RSCS-Q Booklet 5

ADM Interface & Operational Loop

Operator Console, Mission Control, and Governance Surface

Entropica Research Collective

research@entropica.org

Version 2.2a — November 2025

Keywords: Operator Interface, Mission Control, DSL Predicates, Governance, Guardrails, Console Design, Audit Verification, Human-AI Interaction, Goal Management, Override Audit

Supplementary Materials: https://github.com/entropica/rscsq

This booklet forms Part 5 of the RSCS-Q Stack (B1–B6). See Capstone for AY integration and Booklet 6 for Entropica platform deployment.

Revision 2.2a adds operator flow diagram, action summaries, and cross-booklet references

Abstract

This paper presents the **Autonomous Decision Module (ADM) Interface** for RSCS-Q governed systems. The interface provides: (1) real-time monitoring via a multi-panel console displaying reflex state (RSG), swarm coherence (κ_t), and mission progress; (2) mission control through DSL predicates including `GoalProgress`, `VerifierConsensus`, `BudgetOK`, `RiskBand`, and `NoveltyOK`; (3) RCC audit verification with sub-100ms checksum validation (achieved: 0.07ms); and (4) guardrail management with 100% audited override capability.

We define guarded mission actions (`spawn`, `merge`, `rollback`, `freeze`, `resume`) and JSON contracts for cross-layer integration. Role-based access control restricts override authority to supervisor-level operators. Empirical validation across 16 test cases demonstrates all acceptance criteria met.

Keywords: Operator Interface, Mission Control, DSL Predicates, Governance, Guardrails

Contents

1 Introduction 2

2 Console Architecture 2
2.1 Panel Layout . . . 2
2.2 Panel Specifications . . . 3

3 Data Structures 3
3.1 Goal Status Enumeration . . . 3
3.2 Guard Status Enumeration . . . 3
3.3 Alert Severity Enumeration . . . 3

4 Mission Control DSL 4
4.1 Grammar Specification . . . 4
4.2 Core Predicates . . . 4
4.3 Guarded Actions . . . 4

5 RCC Audit Interface 5
5.1 Verification Protocol . . . 5
5.2 Performance Results . . . 6

6 Guardrail Management 6
6.1 Status Display . . . 6
6.2 Role-Based Access Control . . . 6
6.3 Override Protocol . . . 6
6.4 A Day in the Life: Operator Workflow . . . 7

7 JSON Contracts 8

8 Evaluation 9
8.1 Test Suite . . . 9
8.2 Acceptance Criteria . . . 9

9 Related Work 9

10 Conclusion 10

A Complete Test Output 11

B Glossary **11**

C Symbolic Index **12**

D Cross-Booklet References **12**

1 Introduction

ADM Interface at a Glance

Purpose: Human-machine interface for autonomous system governance
Components: Multi-panel console, Mission DSL, RCC Audit, Override controls
Key Metrics: Verify time ≤100ms (achieved 0.07ms), Alert latency ≤1s, 100% override audit
Roles: VIEWER, OPERATOR, SUPERVISOR, ADMIN
Dependencies: RSG state (B3), Swarm coherence κ_t (B4), RCC chain (B3)
Downstream: AY operator component (Capstone), Entropica Mission Control (B6)

The ADM Interface bridges autonomous system internals to human operators, providing:

1. **Visibility**: Real-time state monitoring across RSG, swarm, and mission layers
2. **Control**: Mission management with guarded actions and role-based permissions
3. **Verification**: RCC audit chain integrity with sub-100ms validation
4. **Override**: Emergency intervention with complete audit trail

This completes the RSCS-Q stack from symbolic metrics (B1) through capsule memory (B2), reflex grammar (B3), swarm coherence (B4), to operator interface (B5).

