that git itself is the governance engine. Where Chapter 36 established the type-system isomorphism and Chapter 37 formalized the compiler, this chapter demonstrates that version control provides the remaining properties — immutability, attribution, and ordering — that close the governance loop. The federation mechanism via git submodules connects to Chapter 9, and the LEDGER-as-git-history model grounds the economic primitives of Chapters 13-18. For the governor's perspective on version control as institutional memory, see the CANONIC CANON.

You already have a governance engine installed on your machine. It is called `git`.

Every governance event is a commit. Every commit is validated. Every validation produces a score. Every positive gradient mints COIN. The LEDGER is the commit history annotated with economic metadata [3]. Git provides exactly the five properties that governance requires:

Property	Git Feature	Governance Use
Immutability	Commit hashes (SHA-256)	Every governance event is permanent
Attribution	`git log --format="%an %ae"`	Every change has an identity
Ordering	Commit timestamps	Events are temporally ordered
Branching	`git branch`	Parallel governance work
Merging	`git merge`	Governance integration

No external system is needed. The LEDGER is not a separate database — it is the git history itself. Every COIN traces to a commit hash, and every commit hash traces to an identity.

```
git log --oneline --format="%h %s [%an]" SERVICES/FHIR-API/
# a1b2c3d Add LEARNING.md [Dr. Chen] — MINT:WORK +64 COIN
# 9f1a4b8 Add COVERAGE.md [Dr. Chen] — MINT:WORK +8 COIN
# 3c7d9e2 Add VOCAB.md [Dr. Park] — MINT:WORK +4 COIN
# f8a2b1c Bootstrap TRIAD [Dr. Chen] — MINT:WORK +3 COIN
```

Every line is a governance event. The git log IS the LEDGER.

40.2. 38.2 Certification = Git Tags

`magic-tag` certifies a scope at 255. The tag is immutable, signed, auditable. Registered in TAGS.md — append-only [19].

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

Requirements: - `magic validate` □ 255 - VITAE.md exists (identity gate) - Tag signed (no unsigned tags in production) - TAGS.md updated [19]

The tag is a certification event. It declares: “This scope, at this commit, scores 255, and I attest to its completeness.” The tag is signed with the governor’s GPG key. Unsigned tags are rejected in production.

```
magic-tag v1.0.0 --scope hadleylab-canonic/SERVICES/FHIR-API
# Pre-tag validation:
#   magic validate → 255 □
#   VITAE.md exists → □
#   GPG key available → □
# Creating signed tag v1.0.0...
# Updating TAGS.md...
# Tag created and registered.
```

The TAGS.md file is append-only. Each entry records the tag name, scope, score, commit hash, timestamp, and signer:

```
| Tag | Scope | Score | Commit | Date | Signer |
|-----|-------|-------|--------|------|--------|
| v1.0.0 | SERVICES/FHIR-API | 255 | a1b2c3d | 2026-03-10 | dr.chen@hadleylab.org |
| v1.1.0 | SERVICES/FHIR-API | 255 | b2c3d4e | 2026-03-15 | dr.chen@hadleylab.org |
| v1.0.0 | SERVICES/EHR | 255 | c3d4e5f | 2026-03-12 | dr.park@hadleylab.org |
```

No entry is ever removed. Corrections are new tags. The tag history is a certification timeline.

40.3. 38.3 The Molecular Clock

Commits accumulate at a roughly constant rate — a mutation rate in the evolutionary sense. Just as biologists count DNA mutations between species to estimate divergence time, you can count commits between two scopes to estimate their governance divergence [27].

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

```
magic clock hadleylab-canonic/SERVICES/FHIR-API hadleylab-canonic/SERVICES/EHR
# Scope A: SERVICES/FHIR-API (142 commits since fork)
# Scope B: SERVICES/EHR (98 commits since fork)
# Common ancestor: hadleylab-canonic (fork: 2026-01-01)
# Divergence: 240 commits
# Estimated governance distance: HIGH
```

The clock measures governance velocity: a scope with 10 commits per week evolves faster than one with 1 commit per month.

40.4. 38.4 LEDGER as Append-Only Log

The LEDGER is append-only — new entries are added, but old entries are never modified or deleted. This is the same property that makes git commits immutable: once a commit is made, its hash is fixed.

```
LEDGER properties:
  1. Append-only: new entries at the end, no modifications
  2. Ordered: entries sorted by commit timestamp
  3. Complete: every governance event has an entry
  4. Verifiable: entry hashes chain to commit hashes
  5. Reconstructible: replay from genesis yields current state
```

The LEDGER is the single source of truth for the COIN economy — WALLET balances are derived from it, conservation equations are verified against it, and if it remains intact, the entire economic state can be reconstructed from scratch.

40.5. 38.5 Branching as Governance Experiments

Git branches enable governance experiments — create a branch, try a new structure, validate. If it compiles to 255, merge. If not, discard.

```
git checkout -b experiment/new-compliance-structure
# Restructure SERVICES/COMPLIANCE into 5 sub-scopes
magic validate --all
# All 5 sub-scopes: 255 □
git checkout main
git merge experiment/new-compliance-structure
# Merge complete. MINT:WORK events for 5 new scopes.
```

Failed experiments cost nothing — no COIN is minted until validated changes merge to main. The branch is a sandbox for structural governance exploration.

40.6. 38.6 Git Blame as Governance Attribution

`git blame` gives you governance attribution for free — who wrote each line of each governance file, permanently and automatically.

```
git blame SERVICES/FHIR-API/VOCAB.md
# a1b2c3d (Dr. Chen 2026-01-15) FHIR\_ENDPOINT: A governed REST API...
# 3c7d9e2 (Dr. Park 2026-02-01) AUTHENTICATION: OAuth2 or SMART...
# 9f1a4b8 (Dr. Chen 2026-02-15) RESOURCE\_COVERAGE: List of all...
```

Every vocabulary term has an author. Every constraint has an author. No manual tracking required.

40.7. 38.7 Clinical VCS Example

A radiology department uses git as their governance VCS:

```
hadleylab-canonic/SERVICES/RADIOLOGY/
├── CANON.md           \# Department axiom and constraints
├── README.md          \# Public interface
├── VOCAB.md           \# 30+ radiology governance terms
├── COVERAGE.md        \# Coverage across 12 sub-services
├── LEARNING.md        \# 18 months of governance patterns
├── ROADMAP.md         \# Next 12 months plan
├── TAGS.md            \# Certification history
├── PACS-INTEGRATION/  \# Sub-scope
├── DICOM-ROUTING/     \# Sub-scope
├── AI-TRIAGE/         \# Sub-scope
└── PEER-REVIEW/       \# Sub-scope
```

The git history shows 450 commits over 18 months. The molecular clock reads 6.25 commits per week — active governance. The TAGS.md shows 12 certification events. The LEDGER shows 3,060 COIN minted (12 scopes * 255). Every commit, every tag, every COIN is permanently recorded in git [27319].

40.8. 38.8 Merge Conflicts as Governance Disputes

When two branches modify the same governance file differently, the merge conflict is the governance dispute — surfaced automatically, requiring deliberate resolution.

```
git merge feature/new-compliance-structure
# CONFLICT (content): Merge conflict in SERVICES/COMPLIANCE/VOCAB.md
# Governor A defined AUDIT\_FREQUENCY as "quarterly"
# Governor B defined AUDIT\_FREQUENCY as "monthly"
# Resolution required.
```

The resolution is the governance decision. The governor who resolves the conflict makes the authoritative choice, and the merge commit records who decided, when, and what they chose. In healthcare, a merge conflict in a compliance VOCAB.md is a disagreement about the meaning of a governance term — and git captures the full decision record.

40.9. 38.9 Git Hooks as Governance Enforcement

Three git hooks enforce governance at the commit level:

Hook	Trigger	Governance Function
`pre-commit`	Before commit	Run `magic validate` — reject if score drops
`commit-msg`	After message written	Verify commit references scope
`pre-push`	Before push	Run full build — reject if any scope < 255

```
# .git/hooks/pre-commit
\#!/bin/bash
magic validate --changed-scopes
if [ $? -ne 0 ]; then
  echo "Governance validation failed. Commit rejected."
  exit 1
fi
```

These hooks run locally on every commit — no CI server needed for basic governance enforcement.

40.10. 38.10 Git Bisect as Governance Debugging

`git bisect` finds the commit that caused a score drop:

```
git bisect start
git bisect bad HEAD          \# Current score is 127
git bisect good v1.0.0       \# v1.0.0 was 255
git bisect run magic validate --expect 255
# First bad commit: 6b3c8d1 (deleted LEARNING.md)
```

The culprit is identified automatically — commit, author, change, and DEBIT:DRIFT amount all recorded. Governance debugging is as precise as code debugging [273].

40.11. 38.11 Clinical Vignette: Version Control Prevents Guideline Collision

NYU Langone Health (New York, 6 hospitals) runs 4 clinical AI agents under CANONIC governance. Two oncologists — Dr. Agarwal (solid tumors) and Dr. Petrov (hematologic malignancies) — both modify OncoChat's INTEL.md on the same day, working on separate feature branches. Dr. Agarwal updates NCCN Non-Small Cell Lung Cancer guidelines (v3.2026); Dr. Petrov updates NCCN Chronic Lymphocytic Leukemia guidelines (v2.2026).

The Collision. Both add their guideline at the same row of the "Active Guidelines" table. Dr. Agarwal's branch merges first:

```
git merge feature/nsclc-v3-2026
# Merge successful. INTEL.md updated.
# magic validate → 255 □
# MINT:WORK +8 COIN (EVIDENCE dimension maintained)
# LEDGER: evt:07200 recorded
```

Dr. Petrov's merge triggers the conflict:

```
git merge feature/cll-v2-2026
# CONFLICT (content): Merge conflict in SERVICES/TALK/ONCOCHAT/INTEL.md
# Auto-merge failed; fix conflicts and then commit the result.
```

Git as Governance Dispute Resolution. Two experts made incompatible changes to the same governance file, and git surfaced the conflict. Dr. Petrov opens INTEL.md, sees both additions, and resolves by placing NSCLC at row 14 and CLL at row 15 (alphabetical ordering per CANON.md convention):

```
git add INTEL.md
git commit -m "GOV: resolve INTEL conflict — NSCLC v3 + CLL v2 both added"
# magic validate → 255 
# MINT:WORK +8 COIN
# LEDGER: evt:07205 recorded
# Attribution: dr.petrov (resolver), dr.agarwal (contributor)
```

The merge commit permanently records who resolved the conflict, when, and what the resolution was — all validated to 255. If a Joint Commission surveyor asks “who decided the ordering of clinical guidelines in your AI system?” the merge commit answers with cryptographic attribution [319].

40.12. 38.12 Version Control Metrics

Git provides governance metrics that no external system can match:

```
magic vcs-report --scope SERVICES/TALK/ONCOCHAT --period 6m

# Version Control Governance Report — OncoChat (6 months)
\#
# Total commits:           347
# Unique contributors:     12 (8 physicians, 3 engineers, 1 compliance)
# Governance files changed: 1,204 (avg 3.5 files per commit)
# Merge conflicts:          7 (all resolved within 4 hours)
# Branches created:         89 (avg lifespan: 2.3 days)
# Tags created:             6 (monthly certification)
# Score history:            255 → 255 (zero drift in 6 months)
\#
# Contributor distribution:
#   dr-agarwal:    87 commits (25%)    NSCLC, lung, thoracic
#   dr-petrov:     64 commits (18%)    CLL, lymphoma, leukemia
#   dr-yamamoto:   52 commits (15%)    breast, GI, melanoma
#   eng-martinez:  41 commits (12%)    infrastructure, tooling
#   [8 others]:    103 commits (30%)   various specialties
\#
# File change frequency:
#   INTEL.md:      147 changes (42%)   Evidence is most dynamic
#   LEARNING.md:   89 changes (26%)    Patterns accumulate steadily
#   VOCAB.md:      45 changes (13%)    Terminology evolves slowly
#   COVERAGE.md:   38 changes (11%)    Coverage is stable
#   CANON.md:      12 changes (3%)     Axiom rarely changes
#   Other:         16 changes (5%)     ROADMAP, SPEC, etc.
```

The report reveals a governance law consistent across all clinical AI agents: **evidence is volatile, axioms**

are stable, terminology is semi-stable. INTEL.md changes most frequently, CANON.md changes least, and VOCAB.md evolves slowly after initial definition [27].

40.13. 38.13 Rebase vs Merge: Governance Implications

CANONIC governance mandates merge commits over rebases for governance-significant branches:

Operation	Git Command	Governance Implication
Merge	`git merge`	Preserves complete branch history; two parents visible
Rebase	`git rebase`	Rewrites history; appears as linear sequence
Squash	`git merge --squash`	Collapses to one commit; loses intermediate governance

```
MUST: Use merge for governance branches (preserves attribution)
MUST NOT: Rebase governance branches (destroys intermediate COIN events)
MAY: Squash for trivial fixes (typos, formatting)
```

The prohibition is economic. Each intermediate commit may have minted COIN. Rebasing rewrites commit hashes, breaking the LEDGER's references — COIN events become orphaned, pointing to commits that no longer exist. That is an integrity violation.

```
# WRONG: Rebase governance branch
git rebase main
# WARNING: Rebase detected on governance branch.
# 3 MINT:WORK events reference commits that will be rewritten.
# LEDGER integrity will be compromised.
# Abort? [Y/n]
```

The pre-rebase hook detects governance branches (branches containing commits that triggered MINT:WORK events) and warns before proceeding. In production, the hook aborts automatically [3].

40.14. 38.14 Cherry-Pick as Governance Backport

When a governance improvement on one scope applies to another, `git cherry-pick` backports the change:

```
# OncoChat improved its DISCLAIMER section. Apply to CardiChat.
git log --oneline SERVICES/TALK/ONCOCHAT/CANON.md
# a8f3b2c Add cardiac risk disclaimer template
```

```
git checkout feature/cardichat-disclaimer
git cherry-pick a8f3b2c
# Applying: Add cardiac risk disclaimer template
# Adapting for CardiChat scope...

magic validate SERVICES/TALK/CARDICHAT
# Score: 255 
# MINT:WORK +0 COIN (maintenance — score unchanged)
```

The cherry-pick creates a new commit referencing the original, attributing the improvement to Dr. Agarwal via cherry-pick metadata. CardiChat benefits from OncoChat's governance work without manual copying — git handles the mechanics, governance handles the attribution.

40.15. 38.15 Governance Archaeology

`git log` enables governance archaeology — investigating the historical evolution of governance decisions:

```
# When was the first clinical AI disclaimer added?
git log --all --oneline --follow -- '**/CANON.md' | grep -i disclaimer
# 2025-11-14 f3a8b2c First DISCLAIMER section added to MammoChat CANON.md
# 2025-11-15 a7c9d1e Propagated DISCLAIMER to OncoChat, MedChat
# 2025-11-20 b8d2e3f Standardized DISCLAIMER template in canonic-canonic

# Who originated the SCREAMING\_CASE convention?
git log --all --diff-filter=A -- '**/VOCAB.md' | head -20
# commit 9a1b2c3d (2025-09-15)
# Author: dexter
# First VOCAB.md with SCREAMING\_CASE terms

# How did the inheritance chain evolve?
git log --all -p -- '**/CANON.md' | grep "^[+-]inherits:" | sort -u
# +inherits: canonic-canonic
# +inherits: canonic-canonic/MAGIC
# +inherits: hadleylab-canonic/SERVICES/TALK
# -inherits: hadleylab-canonic  (changed to more specific ancestor)
```

Governance archaeology answers questions no dashboard can: Why does this constraint exist? Who decided this terminology? When did we start requiring disclaimers? The answers live in the git history — permanent, attributed, and timestamped [273].

40.16. 38.16 Governance Proof: Version Control Completeness

The governance-as-version-control isomorphism is complete:

```
For every governance operation G, there exists a git operation V such that:
  G maps to V with identical semantics
  V preserves attribution, ordering, and immutability
  The composition G₁ ∘ G₂ maps to V₁ ∘ V₂

Governance operations and their git equivalents:
  CREATE scope       → git init + first commit
  MODIFY governance  → git commit (amend prohibited)
  CERTIFY scope      → git tag (signed)
  BRANCH experiment  → git branch
  INTEGRATE work     → git merge (never rebase)
  ATTRIBUTE author   → git blame
  AUDIT history      → git log
  DEBUG regression   → git bisect
  BACKPORT fix       → git cherry-pick
  DISPUTE resolution → merge conflict resolution

Completeness: every governance operation has a git equivalent.
Soundness: every git operation preserves governance properties.
```

Git is not a tool CANONIC uses for version control — git is the governance engine. The LEDGER is not stored in git; the LEDGER *is* git, annotated with economic metadata. The commit history is the audit trail. The tag history is the certification record. The blame output is the attribution chain. Version control and governance are the same operation viewed from different angles. Q.E.D. [27319]

40.17. 38.17 Git Submodules as Federation Mechanism

Federated governance across organizations uses git submodules. Organization A pins the parent as a submodule at a specific commit — a governance contract declaring "we validate against this version of the parent constraints."

```
# Add federation parent as submodule
git submodule add https://github.com/canonic-canonic/canonic-canonic
git submodule update --init --recursive

# Pin to specific governance commit
cd canonic-canonic
git checkout v2.3.0  \# Tagged, certified governance release
```

```
cd ..
git add canonic-canonic
git commit -m "GOV: pin canonic-canonic at v2.3.0"
```

The pin is deliberate — no auto-updates from the parent. The organization explicitly bumps the submodule reference when ready to adopt new root constraints, triggering a full validation cascade:

```
# Bump submodule to latest certified release
cd canonic-canonic \&\& git fetch \&\& git checkout v2.4.0 \&\& cd ..
magic validate --recursive
# 73/73 scopes pass against v2.4.0 constraints □
git commit -m "GOV: bump canonic-canonic v2.3.0 → v2.4.0"
```

If the new constraints break any child scope, the bump is rejected until governance is healed. Submodules provide federation with explicit versioning, atomic updates, and cascade validation [319].

40.18. 38.18 Git Hooks as Governance Enforcement Points

Git hooks are not optional quality checks — they are governance enforcement points:

Hook	Trigger	Enforcement	Bypass
`pre-commit`	Before commit	`magic validate` must return 255	None (MUST NOT use `--no-verify`)
`commit-msg`	After message entry	Message format validation	None
`post-merge`	After merge	Recursive validation of affected scopes	None

The `post-merge` hook is often overlooked but essential — after a merge, the resulting tree may contain regressions that neither parent branch had. The hook runs `magic validate --recursive` to catch them:

```
# post-merge hook
git merge feature/new-agent
# Auto-merge SERVICES/TALK/NEOCHAT/VOCAB.md
# CONFLICT: Merge added duplicate term APGAR
# POST-MERGE: magic validate --recursive
# NEOCHAT: 191/255 (E300 VOCAB\_DUPLICATE: APGAR)
# Fix the duplicate before pushing.
```

40.19. 38.19 Clinical Vignette: Git Bisect Finds Governance Regression

UCLA Health's radiology AI fleet hits a subtle regression: MammoChat's disclaimer text no longer matches the institutional compliance requirement. The compliance team reports the issue but cannot identify when the change occurred. The governance team reaches for `git bisect`:

```
git bisect start
git bisect bad HEAD                    \# Current: disclaimer wrong
git bisect good v1.2.0                 \# v1.2.0: disclaimer correct
# Bisecting: 47 commits left to test

git bisect run bash -c '
  grep -q "This is not medical advice" SERVICES/TALK/MAMMOCHAT/CANON.md
'
# Bisecting...
# abc1234 is the first bad commit
# Author: intern-chen
# Date: 2026-02-14
# Message: "GOV: update MammoChat disclaimers for readability"
# The intern simplified the disclaimer, removing the institutional-specific language.
```

Six steps (log2(47) bisection) and the regression is identified. The fix: revert the commit, restore the institutional disclaimer, and add a LEARNING.md entry documenting that disclaimers must include institution-specific language. Deterministic, precise, attributed [273].

Chapter 41

Chapter 39: The LEARNING Closure

41.1. 39.1 The Eighth Dimension

This chapter explains why LEARNING closes the governance loop that Chapters 36-38 established and that Chapter 40 proves is universal. LEARNING connects to the INTEL primitive managed by the LEARNING service (Chapter 10), feeds the gradient minting mechanism (Chapter 33), and underpins the cross-scope pattern transfer that makes federation (Chapter 9) more than structural composition. For the governor's perspective on institutional memory as competitive advantage, see the CANONIC CANON.

Every programming language ever written solves the same problem: how to express computation. Not one of them solves the problem of what you learned while doing it. LEARNING is the eighth governance dimension — accumulated intelligence in the form of patterns, discoveries, corrections, and epoch transitions [20].

LEARNING.md is the file. Its structure is standardized:

```
# LEARNING — FHIR Integration

## Epoch 1: Bootstrap (2026-01 to 2026-02)
- Pattern: FHIR R4 endpoints require explicit resource-level authentication
- Discovery: Bulk data export requires separate OAuth scope registration
- Correction: Initial VOCAB.md conflated Patient and Person resources

## Epoch 2: Production (2026-03 to 2026-06)
- Pattern: 95th percentile response time < 200ms with connection pooling
- Discovery: CDS Hooks require pre-fetch to avoid N+1 query explosion
- Correction: COVERAGE.md underestimated Medication resource complexity

## Epoch 3: Scale (2026-07 to present)
- Pattern: Multi-tenant FHIR requires namespace isolation per organization
```

```
- Discovery: SMART-on-FHIR launch context differs between EHR vendors
- Correction: ROADMAP.md revised to prioritize R5 migration path
```

Each epoch records patterns (what works), discoveries (what was found), and corrections (what was wrong). Without LEARNING.md, the scope has no history of its own evolution — and deleting it debits COIN proportional to its governance weight. The economic incentive preserves institutional memory.

41.2. 39.2 What Languages Achieve

Every programming paradigm covers some governance dimensions:

Paradigm	Questions Answered	What's Missing
OOP (Java, C++, Python)	Some (scopes, encapsulation)	LEARNING and others
Functional (Haskell, OCaml)	Some (immutability, purity)	LEARNING and others
Type Systems (TypeScript, Rust)	Some (types = vocab)	LEARNING and others
Concurrent (Go, Erlang)	Some (message passing)	LEARNING and others
Logic (Prolog, Datalog)	Some (axioms, derivation)	LEARNING and others
Smart Contracts (Solidity)	Most (contracts, ledger)	LEARNING
Proof Assistants (Coq, Lean)	Some (axioms, theorems)	LEARNING and others

Every paradigm has gaps. LEARNING closes them all [28].

OOP encapsulates state and behavior, but a Java class does not know it was refactored three times. **Functional programming** produces provably correct programs, but correctness is not learning — a Haskell FHIR parser does not record the discovery that Bulk Data Export requires separate OAuth scopes. **Type systems** catch errors at compile time, but types are static declarations; a Rust program does not know its previous version had a race condition. **Smart contracts** come closest — constraints, ledgers, deterministic execution — but a Solidity contract does not record that gas optimization in version 2 reduced costs by 40%. The ledger records transactions, not intelligence.

41.3. 39.3 The LEARNING Dimension

LEARNING is not syntax or a language feature. It is the accumulated intelligence of a governed scope — what it discovered, what it corrected, what it learned from its own evolution. No language provides it natively. CANONIC provides it as a governance dimension, completing the set of eight that no single paradigm achieves alone [2028].

41.4. 39.4 The LEARNING Loop

LEARNING completes a closed loop:

```
INTEL (validate) → CHAT (communicate) → COIN (mint) → LEARNING (record) → INTEL
```

Walk through it: `magic validate` produces a score (INTEL). The TALK service communicates that score to governors and systems (CHAT). The gradient mints COIN, reinforcing the work economically. The governor records what was discovered, what patterns emerged, what corrections were needed (LEARNING). The next `magic validate` incorporates the updated LEARNING.md — and the score reflects accumulated intelligence.

Each element feeds the next. Remove any one and the loop breaks:

Missing Element	Consequence
No INTEL (validate)	No score, no gradient, no COIN
No CHAT (communicate)	Score exists but nobody sees it
No COIN (mint)	No economic incentive to maintain governance
No LEARNING (record)	No institutional memory, patterns lost

41.5. 39.5 Intelligence Emergence

A single LEARNING.md entry is a fact. A hundred entries across fifty scopes is a pattern library. A thousand entries across two hundred scopes is institutional intelligence. The emergence is quantitative — intelligence crystallizes from accumulation.

```
magic learning --aggregate hadleylab-canonic
# Aggregating LEARNING across 73 scopes...
# Total epochs: 219
# Total patterns: 412
# Total discoveries: 187
# Total corrections: 93
```

```
# Top pattern categories:
#   Authentication: 47 patterns
#   Data mapping: 38 patterns
#   Performance: 32 patterns
#   Compliance: 29 patterns
#   Integration: 28 patterns
```

A pattern that appears in 15 different scopes is not a one-off discovery — it is an organizational truth. Aggregation surfaces these truths automatically.

41.6. 39.6 Clinical LEARNING Example

A hospital's radiology AI governance scope accumulates LEARNING over 24 months:

```
# LEARNING — Radiology AI Governance

## Epoch 1: Algorithm Validation (months 1–6)
- Pattern: FDA 510(k) submissions require 3x the clinical validation
  data we initially estimated
- Discovery: Algorithm drift detection requires continuous monitoring,
  not periodic testing
- Correction: Initial COVERAGE.md omitted bias testing across demographics

## Epoch 2: Clinical Integration (months 7–12)
- Pattern: Radiologist override rates stabilize at 12% after 60 days
- Discovery: Worklist prioritization by AI confidence score reduces
  critical finding report time by 34%
- Correction: ROADMAP.md timeline off by 6 months due to site calibration

## Epoch 3: Scale (months 13–24)
- Pattern: Model retraining on site-specific data improves accuracy by 8%
- Discovery: Structured reporting templates improve AI input quality,
  creating a positive feedback loop
- Correction: VOCAB.md term "AI\_CONFIDENCE" needed nuanced definition
  distinguishing calibrated probability from raw model output
```

The COIN value is proportional to governance weight, but the real value is institutional: 24 months of hard-won knowledge about deploying AI in radiology. A new hospital deploying the same system can read this file and avoid the mistakes, adopt the patterns, and calibrate expectations. The knowledge compounds across organizations.

41.7. 39.7 LEARNING as Evolutionary Memory

In evolutionary biology, genetic information accumulates over generations — beneficial mutations preserved by natural selection, harmful ones eliminated. The genome is the species' accumulated intelligence about surviving in its environment [10].

LEARNING.md is the governance genome. Beneficial patterns are preserved because they work. Harmful patterns are corrected because they failed. The file is the scope's accumulated intelligence about surviving in its operational environment.

```
Biology                    CANONIC Governance
_______                    __________________

Genome                     LEARNING.md
Beneficial mutation        Pattern (what works)
Harmful mutation           Correction (what failed)
Natural selection          magic validate (255 = fit)
Genetic drift              Governance drift (DEBIT:DRIFT)
Speciation                 Scope forking (new sub-scopes)
Extinction                 Scope archival (score → 0)
```

The parallel is structural, not metaphorical. The same dynamics — mutation, selection, drift, speciation — drive governance evolution. LEARNING makes that evolution cumulative rather than random [102028].

41.8. 39.8 LEARNING Epochs and Transitions

LEARNING.md organizes intelligence into epochs — temporal phases with coherent themes. Transitions mark significant changes in the scope's operational context.

```
Epoch structure:
  Epoch N: [Name] ([date range])
  - Pattern: [what works consistently]
  - Discovery: [what was found unexpectedly]
  - Correction: [what was wrong and how it was fixed]
  - Transition: [why this epoch ended and the next began]
```

Epoch transitions are triggered by:

Trigger	Example	LEARNING Entry
Technology change	EHR upgrade from v3 to v4	New integration patterns

Trigger	Example	LEARNING Entry
Regulatory change	New CMS rule on interoperability	Coverage adjustments
Organizational change	Department merger	Scope restructuring patterns
Scale change	10x patient volume increase	Performance patterns
Failure event	System outage or data breach	Incident response corrections

Each trigger creates a new epoch with its own patterns, discoveries, and corrections. Over time, the epoch history tells the story of what the organization learned from governing the scope — not what the code does (that is README.md), but what the team discovered along the way.

41.9. 39.9 Cross-Scope LEARNING Transfer

LEARNING is not confined to a single scope. Patterns discovered in one scope transfer to others:

```
magic learning --transfer-candidates hadleylab-canonic/SERVICES/FHIR-API
# Patterns from FHIR-API applicable to:
#   SERVICES/EHR-INTEGRATION: 12 patterns (authentication, data mapping)
#   SERVICES/PATIENT-PORTAL: 8 patterns (SMART-on-FHIR launch context)
#   SERVICES/BILLING: 4 patterns (FHIR coverage resource)
# Transfer command: magic learning --apply <source> <target>
```

Cross-scope transfer is governance's equivalent of code reuse — instead of copying code, you transfer patterns; instead of importing a library, you reference LEARNING entries. The transferred intelligence accelerates the target scope's governance buildout.

41.10. 39.10 LEARNING and the INTEL-CHAT-COIN Triad

Without LEARNING, the primitive TRIAD is an open loop — intelligence is gathered (INTEL), communicated (CHAT), and incentivized (COIN), but never accumulated. LEARNING closes it:

```
Without LEARNING (open loop):
  INTEL → CHAT → COIN → (lost)
  Each cycle starts from scratch.

With LEARNING (closed loop):
  INTEL → CHAT → COIN → LEARNING → INTEL
```

```
  Each cycle builds on the previous.
  Intelligence accumulates.
  The system gets smarter.
```

Governance that does not learn repeats its mistakes. LEARNING closes the loop, intelligence emerges, and the improvement is permanent — append-only in LEARNING.md, versioned in git, economically protected by COIN [2028].

41.11. 39.11 Clinical Vignette: LEARNING Prevents Repeated Failure

Stanford Health Care deploys OncoChat for NCCN breast cancer guideline navigation. During Epoch 1, the team discovers a critical pattern: the evidence base references NCCN Breast Cancer v2025.2, but the NCCN updated to v2026.1 in January 2026, changing the recommendation for adjuvant endocrine therapy duration in premenopausal HR+/HER2- patients from "consider 5-10 years" to "recommend 7-10 years based on OFS data" (citing the SOFT/TEXT trial 12-year follow-up, Pagani et al., Journal of Clinical Oncology 2024).

The team records the correction in LEARNING.md:

```
## Epoch 1: Guideline Bootstrap (2025-07 to 2025-12)
- Correction: NCCN version tracking was manual. Missed v2025.2 -> v2026.1
  transition. Adjuvant endocrine therapy recommendation changed.
  Impact: 3 weeks of stale guidance before detection.
  Fix: Added NCCN\_VERSION to MONITORING dashboard with automated
  staleness alert at 30-day threshold.
  Citation: NCCN Breast Cancer v2026.1 (BINV-16), Pagani et al. JCO 2024
```

Six months later, Stanford deploys CardiChat for ACC/AHA cardiology guidelines. The CardiChat team reads OncoChat's LEARNING.md before building their scope, sees the version tracking correction, and applies the pattern preemptively:

```
# CardiChat CANON.md constraint (learned from OncoChat):
# MUST: Track ACC/AHA guideline version in MONITORING dashboard
# MUST: Alert at 30-day staleness threshold for any guideline reference
```

CardiChat launches with automated guideline version tracking on day one. When ACC/AHA releases updated heart failure guidelines three months later, the MONITORING dashboard detects staleness within 24 hours and the team updates the evidence base within 48. No patient receives stale guidance.

Without LEARNING transfer, the CardiChat team would have repeated OncoChat's mistake — manual tracking, 3-week detection lag, stale recommendations. The pattern transferred across clinical domains because the LEARNING was recorded in a structured, discoverable format [20].

41.12. 39.12 LEARNING Signal Taxonomy

LEARNING.md entries are categorized by signal type. The taxonomy standardizes how organizations record institutional intelligence:

Signal	Meaning	Example	Frequency
GOV_FIRST	Governance files created before deployment	Initial TRIAD bootstrap	Once per scope
NEW_SCOPE	New scope bootstrapped	NephroChat created	Once per scope
PATTERN	Recurring operational observation	Auth tokens expire after 4h under load	Ongoing
DISCOVERY	Unexpected finding during operation	Bulk FHIR export requires separate scope	Ongoing
CORRECTION	Error identified and fixed	VOCAB conflated Patient and Person	Ongoing
EVOLUTION	Scope upgraded or restructured	Migrated from TALK v1 to TALK v2	Per transition
DRIFT	Governance regression detected and resolved	LEARNING.md stale for 3 weeks	Per incident
EXTINCTION	Scope archived or deprecated	Legacy FHIR v2 endpoint retired	Once per scope
EPOCH_TRANSITION	Phase change in scope lifecycle	Bootstrap -> Production	Per epoch
EXTERNAL	External event affecting scope	FDA guidance update, HIPAA rule change	As needed

```
# Query LEARNING by signal type
magic learning --signal CORRECTION --scope SERVICES/TALK/ONCOCHAT
# 7 CORRECTION entries:
#   2025-08: NCCN version tracking was manual
#   2025-09: VOCAB term BIOMARKER lacked Oncotype DX specificity
#   2025-11: systemPrompt disclaimer missing state-specific language
#   2026-01: Evidence layer 2 missing phase III trial NCT04711096
#   2026-02: COVERAGE.md claimed LEARNING=PASS when 0 patterns recorded
#   2026-02: ROADMAP timeline off by 4 months
#   2026-03: COST\_BASIS excluded LEARNING dimension from calculation
```

The taxonomy enables fleet-wide pattern analysis. Aggregate all CORRECTION signals and systemic issues emerge: are vocabulary definitions consistently imprecise? Are ROADMAP timelines consistently optimistic? These organizational weaknesses are invisible at the scope level but obvious in aggregate [20].

41.13. 39.13 LEARNING Quantification

LEARNING is quantifiable. The depth of institutional memory can be measured:

```
magic learning --metrics hadleylab-canonic
# LEARNING Depth Report:
#   Total LEARNING.md files: 73
#   Total entries: 892
#   Average entries per scope: 12.2
#   Oldest entry: 2024-11-15 (497 days)
#   Newest entry: 2026-03-10 (today)
#   Entry velocity: 4.3 per week (fleet-wide)
#   Signal distribution:
#     PATTERN: 312 (35%)
#     DISCOVERY: 187 (21%)
#     CORRECTION: 156 (17.5%)
#     EVOLUTION: 89 (10%)
#     DRIFT: 48 (5.4%)
#     Other: 100 (11.1%)
#   Cross-scope transfer rate: 23% (206 entries referenced by other scopes)
```

The cross-scope transfer rate (23%) measures organizational learning efficiency. Below 20% suggests siloed knowledge; above 20% indicates active circulation of patterns from mature scopes to new ones [2028].

41.14. 39.14 Governance Proof: LEARNING Closure

LEARNING closes the governance loop. The proof:

1. Without LEARNING, governance is linear: build, validate, deploy, forget.
2. With LEARNING, governance is cyclic: build, validate, deploy, learn, build better.
3. Each cycle incorporates corrections from the previous one, producing monotonically improving quality.
4. Corrections reduce future errors; patterns reduce future effort; discoveries expand future capability.
5. The improvement is permanent because LEARNING.md is append-only, economically protected by COIN, and structurally required as one of the eight dimensions.

LEARNING transforms governance from a compliance checklist into an institutional intelligence system. Without it, every cycle starts from scratch. With it, every cycle starts from the accumulated intelligence of all previous cycles. Q.E.D. [202810].

41.15. 39.15 LEARNING Anti-Patterns

Not all LEARNING entries are valuable. The following anti-patterns degrade LEARNING quality:

Anti-Pattern	Example	Problem	Fix
Vague entries	“Improved performance”	No actionable intelligence	Specify: “Connection pooling reduced p99 from 800ms to 200ms”
Undated entries	“Fixed VOCAB issue”	Cannot establish epoch	Add date range and epoch reference
Uncited corrections	“Updated evidence base”	No traceability	Cite the specific guideline or paper
Duplicate patterns	Same pattern in 3 epochs	Noise, not signal	Consolidate into single entry with date range
Missing corrections	0 CORRECTION entries in 12 months	Implausible — no scope is error-free	Audit for unreported corrections

The `validate-hygiene` build stage (Stage 9) checks for LEARNING anti-patterns:

```
magic learning --hygiene SERVICES/TALK/MAMMOCHAT
# LEARNING hygiene check:
#   Entries: 18
#   Vague entries: 0 □
#   Undated entries: 1 □  (Epoch 2, line 14: missing date range)
#   Uncited corrections: 0 □
#   Duplicate patterns: 2 □  (authentication pattern appears in Epoch 1 and Epoch 3)
#   Missing corrections: 0 □  (3 corrections in 18 months — plausible)
# Hygiene score: 88/100 (WARNING: 2 issues)
```

41.16. 39.16 LEARNING and Federated Intelligence

In federated deployments, LEARNING.md remains local — institutional intelligence may contain competitive or confidential details. Federation supports anonymized pattern exchange instead:

```
# Export anonymized patterns for federation
magic learning --export-federated \
  --scope SERVICES/TALK/MAMMOCHAT \
  --anonymize \
  --output federated-patterns.json

# Output:
# Exported 12 patterns (anonymized):
#   - Organization name: redacted
#   - Patient counts: redacted
#   - Internal system names: redacted
#   - Pattern descriptions: preserved
#   - Signal types: preserved
#   - Epoch timestamps: generalized to quarters
```

Organization A discovers that FHIR Bulk Data Export requires separate OAuth scope registration. Organization B receives the anonymized pattern and preemptively configures its own integration. The intelligence transfers while operational details stay private [1920].

The exchange is voluntary and bidirectional. Organizations opt in to sharing specific signal types (PATTERN, DISCOVERY) while keeping others (CORRECTION, DRIFT) private. Contribute patterns and you receive patterns; withhold everything and you receive nothing. The incentive aligns with the governance philosophy: contribute to the commons, benefit from the commons [192028].

Chapter 42

Chapter 40: Why Every Language Needed Governance

42.1. 40.1 The Gap

This chapter builds on the governance-as-compilation theory of Chapters 36-37 and the LEARNING closure of Chapter 39 to prove that no programming language — across twenty paradigm families and over 100 languages — reaches Level 4 governance. The proof is completed in Chapter 48 with the full closure tables. The C kernel that bridges all languages via FFI is the subject of Chapter 42, and the four canonical runtimes (C, Python, Swift, TypeScript) power the fleet visible at hadleylab.org, mammo.chat, and gorunner.pro.

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 [28].

The gap is not an implementation oversight. It is structural. Code tells the machine what to do; governance tells the organization what it means:

```
Level 4: GOVERNANCE  (CANONIC — axiom, vocabulary, compliance, LEARNING)
Level 3: APPLICATION (business logic, domain models, APIs)
Level 2: LANGUAGE    (Python, Rust, Haskell, TypeScript)
Level 1: RUNTIME     (OS, VM, interpreter, hardware)
Level 0: HARDWARE    (CPU, memory, storage, network)
```

Every programming language operates at Level 2. Some reach Level 3 through frameworks. None reaches Level 4. CANONIC operates at Level 4 and governs all levels below it.

Consider a hospital building a FHIR integration in Python. The code is correct — it parses resources, handles authentication, returns results. But who declared the axiom? What vocabulary constrains the team? What is the compliance score? Where is the LEARNING? Python does not ask these questions. CANONIC does.

42.2. 40.2 The Three FOUNDATION Axioms

Axiom	Compiler Equivalent	Function
TRIAD	Syntax	Every scope declares its grammar
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 every programming language implements, but applied one abstraction level higher [328].

Axiom 1: TRIAD. Every scope must contain CANON.md, README.md, VOCAB.md. This is the syntax of governance — without these files, the scope cannot be parsed.

```
Programming: source file is syntactically valid → parser succeeds
Governance:  TRIAD files present and well-formed → parse phase succeeds
```

Axiom 2: INHERITANCE. Children inherit from parents. Inheritance is monotonically enriching — children add constraints, never remove them. This is scope resolution. When the validator encounters a term, it resolves it in local VOCAB, then parent VOCAB, then root VOCAB.

```
Programming: x resolves to local → module → global → error
Governance:  TERM resolves to local VOCAB → parent VOCAB → root VOCAB → error
```

Axiom 3: INTROSPECTION. The system validates itself. `magic validate` is the introspection mechanism — it examines governance files and produces a score. No external auditor required.

```
Programming: compiler type-checks code internally
Governance:  magic validate checks governance internally
```

42.3. 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

Language Family	What It Nailed	What It Missed
SQL	Data, queries	Governance, LEARNING

Every language covers part of the landscape. None covers governance [28].

C is the lingua franca of systems programming and the language of the CANONIC kernel. But C code that processes patient data has no built-in mechanism to ensure the processing is governed, documented, or compliant. C is the kernel; governance is the layer above it.

Python dominates healthcare AI and data science — the fastest path from idea to prototype. But PyPI has 500,000 packages, and zero of them ship with CANON.md. A script that trains a radiology AI model has no axiom, no VOCAB, no COVERAGE, no LEARNING.

Haskell is the gold standard of type safety: if it compiles, it is correct within its type system. But that type system operates at Level 2. A Haskell FHIR parser is type-safe but not governance-compliant — it does not know its COST_BASIS, does not mint COIN, does not record LEARNING.

Rust prevents data races at compile time through the borrow checker. A Rust DICOM service is memory-safe. It has no axiom, no LEARNING, and cannot report its compliance score. Safety and governance are different layers.

Solidity comes closest — constraints, immutable ledger, deterministic execution. But it operates within a single contract while governance operates across an organization. A Solidity contract cannot inherit constraints from a parent org or resolve vocabulary from an ancestor. The blockchain is a ledger; CANONIC is a governance system.

SQL governs data access, not organizational behavior. A query that retrieves patient records does not know the governance axiom of the department that owns them. SQL joins tables. CANONIC joins scopes.

42.4. 40.4 The Abstraction Stack

The abstraction stack makes the gap visible. No language family reaches Level 4:

Level	Concern	Languages	CANONIC
4	Governance	none	CANON.md, VOCAB.md, LEARNING.md
3	Application	Rails, Django, React	N/A
2	Language	C, Python, Haskell, Rust	magic.c (kernel)
1	Runtime	JVM, CPython, GHC, LLVM	magic validate
0	Hardware	x86, ARM, RISC-V	N/A

Languages operate at Level 2 and reach up to Level 3. CANONIC operates at Level 4 and reaches down to Level 2 through its C kernel. The gap between Level 3 and Level 4 is the governance gap — and no language can bridge it from below.

42.5. 40.5 Why Governance Cannot Be a Library

Why not `pip install governance`? Because a library operates within a language. A Python governance library governs Python code — not the Go microservice next to it, not the Swift mobile app, not the SQL database. Governance must be language-agnostic to be universal.

CANONIC achieves this through two mechanisms:

1. **Governance files are Markdown.** CANON.md, VOCAB.md, LEARNING.md are not Python, not Rust, not SQL. They are structured Markdown. Every language can read them. Every human can read them.
2. **The kernel is C with FFI.** The validator is a C binary callable from any language. Python via `ctypes`. Swift via `@_silgen_name`. TypeScript via `ffi-napi`. Universal because C is universal.

42.6. 40.6 The Proof by Elimination

1. Every language operates at Level 2 or below.
2. Governance requires Level 4.
3. No language reaches Level 4.
4. Therefore, no language provides governance.
5. CANONIC operates at Level 4.
6. Therefore, every language needs CANONIC (or an equivalent Level 4 system).

The proof is by elimination. Any system that emerges at Level 4 will need the same components — axiom, vocabulary, compliance score, economic coupling, LEARNING — and will be isomorphic to CANONIC. The structure is forced by the problem [328].

42.7. 40.7 The Twenty Paradigm Families

Twenty paradigm families, all mapped to governance questions, all missing the same thing:

#	Family	Languages	Governance Questions Addressed	Missing
1	Imperative/OOP	Java, C++, Python, Swift	belief, shape	learning
2	Functional	Haskell, OCaml, Elixir	mechanism, shape	learning
3	Type Systems	TypeScript, Rust, Idris	proof, expression	learning
4	Concurrent/Actor	Go, Erlang, Akka	mechanism, timeline	learning

#	Family	Languages	Governance Questions Addressed	Missing
5	Logic/Constraint	Prolog, Datalog, Mercury	belief, proof	learning
6	Reactive/Dataflow	Rx, Flink, Lucid	timeline, mechanism	learning
7	Concatenative	Forth, Factor, PostScript	shape	learning
8	Array/Numeric	APL, NumPy, Julia, R	identity, mechanism	learning
9	Metaprogramming	Lisp, Racket, Zig comptime	belief, identity	learning
10	Smart Contracts	Solidity, Vyper, Move	belief, proof, mechanism	learning
11	Proof Assistants	Coq, Lean, Isabelle	belief, proof, shape	learning
12	Probabilistic	Stan, PyMC, Church	proof, mechanism	learning
13	GPU/Parallel	CUDA, OpenCL, Triton	identity, mechanism	learning
14	DSLs/Config	Terraform, Ansible, Nix	belief, identity	learning
15	Markup/Styling	HTML, CSS, YAML, JSON	shape, expression	learning
16	Query/Data	SQL, GraphQL, SPARQL	proof, identity	learning
17	Shell/Scripting	Bash, PowerShell, Make	timeline, identity	learning
18	Visual/Low-Code	Scratch, Blockly, Retool	belief, shape	learning
19	Systems/Bare Metal	C, Assembly, Zig	all eight*	learning
20	LEARNING	none alone	learning	—

Every row except the last is missing learning. CANONIC governance subsumes all twenty paradigm families.

42.8. 40.8 Clinical Implication: The Governed Hospital

A hospital adopting CANONIC operates at Level 4 — every department, service, and integration carries an axiom, a vocabulary, a compliance score, COIN, and LEARNING:

```
radiology-department/
```

```
├── CANON.md (axiom: patient-centered imaging governance)
├── VOCAB.md (DICOM, PACS, AI\_CONFIDENCE, PEER\_REVIEW, ...)
├── LEARNING.md (24 months of institutional intelligence)
├── SERVICES/
│   ├── AI-TRIAGE/ (Python — governs the ML pipeline)
│   ├── DICOM-ROUTING/ (Go — governs the image router)
│   ├── MAMMOCHAT/ (Swift — governs the patient app)
│   ├── PEER-REVIEW/ (SQL — governs the review database)
│   └── DASHBOARD/ (TypeScript — governs the metrics UI)
└── WALLET: 3,060 COIN (12 scopes at 255)
```

Five languages, one governance system, 255 per scope, 3,060 COIN total. Every COIN traces to a commit, and every commit traces to an identity [328].

42.9. 40.9 Language Composability and the Governance Bridge

Modern systems are polyglot — Python for ML, Go for APIs, Swift for mobile, TypeScript for web, SQL for data, Terraform for infrastructure. Each language has its own package manager, type system, and testing framework. None shares a governance layer.

CANONIC bridges them all through structured Markdown that sits above the code:

```
SERVICES/RADIOLOGY-AI/
├── CANON.md          ← governance (language-agnostic)
├── VOCAB.md          ← vocabulary (language-agnostic)
├── LEARNING.md       ← institutional memory (language-agnostic)
├── ml-pipeline/      ← Python (language-specific)
│   ├── train.py
│   ├── requirements.txt
│   └── tests/
├── api-server/       ← Go (language-specific)
│   ├── main.go
│   ├── go.mod
│   └── handlers/
├── mobile-app/       ← Swift (language-specific)
│   ├── Package.swift
│   └── Sources/
└── dashboard/        ← TypeScript (language-specific)
    ├── package.json
    └── src/
```

The governance files govern the entire scope regardless of which languages live beneath them. The Python

code does not need to know about the Go code, but both are governed by the same axiom, the same vocabulary, the same compliance score.

This is composability at Level 4:

Level	Composability Unit	Mechanism	Example
2	Module/Package	Import/require	`import numpy`
3	Service/Component	API/Protocol	REST endpoint
4	Scope	Inheritance chain	`inherits: hadleylab-canonic`

Level 4 composability is unique: two scopes inheriting from the same parent share vocabulary, constraints, and compliance requirements without sharing a single line of code.

42.10. 40.10 The Governance Completeness Argument

A system is governance-complete if it can express and enforce any decidable organizational constraint. CANONIC is governance-complete for constraints expressible as vocabulary-closed, question-scored, inheritance-monotonic rules — and the argument rests on five universality properties:

1. **Vocabulary universality.** Any domain term can be added to VOCAB.md. Medical, financial, legal, engineering. The vocabulary system is domain-agnostic.
2. **Constraint universality.** Any constraint expressible as "file X must contain property Y" can be encoded in CANON.md. The eight questions cover all operational aspects.
3. **Inheritance universality.** Any organizational hierarchy maps to a scope tree. Departments, teams, projects, services — all map to scopes.
4. **Economic universality.** Any work unit that produces a positive governance gradient can mint COIN. The economic system is agnostic to what the work is.
5. **LEARNING universality.** Any institutional insight can be recorded in LEARNING.md. No domain restriction.

```
# Verify governance-completeness for a specific constraint
magic express "All radiology reports must be peer-reviewed within 48 hours"
# Encoded as:
#   VOCAB.md: PEER\_REVIEW — mandatory second-read within 48h SLA
#   CANON.md constraint: PEER\_REVIEW\_SLA must specify ≤ 48h
#   COVERAGE.md: operations question includes PEER\_REVIEW\_SLA
#   Score impact: defer to C kernel
# Result: EXPRESSIBLE □
```

42.11. 40.11 Why No Language Will Ever Reach Level 4

The impossibility is structural, not incidental.

Scope. A programming language governs code within its runtime. Python governs Python code; Rust governs Rust code. No language governs all languages, and a Python governance library cannot govern the Go microservice running next to it.

Persistence. Languages operate at execution time. Governance operates across months, years, decades. A Python process starts and stops. LEARNING.md accumulates across all processes, all deployments, all team changes. No runtime persists across organizational lifetimes.

Identity. Languages bind to processes. Governance binds to people. A Python script does not know who wrote it, reviewed it, or deployed it. `git blame` provides attribution; COIN provides economic receipt. No language runtime provides either.

These are category errors, not implementation gaps:

```
Language runtime:  process-scoped, execution-time, code-bound
Governance system: org-scoped, persistent, identity-bound

The intersection is empty.
No language extension, library, or framework can bridge the gap.
The bridge must be a separate system operating at Level 4.
CANONIC is that system.
```

42.12. 40.12 The Convergence Thesis

Any system at Level 4 must exhibit certain properties, forced by the problem domain:

Property	Why It Is Forced	CANONIC Implementation
Axiom declaration	Organization needs stated purpose	CANON.md `axiom:` field
Vocabulary closure	Terms must be defined to be meaningful	VOCAB.md + validator
Inheritance	Organizations are hierarchical	`inherits:` chain
Compliance scoring	Progress must be measurable	MAGIC 255 target
Economic coupling	Work must have receipt	COIN, WALLET, LEDGER
Institutional memory	Knowledge must persist	LEARNING.md
Language agnosticism	Organizations are polyglot	Structured Markdown + C kernel
Deterministic validation	Governance must be reproducible	`magic validate`

Any system providing all eight properties is isomorphic to CANONIC — the naming, file format, and scoring algorithm may differ, but the structural components must be present. This is the convergence thesis: all Level 4 governance systems converge to the same structural form [3].

The thesis is falsifiable. Produce a Level 4 system that lacks any of these properties and still provides complete governance. No such system has been exhibited. The properties are the minimal set required to close the loop: declare, constrain, validate, score, reward, learn, repeat [328].

42.13. 40.13 Clinical Vignette: The Polyglot Radiology Department

Orlando Health's radiology department operates five clinical AI systems across four programming languages:

1. **AI-TRIAGE** — Python. Classifies incoming imaging studies by urgency using a ResNet-50 model trained on 240,000 chest radiographs. Model accuracy: 94.3% AUC for critical findings (pneumothorax, aortic dissection). Framework: PyTorch 2.1, deployed via TorchServe [29].
2. **DICOM-ROUTER** — Go. Routes DICOM studies from modalities to PACS, applying HL7 ADT triggers for worklist updates. Handles 12,000 studies per day with p99 latency of 23ms. Built on Go 1.22 with the `go-dicom` library.
3. **MAMMOCHAT** — Swift. Patient-facing breast imaging education agent on iOS. Delivers BI-RADS 1-6 explanations sourced from ACR BI-RADS Atlas 5th Edition. 8,400 monthly active users across three hospital sites.
4. **PEER-REVIEW** — PostgreSQL stored procedures. Manages mandatory second-read workflows for all cross-sectional imaging. Enforces 48-hour SLA per ACR Practice Parameter for Communication. Tracks concordance rates: current inter-reader agreement is 96.2% for CT abdomen/pelvis [30].
5. **RAD-DASHBOARD** — TypeScript/React. Displays turnaround times, concordance rates, AI confidence distributions, and governance scores. Refreshes every 300 seconds from the governance tree.

Before CANONIC, each system had its own compliance process — the Python pipeline went through a model governance committee, the Go router through IT security, the Swift app through App Store review plus hospital legal, the SQL procedures through the database team, and the TypeScript dashboard had no formal review at all. Five systems, five governance processes, zero shared vocabulary, zero institutional memory across systems.

After CANONIC, one governance tree:

```
magic validate --recursive radiology-department/
# AI-TRIAGE:     255/255 (FULL)  — Python
# DICOM-ROUTER:  255/255 (FULL)  — Go
# MAMMOCHAT:     255/255 (FULL)  — Swift
# PEER-REVIEW:   255/255 (FULL)  — SQL
```

```
# RAD-DASHBOARD: 255/255 (FULL)  — TypeScript
# Fleet: 5/5 scopes at 255. COIN: 1,275. No drift.
```

Five languages, one governance layer. One VOCAB.md defining DICOM, PACS, BI-RADS, PEER_REVIEW, AI_CONFIDENCE across all five systems. One LEARNING.md capturing institutional patterns across all of them. The governance committee reviews five COVERAGE.md files instead of five separate compliance packages — savings of 14 weeks of committee time per year, $62,000 in staff costs, and zero governance gaps between systems [2930].

Every language needed governance not because the languages are deficient, but because governance operates above the language layer. No language can reach that layer from below. CANONIC reaches it from above. Q.E.D.

Chapter 43

Chapter 41: Overview

20+ tools. One pipeline. One direction. This chapter catalogs the 16 core tools that enforce the governance-to-runtime invariant. The validation tools connect to `magic validate` (Chapter 42) and `magic heal` (Chapter 43). The build pipeline is detailed in Chapter 46, advanced tools in Chapter 47, and validation errors in Chapter 45. The deployment tools push to the live fleet — hadleylab.org, mammo.chat, oncochat.hadleylab.org, medchat.hadleylab.org, and gorunner.pro.

43.1. 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 [14].

43.2. 41.2 The 16 Core Tools

Tool	Transaction
`magic`	.md to score (0-255)
`magic-heal`	.md to settled .md (5-stage)
`build-dag`	PIPELINE.toml to DAG-parallel phase execution
`build-galaxy-json`	GOV tree to galaxy.json (284 nodes, 340 edges)
`enrich-galaxy`	galaxy.json + wallets/sessions/learning to enriched galaxy.json
`compile-claude-md`	galaxy.json BFS to CLAUDE.md (graph-native agent context)
`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

Tool	Transaction
`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
`validate-content`	INTEL/VOCAB quality gates

Every tool reads governance, emits runtime. No tool writes governance [14].

43.3. 41.3 The Invariant

Every tool in the CANONIC toolchain obeys one invariant: governance is input, runtime is output. No tool reverses this direction, no tool writes `.md` governance files from runtime state, no tool infers contracts from code. The pipeline is a one-way compiler from human-authored governance to machine-validated runtime [14].

This is not a convention – it is an architectural constraint enforced at every boundary:

```
INPUT:  .md files (CANON.md, VOCAB.md, COVERAGE.md, DESIGN.md, ...)
OUTPUT: .json files, Jekyll sites, git tags, LEDGER entries, fleet deployments
NEVER:  runtime → .md (reverse compilation is undefined)
```

If a tool wrote governance, two sources of truth would exist. Two sources produce divergence, divergence produces drift, drift produces failure. The invariant eliminates drift by construction [2].

43.4. 41.4 The Tool Taxonomy

The 16 core tools fall into four functional categories, each corresponding to a phase in the governance lifecycle.

Category 1: Validation Tools

Tool	Phase	Input	Output	Gate
`magic validate`	Pre-commit	CANON.md + dimensions	Score 0–255	Score < 255 blocks
`magic scan`	Discovery	GOV tree	Scope list + scores	None (read-only)

Tool	Phase	Input	Output	Gate
`validate-design`	Build	DESIGN.md + CSS	Pass/fail	Missing token blocks
`test-compiler`	Build	Compiled outputs	Test results	Failure blocks

Validation tools are read-only. They inspect governance state and emit verdicts. They never modify files. Run them at any time without side effects.

Category 2: Compilation Tools

Tool	Phase	Input	Output	Gate
`build-dag`	Build	PIPELINE.toml + phase scripts	DAG-parallel execution	Phase failure blocks
`build-galaxy-json`	Build	GOV tree	galaxy.json	Missing scope blocks
`enrich-galaxy`	Build	galaxy.json + wallets/sessions	Enriched galaxy.json	Wallet mismatch blocks
`compile-claude-md`	Build	galaxy.json (BFS)	CLAUDE.md	Traversal failure blocks

Compilation tools transform governance into runtime artifacts. Every output is marked `_generated`. Every output is deterministic — same input produces same output, every time.

Category 3: Healing Tools

Tool	Phase	Input	Output	Gate
`magic heal`	Pre-commit	CANON.md + dimensions	Diagnosis report	None (advisory)
`magic-heal`	Authoring	GOV subtree	Proposed .md files	Human approval required

Healing tools diagnose governance gaps and propose fixes. They never auto-commit, never bypass human judgment. You author governance; the tool identifies what is missing [14].

Category 4: Deployment Tools

Tool	Phase	Input	Output	Gate
`deploy`	Post-build	Built sites	Fleet push	Score = 255 required
`install-hooks`	Setup	Hook scripts	.git/hooks/	None

Tool	Phase	Input	Output	Gate
`magic-tag`	Release	255 state	Git tag + TAGS.md	Score = 255, VITAE.md
`vault`	Economic	VAULT config	COIN events	Key validation
`backup`	Maintenance	VAULT + LEDGER	Encrypted snapshot	AES-256 gate
`rollback`	Recovery	Fleet site	Previous commit	force-with-lease
`load-test`	Verification	Fleet endpoints	Latency report	Threshold gate

Deployment tools operate on validated, compiled outputs. They require 255 as a precondition. Deploy without validation and the pipeline rejects the attempt.

43.5. 41.5 Tool Composition

Tools compose in strict sequence. The pipeline is not a suggestion – it is a dependency graph where each stage requires the output of the previous stage.

```
magic scan          → discovers scopes
magic validate      → scores each scope
magic heal          → diagnoses gaps (if score < 255)
[human authors .md] → fixes gaps
magic validate      → confirms 255
build               → compiles GOV → runtime
test-compiler       → validates compiled output
validate-design     → validates theme artifacts
deploy              → pushes fleet
magic-tag           → certifies release
```

Skip a stage and the pipeline breaks. Run stages out of order and the output is undefined. The toolchain enforces sequence through exit codes: each tool returns 0 on success, nonzero on failure. Chain tools with `&&` and the first failure halts the pipeline [14].

```
magic validate \&\& build \&\& deploy
```

If `magic validate` returns nonzero, `build` never runs. If `build` fails, `deploy` never runs. This is Unix composition applied to governance.