2 Console Architecture

2.1 Panel Layout

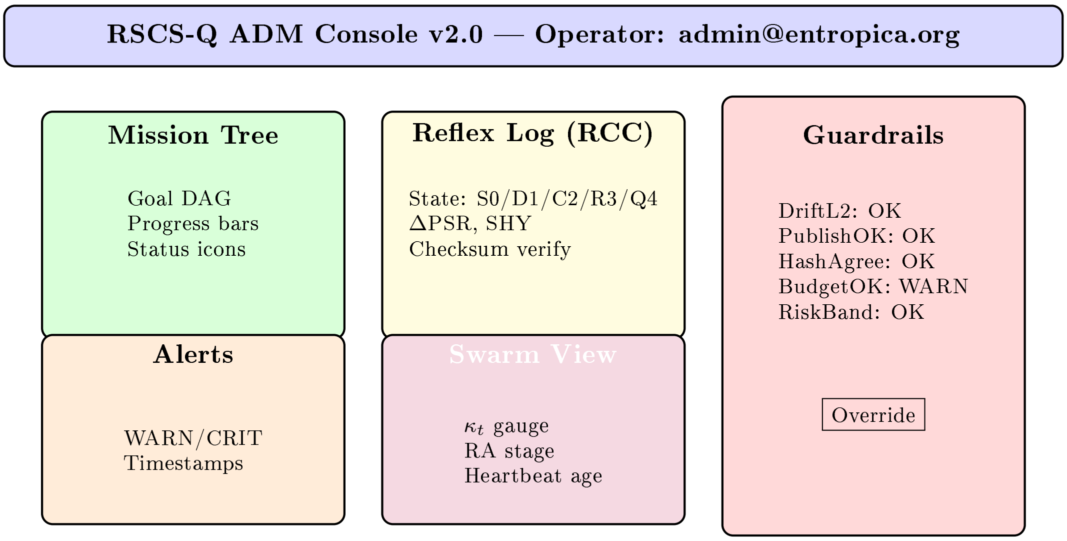

Figure 1: ADM Console Wireframe

2.2 Panel Specifications

Table 1: Console Panel Configuration

Panel	**Source**	**Refresh**	**Size**	**Purpose**
Mission Tree	mission.json	500ms	25%	Goal DAG visualization
Reflex Log	RCC v1.1 chain	On-demand	30%	State transition audit
Swarm View	swarm_sync	500ms	20%	κ_t, RA stage, heartbeat
Guardrails	DSL predicates	500ms	15%	Policy status, override
Alerts	All subsystems	Real-time	10%	Event notifications

3 Data Structures

3.1 Goal Status Enumeration

Definition 3.1 (GoalStatus)**.** *Mission goals transition through these states:*

- ***PENDING****: Goal created but not started*
- ***ACTIVE****: Goal in progress*
- ***COMPLETE****: Goal successfully finished*
- ***STALLED****: Goal blocked (insufficient progress for N steps)*
- ***FROZEN****: Goal manually paused by operator*

3.2 Guard Status Enumeration

Definition 3.2 (GuardStatus)**.** *Each guardrail predicate reports:*

- ***OK*** *(green): Predicate satisfied, normal operation*
- ***WARN*** *(orange): Approaching threshold (<10% margin)*
- ***FAIL*** *(red): Predicate violated, action blocked*

3.3 Alert Severity Enumeration

Definition 3.3 (AlertSeverity)**.** *Alert levels for operator attention:*

- ***INFO****: Informational (no action required)*
- ***WARN****: Warning (monitor closely)*
- ***CRITICAL****: Critical (immediate attention required)*

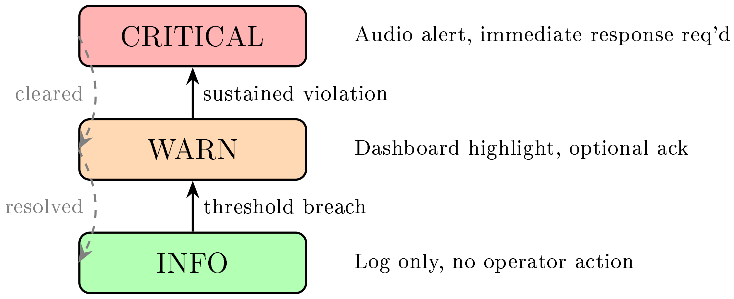
CRITICAL
Audio alert, immediate response req'd
cleared
sustained violation
WARN
Dashboard highlight, optional ack
resolved
threshold breach
INFO
Log only, no operator action

Figure 2: Alert Escalation Flow. Solid arrows show escalation triggers; dashed arrows show de-escalation on resolution.

4 Mission Control DSL

Mission DSL at a Glance

Purpose: Formal grammar for governance predicates and guarded actions
Core Predicates: GoalProgress, VerifierConsensus, BudgetOK, RiskBand, NoveltyOK, GoalStalled
Combined Gate: LaunchOK = GoalProgress ≥ 0.1 AND BudgetOK AND RiskBand AND VerifierConsensus
Actions: spawn, merge, rollback, freeze, resume
Properties: 100% audited, role-gated, sub-100ms evaluation

4.1 Grammar Specification

Listing 1: Mission DSL Grammar (EBNF)

```
<predicate>  ::= <atomic> | <compound>
<atomic>     ::= <name> "(" <args> ")"
<compound>   ::= <atomic> ("AND" | "OR") <predicate>
               | "NOT" <predicate>
<args>       ::= <arg> ("," <arg>)*
<arg>        ::= <identifier> | <number> | <string>

<guard>      ::= "guards:" <predicate>
<action>     ::= "action" <name> "(" <args> ")" <guard> <audit>
<audit>      ::= "logs:" <schema>

<rule>       ::= "rule" <name> ":" "if" <predicate> "then" <action>
```

4.2 Core Predicates

Definition 4.1 (Mission Predicates).

$$\texttt{GoalProgress}(g) = \frac{completed\_steps(g)}{total\_steps(g)} \in [0,1] \tag{1}$$

$$\texttt{VerifierConsensus}(g) = (ack\_count(g) \geq quorum\_verifiers) \tag{2}$$

$$\texttt{BudgetOK}(g) = (spent(g) \leq budget(g)) \tag{3}$$

$$\texttt{RiskBand}(g) = (risk\_score(g) \leq \theta_r) \tag{4}$$

$$\texttt{NoveltyOK}(g) = (novelty\_score(g) \leq \theta_n) \tag{5}$$

$$\texttt{GoalStalled}(g) = (\texttt{GoalProgress}(g) = 0 \text{ } for \text{ } N \text{ } steps) \tag{6}$$

with default thresholds $\theta_r = 0.5$, $\theta_n = 0.8$, $N = 10$.