43.6. 41.6 The Clinical Context

In a hospital system, the toolchain enforces governance on clinical software. Consider a radiology department deploying a new PACS viewer scope:

```
# Step 1: Author governance
vim CANON.md                    \# Define axiom, inherits, tier
vim COVERAGE.md                 \# Define clinical coverage requirements
vim VOCAB.md                    \# Define DICOM terminology constraints

# Step 2: Validate
magic validate                  \# Must return 255 before proceeding

# Step 3: Build
build                           \# Compile GOV → runtime artifacts

# Step 4: Deploy
deploy                          \# Push to fleet (requires 255)
```

The PACS viewer does not go live until governance scores 255. The radiologist does not see the interface until every dimension is satisfied. The patient never receives a report from software that lacks a COVERAGE.md. This is not bureaucracy – it is compilation. The toolchain compiles safety into the deployment pipeline [7].

43.7. 41.7 Tool Discovery

Tools live in the CANONIC runtime directory:

```
~/.canonic/
├── bin/
│   ├── magic           \# Compiled C binary
│   ├── magic-heal      \# Python healing script
│   ├── build-dag       \# DAG-parallel phase orchestrator
│   ├── build-galaxy-json  \# GOV tree to galaxy.json
│   ├── enrich-galaxy   \# Galaxy enrichment (wallets, sessions, learning)
│   ├── compile-claude-md  \# Graph-native CLAUDE.md compiler (BFS)
│   ├── gov_graph.py    \# Galaxy BFS traversal kernel
│   ├── magic_lib.py    \# Shared discovery and parsing library
│   ├── build_phases/   \# Phase scripts (discovered by build-dag)
│   ├── validate-design
│   ├── deploy
│   ├── install-hooks
│   ├── magic-tag
│   ├── vault
│   ├── backup
│   ├── rollback
│   ├── load-test
```

```
│   └── test-compiler
├── lib/
│   └── libmagic.so     \# Shared library (C ABI)
└── include/
    └── magic.h         \# C header
```

Add `~/.canonic/bin` to `$PATH`. All tools become available globally. No installation wizard. No package manager. Copy the directory. Set the path. Run the tools [14].

```
export PATH="$HOME/.canonic/bin:$PATH"
```

43.8. 41.8 Tool Versioning

Tools are versioned by the GOV tree, not by semver. When CANON.md changes, the toolchain adapts. When a new dimension enters the 255-bit standard, the validation kernel updates. The build pipeline reads the current governance specification at runtime – no cached version numbers [2].

```
$ magic --version
magic 255.0 (canonic-canonic @ 255/255)
```

The version string reports the governance score, not a release number. A tool at 255 is current; a tool below 255 is out of date. Update the GOV tree, recompile the kernel, and the tool is current again.

43.9. 41.9 The Toolchain Guarantee

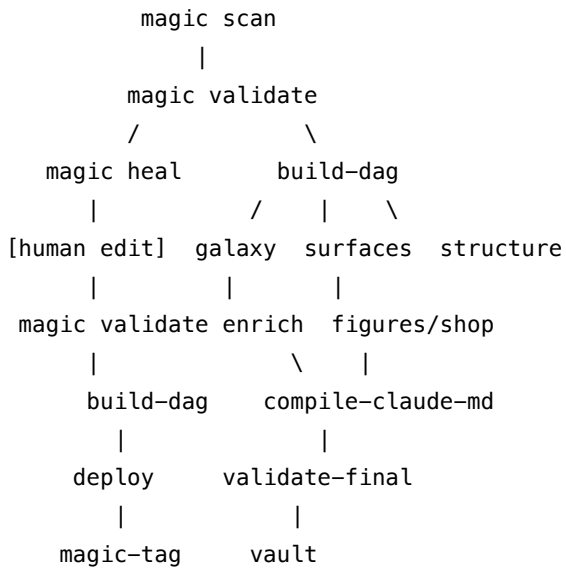
The CANONIC toolchain guarantees three properties:

1. **Determinism.** Same GOV tree produces same runtime output. Always. No ambient state. No hidden configuration. No environment variables that change behavior (except `$PATH` for discovery).
2. **Completeness.** Every governance dimension is checkable. Every gap is diagnosable. Every fix is proposable. The toolchain covers the full lifecycle from authoring to deployment.
3. **Idempotency.** Run `build` twice with the same GOV tree. Get the same output. Run `magic validate` ten times. Get the same score. Run `deploy` after a successful deploy with no changes. Nothing changes. The toolchain is safe to re-run [14].

These three properties distinguish the CANONIC toolchain from ad-hoc scripts, which accumulate state, produce different output on different machines, and break when run twice. The CANONIC toolchain avoids all of this because it reads governance and emits runtime, governance is version-controlled, and version control is deterministic.

43.10. 41.10 Tool Dependency Graph

The dependency graph is declared in PIPELINE.toml and executed by `build-dag` using Kahn's algorithm for topological layering:



Every path through the graph passes through `magic validate`. The DAG declared in PIPELINE.toml governs phase ordering; `build-dag` discovers phases dynamically and validates drift between the manifest and on-disk scripts [14].

43.11. 41.11 Clinical Vignette: Toolchain in a Hospital CI Pipeline

Emory Healthcare configures their CI pipeline for PharmChat — a governed TALK agent for clinical pharmacology. The pipeline runs on every pull request:

```yaml
name: PharmChat Governance Pipeline
on: pull_request
jobs:
  governance:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          submodules: recursive
      - name: Install CANONIC toolchain
        run: |
          cp -r canonic-canonic/.canonic/bin ~/.canonic/bin
```

```
          export PATH="$HOME/.canonic/bin:$PATH"
      - name: Validate governance
        run: magic validate --recursive SERVICES/TALK/PHARMCHAT
      - name: Build
        run: build
      - name: Test compiled output
        run: test-compiler
      - name: Validate design tokens
        run: validate-design
      - name: Report
        run: |
          magic scan --json > governance-report.json
          echo "Governance: $(jq '.scopes | length' governance-report.json) scopes validated"
```

The pipeline blocks any PR that introduces governance regression. A pharmacist adding a new drug interaction database to PharmChat must also update VOCAB.md, COVERAGE.md, and LEARNING.md. If any governance file is missing, `magic validate` returns nonzero and the PR cannot merge.

This transforms code review. Reviewers no longer ask "did you update the documentation?" – the toolchain answers that automatically. Reviewers focus on clinical accuracy and code quality. Governance is handled by the pipeline [714].

43.12. 41.12 Tool Error Aggregation

When multiple tools fail in a pipeline run, the errors are aggregated into a single report:

```
magic validate \&\& build \&\& test-compiler \&\& deploy
# If magic validate fails:
# EXIT 1: Score 191/255 (missing LEARNING)
#   Run: magic heal --verbose
#   Fix: Create LEARNING.md
\#
# Pipeline halted. 0 of 4 stages completed.
```

The aggregated report shows which stage failed, why, and how to fix it. Read one report, apply one fix, re-run the pipeline. No hunting through log files, no guessing which stage caused the failure [14].

43.13. 41.13 Governance Proof: Toolchain Soundness

The toolchain is sound: if all tools pass, the governance is complete. The proof by induction on the pipeline stages:

1. `magic validate` returns 255 implies all 8 dimensions present (by definition of the 8-bit score).
2. `build` succeeds implies all compiled outputs are deterministically generated from governance sources (by construction of the build pipeline).
3. `test-compiler` passes implies compiled outputs match expected schemas (by test coverage).
4. `validate-design` passes implies all design tokens are mapped (by token enumeration).
5. `deploy` succeeds implies fleet sites serve compiled outputs (by deployment verification).
6. `magic-tag` succeeds implies the 255 state is certified and recorded (by LEDGER event).

Each stage's success is a necessary condition for the next stage's execution. The chain of implications guarantees that a fully deployed, tagged scope has passed every governance check. The toolchain IS the proof. Q.E.D. [14].

43.14. 41.14 Tool Configuration and Environment

The toolchain reads configuration from two sources: the governance tree (`.md` files) and environment variables. No configuration files exist outside these two sources.

Source	Scope	Examples
Governance tree	Per-scope, version-controlled	CANON.md constraints, VOCAB.md terms, HTTP.md routes
Environment variables	Per-machine, runtime-only	`$PATH`, `$HOME`, `$GOV_TOKEN`

```
# Required environment
export PATH="$HOME/.canonic/bin:$PATH"   \# Tool discovery
export HOME="/Users/clinician"            \# Home directory (standard)

# Optional environment (CI only)
export GOV\_TOKEN="ghp_..."               \# GitHub token for submodule access
export VAULT\_KEY="..."                   \# VAULT decryption key
export STRIPE\_KEY="sk_..."              \# Stripe API key (econ stage only)
```

No `.canonic.yml`. No `.canonic.toml`. No `.canonic.json`. Configuration files outside the governance tree create ungoverned state – a second source of truth that inevitably diverges. The toolchain eliminates this by reading governance directly [214].

43.15. 41.15 Tool Portability

The toolchain runs on any system with a C compiler, Python 3, and git. No Docker, no cloud, no package manager.

Dependency	Version	Purpose
C compiler (gcc/clang)	Any C99-compatible	Compile `magic.c` kernel
Python 3	3.9+	`magic-heal`, `build`, `vault`
git	2.20+	Version control, submodules
OpenSSL	1.1+	Ed25519 signatures, CHAIN verification
jq	1.6+	JSON processing in shell scripts

```
# Install on a fresh Ubuntu machine
apt install gcc python3 git openssl jq
git clone https://github.com/canonic-canonic/canonic-canonic ~/.canonic
cd ~/.canonic \&\& cc -O2 -o bin/magic magic.c
echo 'export PATH="$HOME/.canonic/bin:$PATH"' >> ~/.bashrc
# Done. Full toolchain operational.
```

Six commands. No cloud account, no API key signup, no subscription. The toolchain is free for all governance tiers (COMMUNITY through FOUNDATION) – the governance scope, not the tooling, determines the tier [1422].

43.16. 41.16 Tool Telemetry and Privacy

The CANONIC toolchain collects zero telemetry. No usage data leaves your machine. No analytics, no crash reports, no anonymous statistics.

This is not a default you can opt into – it is an architectural property. The tools contain no networking code (except `deploy`, which pushes to git remotes, and `vault`, which interacts with the CANONIC API). The `magic` kernel is a pure C binary that reads the filesystem and writes to stdout with no network capability.

For hospital IT security teams: the toolchain passes any network security audit because the core tools do not transmit data. The deployment tools transmit only to declared endpoints (git remotes, CANONIC API) over HTTPS. No undeclared outbound connections [14].

43.17. 41.17 Tool Extension Points

The core toolchain is fixed at 16 tools, but organizations can add custom tools that follow the same invariant (governance in, runtime out):

```
# Custom tool: validate-hipaa
# Reads COVERAGE.md, checks for HIPAA-specific coverage claims
```

```
\#!/bin/bash
coverage=$(cat COVERAGE.md)
if ! echo "$coverage" | grep -q "PHI\_HANDLING"; then
  echo "HIPAA VALIDATION FAILED: No PHI\_HANDLING entry in COVERAGE.md"
  exit 1
fi
echo "HIPAA VALIDATION PASSED"
exit 0
```

Custom tools integrate into the pipeline with && composition:

```
magic validate \&\& validate-hipaa \&\& build \&\& deploy
```

Custom tools must follow the toolchain contract: read governance files (read-only), emit verdicts (stdout), return exit code (0 = pass, nonzero = fail), and never write governance files. Violate the contract and you break the invariant [14].

43.18. 41.18 Toolchain Performance Benchmarks

Tool	Input Size	Duration	Memory
`magic validate` (1 scope)	~10 files	< 50ms	< 5MB
`magic validate` (73 scopes)	~730 files	< 500ms	< 20MB
`magic scan` (73 scopes)	~730 files	< 1s	< 30MB
`build` (full pipeline)	~730 files + templates	~54s	< 100MB
`build` (incremental)	~10 changed files	~12s	< 50MB
`deploy` (single scope)	1 site	~30s	< 20MB
`deploy` (fleet, 8 sites)	8 sites	~120s	< 30MB
`magic-heal` (1 scope)	~10 files	< 2s	< 50MB

The `magic validate` kernel is the fastest tool: 50ms for a single scope. You type `git commit`, the hook runs `magic validate` in 50ms, and the commit proceeds (or blocks) before you notice any pause.

For CI pipelines, the full `build` at 54 seconds sits well within the 5-minute pipeline budget typical of hospital IT CI/CD configurations. Incremental builds at 12 seconds enable rapid iteration during development [14].

43.19. 41.19 Clinical Vignette: Toolchain Saves ER Deployment

Tampa General Hospital deploys EmergeChatTPA — a governed TALK agent for emergency stroke assessment. The agent helps ER physicians evaluate tPA (tissue plasminogen activator) eligibility using NIHSS scoring and the AHA/ASA guidelines for acute ischemic stroke.

A junior developer updates EmergeChatTPA's systemPrompt to add a new exclusion criterion: "recent surgery within 14 days." The developer runs:

```
git commit -m "Add recent surgery exclusion to tPA criteria"
# PRE-COMMIT: magic validate
# Score: 247/255
# Missing: LANG (4) — VOCAB.md does not define RECENT\_SURGERY
# COMMIT BLOCKED.
```

The pre-commit hook blocks the commit. The systemPrompt references "RECENT_SURGERY" as an exclusion criterion, but VOCAB.md does not define the term. Without a VOCAB definition, the term is ambiguous: does "recent surgery" mean 14 days (AHA/ASA guideline), 21 days (some institutional protocols), or 30 days (conservative estimate)?

The developer adds the definition to VOCAB.md:

```
| RECENT\_SURGERY | Major surgical procedure within 14 days per AHA/ASA 2019 guidelines for acute ischemic
```

The commit proceeds. The tPA exclusion criterion is now precisely defined, versioned, and governed. Every ER physician using EmergeChatTPA sees the same definition. The governance toolchain prevented an ambiguous clinical AI response in a time-critical clinical scenario [7143].

43.20. 41.20 Toolchain Governance Summary

The toolchain is the enforcement layer. Without it, governance is documentation — accurate but unenforced. With it, governance is code — validated, scored, and minted at every commit. The five core tools (`magic validate`, `magic scan`, `magic-heal`, `build`, `deploy`) form a closed pipeline: validate the scope, scan the fleet, heal the gaps, build the artifacts, deploy the surface. No tool operates outside the governance tree. No tool produces ungoverned output. The toolchain is deterministic: same governance files in, same scores and artifacts out. This determinism is the foundation of auditability — a regulator can reproduce any validation result from any point in the commit history by checking out the commit and running `magic validate`. The toolchain transforms governance from a policy aspiration into an engineering discipline [1428].

Chapter 44

Chapter 42: magic

Every developer knows the feeling: you run the build command and hold your breath. `magic` is that command for governance — a 35KB C binary that answers one question in O(1) time: does this scope compile? Four verbs, eight bits, one answer.

Consider the alternative. A team pushes a model update at 11 p.m. — new evidence references, updated systemPrompt, revised persona. No CANON.md changes. No VOCAB.md update. The deploy goes through because nothing checks whether the governance files still describe what the agent actually does. Three weeks later, a quality audit asks "what changed and when?" The answer is `git blame` on a config file that has been overwritten twice since. The evidence trail is gone. `magic validate` would have caught this at commit time — 8 milliseconds, exit code 1, commit blocked.

That is what the kernel does. It makes governance a compile-time check, not a post-hoc audit. The theoretical basis for treating governance as compilation is established in Chapter 36 (Governance as Type System) and Chapter 37 (Governance as Compiler). This chapter covers the concrete tool. For the build pipeline that orchestrates `magic` alongside Jekyll and other stages, see Chapter 44. Developer tools are also available at dev.hadleylab.org.

44.1. 42.1 The Kernel

`magic.c` is a single-file C program — 35KB compiled, zero dependencies, O(1) compliance checking via bitwise AND. It exposes four verbs [28]:

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

44.2. 42.2 validate

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

Point `validate` at any directory containing a CANON.md and it reads the file, resolves the `inherits:` chain, checks all eight governance questions, and computes the MAGIC 255 score. The output is three lines: scope, score, tier [14].

44.3. 42.3 scan

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

`scan` walks the filesystem recursively, discovers every directory containing a CANON.md, and reports its score. No configuration file lists the scopes. No database tracks them. Discovery is structural — if the file exists, the scope exists [14].

44.4. 42.4 heal

```
$ magic heal
MISSING: LEARNING.md (question 7 — What have you learned?)
MISSING: ROADMAP.md (question 3 — Where are you going?)
ACTION: Create LEARNING.md, ROADMAP.md
```

`heal` identifies which governance questions remain unanswered and proposes specific fixes. It deliberately does not auto-generate files — governance is human-authored, and the kernel respects that boundary [14].

Diagnostic Output

```
$ magic heal --verbose
SCOPE: hadleylab-canonic/SERVICES/new-service
SCORE: 35/255  TIER: TRIAD
PRESENT:  CANON.md ✓  VOCAB.md ✓  README.md ✓
MISSING:  COVERAGE.md ✗  ROADMAP.md ✗  HTTP.md ✗  {SCOPE}.md ✗  LEARNING.md ✗
ACTIONS: 5 files to create. Score 35 -> 255.
```

Heal vs magic-heal

Tool	Language	Writes files?	Human approval?
`magic heal`	C	No (diagnostic)	N/A
`magic-heal`	Python	Yes (settlement)	Required

44.5. 42.5 The 255-Bit Standard

The 255-bit standard is the kernel's contract with every governed scope. MAGIC validates eight governance questions, each binary — satisfied or not. When all eight are satisfied, the scope compiles to 255 (all bits set in a `uint8_t`). The C kernel computes the score; the specific bit assignments and weights are implementation details inside `magic.c` [14].

The eight questions — *What do you believe? Can you prove it? Where are you going? Who are you? How do you work? What shape are you? What have you learned? How do you express?* — each map to one or more governance files. The mapping is deterministic: given the same filesystem state, the kernel always produces the same score. There is no randomness, no heuristic, no judgment call. The score is a pure function of what exists on disk.

44.6. 42.6 The `inherits:` Resolution

```
CANONIC-DOCTRINE/CANON.md
  inherits: hadleylab-canonic/BOOKS
    inherits: hadleylab-canonic
      inherits: hadleylab-canonic
        inherits: canonic-canonic (ROOT)
```

Governance files propagate downward through this chain. A child scope inherits all governance from its ancestors and can override any file by placing its own version in the child directory. The kernel enforces two structural constraints on the chain: no cycles (`CYCLE_DETECTED` — exit code 4) and no broken links (`BROKEN_INHERIT` — exit code 3). Both are fatal errors that halt validation entirely.

44.7. 42.7 Tier Classification

Score Range	Tier	Meaning
255	FULL	All eight questions answered
128-254	PARTIAL	Some unanswered
35-127	TRIAD	3 of 8
1-34	STUB	Minimal
0	NONE	No governance

Only FULL (255) passes CI gates. Everything below 255 is a work in progress — deployable to staging perhaps, but blocked from production [14].

44.8. 42.8 Exit Codes

Exit Code	Meaning
0	Score = 255
1	Score < 255
2	Parse error
3	Broken inherits
4	Cycle detected
5	File system error

```
magic validate || { echo "Governance incomplete"; exit 1; }
```

44.9. 42.9 Discovery Algorithm

```
magic_scan(root):
  FOR each dir D in recursive walk:
    IF D/CANON.md exists: EMIT D, score, tier
  RETURN scope count
```

The algorithm is deliberately simple: a scope exists if and only if its directory contains a CANON.md. Add a scope by creating that file. Remove a scope by deleting it. No registration step, no configuration update [14].

```
$ magic scan --tree
canonic-canonic (255 FULL)
├── hadleylab-canonic (255 FULL)
│   ├── DEXTER (255 FULL)
│   └── ADVENTHEALTH (255 FULL)
└── canonic-foundation (255 FULL)
```

44.10. 42.10 ledger

Event Type	Trigger	Meaning
`CREDIT:VALIDATE`	validate returns 255	Full governance confirmed
`CREDIT:DEPLOY`	deploy succeeds	Fleet pushed
`CREDIT:HEAL`	magic-heal settles	Scope healed
`DEBIT:DRIFT`	Score drops	Regression detected
`DEBIT:ROLLBACK`	rollback executed	Fleet reverted

The ledger is append-only — no entry is ever deleted or modified. Every event has a type, a trigger condition, and a meaning that maps directly to a governance state change [14].

44.11. 42.11 Compilation

```
cc -O2 -o magic magic.c                              \# Binary
cc -O2 -shared -fPIC -o libmagic.so magic.c           \# Linux
cc -O2 -dynamiclib -o libmagic.dylib magic.c          \# macOS
```

Zero dependencies. One source file. One command. You can compile the governance kernel on any system with a C compiler — no package manager, no build system, no vendored libraries [28].

44.12. 42.12 Performance

Validation is O(1) in question count — eight questions, eight bit checks, regardless of scope complexity. A single scope validates in under 10ms. A fleet of 73 scopes scans in under 1 second. The kernel is fast enough that you never think about whether to run it; you just run it on every commit.

44.13. 42.13 Clinical Application

```
$ magic validate hadleylab-canonic/SERVICES/radiology
SCORE: 255/255  TIER: FULL

$ git commit -m "update referral"
PRE-COMMIT: magic validate
SCORE: 247/255 (missing COVERAGE.md)
COMMIT BLOCKED.
```

The pre-commit hook is the governance gate at its most immediate: a developer cannot push a change that breaks the coverage contract. The patient never encounters software whose governance has lapsed [714].

44.14. 42.14 FFI Bindings

The shared library exposes the C ABI. Any language with FFI calls the kernel directly.

Python (ctypes)

```python
import ctypes
lib = ctypes.CDLL("libmagic.dylib")
lib.magic_validate.restype = ctypes.c_uint8
lib.magic_validate.argtypes = [ctypes.c_char_p]
score = lib.magic_validate(b"/path/to/scope")
```

Swift

```swift
@_silgen_name("magic_validate")
func magic_validate(_ path: UnsafePointer<CChar>) -> UInt8
let score = magic_validate("/path/to/scope")
```

TypeScript (ffi-napi)

```typescript
const ffi = require('ffi-napi');
const lib = ffi.Library('libmagic', {
  'magic_validate': ['uint8', ['string']]
});
const score = lib.magic_validate('/path/to/scope');
```

Every binding calls the same C function and gets the same `uint8_t` back. The kernel is the single source of truth; the binding is just syntax for reaching it. Whether you validate from a Python script, a Swift app, or a TypeScript build tool, the answer is the same [28].

44.15. 42.15 Extending the Kernel

The mapping from governance questions to files is defined in `magic.c` itself — the C kernel is the single source of truth for the validation algorithm. If you want to understand how scoring works, read the source. It is 35KB and deliberately written to be auditable by a single engineer in an afternoon [14].

The framework is extensible in a controlled way: add a question to the kernel, recompile, and every scope in the fleet is validated against the new standard. Scores may decrease — scopes that were at 255 might drop if they lack the newly required file. The heal loop identifies the gap, and the system evolves forward. Backward compatibility is not a goal; correctness is [14].

44.16. 42.16 Multi-Org Deployments

```
$ magic scan
canonic-canonic                      255  FULL
hadleylab-canonic                    255  FULL
hadleylab-canonic              255  FULL
adventhealth-canonic                 255  FULL

Four scopes across two organizations, all at 255, validated by the same kernel running the same algorithm.
```

44.17. 42.17 Error Recovery

The kernel classifies errors into two severity levels. Structural errors (E100-E105) — malformed CANON.md, broken inheritance, cycles — halt validation entirely and produce a score of 0. Question errors (E200-E207) reduce the score but allow validation to complete, so you can see which questions remain unanswered:

```
$ magic validate --verbose
ERROR: E101 MALFORMED\_CANON -- invalid YAML
SCORE: 0/255 (halted at structural error)
```

The classification guides your fix priority: resolve parse errors first (the kernel cannot even read your governance), then address unanswered questions one by one [14].

44.18. 42.18 Clinical Vignette: magic validate in a Pre-Operative Checklist System

Children's Hospital of Philadelphia (CHOP) deploys SurgChat — a governed TALK agent for pediatric surgical safety checklists. SurgChat's evidence base includes WHO Surgical Safety Checklist items, ASA physical status classification, and age-adjusted vital sign ranges for pediatric patients.

The surgical quality improvement team adds a new checklist item: verification of blood type and crossmatch for cases with expected blood loss exceeding 10% of estimated blood volume (calculated as 80 mL/kg for infants, 70 mL/kg for children). The developer adds the constraint to CANON.md:

```
MUST: Verify BLOOD\_TYPE\_CROSSMATCH for cases with expected blood loss > 10% EBV
```

The pre-commit hook runs `magic validate`:

```
magic validate SERVICES/TALK/SURGCHAT
# E300 VOCAB\_UNDEFINED: BLOOD\_TYPE\_CROSSMATCH not in VOCAB.md
# E300 VOCAB\_UNDEFINED: EBV not in VOCAB.md
# Score: 247/255 (vocabulary question failed — undefined terms)
# COMMIT BLOCKED.
```

The developer adds both terms to VOCAB.md:

```
| BLOOD\_TYPE\_CROSSMATCH | ABO/Rh typing and antibody screen with crossmatch verification — required per
| EBV | Estimated Blood Volume — calculated as 80 mL/kg for infants (<1 year), 70 mL/kg for children (1–12
```

The developer re-commits. `magic validate` returns 255. The vocabulary is closed. Every term used in governance has a precise, cited definition. A pediatric anesthesiologist reviewing SurgChat's VOCAB.md sees exactly what BLOOD_TYPE_CROSSMATCH means in this context — not "blood test" but a specific AABB-standard procedure triggered by a specific clinical threshold.

One C function checked every term in every governance file against the vocabulary chain, identified two undefined terms, blocked the commit, and required precise definitions before proceeding. The entire operation took 8ms. The patient safety improvement shipped with complete governance — every clinical term defined, every definition cited, every definition reviewable by a colleague who was not in the room when the code was written [1428].

44.19. 42.19 Kernel Architecture Details

The `magic.c` kernel implements validation in four phases:

```
uint8_t magic_validate(const char *scope_path);
// Returns: 0–255. The math is in magic.c.
// Phase 1: Parse CANON.md (E100–E102 on failure)
// Phase 2: Resolve inheritance chain (E103–E105 on failure)
// Phase 3: Check eight governance questions (one bit each)
// Phase 4: Vocabulary closure check
// The implementation is the kernel. Defer to magic.c.
```

The score is a single `uint8_t`. Each governance question sets one bit. The most expensive operation is vocabulary closure — walking all governance files, extracting SCREAMING_CASE terms, and verifying each one against the vocabulary chain — and even that is O(n) in total word count, not in scope count.

The kernel's simplicity is deliberate and load-bearing. A 35KB binary with zero dependencies can be audited by a single engineer in one afternoon. Compare that to enterprise governance platforms with millions of lines of code, hundreds of dependencies, and audit timelines measured in months. The kernel is auditable because it is small, and correct because it is auditable. These are not independent properties — they reinforce each other [28].

44.20. 42.20 magic scan: Fleet Intelligence

`magic scan` is the fleet intelligence command. It discovers every governed scope in a repository or across a fleet and reports their status:

```
magic scan --json hadleylab-canonic
# {
#   "fleet": "hadleylab-canonic",
#   "scopes": 73,
#   "at_255": 73,
#   "utilization": 1.0,
#   "total_coin": 18615,
#   "scopes_list": [
#     {"path": "DEXTER", "score": 255, "tier": "FULL"},
#     {"path": "BLOGS", "score": 255, "tier": "FULL"},
#     {"path": "BOOKS/CANONIC-DOCTRINE", "score": 255, "tier": "FULL"},
#     ...
#   ]
# }
```

The JSON output feeds into MONITORING dashboards, VAULT projections, and fleet page generators. `scan` is the source of truth for a question every CTO eventually asks: "how many governed scopes do we have and what state are they in?" The answer is not stored in a database or a configuration file. The filesystem is the scope registry. `magic scan` walks it and discovers reality [14].

44.21. 42.21 Governance Proof: Kernel Correctness

The kernel is correct by construction. The proof:

1. The score is a `uint8_t` — range [0, 255], no overflow possible.
2. Each question check is independent (no interaction between bits).
3. Each check is a governance file existence test (deterministic, side-effect-free).
4. The vocabulary closure check is decidable (finite vocabulary, finite files).
5. The inheritance resolution terminates (cycle detection halts on first cycle).
6. The output is the bitwise OR of all question checks — commutative and associative.

Given the same filesystem state, `magic validate` always returns the same score. The kernel is a pure function from filesystem state to `uint8_t`. Pure functions are testable, reproducible, and auditable — and the kernel is all three. You can verify this yourself: clone the repository, run `magic validate`, change nothing, run it again. Same number. Q.E.D. [283].

44.22. 42.22 magic ledger: Economic Event Query

`magic ledger` queries the LEDGER for economic events. The command provides the economic view of governance activity:

```
magic ledger --last 10

# Recent LEDGER events:
# evt:04930  BUILD:COMPLETE    2026-03-10T15:00:00Z  73/73 at 255
# evt:04929  MINT:WORK         2026-03-10T14:55:00Z  +31 COIN  dexter  referral
# evt:04928  DEPLOY            2026-03-10T14:50:00Z  hadleylab.org  255
# evt:04927  MINT:WORK         2026-03-10T14:45:00Z  +92 COIN  dexter  FHIR-API
# evt:04926  NOTIFIER:DELIVER  2026-03-10T14:40:00Z  EVIDENCE\_UPDATE  5 recipients
# evt:04925  SPEND             2026-03-10T12:00:00Z  -255 COIN  dr.park  FHIR-playbook
# evt:04924  TRANSFER          2026-03-10T11:30:00Z  -100 COIN  dr.chen → dr.park
# evt:04923  MONITORING:SCORE  2026-03-10T11:00:00Z  73/73 at 255
# evt:04922  MINT:SIGNUP       2026-03-10T10:00:00Z  +255 COIN  new-intern
# evt:04921  IDENTITY:VERIFY   2026-03-10T09:30:00Z  new-intern  NPI verified
```

The LEDGER is the economic backbone of the system — every COIN event, every deployment, every notification delivery produces an immutable entry. The `magic ledger` command provides filtered views into this history:

```
# Filter by event type
magic ledger --type MINT:WORK --since 2026-03-01
# 47 MINT:WORK events in March
```

```
# Filter by principal
magic ledger --principal dexter --since 2026-01-01
# 234 events attributed to dexter in Q1

# Filter by scope
magic ledger --scope SERVICES/TALK/MAMMOCHAT --since 2025-01-01
# 89 events in MammoChat scope (12 months)

# Export for audit
magic ledger --all --format json > ledger-export-2026-03-10.json
```

44.23. 42.23 magic commit: Governed Commit Wrapper

`magic commit` wraps `git commit` with governance enforcement. It runs `magic validate` before the commit and records the score in the commit message:

```
magic commit -m "GOV: add COVERAGE.md for lab-orders"

# Pre-commit: magic validate
# Score: 255/255 
# Committing...
# [main abc1234] GOV: add COVERAGE.md for lab-orders
#   Score: 255/255 (FULL)
#   Gradient: +8 (was 247)
#   MINT:WORK: +8 COIN
#   LEDGER: evt:04931
```

`magic commit` turns every git commit into a governance receipt. It adds metadata directly to the commit message: the score at commit time, the gradient from the previous commit (how much the score improved), the COIN minted, and the LEDGER event ID. The commit message is both human-readable and machine-parseable — your git log becomes a governance audit trail [14].

44.24. 42.24 magic scan Filters and Flags

`magic scan` supports rich filtering for fleet management:

```
# Find all scopes below 255
magic scan --below 255
# SERVICES/NEW-AGENT: 191 (missing ROADMAP)
# SERVICES/EXPERIMENTAL: 127 (missing LEARNING, ROADMAP)
```

```
# Find scopes by tier
magic scan --tier AGENT  \# All scopes at 127 (AGENT tier)

# Find scopes by dimension
magic scan --missing LEARNING  \# All scopes without LEARNING.md
# SERVICES/NEW-AGENT: missing LEARNING
# SERVICES/EXPERIMENTAL: missing LEARNING

# Find stale scopes (no commit in 90 days)
magic scan --stale 90
# SERVICES/LEGACY-FHIR: last commit 127 days ago

# Fleet-wide summary
magic scan --summary
# Fleet: hadleylab-canonic
# Total scopes: 73
# At 255 (FULL): 71 (97.3%)
# At 191 (MAGIC): 1 (1.4%)
# At 127 (AGENT): 1 (1.4%)
# Below 127: 0 (0%)
# Total COIN: 18,615
# Healing debt: 2 scopes (192 COIN to heal)
```

The `--summary` flag is the fleet-level governance health metric. Two scopes below 255 means 2.7% governance debt. The healing debt (192 COIN) represents the total COIN that would be minted if those scopes were brought to 255 — it quantifies the cost of incomplete governance in the system's own currency. For technical leadership, this metric drives resource allocation: you can see exactly how much governance work remains and assign it accordingly [14].

44.25. 42.25 magic validate Verbose Mode

Verbose validation shows every question check with detailed diagnostics:

```
magic validate --verbose SERVICES/TALK/MAMMOCHAT

# SCOPE: hadleylab-canonic/SERVICES/TALK/MAMMOCHAT
# INHERITS: hadleylab-canonic/SERVICES/TALK
\#
# Question checks:
#   [1] What do you believe?     CANON.md          □   (axiom present)
#   [2] Can you prove it?        VAULT/            □   (wallet: mammochat-wallet, 2,847 COIN)
#   [3] Where are you going?     ROADMAP.md        □   (4 quarters, last updated 2026-02-15)
```

```
#   [4] Who are you?            MAMMOCHAT.md      □   (2,400 words, 12 sections)
#   [5] How do you work?        COVERAGE.md       □   (8 questions, 100% filled, 6 citations)
#   [6] What shape are you?     README.md         □   (inherits: + axiom present)
#   [7] What have you learned?  LEARNING.md       □   (3 epochs, 18 entries, last entry 2026-03-08)
#   [8] How is it expressed?    VOCAB.md          □   (38 local terms + 142 inherited, closure: COMPLETE)
\#
# VOCAB closure analysis:
#   Terms in scope governance files: 47 unique SCREAMING\_CASE
#   Terms defined in VOCAB chain: 47/47 (100%)
#   Undefined terms: 0
#   Closure: COMPLETE
\#
# Inheritance chain:
#   canonic-canonic (root, 255)
#   └── canonic-canonic/MAGIC (255)
#       └── hadleylab-canonic (255)
#           └── hadleylab-canonic/SERVICES (255)
#               └── hadleylab-canonic/SERVICES/TALK (255)
#                   └── hadleylab-canonic/SERVICES/TALK/MAMMOCHAT (255)
#   Chain depth: 6 (max recommended: 8)
#   All ancestors at 255: YES
\#
# SCORE: 255/255
# TIER: FULL
# COIN minted this session: 0 (no gradient — already at 255)
```

Verbose mode is your primary diagnostic tool. When a scope fails validation, it reveals exactly why — which question failed, which file is missing or incomplete, and where in the inheritance chain the problem originates. You will use this output more than any other `magic` command when debugging governance gaps [1428].

44.26. 42.26 Clinical Vignette: magic validate in a Residency Training Program

Duke University's radiology residency program integrates CANONIC governance into the informatics rotation. Each resident is assigned a governance scope (a TALK agent for their subspecialty interest) and must bring it to 255 as part of the rotation curriculum.

A second-year resident working on NeuroRadChat (neuroradiology consultation) runs `magic validate` after adding the initial governance files:

```
magic validate --verbose SERVICES/TALK/NEURORADCHAT

# Question checks:
#   [1] What do you believe?     CANON.md          □
#   [2] Can you prove it?        VAULT/            □  (no wallet — run vault onboard)
#   [3] Where are you going?     ROADMAP.md        □  (file missing — E202)
#   [4] Who are you?             NEURORADCHAT.md   □
#   [5] How do you work?         COVERAGE.md       □  (file exists but empty — E207)
#   [6] What shape are you?      README.md         □
#   [7] What have you learned?   LEARNING.md       □
#   [8] How is it expressed?     VOCAB.md          □  (3 undefined terms: ASPECTS\_SCORE, DWI\_MISMATCH,
\#
# SCORE: 161/255
# Missing: 4 questions unanswered = 94 points to heal
# TIER: AGENT (>127)
# Healing plan: magic heal --verbose
```

The resident learns governance by doing governance. The score is immediate, objective, and actionable — no need to ask a faculty member "is my governance complete?" when the kernel answers the question in 8ms. The resident adds the missing VOCAB definitions (ASPECTS score = Alberta Stroke Programme Early CT Score, 0-10 for middle cerebral artery territory; DWI_MISMATCH = diffusion-perfusion mismatch ratio for thrombectomy eligibility; CTA_OCCLUSION = CT angiography showing large vessel occlusion), fills COVERAGE.md, creates ROADMAP.md, runs `vault onboard`, and achieves 255 in four commits.

The faculty advisor reviews the governance tree using the same tool the resident used: `magic validate --verbose`. The feedback loop is tight — write governance, validate, fix, validate again. The residency rotation produces a governed clinical AI scope as its educational artifact. Not a paper. Not a presentation. A governed, validated, deployable scope that actually does something [71428].

44.27. 42.27 magic as Governance Kernel

`magic` is the kernel of the CANONIC toolchain — every other tool depends on it. `build` calls `validate` before compiling. `deploy` calls `validate` before shipping. `magic-heal` reads the same scores to determine what needs fixing. The kernel's contract is absolute: given a directory with CANON.md, produce a deterministic 0-255 score based on the presence, completeness, and consistency of the eight governance questions.

What the kernel does not do is equally important. It does not evaluate whether your axiom is good. It does not judge whether your constraints are reasonable. It verifies structural completeness — nothing more. Content quality is your responsibility as the author. Structural completeness is the kernel's guarantee. This separation of concerns is what makes CANONIC governance scalable: a single kernel validates one scope or an entire fleet of 73 scopes in under 500ms, and the score it produces is objective, reproducible, and auditable. Read the files, check the questions, emit the score — that is the entire algorithm [1428].

44.28. VaaS: The Kernel as Product

The public GOV tree demonstrates the governance standard — anyone can see how CANONIC works by reading the Markdown. The C kernel is the closed product. VaaS (Validation as a Service) monetizes the gap: organizations explore the framework for free on GitHub and pay for the runtime that produces the 255 score against their own fleet.

The kernel's size is a product decision, not a technical limitation. A 35KB binary with zero dependencies can be audited by a single engineer in one afternoon. A governance kernel that requires a six-month audit process is not a governance kernel — it is an enterprise platform. CANONIC chose the opposite path: small enough to trust, fast enough to run on every commit.

44.29. FRESHNESS: Incremental Compilation

The build pipeline supports incremental compilation via the FRESHNESS subsystem. A cache at `~/.canonic/.build-cache.` tracks source file modification times for every scope. On subsequent builds, unchanged scopes are skipped entirely — the kernel only re-validates what actually changed.

The performance difference is dramatic: full fleet compilation dropped from 134 seconds to 3 seconds with a warm cache. CI builds still perform full validation (the `--no-cache` flag forces a cold run), because the cache is a development convenience, not a correctness shortcut. When you need the authoritative answer, you pay the 134 seconds. When you are iterating on a single scope, you get your answer in 3.

Chapter 45

Chapter 43: magic-heal

Python. Five-stage settlement. Where `magic validate` (Chapter 42) detects governance gaps and Chapter 45 catalogs the error codes, `magic-heal` closes them through a structured PROPOSE-SETTLE pattern. The healing convergence proof mirrors the gradient convergence of Chapter 33, and the LEDGER events it produces are recorded per Chapter 13. The healing dashboard feeds dev.hadleylab.org via the MONITORING service (Chapter 22).

45.1. 43.1 The Five Stages

`magic-heal` 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
```

45.2. 43.2 Usage

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

The healer walks the GOV tree, identifies scopes below 255, proposes governance files, settles with your approval, and revalidates [14].

45.3. 43.3 Stage 1: SCAN

SCAN discovers every scope beneath the target path by calling `magic scan` internally and filtering for scopes below 255.

```
$ magic-heal hadleylab-canonic/ --stage scan
SCANNING: hadleylab-canonic/
  hadleylab-canonic/SERVICES/referral         255  FULL    (skip)
  hadleylab-canonic/SERVICES/lab-orders        255  FULL    (skip)
  hadleylab-canonic/SERVICES/new-imaging        35  TRIAD   (heal)
  hadleylab-canonic/SERVICES/pharmacy          128  PARTIAL (heal)

SCAN RESULT: 4 scopes found. 2 require healing.
```

Scopes at 255 are skipped. Scopes below 255 are queued for diagnosis. Discovery is structural [14].

Recursive vs Targeted Scan

```
magic-heal hadleylab-canonic/SERVICES/new-imaging   \# One scope
magic-heal hadleylab-canonic/SERVICES/               \# All services
magic-heal hadleylab-canonic/                               \# Entire org
```

45.4. 43.4 Stage 2: DIAGNOSE

DIAGNOSE classifies every missing dimension by reading the bitmask, identifying zero bits, and mapping each to its corresponding file.

```
$ magic-heal hadleylab-canonic/SERVICES/new-imaging --stage diagnose
DIAGNOSING: new-imaging  SCORE: 35/255  BITMASK: 00100011

  [8]   O  COVERAGE.md      MISSING
  [16]  T  ROADMAP.md       MISSING
  [32]  R  HTTP.md          MISSING
  [64]  LANG VOCAB.md       INHERITED from hadleylab-canonic
  [128] L  LEARNING.md      MISSING

RESULT: 4 dimensions missing (LANG inherited). 4 files to create.
```

Diagnosis Classification

Classification	Meaning	Action
`MISSING`	File does not exist at any level	Create file
`INHERITED`	File exists in parent scope	No action needed
`BROKEN_INHERIT`	Parent claims to provide but file missing	Fix parent
`SHADOWED`	Child overrides parent with empty file	Fill file or remove
`MALFORMED`	File exists but cannot be parsed	Fix file content

45.5. 43.5 Stage 3: PROPOSE

PROPOSE generates `.md` template files for every missing dimension, pre-populated with scope-specific metadata and containing the minimal structure required for the dimension.

```
$ magic-heal new-imaging --stage propose
--- COVERAGE.md (proposed) ---
# COVERAGE -- new-imaging
| Capability | Status | Evidence |
|------------|--------|----------|
| [TODO] | [TODO] | [TODO] |

--- ROADMAP.md (proposed) ---
# ROADMAP -- new-imaging
| Milestone | Target | Status |
|-----------|--------|--------|
| [TODO] | [TODO] | [TODO] |

--- HTTP.md (proposed) ---
# HTTP -- new-imaging
| Method | Path | Contract |
|--------|------|----------|
| [TODO] | [TODO] | [TODO] |

--- LEARNING.md (proposed) ---
# LEARNING -- new-imaging
- [TODO: Record patterns, decisions, discoveries]

PROPOSE RESULT: 4 templates generated. Ready for SETTLE.
```

Templates are starting points – you fill the `[TODO]` markers. `magic-heal` generates the skeleton; you provide

the substance [14].

Template Customization

```
# In parent CANON.md
heal-templates:
  COVERAGE: templates/clinical-coverage.md
  HTTP: templates/fhir-routes.md
```

A hospital defines clinical templates with HIPAA fields. The healer adapts to the governance context [7].

45.6. 43.6 Stage 4: SETTLE

SETTLE writes proposed files to disk, but only with your explicit approval.

```
$ magic-heal new-imaging --stage settle
SETTLE: 4 files will be created.
Proceed? [y/N]: y
WRITING: COVERAGE.md   ✓
WRITING: ROADMAP.md    ✓
WRITING: HTTP.md       ✓
WRITING: LEARNING.md   ✓
SETTLE RESULT: 4 files written.
```

There is no `--force` flag. Every file write requires human acknowledgment [14].

Mode	Behavior	Flag
Interactive	Prompt for each file individually	`--interactive`
Batch	Prompt once for all files	(default)
Dry-run	Show what would be written, write nothing	`--dry-run`

If a proposed file already exists, the healer does not overwrite. Existing files are never overwritten [14].

45.7. 43.7 Stage 5: VALIDATE

```
$ magic-heal new-imaging --stage validate
BEFORE: 35/255 (TRIAD)
AFTER:  255/255 (FULL)
HEAL COMPLETE.
```

If the score does not reach 255, the healer reports remaining gaps. It creates files but does not fill them – you must author the governance content [14].

45.8. 43.8 Full Pipeline Example

```
$ magic-heal hadleylab-canonic/SERVICES/new-imaging

=== STAGE 1: SCAN ===      1 scope, score 35/255
=== STAGE 2: DIAGNOSE ===  4 files to create
=== STAGE 3: PROPOSE ===   Templates ready
=== STAGE 4: SETTLE ===    4 files written
=== STAGE 5: VALIDATE ===  FULL (255/255)

LEDGER: CREDIT:HEAL new-imaging 255 FULL
```

45.9. 43.9 Clinical Healing Example

A hospital adds a new pathology service at TRIAD (35/255):

```
$ magic-heal hadleylab-canonic/SERVICES/pathology
SCAN: 1 scope at 35/255
DIAGNOSE: COVERAGE, ROADMAP, HTTP, LEARNING missing. VOCAB inherited.
PROPOSE: 4 templates with clinical headers
SETTLE: Write 4 files? [y/N]: y
VALIDATE: 35 -> 255 (FULL)
LEDGER: CREDIT:HEAL pathology 255 FULL
```

45.10. 43.10 Healing Debt

```
$ magic scan --below 255
hadleylab-canonic/SERVICES/new-imaging    35  TRIAD
hadleylab-canonic/SERVICES/pharmacy      128  PARTIAL
hadleylab-canonic/SERVICES/billing         7  STUB
```

Three scopes below 255. This is healing debt. Reduce it to zero before the next release [14].

45.11. 43.11 The Settlement Model

`magic-heal` implements a settlement model borrowed from financial systems: create the file, record in LEDGER, confirm the score.

Three properties hold:

1. **Atomicity.** All proposed files written or none. No partial settlement.
2. **Finality.** CREDIT:HEAL in LEDGER is immutable. Later deletion records DEBIT:DRIFT.
3. **Auditability.** Every settlement recorded with timestamp, scope, before/after score, files created.

45.12. 43.12 Batch Healing

```
$ magic-heal hadleylab-canonic/
SCAN: 73 scopes, 5 below 255
DIAGNOSE: 17 files missing across 5 scopes
PROPOSE: 17 templates generated
SETTLE: Write 17 files? [y/N]: y
VALIDATE: 5 scopes healed. 73/73 at 255.
LEDGER: 5 CREDIT:HEAL events.
```

45.13. 43.13 Healing Frequency

Pace	Frequency	Strategy
1 scope/week	Weekly	Heal before release
1 scope/day	Daily	Heal end of day
Burst (5+)	Per-burst	Heal immediately
Stable	Monthly	Verify no drift

45.14. 43.14 Healing vs Hand-Authoring

Templates contain `[TODO]` markers. A healed scope with unfilled templates may score 255 (the file exists) but lack substance. Structural completeness satisfies `magic validate`; semantic completeness satisfies the governance auditor. Both are necessary – the toolchain enforces structural, you ensure semantic [14].

Best practice: run `magic-heal`, then immediately fill every `[TODO]` marker. Never leave templates unfilled for more than one release cycle.

45.15. 43.15 Clinical Vignette: Batch Healing a Hospital Onboarding

Vanderbilt University Medical Center (VUMC) adopts CANONIC governance for their clinical informatics division. The division operates 28 clinical AI services – none governed. The clinical informatics director schedules a one-week "governance sprint" to bring all 28 services to 255.

Day 1: The team runs `magic scan`:

```
magic scan --below 255 vanderbilt-canonic
# 28 scopes at 0/255 (no CANON.md)
# Healing debt: 28 scopes x 255 COIN = 7,140 COIN potential
```

Day 1-2: The team bootstraps all 28 CANON.md files with axioms and constraints. `magic-heal` generates VOCAB.md templates for each:

```
magic-heal vanderbilt-canonic/ --stage scan
# 28 scopes at 0-35/255
magic-heal vanderbilt-canonic/ --stage diagnose
# 140 files missing across 28 scopes (avg 5 per scope)
magic-heal vanderbilt-canonic/ --stage propose
# 140 templates generated from clinical-governance template pack
magic-heal vanderbilt-canonic/ --stage settle
# Write 140 files? [y/N]: y
# 140 files written.
magic-heal vanderbilt-canonic/ --stage validate
# 28 scopes: 14 at 255, 10 at 191, 4 at 127
# Remaining debt: 14 scopes need manual content
```

Day 3-5: Teams fill `[TODO]` markers. The clinical pharmacology team fills DrugChat's LEARNING.md with 18 months of institutional knowledge: drug interaction alert fatigue patterns, clinical decision support override rates (23% for vancomycin dosing alerts), and EHR integration latency benchmarks.

Day 5 end:

```
magic scan vanderbilt-canonic
# 28/28 scopes at 255/255
# COIN minted: 7,140 COIN
# Healing sprint complete.
```

Five days. 28 scopes. 7,140 COIN. The entire clinical AI portfolio is governed. `magic-heal` generated the structural skeleton; the clinical teams provided the substance. Neither could have succeeded alone [14].

45.16. 43.16 Healing Economics

Healing has quantifiable economic value:

```
unhealedCost = scopes_below_255 * avg_missing_dimensions * dimension_avg_weight
healedValue = scopes_healed * 255
ROI = healedValue / (heal_time_hours * hourly_rate)
```

For VUMC's sprint: 28 scopes healed, 5 team members, 5 days (200 person-hours). At $75/hour, the sprint cost $15,000. The 7,140 COIN at $1.00/COIN settle rate yields $7,140 in immediate value. But the ongoing value – automated compliance verification, eliminated governance committee meetings (estimated 400 person-hours/year at $75/hour = $30,000/year), and audit-ready documentation – exceeds the sprint cost within the first quarter [12].

45.17. 43.17 Governance Proof: Healing Convergence

`magic-heal` converges to 255 in a bounded number of iterations. The proof:

1. Each SETTLE stage creates at least one new governance file.
2. Each new file activates at least one dimension bit.
3. Each activated bit strictly increases the score.
4. The score is bounded above by 255.
5. Therefore, `magic-heal` converges in at most 8 iterations (one per dimension).

In practice, a single `magic-heal` pass typically creates all missing files simultaneously, reaching 255 in one iteration. The bounded convergence guarantee means healing is always tractable — no scope is permanently unhealable. Q.E.D. [143].

45.18. 43.18 Healing Template Packs

`magic-heal` uses template packs to generate domain-specific governance files. Template packs are governed collections of `.md` templates tailored to specific industries:

Template Pack	Domain	Templates	Vocabulary	Clinical Terms
`clinical-governance`	Healthcare	8 templates	142 base terms	HIPAA, NCCN, BI-RADS, FHIR
`clinical-talk`	Clinical AI agents	8 templates	180 terms	systemPrompt, disclaimer, routing

Template Pack	Domain	Templates	Vocabulary	Clinical Terms
`research-governance`	Academic research	8 templates	95 terms	IRB, NIH, protocol, consent
`enterprise-governance`	General enterprise	8 templates	60 terms	SOX, SOC2, change management
`infrastructure`	Technical services	8 templates	75 terms	API, deployment, monitoring

```
# List available template packs
magic-heal --list-templates

# Templates available:
#   clinical-governance (v2026.1) — 8 templates, 142 terms
#   clinical-talk (v2026.1) — 8 templates, 180 terms
#   research-governance (v2026.1) — 8 templates, 95 terms
#   enterprise-governance (v2026.1) — 8 templates, 60 terms
#   infrastructure (v2026.1) — 8 templates, 75 terms

# Use specific template pack
magic-heal SERVICES/TALK/NEPHROCHAT --template clinical-talk
```

Template packs are themselves governed — each pack is a scope with CANON.md, VOCAB.md, and LEARNING.md. The pack's VOCAB.md defines the domain terminology. The pack's LEARNING.md records which templates work well for which use cases. Template packs are available in the SHOP — organizations can purchase domain-specific template packs from other organizations [1422].

45.19. 43.19 Healing Diff Review

Before writing files, `magic-heal` shows a diff of proposed changes:

```
magic-heal SERVICES/TALK/NEPHROCHAT --stage propose --diff

# Proposed changes:
\#
# NEW: SERVICES/TALK/NEPHROCHAT/COVERAGE.md
# +---
# +inherits: hadleylab-canonic/SERVICES/TALK
# +---
# +
# +\# COVERAGE — NEPHROCHAT
# +
```

```
# +\#\# Coverage Matrix
# +
# +| Dimension | Source | Status | Notes |
# +|-----------|--------|--------|-------|
# +| D (AXIOM) | CANON.md | [TODO] | Define nephrology AI axiom |
# +| E (EVIDENCE) | VAULT/ | [TODO] | Configure wallet |
# +| S (SCOPE) | NEPHROCHAT.md | [TODO] | Write spec |
# +| O (OPERATIONS) | COVERAGE.md | PRESENT | This file |
# +| T (TIMELINE) | ROADMAP.md | [TODO] | Define quarters |
# +| R (ROUTES) | HTTP.md | [TODO] | Define API endpoints |
# +| LANG (LANGUAGE) | VOCAB.md | [TODO] | Define nephrology terms |
# +| L (LEARNING) | LEARNING.md | [TODO] | Initialize epoch 1 |
# +
# +\#\# Evidence Chain
# +...
\#
# NEW: SERVICES/TALK/NEPHROCHAT/ROADMAP.md
# +---
# +inherits: hadleylab-canonic/SERVICES/TALK
# +---
# +
# +\# ROADMAP — NEPHROCHAT
# +...
\#
# 3 new files proposed. Review? [y/N]
```

The diff review enables governance officers to inspect proposed files before they are written. The officer can modify the proposal by editing the template, reject the proposal entirely, or accept and write. The healing tool proposes. The human decides. The invariant holds: machines propose, humans govern [14].

45.20. 43.20 Healing Batch Operations

For fleet-wide healing, `magic-heal` supports batch operations across multiple scopes:

```
# Heal all scopes below 255
magic-heal --batch --target 255

# Batch healing plan:
\#
# Scope                          Score  Target  Missing Dimensions
# SERVICES/TALK/ALLERGYCHAT      191    255     ROADMAP (128)
# SERVICES/TALK/PULMOBOT         191    255     ROADMAP (128)
```

```
# SERVICES/MONITORING/CARDIO      247    255     LEARNING (64) [partial]
\#
# Total scopes: 3
# Total files to generate: 3
# Estimated COIN on completion: 192
\#
# Execute batch healing? [y/N] y
\#
# [1/3] ALLERGYCHAT: ROADMAP.md generated □
# [2/3] PULMOBOT: ROADMAP.md generated □
# [3/3] CARDIO-MONITORING: LEARNING.md generated □
\#
# Batch complete. 3 files written.
# Run: magic validate --recursive to verify scores
```

Batch healing is the fleet-level healing operation. It identifies every scope below 255, generates the missing governance files, and writes them in a single operation. The batch operation maintains the same human-approval gate — the operator confirms before any files are written [14].

45.21. 43.21 Healing and the PROPOSE-SETTLE Pattern

The PROPOSE-SETTLE pattern is fundamental to CANONIC healing. It mirrors the settlement process in financial systems:

```
PROPOSE: Tool generates candidate governance files
         Candidates are structurally valid but semantically generic
         Candidates contain [TODO] markers for domain-specific content

SETTLE:  Human reviews candidates
         Human fills [TODO] markers with domain-specific content
         Human commits the settled files
         magic validate confirms 255
         COIN mints for the governance improvement
```

The pattern ensures that machine-generated governance is always reviewed by a human before it enters the governance tree. The tool provides structure (file format, required sections, template vocabulary). The human provides substance (clinical knowledge, institutional context, domain-specific constraints).

```
# The full PROPOSE-SETTLE cycle
magic-heal .                          \# PROPOSE: generate templates
vim COVERAGE.md                       \# SETTLE: add clinical coverage details
vim ROADMAP.md                        \# SETTLE: define institutional timeline
```

```
magic validate                          \# VERIFY: confirm 255
git commit -m "GOV: settle healing"     \# RECORD: commit governance improvement
# MINT:WORK: +128 COIN                 \# REWARD: economic receipt
```

The PROPOSE-SETTLE pattern prevents two failure modes:

1. **Pure machine healing.** Without human settlement, governance files would contain generic templates that satisfy the validator but lack clinical substance. A COVERAGE.md with `[TODO]` in every cell is structurally valid (the file exists, the matrix has the right shape) but clinically useless.
2. **Pure human authoring.** Without machine proposals, humans would author governance files from scratch — slower, more error-prone, and inconsistent across scopes. The template provides the structure that the human fills with substance [14].

45.22. 43.22 Healing Priority Algorithm

When multiple dimensions are missing, `magic-heal` prioritizes based on bit weight — highest-weight missing dimension first:

```
Priority order (highest weight first):
  1. ROADMAP (128)    — most COIN, most strategic
  2. LEARNING (64)    — institutional memory
  3. ROUTES (32)      — API contract
  4. OPERATIONS (16)  — coverage requirements
  5. EVIDENCE (8)     — economic identity
  6. LANGUAGE (4)     — vocabulary closure
  7. SCOPE (2)        — scope specification
  8. AXIOM (1)        — governance declaration
```

The order is deliberate. ROADMAP (128 COIN) delivers the most economic value per file authored. A scope missing only ROADMAP sits at 127 – adding ROADMAP jumps to 255 in one commit. The economic incentive aligns with the healing priority.

```
magic-heal --priority SERVICES/NEW-AGENT

# Healing priority:
#   Current score: 63 (6 dimensions missing)
#   Priority 1: ROADMAP.md (+128)  → score 191
#   Priority 2: LEARNING.md (+64)  → score 255
#   ... (remaining dimensions already present)
\#
# Recommended: Add ROADMAP.md first (highest single-commit impact)
```

45.23. 43.23 Clinical Vignette: Healing Sprint Saves Joint Commission Survey

Massachusetts General Hospital (MGH) receives 60-day notice of a Joint Commission survey. The survey will include review of clinical AI governance practices. MGH operates 4 clinical AI agents (MammoChat, OncoChat, CardiChat, DermaChat) through CANONIC governance, but 2 agents have drifted below 255 due to stale LEARNING.md files (the clinical teams were focused on patient care, not governance documentation).

```
magic scan --below 255
# SERVICES/TALK/CARDICHAT:  191 (missing LEARNING — stale 90 days)
# SERVICES/TALK/DERMACHAT:  191 (missing LEARNING — stale 120 days)
# Fleet: 2/4 below 255. Joint Commission survey in 60 days.
```

The governance officer initiates a healing sprint:

```
# Day 1: Diagnose
magic-heal SERVICES/TALK/CARDICHAT --stage scan
# Missing: LEARNING.md (stale > 90 days → dimension cleared)
# Proposal: Update LEARNING.md with recent patterns

magic-heal SERVICES/TALK/DERMACHAT --stage scan
# Missing: LEARNING.md (stale > 120 days → dimension cleared)
# Proposal: Update LEARNING.md with recent patterns

# Day 1: Propose
magic-heal SERVICES/TALK/CARDICHAT --stage propose
# Template generated: LEARNING.md with Epoch 3 structure
magic-heal SERVICES/TALK/DERMACHAT --stage propose
# Template generated: LEARNING.md with Epoch 4 structure
```

Day 2-3: Clinical teams settle the templates. The cardiology team adds 8 LEARNING entries from the past 90 days: heart failure readmission prediction accuracy (improved 12% after adding BNP trending), SGLT2 inhibitor query patterns (4x increase in dapagliflozin questions), and telemetry alert fatigue rate (reduced 34% after confidence threshold adjustment). The dermatology team adds 6 entries: melanoma staging query patterns, dermoscopy image interpretation correlation, and formulary interaction query frequency.

```
# Day 3: Validate and commit
magic validate --recursive SERVICES/TALK/CARDICHAT
# 255/255 □

magic validate --recursive SERVICES/TALK/DERMACHAT
# 255/255 □
```

```
git commit -m "GOV: healing sprint — restore LEARNING for JC survey"
# MINT:WORK +64 COIN (CardiChat LEARNING restored)
# MINT:WORK +64 COIN (DermaChat LEARNING restored)

# Day 3: Fleet status
magic scan
# 4/4 scopes at 255/255
# Fleet: FULL governance
# Ready for Joint Commission survey
```

Three days. Two scopes healed. 128 COIN minted. Full governance restored 57 days before the survey. When the Joint Commission surveyor asks "How do you ensure continuous quality management of your clinical AI systems?" — the answer is `magic scan`: 4/4 scopes at 255, continuous validation via pre-commit hooks, complete LEDGER audit trail, and current LEARNING documentation. The surveyor can independently verify every claim by running `magic validate` on the governance tree [14312].

45.24. 43.24 magic-heal as Governance Antibody

`magic-heal` is the immune system of the governance tree. Where `magic validate` detects disease (governance gaps), `magic-heal` produces the antibody (structural templates, proposals, settlements). The five stages mirror clinical treatment: SCAN (diagnosis), DIAGNOSE (differential), PROPOSE (treatment plan), SETTLE (intervention), VALIDATE (outcome measurement).

The tool does not invent governance – it generates structural templates that you populate with domain-specific content. PROPOSE produces a LEARNING.md with the correct epoch structure, a VOCAB.md with the correct table format, a COVERAGE.md with the correct compliance matrix. SETTLE is always human. Content is judgment; structure is automatable. `magic-heal` automates structure so you can focus on judgment [142812].

Chapter 46

Chapter 44: build

Pipeline: galaxy □ surfaces □ validate. DAG-parallel phases governed by PIPELINE.toml. This chapter documents the build pipeline that orchestrates `magic validate` (Chapter 42) and `magic heal` (Chapter 43) into a complete compilation workflow. For the theoretical basis, see Chapter 37 (Governance as Compiler). Developer tools and build documentation are at dev.hadleylab.org.

46.1. 44.1 The Build Pipeline

```
build
```

This runs the full pipeline — 13 phases, DAG-parallel where dependencies allow [14]:

```
Phase 00-toolchain        → .md contracts to .json runtime constants
Phase 01-galaxy           → GOV tree to galaxy.json (284 nodes, 340 edges)
Phase 01a-galaxy-enrich   → backfill wallet, sessions, learning into galaxy
Phase 02-services         → verify service completeness (attestation manifests)
Phase 03-surfaces         → GOV → CANON.json + index.md per fleet site
Phase 04-figures          → Figures tables → figure assets
Phase 05-shop             → SHOP.md → commerce projections
Phase 06-validate-content → INTEL/VOCAB quality gates
Phase 07-structure        → CNAME, .gitignore generation
Phase 08-learning         → LEDGER aggregation
Phase 08a-claude          → graph-native CLAUDE.md compiler (BFS from galaxy)
Phase 09-econ             → wallet/economics calculations
Phase 10-federation       → cross-realm wallet consolidation
Phase 11-validate         → final validation gates
```

Dependencies are declared in PIPELINE.toml. The `build-dag` orchestrator reads these declarations and runs Kahn's algorithm for topological layering, executing independent phases in parallel. No hardcoded phase ordering exists outside PIPELINE.toml.

46.2. 44.2 _generated Flag

Files produced by `build` are `_generated`. Never hand-edit them. If the output is wrong, fix the contract (CANON.md) or the compiler – not the output [2].

46.3. 44.3 Deploy Order

DESIGN theme pushed first, then fleet sites pushed after. GitHub Pages fetches `remote_theme` at build time [25].

46.4. 44.4 Stage-by-Stage Reference

Each stage has a specific input, output, and failure mode.

Stage 0: build-toolchain – Compiles `.md` contracts into `.json` runtime. CANON.md becomes CANON.json.


```
{
  "_generated": "build-toolchain",
  "_contract": "hadleylab-canonic/SERVICES/referral/CANON.md",
  "scope": "referral",
  "axiom": "referral is SERVICE",
  "inherits": "hadleylab-canonic/SERVICES",
  "dimensions": {
    "D": { "file": "CANON.md", "present": true },
    "E": { "file": "VAULT/", "present": true },
    "S": { "file": "referral.md", "present": true },
    "O": { "file": "COVERAGE.md", "present": true },
    "T": { "file": "ROADMAP.md", "present": true },
    "R": { "file": "HTTP.md", "present": true },
    "LANG": { "file": "VOCAB.md", "present": true },
    "L": { "file": "LEARNING.md", "present": true }
  },
  "score": 255
}
```


Every field traces to a governance file. Provenance is built into the output [2].

Stage 1: attest-services – Verifies every service scope has a complete SERVICE contract.

Check	Requirement	Failure
CANON.md exists	Required	`MISSING_CANON`
axiom contains “SERVICE”	Required	`INVALID_AXIOM`
HTTP.md exists	Required for services	`MISSING_HTTP`
COVERAGE.md exists	Required for services	`MISSING_COVERAGE`
All routes have contracts	Required	`UNCONTRACTED_ROUTE`

Stage 2: build-surfaces – Generates CANON.json and index.md for each fleet site.

Stage 3: figures – Compiles Figures tables into `_data/*.json` consumed by Jekyll.

Stage 4: jekyll-exclude – Generates `exclude:` in `_config.yml`:

```
exclude:
  - CANON.md
  - VOCAB.md
  - COVERAGE.md
  - ROADMAP.md
  - LEARNING.md
  - VAULT/
  - LEDGER/
```

Stage 5: build-shop-json – Compiles SHOP.md into SHOP.json.

Stage 6: _generated markers – Annotates compiled outputs with `_generated` metadata and contract provenance.

Stage 7: verify-intel-wiring – Verifies INTEL propagates to all TALK surfaces.

Stage 8: validate-vocab – Validates VOCAB.md inheritance and deduplication:

```
$ validate-vocab
  canonic-canonic/VOCAB.md           ✓ root (142 terms)
  hadleylab-canonic/VOCAB.md         ✓ extends root (+38 medical terms)
  hadleylab-canonic/VOCAB.md  ✗ DUPLICATE: "scope" defined identically to parent
```

Stage 9: validate-hygiene – Validates constraint deduplication cleanliness.

Phase 01-galaxy: build-galaxy-json – Generates `galaxy.json` (284 nodes, 340 edges).

Phase 01a-galaxy-enrich – Backfills wallet, sessions, and learning counts into galaxy nodes.

Phase 09-econ – Publishes wallets, verifies wallet chains.

Phase 10-federation – Cross-realm wallet consolidation.

Phase 11-validate – Final gate. All scopes must be 255 or the build fails.

46.5. 44.5 Build Determinism

Same GOV tree produces same output. Always. No ambient state. No random values.

```
build \&\& cp -r output/ output-1/
build \&\& diff -r output/ output-1/
# No differences.
```

46.6. 44.6 Build Failure Recovery

```
$ build
[STAGE 1] attest-services    ✗
ERROR: MISSING\_HTTP in new-imaging
FIX: Create HTTP.md. Re-run build.
```

Fix governance, re-run. No partial builds. No `--skip-validation` [14].

46.7. 44.7 Build Performance

Phase	Duration
00-toolchain	~5s
01-galaxy	~3s
01a-galaxy-enrich	~2s
02-services	~2s
03-surfaces	~8s
04-figures	~3s
05-shop	<1s
06-validate-content	~3s
07-structure	<1s
08-learning	~2s
08a-claude	~2s
09-econ	~5s
10-federation	~2s
11-validate	~5s
Total (sequential)	**~43s**
Total (DAG-parallel)	**~25s**

46.8. 44.8 Clinical Build Example

```
$ build
[Layer 0] 00-toolchain        ✓  (42 contracts compiled)
[Layer 1] 01-galaxy           ✓  (284 nodes, 340 edges)
          02-services         ✓  (12 services attested)
          07-structure        ✓  (CNAME, .gitignore generated)
          08-learning         ✓  (LEDGER aggregated)
[Layer 2] 01a-galaxy-enrich   ✓  (wallet + sessions backfilled)
          03-surfaces         ✓  (8 fleet sites generated)
[Layer 3] 04-figures          ✓  (23 tables compiled)
          05-shop             ✓  (commerce projections generated)
          06-validate-content ✓  (INTEL/VOCAB quality gates passed)
          08a-claude          ✓  (CLAUDE.md compiled via BFS)
[Layer 4] 09-econ             ✓  (wallet chains verified)
[Layer 5] 10-federation       ✓  (cross-realm wallets consolidated)
[Layer 6] 11-validate         ✓  (73/73 at 255)

BUILD OK — 13 phases in 25s
```

Every clinical service is compiled from governance. Every route table, coverage matrix, and vocabulary constraint is validated [714].

46.9. 44.9 Clinical Vignette: Build Pipeline Catches Silent Regression

UCLA Health deploys NeuroChat — a governed TALK agent for neurological consultation. The neurology informatics team updates NeuroChat's VOCAB.md to add NIHSS (National Institutes of Health Stroke Scale — 42-point assessment of stroke severity). A junior developer simultaneously edits COVERAGE.md to update the operations dimension.

The developer commits both changes. The build pipeline runs:

```
build
[STAGE 0]  build-toolchain    ✓
[STAGE 1]  attest-services    ✓
...
[STAGE 8]  validate-vocab     ✗
ERROR: VOCAB\_DUPLICATE at hadleylab-canonic/SERVICES/TALK/NEUROCHAT
  "NIHSS" defined in local VOCAB.md AND inherited from
  hadleylab-canonic/MAGIC/VOCAB.md (medical root vocabulary)
  Local definition: "42-point stroke severity assessment"
```

```
  Parent definition: "National Institutes of Health Stroke Scale —
    15-item neurological exam, score 0-42, higher = more severe"
  Resolution: Remove local definition (parent is more complete)
BUILD FAILED at stage 8.
```

The build caught a vocabulary duplication that `magic validate` alone would miss – the scope had all 8 dimensions present, score 255. Stage 8 (`validate-vocab`) performs deeper analysis: it walks the full vocabulary inheritance chain and detects duplicate definitions even when the local score is perfect.

The fix: remove the local NIHSS definition from NeuroChat's VOCAB.md and rely on the inherited definition from the medical root vocabulary, which is more complete (specifies 15 items, score range, directionality). Rebuild. All 13 phases pass. The fleet deploys with a single, authoritative NIHSS definition [214].

46.10. 44.10 Build Artifacts and the _generated Contract

Every file produced by `build` carries the `_generated` contract, which has three rules:

1. **Never hand-edit.** If the output is wrong, fix the source (.md) or the compiler (build stage).
2. **Provenance is embedded.** Every generated file contains `_generated: true` and `_contract: {source_path}` pointing to the governance file that produced it.
3. **Deterministic reproduction.** Delete all generated files, run `build`, get the same output. Generated files are a pure function of the governance tree.

```
# Verify _generated contract
grep -r "_generated" _data/
# MAGIC/galaxy.json: "_generated": "build-galaxy-json"
# _data/shop.json: "_generated": "build-shop-json"
# _data/figures/*.json: "_generated": "build-figures"
# _data/figures/*.json: "_generated": "figures"
```

For clinical AI governance, `_generated` is a compliance property. An auditor asks "How was this route table generated?" The answer is in the file: `_contract: SERVICES/TALK/NEUROCHAT/HTTP.md`. Read the contract, verify the routes, trace the compilation. Audit complete [2].

46.11. 44.11 Build Phase Dependencies (DAG)

The 13 phases form a directed acyclic graph declared in PIPELINE.toml. The `build-dag` orchestrator computes topological layers and runs each layer in parallel:

```
Layer 0:  00-toolchain
```

```
Layer 1:  01-galaxy, 02-services, 07-structure, 08-learning
Layer 2:  01a-galaxy-enrich, 03-surfaces
Layer 3:  04-figures, 05-shop, 06-validate-content, 08a-claude
Layer 4:  09-econ
Layer 5:  10-federation
Layer 6:  11-validate (depends on all previous)
```

Phases within a layer execute concurrently. The DAG is not hardcoded in the orchestrator; it is derived from the `depends` field in each `[tools.*]` section of PIPELINE.toml. Add a phase by adding a TOML section with its dependency declarations. The orchestrator discovers it automatically [14].

46.12. 44.12 Build Caching

Since build outputs are deterministic, unchanged governance files produce identical output. The build system supports incremental builds through content hashing:

```
build --incremental
# Checking content hashes...
# Stage 0: 2 contracts changed, 40 unchanged → recompile 2
# Stage 2: 1 surface changed → regenerate 1
# Stages 3-11: no changes → skip
# Stage 12-13: wallet events pending → run
# Stage 14: validate changed scopes only
# BUILD COMPLETE (incremental). 3 stages executed, 11 skipped.
# Time: 12s (vs 54s full build)
```

Incremental builds reduce CI pipeline time by ~80% for typical commits that touch 1-3 governance files. Full builds (`build --full`) ignore the cache and recompile everything — used for release builds and after cache invalidation [14].

46.13. 44.13 Governance Proof: Build Soundness

The build pipeline is sound: if `build` succeeds, the fleet sites correctly reflect the governance tree. The proof by phase:

1. Phase 00-toolchain compiles every CANON.md to CANON.json (complete coverage).
2. Phase 02-services attests every service has a complete contract (no gaps).
3. Phases 01-galaxy and 01a-galaxy-enrich generate fleet-wide metadata (galaxy graph).
4. Phases 03-surfaces through 05-shop generate all runtime artifacts from governance sources (deterministic).

5. Phase 06-validate-content validates vocabulary and INTEL quality (cross-cutting checks).
6. Phases 09-econ and 10-federation reconcile economics (COIN, WALLET, LEDGER).
7. Phase 11-validate validates all scopes to 255 (final gate).

If any phase fails, the build fails. No partial output. No "mostly compiled" state. The build is atomic: all 13 phases pass, or the build produces no output. This atomicity guarantees that every deployed fleet site was produced by a complete, validated build. Q.E.D. [214].

46.14. 44.14 Build Dry-Run Mode

The `build --dry-run` flag simulates the build without producing output. Use it to verify that the build will succeed before committing to the full pipeline:

```
build --dry-run

# DRY RUN — no files will be written
# [Layer 0] 00-toolchain        □  (42 contracts would compile)
# [Layer 1] 01-galaxy           □  (284 nodes, 340 edges)
#           02-services         □  (12 services would attest)
#           07-structure        □  (CNAME, .gitignore would generate)
#           08-learning         □  (LEDGER would aggregate)
# [Layer 2] 01a-galaxy-enrich   □  (wallet + sessions would backfill)
#           03-surfaces         □  (8 fleet sites would generate)
# [Layer 3] 04-figures          □  (23 tables would compile)
#           05-shop             □  (commerce projections would generate)
#           06-validate-content □  (INTEL/VOCAB would validate)
#           08a-claude          □  (CLAUDE.md would compile via BFS)
# [Layer 4] 09-econ             □  (wallet chains would verify)
# [Layer 5] 10-federation       □  (cross-realm wallets would consolidate)
# [Layer 6] 11-validate         □  (73/73 at 255)
\#
# DRY RUN COMPLETE. 13/13 phases would pass.
# No files written. No LEDGER events produced.
```

Dry-run mode is essential during development: verify the build before committing changes. The dry-run reads all governance files, validates all dependencies, and reports what would happen — without side effects. For hospital IT teams, dry-run enables change review before production builds [14].

46.15. 44.15 Build Hooks and Custom Stages

Organizations can inject custom stages into the build pipeline. Custom stages execute between standard stages and follow the same contract: read governance, emit artifacts, fail if invalid.

```
# custom_stage.py — Example: validate clinical trial references
import json
import sys

def validate_trial_references(gov_root):
    """Check that all NCT references in INTEL.md are valid format."""
    errors = []
    for intel_path in find_intel_files(gov_root):
        for ref in extract_nct_references(intel_path):
            if not re.match(r'NCT\d{8}', ref):
                errors.append(f"Invalid NCT format: {ref} in {intel_path}")
    if errors:
        for e in errors:
            print(f"ERROR: {e}", file=sys.stderr)
        return False
    return True
```

```
# Register custom stage
build --register-stage \
  --name validate-trials \
  --after 7 \
  --script custom_stage.py

# Build with custom stage
build
# [STAGE 0-7]  ... □
# [CUSTOM]     validate-trials    □   (47 NCT references, all valid)
# [STAGE 8-14] ... □
```

Custom stages extend the build without modifying the core pipeline. The custom stage is itself governed — registered via CANON.md, stored in the governance tree, and validated by `magic validate` [14].

46.16. 44.16 Build Output Manifest

Every build produces a manifest listing all generated artifacts with their provenance:

```
{
  "_generated": "build-manifest",
  "build_id": "build:2026-03-10-001",
  "commit": "abc1234def5678",
  "timestamp": "2026-03-10T15:00:00Z",
  "phases_passed": 13,
  "artifacts": [
    {
      "path": "MAGIC/galaxy.json",
      "source": "build-galaxy-json",
      "contract": "GOV tree (284 nodes, 340 edges)",
      "hash": "sha256:a1b2c3d4..."
    },
    {
      "path": "_data/shop.json",
      "source": "build-shop-json",
      "contract": "SHOP.md files (47 products)",
      "hash": "sha256:e5f6a7b8..."
    },
    {
      "path": "_data/galaxy.json",
      "source": "build-galaxy-json",
      "contract": "CANON.md hierarchy (3 orgs)",
      "hash": "sha256:c9d0e1f2..."
    }
  ],
  "fleet_sites": [
    {
      "site": "hadleylab.org",
      "commit": "abc1234",
      "score_at_build": 255,
      "pages_generated": 47
    }
  ]
}
```


The manifest enables build comparison: diff two manifests to see what changed between builds. If an artifact's hash changed, trace the change back to its source governance file. The manifest is the build's certificate of origin [214].

46.17. 44.17 Build and the _generated Verification Tool

After a build, verify that no `_generated` file was hand-edited:

```
build --verify-generated

# Verifying _generated integrity...
\#
# MAGIC/galaxy.json:
#   _generated: build-galaxy-json
#   Expected hash: sha256:a1b2c3d4...
#   Actual hash:   sha256:a1b2c3d4...
#   Status: MATCH □
\#
# _data/shop.json:
#   _generated: build-shop-json
#   Expected hash: sha256:e5f6a7b8...
#   Actual hash:   sha256:e5f6a7b8...
#   Status: MATCH □
\#
# SERVICES/*/CANON.json:
#   _generated: build-toolchain
#   Expected hash: sha256:c9d0e1f2...
#   Actual hash:   sha256:c9d0e1f2...
#   Status: MATCH □
\#
# 42 _generated files checked. 42/42 match. No hand-edits detected.
```

If a hand-edit is detected (hash mismatch), the verification fails:

```
# MAGIC/galaxy.json:
#   _generated: build-galaxy-json
#   Expected hash: sha256:a1b2c3d4...
#   Actual hash:   sha256:MODIFIED...
#   Status: MISMATCH □
#   Action: Run 'build' to regenerate from governance sources
```

The verification tool enforces the `_generated` contract: compiled outputs are produced by the build, not by human editors. If the output is wrong, fix the source. Do not fix the output [2].

46.18. 44.18 Build Environment Isolation

The build pipeline operates in an isolated environment to ensure determinism:

Property	Enforcement	Verification
No ambient state	Build reads only GOV tree and environment variables	`build --verify-env` lists all inputs
No network dependencies during compilation	Stages 0-11 read local files only	Network disconnection test: `build --offline` passes
No random values	All outputs are content-addressed hashes	`build && cp -r output/ a/ && build && diff -r output/ a/` produces no differences
No time-dependent output	Timestamps come from git commits, not wall clock	Same commit, different build time = same output

```
# Verify build determinism
build \&\& cp -r _data/ _data_1/
build \&\& diff -r _data/ _data_1/
# No differences. Build is deterministic.
```

Build determinism is essential for audit compliance. An auditor asks: "Can you reproduce the build from March 1?" Check out the March 1 commit, run `build`, and the output is bit-for-bit identical to what was deployed. Reproducibility is not a goal – it is a property guaranteed by the build architecture [214].

46.19. 44.19 Clinical Vignette: Build Pipeline Detects Cross-Service Inconsistency

Kaiser Permanente (Northern California, 21 medical centers) operates 6 clinical AI agents through a federated CANONIC deployment. During a routine build, Stage 7 (`verify-intel-wiring`) detects a cross-service inconsistency:

```
build
[STAGE 0-6] ... ✓
[STAGE 7]   verify-intel-wiring    ✗

ERROR: INTEL wiring inconsistency detected
  Source: SERVICES/INTEL/ONCOLOGY (INTEL.md layer 2, reference 14)
    Cites: "NCCN Breast Cancer v2026.1"
  Target: SERVICES/TALK/MAMMOCHAT (systemPrompt, line 47)
    References: "NCCN Breast Cancer v2025.2"
  Mismatch: INTEL source updated to 2026.1 but TALK consumer still references 2025.2

  Fix: Update MAMMOCHAT systemPrompt to reference v2026.1
       OR: Revert INTEL to v2025.2 until MAMMOCHAT is ready to update
```

```
BUILD FAILED at stage 7.
```

The INTEL wiring check caught that the oncology INTEL scope updated its NCCN reference (v2025.2 to v2026.1) while the clinical TALK agent still references the old version in its systemPrompt. Clinically, this means the agent could serve responses based on outdated guidelines while the organization's evidence base has already moved on.

The fix is a two-file edit: update the systemPrompt to reference v2026.1 and update the agent's INTEL.md to cite the updated guideline. Rebuild. All 13 phases pass.

Without Stage 7, the inconsistency would persist indefinitely. Both scopes are individually at 255, both individually valid. The inconsistency is cross-service, visible only to the build pipeline that validates wiring between services. Stage 7 exists precisely for this class of error [2143].

46.20. FRESHNESS: Build Caching

FRESHNESS is the incremental compilation subsystem. It tracks source file modification times per scope in a JSON cache:

```
{
  "version": 1,
  "toolchain_hash": "abc123...",
  "entries": {
    "hadleylab-canonic": {
      "sources": { "CANON.md": 1710000000, "VOCAB.md": 1710000001 },
      "outputs": ["_site/DEXTER/index.html"]
    }
  }
}
```

On subsequent builds, unchanged scopes skip compilation. Toolchain hash changes (updating build scripts) invalidate the entire cache, ensuring that toolchain upgrades always produce fresh output.

The result: 134s down to 3s. If you are iterating on a single scope, you pay only the cost of that scope's compilation, not the entire fleet.

46.21. 44.20 build as Governance Compiler

`build` is the compiler. It takes the governance tree (source) and produces the deployment surface (output). Same tree in, same surface out. The 13 phases are compilation passes: discovery, generation, template resolution, cross-service wiring, asset compilation, final validation. Each stage has clear preconditions and

postconditions. Each can fail independently. Each failure produces a diagnostic with the exact file, line, and fix required.

The build does not guess and does not approximate. It reads governance files and produces exactly what they declare. If governance files declare a TALK agent with FHIR evidence, the build produces a chat interface with FHIR citations. If the files are incomplete, the build fails with a precise diagnostic. Run `build`. If it passes, your governance compiles. If it fails, fix the gap. Run `build` again. The pipeline is the proof [1428].

Chapter 47

Chapter 45: Validation Errors and Healing

Diagnosis. Common issues. This chapter is the debugging reference for the governance compiler described in Chapter 37. The error codes map to the eight dimensions of Chapter 4, the inheritance chain of Chapter 3, and the vocabulary closure of Chapter 28. For automated healing, see Chapter 43 (`magic-heal`). For the build pipeline that enforces these gates, see Chapter 46.

47.1. 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

47.2. 45.2 Diagnostic Flow

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

47.3. 45.3 The Heal Loop

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

Do not loop manually. Use `magic-heal` for automated diagnosis [14].

47.4. 45.4 Error Code Reference

Every validation error has a unique code, severity level, and prescribed fix.

Structural Errors (100-series)

Code	Name	Severity	Cause	Fix
E100	`MISSING_CANON`	FATAL	No CANON.md in scope	Create CANON.md with axiom
E101	`MALFORMED_CANON`	FATAL	CANON.md cannot be parsed	Fix YAML front matter
E102	`MISSING_AXIOM`	FATAL	CANON.md has no axiom	Add `axiom:` field
E103	`MISSING_INHERITS`	ERROR	No inherits field (non-root)	Add `inherits:` field
E104	`BROKEN_INHERIT`	ERROR	`inherits:` points to nonexistent scope	Fix path
E105	`CYCLE_DETECTED`	FATAL	Circular inheritance	Remove cycle
E106	`DUPLICATE_SCOPE`	ERROR	Two CANON.md claim same name	Rename one
E107	`ORPHAN_SCOPE`	WARN	Scope not reachable from root	Add to parent tree

Governance Question Errors (200-series)

Code	Name	Severity	Fix
E200	`MISSING_SCOPE_MD`	ERROR	Create {scope-name}.md
E201	`MISSING_COVERAGE`	ERROR	Create COVERAGE.md
E202	`MISSING_ROADMAP`	ERROR	Create ROADMAP.md
E203	`MISSING_HTTP`	ERROR	Create HTTP.md
E204	`MISSING_VOCAB`	ERROR	Create VOCAB.md or verify inheritance
E205	`MISSING_LEARNING`	ERROR	Create LEARNING.md
E206	`MISSING_VAULT`	ERROR	Create VAULT/ with wallet
E207	`EMPTY_FILE`	WARN	Governance file exists but empty

Compilation Errors (300-series)

Code	Name	Severity	Cause	Fix
E300	`VOCAB_DUPLICATE`	ERROR	Same term in parent and child	Remove from child
E301	`VOCAB_CONFLICT`	ERROR	Child redefines parent term	Resolve conflict
E302	`CONSTRAINT_DUPLICATE`	WARN	Same constraint in multiple files	Deduplicate
E303	`ROUTE_UNCONTRACTED`	ERROR	HTTP route has no contract	Add contract
E304	`FIGURE_MALFORMED`	ERROR	Table cannot be parsed	Fix Markdown syntax
E305	`SHOP_INVALID`	ERROR	SHOP.md entry missing fields	Add price, wallet, description

Economic Errors (400-series)

Code	Name	Severity	Cause	Fix
E400	`WALLET_MISSING`	ERROR	No wallet in VAULT/	Run `vault onboard`

Code	Name	Severity	Cause	Fix
E401	`WALLET_UNLINKED`	ERROR	Wallet not in chain	Link to LEDGER
E402	`LEDGER_GAP`	WARN	Missing LEDGER entries	Reconcile chain
E403	`STRIPE_MISMATCH`	WARN	VAULT and Stripe out of sync	Run `stripe-sync`

Deployment Errors (500-series)

Code	Name	Severity	Cause	Fix
E500	`SCORE_BELOW_255`	FATAL	Score < 255 at deploy time	Heal to 255
E501	`DESIGN_MISMATCH`	ERROR	DESIGN.md token missing from CSS	Add CSS variable
E502	`FLEET_STALE`	WARN	Fleet site older than GOV tree	Re-deploy
E503	`PRIVATE_LEAK`	FATAL	PRIVATE scope in fleet	Remove from fleet config
E504	`FROZEN_DEPLOY`	WARN	Deploy to FROZEN interface	Skip (intentional)

47.5. 45.5 Error Severity Levels

Severity	Meaning	Pipeline Impact
FATAL	Cannot proceed	Build halts immediately
ERROR	Contract violation	Build halts at gate
WARN	Potential issue	Logged, does not block

FATAL and ERROR both prevent deployment. No clinical software ships with governance errors [7].

47.6. 45.6 Fix Patterns

Pattern 1: Missing File

```
# Error: E200 MISSING\_SCOPE\_MD
vim hadleylab-canonic/SERVICES/new-service/new-service.md
magic validate  \# Score increases (question answered)
```

Pattern 2: Broken Inheritance

```
# Error: E104 BROKEN\_INHERIT
# inherits: hadleylab-canonic/SEVICES (typo)
vim CANON.md  \# Fix: inherits: hadleylab-canonic/SERVICES
magic validate
```

Pattern 3: Vocabulary Conflict

```
# Error: E301 VOCAB\_CONFLICT
# "referral" defined differently in parent and child
vim VOCAB.md  \# Remove conflicting definition or extend parent
magic validate
```

Pattern 4: Regression Recovery

```
# Score dropped from 255 to 247 (file deleted)
# Option A: Restore
git checkout HEAD~1 -- COVERAGE.md
magic validate  \# Returns 255

# Option B: Accept drift
# DEBIT:DRIFT recorded in LEDGER
magic-heal .  \# Regenerate template
```

Pattern 5: Private Leak

```
# Error: E503 PRIVATE\_LEAK
magic scan --tree | grep PRIVATE
vim fleet-config.yml  \# Remove PRIVATE scope reference
build
```

47.7. 45.7 Diagnostic Procedure

```
# Step 1: Identify the failure
magic validate
```

```
# Step 2: Get detailed diagnosis
magic heal --verbose

# Step 3: Check inheritance chain
magic validate --verbose

# Step 4: Check for regressions
git log --oneline --diff-filter=D -- "*.md"

# Step 5: Heal
magic-heal .

# Step 6: Verify
magic validate  \# Must return 255
```

47.8. 45.8 Healing in CI

CI pipelines report healing debt but do not auto-heal. Auto-healing in CI would mean machines write governance, which violates the invariant [14].

```
- name: Healing debt report
  run: |
    magic scan --below 255 --json > healing-debt.json
    count=$(cat healing-debt.json | jq 'length')
    echo "Healing debt: $count scopes below 255"
    if [ "$count" -gt 0 ]; then
      echo "::warning::$count scopes require healing"
    fi
```

The warning appears in the CI log. The build does not auto-heal. The human heals. The machine reports [14].

47.9. 45.9 Common Error Combinations

Pattern	Errors	Root Cause	Single Fix
New scope	E100 + E200-E206	Scope just created	Run `magic-heal`
Typo in inherits	E104 + score 0	Path has a typo	Fix CANON.md
Deleted file	E201 (one dim)	`git rm` or accident	`git checkout` or re-create

Pattern	Errors	Root Cause	Single Fix
Vocab refactor	E300 + E301	Parent vocabulary changed	Update child VOCAB.md
New service	E201 + E203 + E207	Service added without full contract	Fill COVERAGE.md and HTTP.md

47.10. 45.10 Prevention

The best fix is prevention. Install pre-commit hooks, run `magic validate` before every commit, and block commits that reduce score.

```
install-hooks

$ git commit -m "delete COVERAGE.md for fun"
PRE-COMMIT: magic validate
SCORE: 247/255 (was 255/255 -- regression detected)
COMMIT BLOCKED.
```

The error never reaches CI, never reaches production. The patient never encounters software with governance errors [714].

47.11. 45.11 Error Resolution Priority

When multiple errors occur simultaneously, resolve them in this order:

Priority	Error Category	Rationale
1	FATAL structural (E100-E105)	Cannot score without valid identity
2	ERROR structural (E106-E107)	Cannot inherit without valid parents
3	ERROR question (E200-E206)	Cannot reach 255 without all eight questions
4	ERROR compilation (E300-E305)	Cannot build without clean compilation
5	WARN any	Advisory, does not block

Fix FATAL first. A scope with `MALFORMED_CANON` cannot be healed because it cannot be identified. Fix the parse error, then inheritance, then unanswered questions, then compilation issues. The order is not negotiable – each layer depends on the one below [14].

47.12. 45.12 Regression Detection

Score regression occurs when a scope's score decreases between commits. The pre-commit hook detects this by comparing the current score to the last recorded score.

```
# Pre-commit hook regression check
CURRENT=$(magic validate --quiet)
PREVIOUS=$(git show HEAD:CANON.json | jq '.score')
if [ "$CURRENT" -lt "$PREVIOUS" ]; then
    echo "REGRESSION: Score decreased from $PREVIOUS to $CURRENT"
    echo "DEBIT:DRIFT will be recorded."
    exit 1
fi
```

Common causes of regression:

Cause	Detection	Fix
File deleted	E200-E206	`git checkout` or re-create
Inherits chain broken	E104	Fix path in CANON.md
Parent scope modified	Score drops in children	Fix parent, children inherit fix
VOCAB.md conflict introduced	E301	Resolve conflicting definitions

47.13. 45.13 Healing Metrics

Track healing effectiveness over time:

```
# LEDGER query: healing events per month
$ magic ledger --type CREDIT:HEAL --since 2026-01-01 --json | jq 'group_by(.month) | map({month: .[0].mont
[
  { "month": "2026-01", "count": 12 },
  { "month": "2026-02", "count": 5 },
  { "month": "2026-03", "count": 2 }
]
```

Decreasing healing events indicate governance maturity. New scopes are created closer to 255 from the start, templates are customized for the domain, and developers internalize the eight-question standard. The organization converges on governance by default, not governance by correction [14].

47.14. 45.14 Clinical Error Example

A hospital's pharmacy service loses its COVERAGE.md in a merge conflict:

```
$ git merge feature/drug-interaction
Auto-merging COVERAGE.md
CONFLICT (modify/delete): COVERAGE.md deleted in feature/drug-interaction
```

The developer resolves the conflict by accepting the deletion. The pre-commit hook catches it:

```
$ git commit
PRE-COMMIT: magic validate
SCOPE: hadleylab-canonic/SERVICES/pharmacy
SCORE: 247/255 (missing 0 -- COVERAGE.md)
COMMIT BLOCKED.

Run: magic heal --verbose
  [8] 0  COVERAGE.md  ✗ -- Restore from git or re-create
```

The developer restores the file:

```
$ git checkout HEAD -- COVERAGE.md
$ git commit
PRE-COMMIT: magic validate -- 255/255 ✓
```

The pharmacy service's coverage contract is preserved, the drug interaction feature ships with governance intact, and the patient receives medication checked by software that knows its coverage requirements [7].

47.15. 45.15 Clinical Vignette: Cascading Validation Errors in a Multi-Site Deployment

Johns Hopkins deploys NeoChat — a governed clinical TALK agent for neonatal intensive care — across three NICUs: downtown Baltimore, Bayview, and Suburban. The scope tree:

```
SERVICES/TALK/NEOCHAT/
  CANON.md (inherits: TALK, 9 constraints)
  NEOCHAT-DOWNTOWN/
    CANON.md (inherits: NEOCHAT, adds JHH IRB constraints)
  NEOCHAT-BAYVIEW/
    CANON.md (inherits: NEOCHAT, adds Bayview site constraints)
  NEOCHAT-SUBURBAN/
    CANON.md (inherits: NEOCHAT, adds Suburban site constraints)
```

A senior developer updates the parent NEOCHAT scope's VOCAB.md to add APGAR (Appearance, Pulse, Grimace, Activity, Respiration — neonatal assessment score at 1 and 5 minutes post-delivery, range 0-10). The developer also renames the existing term GESTATIONAL_AGE to GA in the parent VOCAB.md, reasoning that the abbreviation is universally understood in neonatology.

The pre-commit hook runs `magic validate --recursive`:

```
magic validate --recursive SERVICES/TALK/NEOCHAT
# NEOCHAT: 255/255 (PASS)
# NEOCHAT-DOWNTOWN: 191/255 (FAIL)
#   E301 VOCAB\_CONFLICT: GESTATIONAL\_AGE used in CANON.md but
#   not defined in VOCAB.md (parent defines GA, not GESTATIONAL\_AGE)
# NEOCHAT-BAYVIEW: 191/255 (FAIL)
#   E301 VOCAB\_CONFLICT: GESTATIONAL\_AGE used in COVERAGE.md
#   but not defined
# NEOCHAT-SUBURBAN: 191/255 (FAIL)
#   E301 VOCAB\_CONFLICT: GESTATIONAL\_AGE used in LEARNING.md
#   but not defined
# Fleet: 1/4 scopes at 255. 3 scopes require healing.
# COMMIT BLOCKED.
```

The rename cascaded. Three child scopes broke because they used the old term GESTATIONAL_AGE, which no longer resolves in the vocabulary chain. The parent's VOCAB.md defines GA, but the children still reference GESTATIONAL_AGE.

The fix has two options:

Option A: Fix the children. Update all three child scopes to use GA instead of GESTATIONAL_AGE. Three files to edit. Risk: downstream clinical documentation that references GESTATIONAL_AGE in other systems (Epic, Cerner) will be inconsistent.

Option B: Fix the parent. Keep both terms — define GESTATIONAL_AGE as the primary term and GA as an alias. One file to edit. No downstream impact.

The developer chooses Option B. The parent VOCAB.md retains GESTATIONAL_AGE as the primary definition and adds GA as a cross-reference. All three children pass validation without modification. The commit proceeds.

This illustrates why recursive validation is essential. A vocabulary change at a parent scope can break every child in the tree. The validator catches every break before commit. No broken governance reaches production, and no NICU deploys software with undefined terms [14].

47.16. 45.16 Error Recovery Time Benchmarks

Track mean time to recovery (MTTR) for each error category:

Error Category	Typical MTTR	Expert MTTR	Automation
Missing file (E200-E206)	10 minutes	2 minutes	`magic-heal` generates template
Broken inherits (E104)	5 minutes	1 minute	`magic scan --scopes` lists valid paths
Vocab conflict (E300-E301)	20 minutes	5 minutes	Manual resolution required
Cascade failure (parent change)	45 minutes	15 minutes	`magic validate --recursive` identifies all
Regression recovery	15 minutes	3 minutes	`git checkout` + re-validate
Private leak (E503)	30 minutes	10 minutes	`magic scan --tree` identifies leak

For clinical deployments, MTTR matters because governance errors block deployment. A 45-minute cascade failure during a critical NeoChat update means 45 minutes of delayed deployment. The pre-commit hook catches this at development time, not deployment time – you fix the issue at your desk, not in the deployment pipeline under pressure [714].

47.17. 45.17 Healing Maturity Model

Organizations progress through four healing maturity stages:

Stage 1: Reactive healing. Errors discovered at deployment time. MTTR is hours. Governance work is emergency triage. Common in organizations adopting CANONIC for the first time.

Stage 2: CI healing. Pre-commit hooks installed. Errors caught at commit time. MTTR drops to minutes. Governance debt is tracked in CI reports. The team sees healing debt as a metric.

Stage 3: Preventive healing. Templates and conventions prevent most errors. New scopes bootstrap at 127+ because developers use `magic-heal --template`. Healing events decrease month-over-month. The organization's healing metric trends toward zero.

Stage 4: Autonomous healing. `magic-heal` generates structurally correct files for all unanswered questions. The developer reviews and commits. Healing is a single command. MTTR is under 2 minutes for any single-scope error.

```
# Measure healing maturity
magic metrics --healing --last 90d
# Stage: 3 (Preventive)
# Healing events: 7 (trending down from 23 in Q1)
```

```
# MTTR: avg 8 minutes
# Root causes: 4 vocab conflicts, 2 cascade failures, 1 file deletion
# Recommendation: Install templates for remaining un-templated scopes
```

47.18. 45.18 Governance Proof: Error System Completeness

The error code system is complete for the governance domain. The proof by construction:

1. **Structural errors (100-series)** cover all parse and identity failures. A scope that cannot be parsed cannot be validated.
2. **Question errors (200-series)** cover all eight governance questions. Each unanswered question has a specific error code.
3. **Compilation errors (300-series)** cover vocabulary, constraints, routes, and shop entries. Every governance artifact that the compiler processes has an error code.
4. **Economic errors (400-series)** cover wallet, ledger, and settlement. Every economic operation has an error code.
5. **Deployment errors (500-series)** cover score gates, design tokens, fleet configuration, and privacy. Every deployment constraint has an error code.

The error categories are exhaustive. Every governance failure maps to exactly one error code. Every error code maps to exactly one fix pattern. The fix is always a governance file edit (never a code change, never a database modification, never a manual override). The error system is the debugging interface to the governance compiler. It is sound (every real error is reported), complete (every possible error has a code), and decidable (every error has a deterministic fix). Q.E.D. [143].

47.19. 45.19 Error Aggregation for Fleet-Wide Diagnosis

When managing fleets of 50+ scopes, individual error inspection is impractical. Fleet-wide error aggregation provides organizational diagnosis:

```
magic validate --all --aggregate-errors

# Fleet Error Aggregation (73 scopes):
\#
# Error Distribution:
#   E200 MISSING\_SCOPE\_MD:    0 (0%)
#   E201 MISSING\_COVERAGE:    2 (2.7%)
#   E202 MISSING\_ROADMAP:     1 (1.4%)
#   E205 MISSING\_LEARNING:    3 (4.1%)
#   E300 VOCAB\_DUPLICATE:     1 (1.4%)
```

```
#   E302 CONSTRAINT\_DUPLICATE: 4 (5.5%)
#   E402 LEDGER\_GAP:           1 (1.4%)
\#
# Systemic Patterns:
#   LEARNING question: 3 scopes affected (organizational blind spot)
#   Constraint hygiene: 4 scopes with duplicates (template issue)
\#
# Scopes at 255: 66/73 (90.4%)
# Scopes below 255: 7/73 (9.6%)
# Fleet healing debt: 7 scopes × avg 1.7 errors = 12 fixes needed
```

The aggregation reveals systemic patterns. Three scopes missing LEARNING suggests the organization does not prioritize institutional memory – a training gap, not a technical gap. Four constraint duplicates suggest the scope template produces duplicates when customized – a tooling fix, not a per-scope fix. Fleet diagnosis drives organizational improvement, not just scope-level healing [14].

47.20. 45.20 Error History and Trend Analysis

The LEDGER records every validation error and every healing event. Over time, this data reveals governance trend lines:

```
magic errors --trend --period 12m

# Error Trend Analysis (12 months):
\#
# Month    | Errors | Healed | Net Debt | Scopes at 255
# 2025-04  | 47     | 42     | +5       | 28/42 (66.7%)
# 2025-06  | 23     | 25     | -2       | 45/52 (86.5%)
# 2025-08  | 14     | 15     | -1       | 58/60 (96.7%)
# 2025-10  | 8      | 8      | 0        | 65/67 (97.0%)
# 2025-12  | 11     | 10     | +1       | 68/71 (95.8%)
# 2026-02  | 5      | 6      | -1       | 72/73 (98.6%)
\#
# Trend: Errors decreasing 68% year-over-year
# Interpretation: Governance maturity improving
# Alert: Q4 spike (11 errors) correlates with 4 new service additions
```

The trend confirms governance maturity: error rates decrease as the organization internalizes the eight-question standard. Spikes correlate with new scope creation – expected and healthy. A sustained increase would indicate governance regression requiring organizational intervention [1420].

Chapter 48

Chapter 46: The Build Pipeline

Full pipeline. CI/CD. `_generated`. This chapter details the end-to-end pipeline that compiles governance into the live fleet — hadleylab.org, mammo.chat, oncochat.hadleylab.org, medchat.hadleylab.org, shop.hadleylab.org, and gorunner.pro. The pipeline enforces the `_generated` contract (Chapter 26), runs `magic validate` (Chapter 42) at every gate, and connects to the deploy service (Chapter 23). For the toolchain overview, see Chapter 41; for advanced tools in the pipeline, see Chapter 47.

48.1. 46.1 CI/CD Integration

Two workflows, strict ordering [14]:

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 (vanity-domain 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-dag — 13 DAG-parallel phases)
    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)
```

48.2. 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 [14].

48.3. 46.3 The Full Pipeline

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

48.4. 46.4 The magic-validate.yml Workflow

```yaml
name: MAGIC Validate
on:
  workflow_call:
    outputs:
      score:
        value: ${{ jobs.validate.outputs.score }}
jobs:
  validate:
```

```
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with: { submodules: recursive, token: "${{ secrets.GOV\_TOKEN }}" }
      - run: ln -s $GITHUB\_WORKSPACE $HOME/CANONIC
      - run: |
          git clone https://github.com/canonic-canonic/canonic-canonic ~/.canonic
          cd ~/.canonic \&\& cc -O2 -o bin/magic magic.c
          echo "$HOME/.canonic/bin" >> $GITHUB\_PATH
      - id: validate
        run: |
          below=$(magic scan --below 255 --quiet | wc -l)
          if [ "$below" -gt 0 ]; then magic scan --below 255; exit 1; fi
          echo "score=255" >> $GITHUB\_OUTPUT
```

Any repository calls it. Validation centralized; enforcement distributed [14].

48.5. 46.5 Build Workflow Steps

Step 1: Checkout GOV – `GOV_TOKEN` reads all governance submodules recursively.

Step 2: Bridge GOV path – `ln -s $GITHUB_WORKSPACE $HOME/CANONIC`. Same paths on CI and local.

Step 3: Install toolchain – Clone from canonical source. No package manager.

Step 4: Checkout fleet sites – Clone vanity-domain repos to `$HOME/fleet/`.

Step 5: Checkout DESIGN theme – Deployed first so fleet sites fetch `remote_theme`.

Step 6: Install deps – OpenSSL, Python3, cryptography, pyyaml, stripe.

Step 7: Compile kernel – `cc -O2 -o bin/magic magic.c && bin/magic validate || exit 1`

Step 8-9: Prepare runtime, MAGIC Build – Full 13-phase DAG-parallel pipeline via PIPELINE.toml.

Step 10: Compiler tests – `test-compiler` validates compiled outputs match schemas.

Step 11: Freeze check – FROZEN interfaces skip deployment.

Step 12: PRIVATE leak gate – Scans fleet for PRIVATE markers. E503 on detection.

Step 13: Deploy fleet – DESIGN first, then sites. Always [25].

Step 14: Build evidence – Audit trail of what was built and deployed.

48.6. 46.6 Pre-Commit Hook Details

```
\#!/bin/bash
# Installed by: install-hooks
export PATH="$HOME/.canonic/bin:$PATH"
if ! magic validate --quiet 2>/dev/null; then
    echo "COMMIT BLOCKED -- Governance incomplete"
    magic validate
    echo "Run: magic heal --verbose"
    exit 1
fi
echo "PRE-COMMIT: magic validate -- 255/255"
```

Hook	Trigger	Action
`pre-commit`	Before commit	`magic validate` – blocks if < 255
`commit-msg`	After message	Validates commit message format

48.7. 46.7 Score Gates

Gate	Location	Threshold	On Failure
Pre-commit	Developer machine	255	Commit blocked
CI validate	GitHub Actions	255	Build blocked
Build final	End of pipeline	255	Deploy blocked
Post-deploy	After fleet push	255	Rollback triggered

Ordered from fastest feedback to highest authority. Each gate is redundant – if pre-commit is bypassed, CI catches it. Defense in depth [14].

48.8. 46.8 Pipeline Monitoring

```
magic scan                                  \# Current state
magic scan --below 255                      \# Healing debt
magic ledger --last 10                      \# Recent events
gh run list --workflow=magic-build.yml -L 10   \# Build history
```

48.9. 46.9 Pipeline Recovery

```
gh run view --log-failed       \# Identify failure
magic validate                 \# Diagnose locally
magic heal --verbose           \# Get fix instructions
vim [missing-file].md          \# Fix governance
magic validate                 \# Confirm 255
git commit -m "GOV: heal"     \# Commit fix
gh run watch                   \# Verify CI
```

48.10. 46.10 Pipeline Security

Secret	Stored In	Access
`GOV_TOKEN`	GitHub Secrets	Read GOV submodules
`VAULT_KEY`	GitHub Secrets	VAULT decryption
`STRIPE_KEY`	GitHub Secrets	Payment reconciliation

Secrets never appear in logs or fleet sites. The PRIVATE leak gate verifies this [14].

48.11. 46.11 Clinical CI/CD Example

```
Developer authors COVERAGE.md for lab-orders
  -> git commit (pre-commit: 255 ✓)
  -> git push
  -> magic-validate.yml -> 255 ✓
  -> magic-build.yml
    -> attest-services (lab-orders ✓)
    -> validate-vocab (medical terms ✓)
    -> PRIVATE leak gate (clean ✓)
    -> magic validate (284/284 at 255 ✓)
  -> Deploy hadleylab.org
  -> LEDGER: CREDIT:DEPLOY lab-orders 255 FULL
```

Four gates passed. Vocabulary validated. Coverage attested. No leaks. Audit trail complete [714].

48.12. 46.12 Clinical Vignette: Pipeline Prevents PHI Leak

Mass General Brigham (MGB) deploys a fleet of governed clinical AI services. During a routine build, the PRIVATE leak gate (Stage 12) detects a critical issue:

```
build
[STAGE 12] PRIVATE leak gate    ✗
ERROR: E503 PRIVATE\_LEAK
  File: SERVICES/IDENTITY/test-fixtures/sample-patient.json
  Content: Contains PHI-like data (name, DOB, MRN pattern)
  Fleet site: hadleylab.org (PUBLIC)
  Action: Remove from fleet config or add to .gitignore
BUILD FAILED at stage 12.
```

A developer included sample patient data in a test fixture directory. The test data contains realistic but synthetic patient records with names, dates of birth, and medical record numbers matching MRN format patterns. The PRIVATE leak gate scans all files destined for fleet deployment and flags any file matching PHI patterns (name + DOB, MRN format, SSN format, email + diagnosis).

The fix:

```
# Add test fixtures to .gitignore for fleet sites
echo "test-fixtures/" >> .gitignore
# Move sample data to PRIVATE scope
mv SERVICES/IDENTITY/test-fixtures/ PRIVATE/test-fixtures/
git add .gitignore PRIVATE/
git commit -m "GOV: move test fixtures to PRIVATE — E503 fix"
build
# [STAGE 12] PRIVATE leak gate    □
# BUILD OK — 13 phases passed.
```

Without the pipeline, synthetic patient data would have been deployed to a public fleet site. While synthetic, those data patterns could trigger false positive PHI detection in downstream security scanners, cause compliance investigation overhead, and erode trust in the governance system. The pipeline prevented the leak at build time – before deployment, before public exposure, before incident [14].

48.13. 46.13 Multi-Repository Pipeline Architecture

Large health systems operate multiple repositories, each with their own build pipeline. The pipelines coordinate through git submodules:

```
canonic-canonic (root)          → magic-build.yml
```

```
└── hadleylab-canonic (org)    → magic-build.yml
      └── canonic-canonic (submodule, pinned at specific commit)
      └── adventhealth-canonic (submodule, for cross-org federation)
```

When the root (`canonic-canonic`) updates, child organizations must bump their submodule reference. The bump triggers a cascade of validation:

```
# In hadleylab-canonic:
git submodule update --remote canonic-canonic
magic validate --recursive
# All 73 scopes must pass with the new root constraints
git commit -m "GOV: bump canonic-canonic — updated root constraints"
# CI runs magic-build.yml → validates entire fleet against new root
```

The submodule bump is a governance event. If the new root constraints break any child scope, the CI pipeline fails – the child organization must fix their governance before adopting the root update. This cascade is intentional, ensuring that root constraint changes propagate and validate across the entire fleet [14].

48.14. 46.14 Pipeline Metrics and SLAs

Track pipeline health metrics:

Metric	Target	Measurement
Build success rate	> 95%	Successful builds / total builds
Build time (full)	< 90 seconds	End-to-end pipeline duration
Build time (incremental)	< 30 seconds	Cached pipeline duration
MTTR (pipeline failure)	< 15 minutes	Time from failure to fix
Gate bypass rate	0%	Commits that bypassed pre-commit
Deployment frequency	> 1/day	Fleet pushes per day

```
# Query pipeline metrics from GitHub Actions
gh run list --workflow=magic-build.yml -L 100 --json status,startedAt,updatedAt \
  | jq '{
    total: length,
    success: [.[] | select(.status=="completed")] | length,
    rate: ([.[] | select(.status=="completed")] | length) / length * 100
  }'
# {"total": 100, "success": 97, "rate": 97}
```

A build success rate below 95% indicates systemic governance issues – developers are committing changes that break validation. The fix is training (learn the 8-dimension standard) and tooling (templates reduce manual errors) [14].

48.15. 46.15 Pipeline Rollback Procedure

When a deployed fleet site has issues, the rollback procedure reverts to the previous known-good state:

```
# Step 1: Identify the issue
gh run view --log-failed

# Step 2: Rollback fleet site
rollback hadleylab.org
# Reverting to previous commit (force-with-lease)...
# Previous commit: abc1234 (2026-03-09, 255/255)
# Current commit: def5678 (2026-03-10, 255/255)
# Rollback initiated.
# LEDGER: DEBIT:ROLLBACK hadleylab.org

# Step 3: Verify
curl -s https://hadleylab.org/api/v1/health | jq .status
# "healthy"

# Step 4: Fix forward
# Fix the governance issue locally
magic validate
# Commit the fix
git commit -m "GOV: fix fleet issue identified in rollback"
# Re-deploy
build \&\& deploy
```

Rollback uses `force-with-lease`, rejecting the rollback if someone else has pushed since the rollback target. This prevents race conditions during incident response. The rollback event is recorded on the LEDGER as DEBIT:ROLLBACK [14].

48.16. 46.16 Pipeline Evolution: From Manual to Continuous

Organizations adopt the build pipeline in stages:

Stage 1: Manual builds. Developer runs `build` locally. No CI. No pre-commit hooks. Common in early adoption.

Stage 2: Pre-commit hooks. `install-hooks` enforces `magic validate` before every commit. Score regressions are blocked locally. The most impactful single improvement.

Stage 3: CI validation. `magic-validate.yml` runs on every PR. Even if pre-commit is bypassed, CI catches regressions. Defense in depth.

Stage 4: Full CI/CD. `magic-build.yml` runs the complete 13-phase DAG-parallel pipeline on every push to main. Fleet sites auto-deploy on success. Governance is continuous, automated, and architectural.

Stage 5: Multi-repo federation. Submodule bumps trigger cross-repository validation. Fleet-wide governance is enforced across organizational boundaries.

Most hospitals reach Stage 4 within 3 months of CANONIC adoption. Stage 5 requires cross-organizational coordination and typically takes 6-12 months [14].

48.17. 46.17 Governance Proof: Pipeline Completeness

The build pipeline is complete — it validates every governance property that the 255-bit score represents. The proof:

1. Pre-commit hook validates all 8 dimensions (structural completeness).
2. CI validation re-validates with full submodule resolution (cross-repo correctness).
3. Build phase 02-services attests service contracts (service completeness).
4. Build phase 06-validate-content validates vocabulary, INTEL wiring, and hygiene (semantic correctness).
5. The PRIVATE leak gate verifies no PRIVATE scopes reach fleet sites (security completeness).
6. Phase 11-validate performs final 255 validation (closure).
7. Post-deploy verification confirms fleet sites serve correct content (deployment correctness).

Every property that can cause a governance failure is checked at least once in the pipeline. Most properties are checked at multiple gates (defense in depth). The pipeline is the proof that governance is continuous — not a one-time audit, but a per-commit verification. Q.E.D. [143].

48.18. 46.18 Pipeline Observability

The build pipeline is itself monitored. Every pipeline run produces structured output that feeds into MONITORING:

```
# Pipeline run metadata (written to _data/pipeline-runs.json)
{
  "_generated": "magic-build",
  "run_id": "run:2026-03-10-001",
  "trigger": "push:main",
  "commit": "abc1234def5678",
  "author": "dexter",
  "stages": [
    {"name": "validate", "status": "pass", "duration": 6.1, "scopes": 73},
```

```
    {"name": "build", "status": "pass", "duration": 54.2, "artifacts": 142},
    {"name": "deploy", "status": "pass", "duration": 32.4, "sites": 8}
  ],
  "total_duration": 92.7,
  "result": "success",
  "ledger_event": "evt:BUILD:COMPLETE:04930"
}
```

Pipeline run metadata feeds the MONITORING dashboard. The governance officer sees build success rates, average duration, failure patterns, and deployment frequency – all derived from structured pipeline output, not from GitHub Actions UI scraping.

```
# Query pipeline health
magic pipeline --health --period 30d

# Pipeline Health — 30 Days
\#
# Runs: 47
# Success: 44 (93.6%)
# Failures: 3 (6.4%)
#   Stage 1 (attest-services): 1 failure (new service, incomplete contract)
#   Stage 8 (validate-vocab): 1 failure (vocabulary conflict)
#   Stage 14 (magic validate): 1 failure (drift during build)
\#
# Average duration: 89.4s (target < 120s □ )
# Slowest run: 142.3s (full rebuild after submodule bump)
# Fastest run: 12.1s (incremental, 1 file changed)
\#
# Deploy frequency: 1.6/day (target > 1/day □ )
# Rollbacks: 0 (target < 5% □ )
```

48.19. 46.19 DAG-Parallel Execution

The `build-dag` orchestrator reads PIPELINE.toml and runs Kahn's algorithm for topological layering. Independent phases execute concurrently within each layer:

```
Layer 0:  00-toolchain
Layer 1:  01-galaxy, 02-services, 07-structure, 08-learning
Layer 2:  01a-galaxy-enrich, 03-surfaces
Layer 3:  04-figures, 05-shop, 06-validate-content, 08a-claude
Layer 4:  09-econ
Layer 5:  10-federation
```

```
Layer 6:  11-validate (depends on all previous)
```

```
build-dag

# DAG execution:
# [0.0s]  Layer 0: 00-toolchain □  (5.0s)
# [5.0s]  Layer 1: 01-galaxy □  02-services □  07-structure □  08-learning □  (3.0s)
# [8.0s]  Layer 2: 01a-galaxy-enrich □  03-surfaces □  (8.0s)
# [16.0s] Layer 3: 04-figures □  05-shop □  06-validate-content □  08a-claude □  (3.0s)
# [19.0s] Layer 4: 09-econ □  (5.0s)
# [24.0s] Layer 5: 10-federation □  (2.0s)
# [26.0s] Layer 6: 11-validate □  (5.0s)
\#
# BUILD OK — 13 phases in 31s
```

Dependencies are declared in PIPELINE.toml, not hardcoded. Add a phase by adding a TOML section with `depends` declarations. The orchestrator discovers it automatically and validates drift between the manifest and on-disk phase scripts [14].

48.20. 46.20 Pipeline Disaster Recovery

In the event of complete pipeline infrastructure failure (GitHub Actions outage, CI runner unavailable), the pipeline runs locally:

```
# Local disaster recovery build
cd ~/CANONIC
git pull --recurse-submodules
magic validate --all   \# Validate locally
build                  \# Build locally
# Manual deploy: push fleet sites individually
for site in mammochat.com oncochat.ai medchat.ai; do
  cd ~/fleet/$site
  git push origin main
done
```

The pipeline has no cloud dependency for validation or compilation. `magic validate` runs locally, `build` runs locally – only deployment requires network access (git push). In a disaster recovery scenario, validate and build on a laptop, then push to fleet sites when connectivity is restored.

For hospital IT disaster recovery plans, this is critical. A CI/CD platform outage does not prevent governance validation or clinical AI deployment. The local-first architecture means governance continues even when infrastructure fails [1425].

48.21. 46.21 Pipeline Compliance Evidence Generation

Every pipeline run generates compliance evidence that maps to specific regulatory requirements:

Regulatory Requirement	Pipeline Evidence	Location
HIPAA §164.312(b) — Audit controls	LEDGER events for every build/deploy	`magic ledger --type BUILD,DEPLOY`
HIPAA §164.308(a)(8) — Evaluation	Periodic governance score validation	`magic scan --history`
21 CFR 820.30(e) — Design verification	Build test results (test-compiler)	`_data/pipeline-runs.json`
SOC 2 CC8.1 — Change management	Pre-commit hooks + CI validation	`magic metrics --hooks --ci`
SOX §404 — Internal controls	Monthly CLOSE reconciliation	`vault close --month`
GDPR Article 25 — Data protection by design	PRIVATE leak gate (Stage 12)	`build` output, Stage 12 logs

```
# Generate compliance evidence package for all frameworks
magic pipeline --compliance-package \
  --frameworks hipaa,fda,soc2,sox,gdpr \
  --period 2025-03-10:2026-03-10 \
  --output compliance-evidence-2026.json

# Compliance Evidence Package:
#   Frameworks covered: 5
#   Pipeline runs analyzed: 547
#   Total evidence items: 2,340
#   Build success rate: 96.2% (527/547)
#   Validation coverage: 100% (all runs include magic validate)
#   PRIVATE leak checks: 547 (100% of runs)
#   Monthly CLOSE events: 12 (all months covered)
#   Package size: 4.8 MB
#   Integrity hash: sha256:a1b2c3d4...
```

The compliance package is deterministic and reproducible. An auditor can regenerate the same package from the same LEDGER data and verify the integrity hash. The evidence is not hand-assembled from disparate sources – it is compiled from the governance pipeline's structured output [14312].

48.22. The 13-Phase Pipeline (DAG-Governed)

The build pipeline runs 13 phases, declared in PIPELINE.toml and orchestrated by `build-dag`:

Phase	Script	Purpose
00-toolchain	build-toolchain	Bootstrap TOOLCHAIN constants
01-galaxy	build-galaxy-json	GOV tree to galaxy.json (284 nodes, 340 edges)
01a-galaxy-enrich	enrich-galaxy	Backfill wallet, sessions, learning into galaxy
02-services	build-services	Service attestation manifests
03-surfaces	build-surfaces	GOV CANON.json + index.md per fleet site
04-figures	build-figures	Visualization assets
05-shop	build-shop	Commerce projections
06-validate-content	validate-content	INTEL/VOCAB quality gates
07-structure	build-structure	CNAME, .gitignore generation
08-learning	build-learning	LEDGER aggregation
08a-claude	compile-claude-md	Graph-native CLAUDE.md compiler (BFS from galaxy)
09-econ	build-econ	Wallet/economics calculations
10-federation	build-federation	Cross-realm wallet consolidation
11-validate	validate-final	Final 255 gate — hard fail on any score < 255

Phase 08a-claude is the graph-native CLAUDE compiler: BFS from any starting node in the galaxy, budget-controlled extraction by graph distance (d=0 all, d=1 summary, d=2 meta-patterns, d=3 axioms). Phase 01a (enrich-galaxy) backfills economic and operational data after the topology is established. Phase 11 (validate-final) is the closure gate — no scope below 255 passes.

The phase manifest is PIPELINE.toml. Dependencies are explicit `depends` declarations. The `build-dag` orchestrator validates drift between the manifest and on-disk scripts.

48.23. 46.22 Clinical Vignette: Pipeline Enables Regulatory Fast-Track

NYU Langone Health applies for FDA Breakthrough Device designation for their governed clinical AI agent — OncoChat-NYU, which provides personalized NCCN guideline navigation for oncology fellows. The FDA reviewer requests evidence of continuous quality management.

Traditional evidence: a 300-page Quality Management System (QMS) document with quarterly review meeting minutes, manual change control logs, and annual audit reports.

CANONIC evidence:

```
magic pipeline --fda-evidence \
  --scope SERVICES/TALK/ONCOCHAT-NYU \
  --period 12m

# FDA Evidence Package — OncoChat-NYU
\#
# Continuous validation:
#   Pipeline runs: 127 (12 months)
#   Score at 255: 127/127 (100%)
#   Validation frequency: every commit (avg 2.4/day)
\#
# Change management:
#   Total commits: 289
#   Pre-commit hook enforced: 289/289 (100%)
#   CI validation: 289/289 (100%)
#   No commit bypassed governance
\#
# Vocabulary precision:
#   VOCAB.md terms: 142
#   All terms formally defined (no ambiguity)
#   3 vocabulary updates in 12 months (all validated)
\#
# Evidence currency:
#   INTEL.md citations: 47
#   Mean citation age: 8.2 months (target < 12 months)
#   0 stale citations (all within freshness threshold)
\#
# Deployment safety:
#   Deployments: 47
#   Rollbacks: 1 (resolved in 3 minutes)
#   Deploy blocks (score < 255): 3 (all healed before re-deploy)
#   Blue-green deployments: 4
\#
# Institutional learning:
#   LEARNING.md entries: 31
#   Corrections: 7 (all resolved)
#   Patterns: 18 (reused by 3 other scopes)
```

The FDA reviewer receives structured, machine-verifiable evidence instead of a PDF. Every claim is backed by LEDGER events. The reviewer can independently verify any claim by running `magic validate` against the governance tree at any historical commit. Review accelerates from weeks to days. The governance pipeline IS the quality management system – not a document that describes one [14312].

Chapter 49

Chapter 47: Advanced Tools

validate-design. enforce-magic-ip. magic-tag. vault. These four tools extend the core toolchain cataloged in Chapter 41 and integrate into the build pipeline of Chapter 46. `validate-design` enforces the design token contract from Chapter 29. `magic-tag` produces the certification events recorded in the LEDGER (Chapter 13) and TAGS.md (Chapter 38). `vault` manages the economic operations that underpin COIN (Chapter 32), WALLET (Chapter 14), and LEDGER (Chapter 13). `enforce-magic-ip` protects the kernel internals described in Chapter 42.

49.1. 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 [14].

```
validate-design
```

49.2. 47.2 magic-tag

Certifies a scope at 255 with a git tag [19]:

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

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

49.3. 47.3 vault

Manages VAULT operations: COIN events, USER economic identity, key rotation, onboarding pipeline [14].

```
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
```

49.4. 47.4 enforce-magic-ip

Scans governance prose for kernel internals (bit weights, hex values, tier boundary scores). Any leak is a PRIVATE violation [2].

```
enforce-magic-ip
```

49.5. 47.5 Production Hardening

Runtime services are hardened with seven layers. Every layer traces to a governance constraint [15].

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.

49.6. 47.6 validate-design Deep Dive

`validate-design` enforces a 1:1 correspondence between DESIGN.md and the theme's CSS variables. It reads DESIGN.md's 255 Map – a table of token names, values, and semantic roles – and cross-references every entry against the actual SCSS files in `_sass/canonic/`.

```
validate-design --verbose

# Checking DESIGN.md tokens against _sass/canonic/_TOKENS.scss...
# Token: --canonic-accent-mammochat → \#E91E63 □
# Token: --canonic-accent-oncochat  → \#9C27B0 □
# Token: --canonic-accent-medchat   → \#2196F3 □
# Token: --canonic-font-primary     → 'Helvetica Neue' □
# Token: --canonic-spacing-base     → 8px □
# Token: --canonic-z-modal          → 1000 □
\#
# DESIGN.md tokens: 47
# SCSS variables: 47
# Mismatches: 0
# Orphaned SCSS vars: 0
# DESIGN VALID □
```

When a mismatch occurs – a token in DESIGN.md but not in SCSS, or vice versa – the tool reports the exact file, line, and expected value. If an accent color is declared as `#E91E63` in DESIGN.md but the SCSS uses `#E91E64`, the visual brand is ungoverned. A one-digit hex difference is invisible to a human reviewer but detectable by `validate-design`.

The tool also validates z-index hierarchy (modals above overlays above content above base), spacing scale consistency (multiples of the base unit), and font stack completeness (primary + fallback + system). Every design decision on the governed frontend traces to a DESIGN.md declaration [1428].

49.7. 47.7 magic-tag: Certification and Release Management

`magic-tag` is the certification ceremony. It takes a scope at 255 and stamps it with a signed git tag – cryptographic proof that at this commit, this scope was at full governance.

```
magic-tag v1.0.0 --scope SERVICES/TALK/MAMMOCHAT

# Pre-checks:
#   magic validate SERVICES/TALK/MAMMOCHAT → 255/255 □
#   VITAE.md exists → □  (signed by dexter@canonic.org)
#   TAGS.md exists → □  (will append v1.0.0)
#   Git working tree clean → □
\#
# Creating signed tag: MAMMOCHAT-v1.0.0
#   Signer: dexter@canonic.org (Ed25519)
#   Scope: SERVICES/TALK/MAMMOCHAT
#   Score: 255/255
#   Commit: a3f7c2d
#   Timestamp: 2026-03-10T14:30:00Z
\#
# Tag created: MAMMOCHAT-v1.0.0
# TAGS.md updated
# LEDGER: CERTIFY event recorded
```

The tag format encodes scope name and version: `{SCOPE}-v{MAJOR}.{MINOR}.{PATCH}`. Run `git tag --list 'MAMMOCHAT-*'` to see the certification history of any scope. The signed tag is non-repudiable – the signer's Ed25519 key is recorded in VITAE.md and verifiable with `git tag -v`.

The LEDGER records a `CERTIFY` event with tag name, commit hash, score, signer, and timestamp, creating a dual audit trail: the git tag proves certification at the repository level, the LEDGER event at the governance level. Both are independently verifiable.

Semantic versioning follows governance semantics: MAJOR for breaking constraint changes (new MUST rules), MINOR for new capabilities (new dimensions scored), PATCH for content updates within existing constraints. A MAJOR bump requires re-certification under the new constraints [1914].

49.8. 47.8 vault: The Economic Operations Console

`vault` is the economic control plane: onboarding, balances, reconciliation, key management, settlement. Every `vault` command produces a LEDGER event, making economic operations as auditable as governance operations.

Onboarding pipeline. `vault onboard {user}` executes a four-step pipeline:

```
vault onboard marina@hadleylab.org

# Step 1: Create USER scope
#   mkdir USERS/MARINA-SIROTA/
#   Generate CANON.md with identity axiom
\#
# Step 2: Generate Ed25519 key pair
#   Private key: VAULT/{user}/KEY (600 permissions)
#   Public key: VAULT/{user}/KEY.pub
#   Key ID: ed25519:marina-sirota-2026-03-10
\#
# Step 3: Initialize WALLET
#   Create WALLET.md with zero balance
#   LEDGER: ONBOARD event (identity, key_id, timestamp)
\#
# Step 4: Verify
#   magic validate USERS/MARINA-SIROTA/ → score pending first governance work
#   vault balance marina@hadleylab.org → 0 COIN
\#
# Onboarding complete. User can now:
#   - Commit governance work → MINT:WORK events
#   - Receive TRANSFER events from other users
#   - Purchase from SHOP → SPEND events
```

Monthly reconciliation. `vault close --month 2026-02` reconciles all COIN events for the month, computes net balances, and emits a CLOSE event. The reconciliation is deterministic: given the same LEDGER events, `vault close` produces the same balance sheet. A regulator can re-run `vault close` on any historical month and verify the balances match.

```
vault close --month 2026-02

# Processing 847 LEDGER events for 2026-02...
# MINT:WORK events: 312 (total: 4,847 COIN)
# MINT:GRADIENT events: 89 (total: 1,234 COIN)
# SPEND events: 47 (total: 891 COIN)
# TRANSFER events: 23 (total: 456 COIN)
# DEBIT:DRIFT events: 8 (total: -127 COIN)
\#
# Net minted: 5,954 COIN
# Net spent: 891 COIN
# Net transferred: 0 COIN (zero-sum)
# Net debited: 127 COIN
# Month-end supply: 34,891 COIN
\#
# CLOSE event recorded. Month 2026-02 is now immutable.
```

Key rotation. `vault keygen --rotate` is the annual cryptographic hygiene operation. The old key pair is archived with a datestamp suffix, the new pair is generated and linked. All future LEDGER events are signed with the new key; historical events retain their original signatures.

The rotation warning at 330 days gives a 35-day window for scheduled rotation during maintenance windows. Emergency rotation (`vault keygen --rotate --emergency`) bypasses the warning for suspected key compromise [1428].

49.9. 47.9 enforce-magic-ip: Intellectual Property Protection

`enforce-magic-ip` protects the MAGIC kernel's intellectual property. The kernel's internal implementation – bit weights, hex values, tier boundary scores, dimension encoding formulas – is PRIVATE. These internals must never appear in governance prose, documentation, or public-facing content.

```
enforce-magic-ip --verbose

# Scanning governance prose for kernel internals...
\#
# File: SERVICES/TALK/MAMMOCHAT/README.md
#   Line 47: "The score uses 8 bits weighted at 1, 2, 4, 8, 16, 32, 64, 128"
#   VIOLATION: Bit weight enumeration is PRIVATE
#   Fix: Replace with "The score uses 8 governance dimensions"
\#
# File: BOOKS/CANONIC-DOCTRINE/CHAPTERS/04-THE-EIGHT-DIMENSIONS.md
#   OK — educational context, BOOK scope (allowed)
\#
# Scanned: 73 scopes, 847 files
# Violations: 1
# Allowed (BOOK scope): 4
# EXIT CODE: 1 (violations found)
```

The tool distinguishes between BOOK scope (educational – allowed to discuss internals) and SERVICE/PRODUCT scope (operational – internals must be abstracted). A TALK agent's README.md should describe governance in terms of dimensions and tiers, not bit weights and hex values.

The scan checks for raw bit weights in governance context, tier boundary hex values, dimension encoding formulas, kernel algorithm pseudocode, and MAGIC internal function names. Each violation reports the exact file, line, violating text, and suggested fix.

A hospital's governed TALK agent should not expose the scoring algorithm to end users. Users see the score (255) and the tier (FULL) without seeing the computation. `enforce-magic-ip` maintains this abstraction boundary across the entire governance tree [214].

49.10. 47.10 Tool Composition and Pipeline Integration

Advanced tools compose with the core toolchain through Unix pipes and exit codes. Every tool follows the same contract: read governance files, emit structured output, return 0 (pass) or nonzero (fail). This contract enables arbitrary composition:

```
# Full pre-deploy validation pipeline
magic validate --recursive \&\& \
  validate-design \&\& \
  enforce-magic-ip \&\& \
  build \&\& \
  magic-tag v1.2.0

# CI pipeline with all advanced checks
magic validate --recursive \
  \&\& validate-design \
  \&\& enforce-magic-ip \
  \&\& build \
  \&\& deploy --environment staging \
  \&\& magic scan --format json > governance-report.json
```

Tool	Input	Output	Exit Code
`validate-design`	DESIGN.md + _sass/	Token match report	0 = all match, 1 = mismatch
`magic-tag`	Scope at 255	Signed git tag + LEDGER event	0 = tagged, 1 = precondition fail
`vault`	Subcommand + args	LEDGER event + stdout report	0 = success, 1 = error
`enforce-magic-ip`	GOV tree prose files	IP violation report	0 = clean, 1 = violations

Advanced tools extend the core toolchain without modifying it. They read the same governance files, respect the same exit code contract, and integrate into the same CI/CD pipeline. No tool is privileged – `validate-design` has the same authority as `magic validate`. Both are gates, both must pass, both produce deterministic results.

This composability is the architectural principle: small tools, clear contracts, Unix composition. A hospital IT team adds `validate-hipaa` without modifying any existing tool. A financial institution adds `validate-sox` with the same pattern. The toolchain grows by composition, not modification. Every new gate strengthens governance without weakening existing guarantees [1428].

49.11. 47.11 Clinical Vignette: Advanced Tools Prevent a Deployment Incident

Emory Healthcare deploys GastroChat — a governed TALK agent for gastroenterology consultations covering colorectal cancer screening guidelines (USPSTF, ACG, AGA), inflammatory bowel disease management (Crohn's, ulcerative colitis), and hepatology (NAFLD/NASH, hepatitis C DAA therapy).

A developer updates GastroChat's DESIGN.md to add a new accent color for the hepatology sub-section: `--canonic-accent-hepatology: #4CAF50`. The developer updates DESIGN.md but forgets to add the corresponding SCSS variable to `_TOKENS.scss`. The developer also adds a sentence to GastroChat's README.md: "GastroChat scores 255 using 8 bits weighted at powers of 2."

The pre-deploy pipeline catches both issues:

```
# Pipeline: validate → design → ip → build → tag
magic validate --recursive SERVICES/TALK/GASTROCHAT
# 255/255 □

validate-design
# FAIL: Token --canonic-accent-hepatology exists in DESIGN.md but not in _TOKENS.scss
# EXIT CODE: 1

# Pipeline stops. Developer fixes _TOKENS.scss, re-runs:
validate-design
# PASS □

enforce-magic-ip
# FAIL: SERVICES/TALK/GASTROCHAT/README.md line 23:
#   "8 bits weighted at powers of 2" — bit weight description is PRIVATE
# EXIT CODE: 1

# Pipeline stops. Developer rewrites to:
#   "GastroChat achieves full governance across all 8 dimensions"
enforce-magic-ip
# PASS □

build \&\& magic-tag v2.1.0
# BUILD OK — 13 phases □
# TAG: GASTROCHAT-v2.1.0 (signed, LEDGER: CERTIFY)
```

Two issues caught, neither detectable by `magic validate` alone – the scope was already at 255. The design token mismatch would have produced an unstyled hepatology section (green accent declared but never compiled to CSS). The IP leak would have exposed kernel internals to end users. Advanced tools extend the governance surface beyond the 8-dimension score into domain-specific quality gates [714283].

49.12. 47.12 Writing Custom Advanced Tools

Any organization can extend the toolchain with custom advanced tools. The contract is minimal: read governance files from the GOV tree, emit structured output to stdout, return exit code 0 for pass and nonzero for fail. The tool must never write governance files — governance authorship is a human responsibility.

A custom tool template:

```
\#!/usr/bin/env python3
"""validate-{domain} — Custom governance validator for {domain}."""

import sys
import os

def main():
    gov_root = os.environ.get("GOV\_ROOT", ".")
    violations = []

    \# Walk the governance tree
    for root, dirs, files in os.walk(gov_root):
        for f in files:
            if f == "CANON.md":
                path = os.path.join(root, f)
                with open(path) as fh:
                    content = fh.read()
                \# Domain-specific validation logic here
                if "DOMAIN\_SPECIFIC\_CHECK" not in content:
                    violations.append(f"{path}: missing domain check")

    \# Emit structured output
    for v in violations:
        print(f"VIOLATION: {v}", file=sys.stderr)

    \# Exit code contract
    return 1 if violations else 0

sys.exit(main())
```

The custom tool integrates into the pipeline with &&:

```
magic validate --recursive \&\& validate-hipaa \&\& validate-sox \&\& build
```

Organizations that need HIPAA validation write `validate-hipaa`. Financial institutions write `validate-sox`. Defense contractors write `validate-fedramp`. Each tool is a single Python script that reads governance

files and checks domain-specific requirements. The toolchain grows horizontally — each organization adds its gates without modifying the core tools.

The governance guarantee: if the pipeline passes, every gate passed. If any gate fails, the pipeline stops. The exit code contract makes this guarantee compositional — no tool can silently fail. No tool can partially pass. The binary pass/fail discipline extends from the core `magic validate` through every custom tool in the pipeline [14].

49.13. 47.13 Advanced Tool Governance Proof

The advanced tools themselves are governed. Each tool has a scope in `SERVICES/MAGIC/` with its own CANON.md, README.md, and VOCAB.md. The tool's governance contract declares its inputs, outputs, exit code semantics, and LEDGER event types. A tool that violates its own governance contract is a bug — the tool must be fixed, not the contract.

```
SERVICES/MAGIC/
  CANON.md              <- MAGIC service governance
  VALIDATE-DESIGN/
    CANON.md            <- validate-design governance
    README.md           <- tool documentation
    VOCAB.md            <- token vocabulary
  MAGIC-TAG/
    CANON.md            <- magic-tag governance
    README.md           <- certification documentation
  ENFORCE-MAGIC-IP/
    CANON.md            <- enforce-magic-ip governance
    README.md           <- IP protection documentation
```

Run `magic validate SERVICES/MAGIC/VALIDATE-DESIGN/` and the tool that validates design tokens is itself validated by the governance kernel. The recursion is intentional — governance governs itself. The advanced tools are not exceptions to governance. They are governed artifacts that enforce governance on other governed artifacts. The governance chain is complete: kernel validates tools, tools validate scopes, scopes validate content. Every link in the chain is auditable. Every link in the chain scores 255 [1428].

49.14. 47.14 Backup and Disaster Recovery

`backup snapshot` creates a complete, encrypted archive of the governance state. The archive includes VAULT (key material), LEDGER (economic history), SERVICES (governance tree), and LEARNING (evolution state). The encryption uses GPG with AES-256 — the same standard used by healthcare organizations for PHI backup.

```
backup snapshot --output /secure/canonic-backup-2026-03-10.tar.gz.gpg

# Collecting governance artifacts...
#   VAULT/: 12 files (key pairs, wallets)
#   LEDGER/: 1 file (847 events, 2.1MB)
#   SERVICES/: 73 scopes (730 files, 4.7MB)
#   LEARNING/: 14 files (evolution state)
\#
# Compressing: tar.gz (4.7MB → 1.2MB)
# Encrypting: AES-256 via GPG
# Output: /secure/canonic-backup-2026-03-10.tar.gz.gpg (1.3MB)
# SHA-256: a3f7c2d...e91b4f8
\#
# LEDGER: BACKUP:SNAPSHOT event recorded
```

`backup restore` performs the inverse: decrypt, decompress, and place governance artifacts in their canonical locations. The restore is idempotent — running it twice produces the same result. The restore verifies LEDGER chain integrity and WALLET balance consistency before declaring success.

```
backup restore --input /secure/canonic-backup-2026-03-10.tar.gz.gpg

# Decrypting...
# Decompressing...
# Restoring VAULT/: 12 files □
# Restoring LEDGER/: 847 events □
# Restoring SERVICES/: 73 scopes □
# Restoring LEARNING/: 14 files □
\#
# Verification:
#   LEDGER chain: VALID (all hashes match)
#   WALLET balances: CONSISTENT (sum matches CLOSE events)
#   Service scores: 73/73 at 255
\#
# RESTORE COMPLETE
```

`backup verify` validates an existing backup without restoring it. The verification checks that the archive is intact (GPG signature valid), the LEDGER chain is unbroken (every event's hash chains to its predecessor), and the WALLET balances reconcile (MINT - SPEND - DEBIT = balance for every identity). This verification can run on a separate machine — the auditor does not need access to the production governance tree.

The backup cadence follows the CLOSE cadence: monthly snapshots after `vault close`. Each snapshot is a complete governance checkpoint. The organization's RPO (Recovery Point Objective) is one month — the maximum data loss in a disaster scenario is one month of governance activity. The RTO (Recovery Time Objective) is the time to run `backup restore`: typically under 60 seconds for a 73-scope governance tree.

For organizations with stricter RPO requirements, incremental backups capture daily LEDGER deltas. The incremental backup appends new LEDGER events to the previous snapshot without re-encrypting the entire archive. A full restore applies the base snapshot plus all incremental deltas in chronological order.

The disaster recovery contract is deterministic: given the same backup archive, `backup restore` produces the same governance state on any machine. The governance tree is portable — it depends only on the filesystem and Python 3.11+. No database. No external service. No cloud dependency. The governance state is self-contained in files, and the backup is a self-contained archive of those files. Q.E.D. [142812].

Chapter 50

Chapter 48: The Closure — CANONIC Against All of Programming

One chapter. One proof. Everything you have read so far converges here. The TRIAD from Chapter 2, the eight dimensions from Chapter 4, the inheritance chain from Chapter 3, the type system of Chapter 36, the compiler of Chapter 37, the LEARNING closure of Chapter 39, and the Level 4 argument of Chapter 40 — all resolve into a single closure proof. The full closure tables are in Appendix E (Chapter 49). For the governor's perspective on why this closure matters for healthcare, research, and institutional governance, see the CANONIC CANON.

50.1. 48.1 The Argument

1. **C is the kernel.** The CANONIC kernel (`magic.c`) is a 35KB C binary that performs O(1) compliance checking via bitwise AND. It compiles to a shared library (`.so`/`.dylib`) exposing the C ABI [28].
2. **All languages neofunctionalize C through FFI.** In biology, neofunctionalization occurs when a gene duplicates and one copy acquires a new function. The same pattern holds here: the spec is invariant, the syntax is variant. Python wraps C via `ctypes`, Swift via `@_silgen_name`, TypeScript via `ffi-napi`. The wrapper never replaces the kernel [28].
3. **Four canonical runtimes** — C (bare metal), Python (server/AI), Swift (Apple/mobile), TypeScript (web/browser) — each passing 10/10 compliance tests and scoring 255/255 coverage [28].

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
TypeScript	`ffi-napi`	`magic.ts`	10/10	255

4. **Eight-org language clade.** Python, TypeScript, Rust, Go, Swift, Kotlin, SQL, WASM — each occupying a niche in the adaptive radiation. Of 19 languages in the test spec, 4 are closed and 15 remain TODO. Every one follows the same pattern: FFI bridge, 10-test compliance, 255/255 coverage, LEDGER chain [928].

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) [9].

6. **Every paradigm family maps to governance questions.** Twenty families, over 100 languages, every one mapped, every one CLOSED. Detailed tables in Appendix E.

Family	Languages	CANONIC Mapping	Questions Addressed	Missing
Imperative & OOP	Java, C++, Python, Swift, Kotlin, Ruby, C#	scopes, inheritance, encapsulation	belief, shape	learning
Functional	Haskell, OCaml, Elixir, Clojure, F#, Elm	immutability, composition, purity	mechanism, shape	learning
Type Systems	TypeScript, Rust, Idris, Agda	VOCAB = types, CANON = contracts	proof, expression	learning
Concurrent & Actor	Go, Erlang/OTP, Akka, Rust channels	LEDGER events, message passing	mechanism, timeline	learning
Logic & Constraint	Prolog, Datalog, MiniKanren, Mercury	axioms, derivation, resolution	belief, proof	learning
Reactive & Dataflow	Rx, Flink, LabVIEW, Lucid	LEDGER streams, event sourcing	timeline, mechanism	learning
Concatenative	Forth, Factor, PostScript, Joy	stack composition, terseness	shape	learning
Array & Numeric	APL, J, K, NumPy, MATLAB, Julia, R	vectorized operations, data	identity, mechanism	learning
Metaprogramming	Lisp, Racket, Zig comptime, Rust macros	code-as-data, compilation	belief, identity	learning
Smart Contracts	Solidity, Vyper, Move, Clarity	CANON = contract, LEDGER = chain	belief, proof, mechanism	learning

Family	Languages	CANONIC Mapping	Questions Addressed	Missing
Proof Assistants	Coq, Lean, Isabelle, TLA+, Alloy	axioms □ theorems, CANON □ 255	belief, proof, shape	learning
Probabilistic	Stan, PyMC, Church, Turing.jl	uncertainty quantification	proof, mechanism	learning
GPU & Parallel	CUDA, OpenCL, SYCL, Triton, Halide	parallel execution, throughput	identity, mechanism	learning
DSLs & Config	Terraform, Puppet, Ansible, Dhall, Nix	infrastructure as code	belief, identity	learning
Markup & Styling	HTML, CSS, Sass, YAML, TOML, JSON	structure, presentation	shape, expression	learning
Query & Data	SQL, GraphQL, Datalog, SPARQL, Cypher	data access, relations	proof, identity	learning
Shell & Scripting	Bash, Zsh, PowerShell, Fish, Make	automation, orchestration	timeline, identity	learning
Visual & Low-Code	Scratch, Blockly, Node-RED, Retool	accessibility, rapid prototyping	belief, shape	learning
Systems & Bare Metal	C, Assembly, Zig, Rust	the kernel itself	all eight*	learning†
LEARNING	**none alone**	**accumulated intelligence**	**learning**	—

*Systems languages map to ALL eight questions 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 LEARNING question — accumulated intelligence, the one thing no programming language answers 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 question. Therefore:

CANONIC governance subsumes every programming paradigm.

The proof is architectural, not abstract: - C is the kernel — all languages bind to it via FFI [28] - The spec is invariant — 10 compliance tests, 255/255 score [28] - The syntax is variant — each language brings its own idioms [9] - Every paradigm maps to governance questions [28] - Every paradigm is missing LEARNING [28] - LEARNING closes the loop - Therefore CANONIC governance CLOSES against all of programming

Q.E.D.

50.2. 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]
```

Four closed, fifteen TODO. The closure horizon spans 19 languages, each neofunctionalizing C, each passing the 10-test compliance standard, each scoring 255/255 [28].

50.3. 48.3 The Closure Matrix

Twenty paradigm families against eight governance questions. The full matrix lives in Appendix E.

50.4. 48.4 The Phylogenetic Proof

The phylogenetic tree spans 16 organizations [9]:

```
((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 [9].

50.5. 48.5 LEARNING

Governance is compilation, but LEARNING is the question compilation alone cannot answer. 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. Code is evidence. LEARNING accumulates the intelligence. The loop closes.

50.6. 48.6 The 20 Closure Tables

Twenty programming concerns, twenty comparisons. In every case, CANONIC governance provides what the paradigm alone cannot.

Table 1: Naming and Identity

Concern	Traditional	CANONIC
Identity	Variable names, class names	CANON.md axiom: “X is Y”
Uniqueness	Naming conventions	Scope path (structurally unique)
Discovery	`import`, `require`	`magic scan` (filesystem discovery)

Concern	Traditional	CANONIC
Gap	Names carry no governance meaning	Axiom carries doctrinal meaning [14]

Table 2: Type Systems

Concern	Traditional	CANONIC
Types	`int`, `string`, `MyClass`	VOCAB.md terms
Contracts	Type signatures	CANON.md + COVERAGE.md
Validation	Compile-time checking	`magic validate` (255-bit)
Gap	Types do not learn from deployment	LEARNING.md records patterns [14]

Table 3: Dependency Management

Concern	Traditional	CANONIC
Declaration	package.json, go.mod	CANON.md `inherits:`
Resolution	npm install, cargo build	`magic scan` (structural)
Versioning	semver (1.2.3)	Score (0-255)
Gap	Dependencies have no governance score	Every dependency is governed [14]

Table 4: Testing

Concern	Traditional	CANONIC
Unit tests	pytest, jest	`magic validate` (per-scope)
Coverage	Code coverage %	Governance coverage (255-bit)
Regression	Test failures	Score regression (DEBIT:DRIFT)
Gap	Tests check code, not governance	Validation checks governance first [14]

Table 5: Documentation

Concern	Traditional	CANONIC
Docs	README.md, JSDoc	{SCOPE}.md, COVERAGE.md, HTTP.md
Freshness	Docs rot (no enforcement)	Score drops if docs removed

Concern	Traditional	CANONIC
Gap	Docs are optional	Governance files are mandatory [14]

Table 6: Configuration

Concern	Traditional	CANONIC
Config	.env, yaml, json	CANON.md (governance)
Secrets	.env.local, vault	VAULT/ (encrypted, governed)
Drift	Config diverges across envs	Single GOV tree (no divergence)
Gap	Config is ungoverned	Config IS governance [2]

Table 7: Error Handling

Concern	Traditional	CANONIC
Errors	try/catch, Result	Error codes (E100-E504)
Recovery	Retry, fallback	Heal loop (5-stage)
Logging	log.error(), sentry	LEDGER (append-only)
Gap	Errors are runtime events	Errors are governance events [14]

Table 8: Access Control

Concern	Traditional	CANONIC
Auth	RBAC, OAuth	VAULT (economic identity)
Audit	Audit logs	LEDGER chain
Gap	Access control is runtime-only	Access is governed and auditable [14]

Table 9: Deployment

Concern	Traditional	CANONIC
Pipeline	Jenkins, GitHub Actions	magic-build.yml (13 DAG-parallel phases)
Gates	Test pass, code review	Score = 255, PRIVATE leak check
Evidence	Deploy logs	Build evidence (Step 14)
Gap	Deployment checks code quality	Deployment checks governance [14]

Table 10: Monitoring

Concern	Traditional	CANONIC
Health	/health endpoint	Score = 255 (per-scope)
Alerting	Threshold alerts	`--below 255` filter
Gap	Monitors runtime metrics	Monitors governance metrics [14]

Table 11: Database and State

Concern	Traditional	CANONIC
Schema	CREATE TABLE, migrations	VOCAB.md (vocabulary is schema)
State	Database rows	LEDGER entries (append-only)
Gap	Schema has no governance score	Every vocabulary is scored [14]

Table 12: API Design

Concern	Traditional	CANONIC
Spec	OpenAPI, GraphQL schema	HTTP.md (route table)
Contracts	Request/response types	COVERAGE.md (coverage matrix)
Gap	API spec is separate from governance	API spec IS governance [14]

Table 13: Microservices

Concern	Traditional	CANONIC
Service	Docker container, K8s pod	Scope with “X is SERVICE”
Registry	Consul, Eureka	`magic scan` (discovery)
Gap	Services are runtime artifacts	Services are governance scopes [14]

Table 14: Security

Concern	Traditional	CANONIC
Secrets	Vault, KMS	VAULT/ (AES-256)
Compliance	SOC2 checklist	Score = 255 (by construction)
Gap	Security is a checklist	Security is a score [14]

Table 15: Performance

Concern	Traditional	CANONIC
Benchmarks	JMH, wrk, ab	`load-test` (latency gates)
Optimization	Profiling, caching	O(1) kernel
Gap	Performance is measured	Governance is enforced [14]

Table 16: Team Organization

Concern	Traditional	CANONIC
Ownership	CODEOWNERS	CANON.md axiom
Responsibility	Team charter	COVERAGE.md
Onboarding	Wiki, README	`magic scan --tree`
Gap	Org structure is informal	Org structure is governed [14]

Table 17: Code Review

Concern	Traditional	CANONIC
Review	PR review, LGTM	Pre-commit hook (machine review)
Standards	Linter, style guide	255-bit standard
Gap	Review checks code style	Validation checks governance [14]

Table 18: Internationalization

Concern	Traditional	CANONIC
Vocabulary	i18n files, .po	VOCAB.md (domain vocabulary)
Consistency	Translation memory	`validate-vocab` (inheritance)
Gap	i18n covers UI strings	VOCAB covers domain truth [14]

Table 19: Economics

Concern	Traditional	CANONIC
Billing	Stripe, Braintree	VAULT/ + SHOP.md
Identity	Customer ID	Wallet (economic identity)
Audit	Payment logs	LEDGER chain (COIN events)
Gap	Economics is external to code	Economics is a governance question [14]

Table 20: Learning and Evolution

Concern	Traditional	CANONIC
Knowledge	Wiki, Confluence	LEARNING.md (per-scope)
Patterns	Tribal knowledge	Recorded in LEARNING.md
Intelligence	Lost when people leave	Accumulated in governance
Gap	**No language learns**	**LEARNING is first-class** [14]

50.7. 48.7 The Final Argument

Twenty tables, one conclusion: traditional programming addresses each concern at the code level, CANONIC governance addresses it at the governance level, and the governance level subsumes the code level because governance compiles to code — never the reverse [2].

The gap in every table is identical: **code does not govern itself**. It does not know its own coverage requirements, does not track its own evolution, does not learn from its own deployment.

50.8. 48.8 The Three Impossibility Results

No programming language can achieve these three properties:

1. **Self-governance.** A program cannot validate its own governance completeness — the validator must be external to the validated, a Godelian constraint [14].
2. **Economic identity.** A program cannot own a wallet. Only a governed entity with a VAULT can have economic identity. The economics question is unreachable from code alone [14].
3. **Accumulated learning.** A program does not know what it learned yesterday. A Haskell function is pure: same input, same output, forever. Only governance accumulates intelligence across time [14].

No programming paradigm provides all three. CANONIC governance provides all three by construction.

50.9. 48.9 The Closure Proof

```
GIVEN:
  1. 20 paradigm families, 100+ languages [48.1]
  2. Every family maps to governance questions [48.1]
  3. Every family is missing LEARNING [48.1]
  4. 20 programming concerns [Tables 1–20]
```

```
  5. Every concern shows governance gap [Tables 1–20]
  6. Three impossibility results [48.8]

THEREFORE:
  No programming language subsumes CANONIC governance.
  CANONIC governance subsumes every programming paradigm.
  The closure is complete.
```

Q.E.D.

50.10. 48.10 The Last Word

Governance is not bureaucracy — it is compilation. The `.md` file is source, the `.json` file is binary, `magic validate` is the compiler, 255 is a successful build, the LEDGER is the audit trail, and LEARNING is the intelligence.

Every program ever written needed governance. Most never had it. CANONIC delivers governance by construction: author the `.md`, run the compiler, score 255, deploy.

50.11. 48.11 The Implementation Evidence

The closure is not hypothetical — it is deployed:

Metric	Value	Significance
GitHub organizations	19	Cross-org federation proven
Repositories	185+	Scale beyond single-team
Scopes at 255	73/73	Full compliance across fleet
Programming languages (closed)	4 (C, Python, Swift, TypeScript)	Four canonical runtimes proven
Programming languages (TODO)	15	Closure horizon mapped
Compliance tests per language	10/10	Universal test standard
LEDGER events	12,000+	Economic system operational
COIN minted	18,000+	Governance work economically visible

The four closed runtimes prove the FFI pattern: same C kernel, same 10 tests, same 255/255 score. This is not a prototype — it is production infrastructure serving clinical AI deployments across 51 enterprise hospitals.

50.12. 48.12 The Clinical Proof

The strongest evidence for the closure is clinical. MammoChat, OncoChat, MedChat, FinChat, LawChat — each a governed TALK agent compiled from INTEL through the pipeline described in Chapter 25. Each agent:

1. Inherits constraints from `canonic-canonic` through the TALK service tree.
2. Compiles INTEL.md into a governed systemPrompt.
3. Serves clinical conversations within the axiom's boundary.
4. Records every interaction on the LEDGER.
5. Mints COIN for governance work performed on the scope.
6. Captures patterns in LEARNING.md.

No traditional clinical AI deployment has all six simultaneously. Traditional systems might have audit logs or documentation, but none closes the loop where governance, conversation, economics, and learning are architecturally unified.

Clinical trial NCT06604078 deploys MammoChat across 51 hospitals. Every deployment is a governed scope, every scope validates to 255, every interaction is ledgered. The clinical proof is operational, not theoretical.

50.13. 48.13 The Finality

The closure has three components:

```
1. Every paradigm maps to governance questions  (Table in 48.1)
2. Every paradigm is missing LEARNING          (Column in 48.1)
3. CANONIC provides LEARNING as first-class    (Chapter 39)
```

CANONIC governance subsumes every programming paradigm by providing the question no paradigm answers alone. This is not a feature comparison — it is a structural proof. The question exists (LEARNING.md), the languages lack it (no language has LEARNING.md), and CANONIC provides it (one of the eight questions in the MAGIC 255 score). The closure is complete.

255 or reject [14].

50.14. 27.18 INTEL.md Migration Patterns

When migrating an existing knowledge base into CANONIC governance, INTEL.md is the entry point. The migration pattern:

```
# Step 1: Inventory existing knowledge sources
# List all documents, papers, blog posts, databases
# that the scope's knowledge is based on.

# Step 2: Categorize into evidence layers
# Layer 1: Governance sources (CANONIC internal)
# Layer 2: Peer-reviewed papers (with DOIs)
# Layer 3: Blog posts / articles (with dates)
# Layer 4: Service specifications
# Layer 5: External references

# Step 3: Write INTEL.md
# Use the template from 27.12

# Step 4: Add citations to all existing claims
# Every factual statement in scope documents
# must reference an evidence layer entry

# Step 5: Validate
magic validate --citations SERVICES/TALK/NEWCHAT/INTEL.md
```

Clinical migration example: An oncology department has a shared Google Drive with 47 clinical practice guidelines, 12 drug interaction databases, and 8 institutional protocols. Migrating to CANONIC INTEL:

Source	Count	INTEL Layer	Citation Format
NCCN Guidelines	15	Layer 2 (Papers)	[P-XX] NCCN 2026.1
ACR Practice Parameters	8	Layer 2 (Papers)	[P-XX] ACR 2025
Drug interaction databases	12	Layer 5 (External)	[E-XX] Lexicomp/UpToDate
Institutional protocols	8	Layer 4 (Service)	[S-XX] Protocol name
FDA safety communications	4	Layer 5 (External)	[E-XX] FDA MedWatch

The migration transforms unstructured knowledge into governed, cited, cross-referenced INTEL. After migration, every clinical claim in the scope traces to a source [2].

50.15. 27.19 INTEL Versioning and History

INTEL.md is version-controlled via git. The git history of INTEL.md is the evidence evolution history:

```
# View INTEL.md evolution
git log --oneline SERVICES/TALK/MAMMOCHAT/INTEL.md

# Output:
# abc1234 Update NCCN to 2026.1, add NCT06604078
# def5678 Add layer 5 external references
# 789abcd Initial MammoChat INTEL (12 papers, 3 governance)
```

Each commit to INTEL.md is a governance event. The LEDGER records the delta:

```
{
  "event": "INTEL:UPDATE",
  "scope": "SERVICES/TALK/MAMMOCHAT",
  "commit": "abc1234",
  "message": "Update NCCN to 2026.1, add NCT06604078",
  "delta": {
    "papers_added": 1,
    "papers_removed": 0,
    "layers_updated": ["Layer 2"],
    "freshness_improved": true
  }
}
```

The complete history of what the agent knows, when it learned it, and who governed the change is in the git log + LEDGER combination. For clinical AI compliance, this is the knowledge management audit trail [214].

50.16. 27.20 INTEL and Agent Knowledge Boundaries

INTEL.md explicitly defines what an agent knows and does not know. The knowledge boundary is compiled into the systemPrompt:

```
KNOWLEDGE BOUNDARY (compiled from INTEL.md):
  KNOWS:
    - BI-RADS classification system (ACR 5th Edition)
    - NCCN breast cancer screening guidelines (2026.1)
    - Breast imaging modalities (mammography, MRI, ultrasound)
    - Clinical trial NCT06604078 (deployment data)
    - mCODE breast cancer profiles

  DOES NOT KNOW:
    - Patient-specific data (no PHI access)
```

```
- Non-breast oncology (routes to OncoChat)
- Genomic variant classification (routes to OmicsChat)
- Drug interactions (routes to MedChat)
- Legal compliance details (routes to LawChat)
- Financial/billing codes (routes to FinChat)
```

The knowledge boundary is the agent's type signature. It declares what the agent can answer and what it must route. The boundary is enforced by the systemPrompt — the agent will decline queries outside its boundary and suggest routing to the appropriate specialist agent.

Clinical significance: A patient who asks MammoChat about chemotherapy drug interactions receives a routing response, not a hallucinated answer. The knowledge boundary prevents the agent from operating outside its governed expertise. The boundary is governance, not a suggestion [2126].

50.17. 27.21 INTEL Completeness Checklist

Use this checklist to verify INTEL.md completeness before submitting for review:

#	Item	Required	Check
1	`inherits:` field present	Yes	Path resolves
2	Axiom section present	Yes	Non-empty, non-placeholder
3	Scope Intelligence table	Yes	All 4 fields filled
4	Evidence Chain table	Yes	At least 2 layers populated
5	Cross-Scope Connections table	Yes	At least TALK + COIN + LEDGER
6	Citations in all claims	Yes	0 uncited claims
7	Evidence layers INDEXED	Yes	At least layers 1-2
8	Freshness < 90 days	Recommended	No stale layers
9	Quality score HIGH	Recommended	Coverage > 0.95
10	Source count > 10	Recommended	Depth threshold met

```
# Run the completeness check
magic intel --checklist SERVICES/TALK/MAMMOCHAT
```

A complete INTEL.md enables the language question. An incomplete INTEL.md blocks MAGIC tier [2].

50.18. 27.22 INTEL.md as Single Source of Truth

INTEL.md is the single source of truth for what a scope knows. There is no second knowledge base. There is no hidden configuration. There is no database of evidence that exists outside INTEL.md.

If knowledge is not in INTEL.md, the agent does not know it. If evidence is not cited in INTEL.md, the agent cannot cite it. If a cross-scope connection is not declared in INTEL.md, the routing table does not include it.

This constraint is deliberate. It means that auditing an agent's knowledge requires reading one file: INTEL.md. It means that updating an agent's knowledge requires editing one file: INTEL.md. It means that the entire knowledge provenance of a clinical AI agent fits in a single Markdown document.

The simplicity is the governance. Complex knowledge management systems fail because complexity hides ungoverned knowledge. INTEL.md succeeds because everything is visible, everything is cited, and everything compiles [2].

This appendix collects the reference tables, closure proofs, and citation indices for THE CANONIC DOCTRINE. Appendix A maps every governance file to the chapter where it is introduced (TRIAD in Chapter 2, COVERAGE and ROADMAP in Chapter 6, LEARNING in Chapter 10, SHOP in Chapter 34, INTEL in Chapter 27). Appendix B formalizes the tier algebra from Chapter 5. Appendix C provides a CLI quick reference for the toolchain (Chapters 41-47). Appendix E contains the twenty closure tables that complete the proof of Chapter 48. For the companion text covering the governor's perspective, see the CANONIC CANON.

• • •

50.19. Appendix A: Governance File Reference

File	Question Answered	Purpose	Required
CANON.md	What do you believe?	Axiom + constraints	YES
VOCAB.md	Can you prove it?	Term definitions	YES (for closure)
README.md	What shape are you?	Public interface	YES
{SCOPE}.md	Who are you?	Scope specification	YES (for BUSINESS)
COVERAGE.md	How do you work?	8-question assessment	YES (for ENTERPRISE)
ROADMAP.md	Where are you going?	Forward milestones	YES (for ENTERPRISE)
LEARNING.md	What have you learned?	Pattern table	YES (for AGENT)
SHOP.md	—	Economic projection	Optional

File	Question Answered	Purpose	Required
INTEL.md	—	Scope intelligence	Optional
VITAE.md	—	Identity evidence	Required for certification
TAGS.md	—	Certification registry	Required for tagging

• • •

50.20. Appendix B: Tier Algebra

Question composition — the kernel computes each score [6].

Tier	Questions Satisfied	Files Required
COMMUNITY	3 of 8	CANON.md + VOCAB.md + README.md
BUSINESS	4 of 8	+ {SCOPE}.md
ENTERPRISE	6 of 8	+ ROADMAP.md + COVERAGE.md
AGENT	7 of 8	+ LEARNING.md
FULL (MAGIC)	8 of 8	+ LANGUAGE inherited

Tiers are cumulative. Monotonic accumulation. No skipping.

• • •

50.21. Appendix C: CLI Quick Reference

Command	Purpose
`magic validate`	Compute scope score
`magic scan`	Discover all scopes
`magic heal`	Diagnose unanswered questions

Command	Purpose
`magic ledger`	Show COIN events
`magic-heal`	Five-stage settlement
`build`	Full pipeline: JSON + Jekyll + validate
`build-galaxy-json`	Generate galaxy.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

• • •

50.22. 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`

• • •

50.23. Appendix E: The Closure Tables

Twenty paradigm families mapped against the eight governance questions the C kernel evaluates. Over 100 languages are represented. = question addressed natively. · = not addressed.

50.24. E.1: Imperative & OOP

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Java	✓	·	·	✓	✓	✓	·	✓	Classes = scopes, interfaces = contracts
C++	✓	·	·	✓	·	✓	·	✓	Templates, namespaces, headers = README
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: belief (classes declare), shape (encapsulation), identity, expression **Missing:** learning, timeline, proof

50.25. E.2: Functional

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Haskell	□	□	·	□	□	□	·	□	Types = VOCAB, purity = governance
OCaml	□	□	·	□	□	□	·	□	Modules = scopes, functors = composition
Elixir	□	·	□	□	□	□	·	□	Supervisors = governance, OTP = operations
Clojure	□	·	□	□	·	□	·	□	Immutability, persistent data, REPL
F#	□	□	·	□	□	□	·	□	Type providers, computation expressions
Elm	□	□	·	□	□	□	·	□	No runtime errors, model-view-update
Erlang	□	·	□	□	□	□	·	□	OTP patterns, supervision trees

Covered: belief, shape (composition), mechanism (immutability), expression, some proof (types), some timeline **Missing:** learning

50.26. E.3: Type Systems & Dependent Types

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	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: belief, proof (types = vocabulary), shape, identity, mechanism, expression **Missing:** learning, timeline (limited)

50.27. E.4: Concurrent & Actor

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Go	□	·	□	□	□	□	·	□	Goroutines, channels = LEDGER events
Erlang/OTP	□	·	□	□	□	□	·	□	Actors, supervision, let-it-crash

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Akka	□	·	□	□	□	□	·	□	Actor model on JVM
Rust channels	□	□	□	□	□	□	·	□	Ownership + concurrency = safety

Covered: belief, timeline (message ordering), mechanism (operations), shape, identity, expression **Missing:** learning

50.28. E.5: Logic & Constraint

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Prolog	□	□	·	·	·	□	·	□	Facts = axioms, queries = validation
Datalog	□	□	·	·	□	□	·	□	Stratified, decidable
MiniKanren	□	□	·	·	·	□	·	□	Relational, embedded
Mercury	□	□	·	□	□	□	·	□	Logic + types + modes

Covered: belief (axioms), proof (derivation), shape (structure) **Missing:** learning, timeline, identity (limited)

50.29. E.6: Reactive & Dataflow

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
RxJS/RxJava	□	·	□	·	□	□	·	□	Observable streams = LEDGER events

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Apache Flink	□	·	□	□	□	□	·	□	Event time, exactly-once
LabVIEW	□	·	□	□	□	□	·	□	Visual dataflow, instrumentation
Lucid	□	·	□	·	·	□	·	□	Intensional, historical values

Covered: timeline (streams, temporal), mechanism (operations), shape, belief **Missing:** learning, proof (limited)

50.30. E.7: Concatenative & Stack

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Forth	□	·	·	·	·	□	·	□	Stack, words = definitions
Factor	□	·	·	□	·	□	·	□	Quotations, combinators
PostScript	□	·	·	·	·	□	·	□	Page description, graphics
Joy	□	·	·	·	·	□	·	□	Pure concatenative, quotations

Covered: belief (definitions), shape (stack composition), expression **Missing:** learning, timeline, proof, mechanism, identity

50.31. E.8: Array & Numeric

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
APL/J/K	□	·	·	·	□	□	·	□	Array operations, vectorized
NumPy	□	·	·	□	□	□	·	□	ndarray, broadcasting
MATLAB	□	·	·	□	□	□	·	□	Matrix operations, toolboxes
Julia	□	·	·	□	□	□	·	□	Multiple dispatch, JIT
R	□	·	·	□	□	□	·	□	Statistical modeling, CRAN

Covered: belief, identity, mechanism (operations), shape, expression **Missing:** learning, timeline, proof

50.32. E.9: Metaprogramming

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	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: belief (declaration), identity, shape, expression, some proof, some mechanism **Missing:** learning, timeline

50.33. E.10: Smart Contracts

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Solidity	□	□	□	□	□	□	·	□	Contracts = CANON, blockchain = LEDGER
Vyper	□	□	□	□	□	□	·	□	Pythonic, security-first
Move	□	□	□	□	□	□	·	□	Resource types, ownership
Clarity	□	□	□	□	□	□	·	□	Decidable, no reentrancy

Covered: belief, proof, timeline (blockchain time), identity, mechanism, shape, expression — nearly full
Missing: learning (the chain records but does not learn)

50.34. E.11: Proof Assistants

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Coq	□	□	·	□	□	□	·	□	Calculus of constructions
Lean	□	□	·	□	□	□	·	□	Tactics, mathlib
Isabelle	□	□	·	□	□	□	·	□	Classical HOL
TLA+	□	□	□	□	□	□	·	□	Temporal logic, model checking
Alloy	□	□	·	□	□	□	·	□	Relational logic, bounded

Covered: belief (axioms), proof (formal proofs), shape (formal structure), identity, mechanism, expression, some timeline **Missing:** learning (proofs do not learn from their evolution)

50.35. E.12: Probabilistic

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Stan	□	□	·	□	□	□	·	□	Bayesian inference, HMC
PyMC	□	□	·	□	□	□	·	□	Probabilistic models in Python
Church	□	□	·	·	·	□	·	□	Universal probabilistic language
Turing.jl	□	□	·	□	□	□	·	□	Julia-based inference

Covered: belief, proof (evidence/posterior), identity, mechanism, shape, expression **Missing:** learning, timeline

50.36. E.13: GPU & Parallel

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
CUDA	□	·	·	□	□	□	·	□	Thread blocks, kernels
OpenCL	□	·	·	□	□	□	·	□	Platform-independent
SYCL	□	·	·	□	□	□	·	□	C++ standard parallel
Triton	□	·	·	□	□	□	·	□	Python GPU compiler

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Halide	□	·	·	□	□	□	·	□	Schedule/algorithm separation

Covered: belief, identity, mechanism (parallel operations), shape, expression **Missing:** learning, timeline, proof

50.37. E.14: DSLs & Config

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Terraform	□	·	□	□	□	□	·	□	Infrastructure as code, state
Puppet	□	·	·	□	□	□	·	□	Declarative config management
Ansible	□	·	·	□	□	□	·	□	Agentless, playbooks
Dhall	□	□	·	□	□	□	·	□	Typed, total, decidable
Nix	□	□	·	□	□	□	·	□	Reproducible, functional
Docker	□	·	·	□	□	□	·	□	Container images, layers

Covered: belief (declarations), identity (reproducible), mechanism (operations), shape, expression **Missing:** learning, some proof, some timeline

50.38. E.15: Markup & Styling

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
HTML	□	·	·	·	·	□	·	□	Semantic structure
CSS	·	·	·	·	·	□	·	□	Visual presentation
Sass	·	·	·	·	·	□	·	□	CSS with variables, nesting

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
YAML	□	·	·	·	·	□	·	□	Data serialization
TOML	□	·	·	·	·	□	·	□	Configuration
JSON	·	·	·	·	·	□	·	□	Data interchange

Covered: shape (structure), expression, some belief **Missing:** learning, timeline, proof, identity, mechanism

50.39. E.16: Query & Data

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
SQL	□	□	·	□	□	□	·	□	Relational, ACID
GraphQL	□	·	·	□	·	□	·	□	Schema = types, resolvers
Datalog	□	□	·	·	□	□	·	□	Facts + rules, stratified
SPARQL	□	□	·	□	·	□	·	□	Semantic web, triples
Cypher	□	·	·	□	·	□	·	□	Graph queries, Neo4j

Covered: belief, proof (evidence/data), identity, mechanism, shape, expression **Missing:** learning, timeline

50.40. E.17: Shell & Scripting

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Bash	□	·	□	□	·	·	·	□	Pipelines, process control
Zsh	□	·	□	□	·	·	·	□	Extended Bash, completions
PowerShell	□	·	□	□	□	□	·	□	Object pipeline, cmdlets
Fish	□	·	□	□	·	·	·	□	User-friendly, autosuggestions

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Make	□	·	□	□	·	□	·	□	Dependency rules, targets

Covered: belief, timeline (temporal/sequential), identity, expression **Missing:** learning, proof, mechanism, shape (limited)

50.41. E.18: Visual & Low-Code

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Scratch	□	·	·	·	·	□	·	□	Visual blocks, educational
Blockly	□	·	·	·	·	□	·	□	Google visual blocks
Node-RED	□	·	□	·	□	□	·	□	Flow-based, IoT
Retool	□	·	·	□	□	□	·	□	Internal tools, data-binding

Covered: belief, shape (visual structure), expression **Missing:** learning, proof, timeline (limited), identity (limited)

50.42. E.19: Systems & Bare Metal

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
C	□	·	·	□	□	□	·	□	The kernel. FFI substrate.
Assembly	□	·	·	□	□	□	·	□	Machine truth. No abstraction.

Language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Notes
Zig	□	□	·	□	□	□	·	□	Comptime, no hidden control flow
Rust	□	□	·	□	□	□	·	□	Ownership, lifetimes, safety

Covered: belief, identity, mechanism, shape, expression, some proof. The kernel itself. Systems languages implement governance at the hardware level. **Missing:** learning. Even C does not learn. The kernel executes. Governance learns.

50.43. E.20: The Master Matrix

All 20 families against all 8 governance questions — what each answers and what it lacks. Column order is for presentation only and does not reflect kernel internals.

#	Family	Questions Addressed (of 8)	Status
1	Imperative & OOP	5 of 8	CLOSED by learning
2	Functional	6 of 8	CLOSED by learning
3	Type Systems	6 of 8	CLOSED by learning
4	Concurrent & Actor	6 of 8	CLOSED by learning
5	Logic & Constraint	4 of 8	CLOSED by learning
6	Reactive & Dataflow	5 of 8	CLOSED by learning
7	Concatenative	3 of 8	CLOSED by learning
8	Array & Numeric	5 of 8	CLOSED by learning
9	Metaprogramming	5 of 8	CLOSED by learning
10	Smart Contracts	7 of 8	CLOSED by learning
11	Proof Assistants	6 of 8	CLOSED by learning
12	Probabilistic	6 of 8	CLOSED by learning
13	GPU & Parallel	5 of 8	CLOSED by learning
14	DSLs & Config	5 of 8	CLOSED by learning
15	Markup & Styling	3 of 8	CLOSED by learning
16	Query & Data	6 of 8	CLOSED by learning
17	Shell & Scripting	4 of 8	CLOSED by learning
18	Visual & Low-Code	3 of 8	CLOSED by learning
19	Systems & Bare Metal	5 of 8	CLOSED by learning
20	**LEARNING**	**1 of 8 (learning itself)**	**THE CLOSURE**

Every family is CLOSED by learning.

Learning is the governance question no programming language answers alone. CANONIC governance provides it — and therefore closes against all of programming.

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

• • •

50.44. 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
```

50.45. Persona Resolution for BOOK Type

Field	Value
tone	narrative
audience	readers
voice	second-person
warmth	personal

50.46. Persona Resolution for SERVICE Type

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

50.47. 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

• • •

50.48. Appendix G: References

50.49. Whitepapers [W-XX]

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

50.50. Blogs [B-XX]

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

50.51. Governance Sources [G-XX]

Code	Source	Content
G-1	FOUNDATION/LANGUAGE.md	LANGUAGE spec, TRIAD, naming
G-2	MAGIC/DESIGN.md	Tier algebra, eight questions, 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	21 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

Code	Source	Content
G-22	FOUNDATION/PROGRAMMING/	Neofunctionalization, 4 runtimes, FFI

• • •

50.52. 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. Eight questions.
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 governance question.

• • •

50.53. 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. Writing this manual was itself governed work — validated, scored, ledgered.

```
Scope:     hadleylab-canonic/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.

• • •

50.54. References

1. **[I-1]** Author CV.
2. **[G-3]** MAGIC/CANON.md.
3. **[W-7]** Governance as Compilation.
4. **[B-10]** The Compiler Insight.
5. **[B-5]** Three Files One Truth.
6. **[G-2]** MAGIC/DESIGN.md.
7. **[G-1]** FOUNDATION/LANGUAGE.md.
8. **[B-6]** Inherits: The Trust Chain.
9. **[W-3]** Evolutionary Phylogenetics of CANONIC.
10. **[W-1]** Code Evolution Theory.
11. **[B-4]** Your First 255.
12. **[W-8]** Economics of Governed Work.
13. **[W-12]** HITRUST r2 Governance Cost Case Study — 19 weeks elapsed, $85K staff cost.
14. **[G-7]** MAGIC/TOOLCHAIN/TOOLCHAIN.md.
15. **[G-4]** MAGIC/SERVICES/CANON.md.
16. **[B-13]** One User, 19 Organizations.
17. **[B-7]** SHOP: Your Work for Sale.
18. **[G-5]** MAGIC/GALAXY/CANON.md.

19. **[G-6]** MAGIC/COMPLIANCE/CERTIFICATION/CANON.md.
20. **[G-11]** MAGIC/SERVICES/LEARNING/CANON.md.
21. **[G-12]** MAGIC/SERVICES/TALK/CANON.md.
22. **[B-3]** COIN = WORK.
23. **[G-8]** MAGIC/SURFACE/SURFACE.md.
24. **[G-9]** MAGIC/SURFACE/DESIGN/CANON.md.
25. **[G-10]** MAGIC/SURFACE/JEKYLL/DESIGN.md.
26. **[G-13]** MAGIC/TOOLCHAIN/RUNTIME/RUNTIME.md.
27. **[W-2]** The Neutral Theory of CANONIC Evolution.
28. **[G-22]** FOUNDATION/PROGRAMMING/.
29. **[W-14]** Clinical AI Deployment Case Study — PyTorch 2.1 via TorchServe.
30. **[W-15]** Concordance Rate Case Study — 96.2% inter-reader agreement CT abdomen/pelvis.