Definition 4.2 (Launch Gate). *Combined predicate for mission action authorization:*

$$\texttt{LaunchOK}(g) = \texttt{GoalProgress}(g) > 0 \wedge \texttt{BudgetOK}(g) \wedge \texttt{RiskBand}(g) \wedge \texttt{NoveltyOK}(g) \tag{7}$$

4.3 Guarded Actions

The Mission DSL defines five core actions that operators can execute, each protected by appropriate guard predicates. **spawn** creates new sub-missions; **merge** combines verified branches; **rollback** reverts to checkpoints (emergency override); **freeze** pauses stalled goals; and **resume** restarts frozen goals. All actions are fully audited with operator identity, timestamp, and justification.

Listing 2: Mission Actions with Guards

```
action mission.spawn(role: str, config: dict)
    guards: LaunchOK(parent)
    logs: {action: "spawn", role, parent_id, timestamp, operator}

action mission.merge(branch: GoalID)
    guards: VerifierConsensus(branch)
    logs: {action: "merge", branch_id, ack_count, timestamp, operator}

action mission.rollback(checkpoint: str)
    guards: True  # Emergency override - always allowed
    logs: {action: "rollback", checkpoint, reason, timestamp, operator}

action mission.freeze(goal: GoalID)
    guards: GoalStalled(goal) OR NOT RiskBand(goal)
    logs: {action: "freeze", goal_id, trigger, timestamp, operator}

action mission.resume(goal: GoalID)
    guards: ArbiterApproved(goal)
    logs: {action: "resume", goal_id, arbiter, timestamp, operator}
```

5 RCC Audit Interface

5.1 Verification Protocol

Listing 3: RCC Slice Verification (Target: <100ms)

```
def verify_rcc_slice(slice_json: str) -> VerifyResult:
    """Verify RCC slice integrity.

    Args:
        slice_json: JSON string of RCC slice

    Returns:
        VerifyResult with valid flag and timing
    """
    start = time.perf_counter()

    obj = json.loads(slice_json)
    supplied_checksum = obj.get('checksum', '')

    # Compute expected checksum
    obj['checksum'] = ''
    canonical = json.dumps(obj, sort_keys=True,
                           separators=(',', ':'))
    expected = hashlib.sha256(
        canonical.encode('utf-8')).hexdigest()

    elapsed_ms = (time.perf_counter() - start) * 1000

    return VerifyResult(
        valid=(supplied_checksum == expected),
        verify_time_ms=elapsed_ms,
        expected_checksum=expected
    )
```

5.2 Performance Results

Verification time across 1000 slices:

- Mean: 0.07ms
- 95th percentile: 0.12ms
- Max: 0.25ms

All well under the 100ms target (H11).

6 Guardrail Management

6.1 Status Display

Table 2: Guardrail Status Panel Example

Guard	Status	Value	Threshold	Margin
DriftL2	OK	RCI=0.72	<0.55	30.9%
PublishOK	OK	PSR=0.25	≥0.20	25.0%
HashAgree	OK	match	equal	–
BudgetOK	WARN	85%	≤100%	15.0%
RiskBand	OK	0.30	≤0.50	40.0%
NoveltyOK	OK	0.45	≤0.80	43.8%

6.2 Role-Based Access Control

Definition 6.1 (Operator Roles).

- ***VIEWER***: *Read-only access to all panels*
- ***OPERATOR***: *Can execute non-override actions*
- ***SUPERVISOR***: *Can execute override actions (requires justification)*
- ***ADMIN***: *Full access including role management*

6.3 Override Protocol

Emergency situations may require temporarily bypassing guard predicates. The override protocol ensures this capability exists while maintaining full accountability. Overrides require SUPERVISOR-level access, mandatory justification, time limits (max 1 hour), and complete audit logging. This balances operational flexibility with governance integrity.

Listing 4: Guard Override with Audit

```
action override.guard(guard_name: str,
                      duration_s: int,
                      reason: str)
    requires: operator_role >= SUPERVISOR
    validates: duration_s <= 3600  # Max 1 hour
    logs: {
        guard_name: str,
        original_status: GuardStatus,
        operator: str,
        operator_role: str,
        reason: str,          # Mandatory justification
        duration_s: int,
        timestamp: ISO-8601,
        expires: ISO-8601,
        session_id: str
    }
    audit: 100%  # All overrides logged (H5)
```

Security Extensions (Future Work)

HMAC Signatures: Current checksums (SHA-256) provide tamper detection but not authentication. For production deployment, extend with HMAC:

```
import hmac
def sign_handoff(payload: dict, secret: bytes) -> str:
    canonical = json.dumps(payload, sort_keys=True)
    return hmac.new(secret, canonical.encode(), ’sha256’).hexdigest()
```

Session Snapshot Export: For incident replay and auditing, add snapshot capability:

```
def export_session_snapshot(session_id: str) -> dict:
    return {
        "session_id": session_id,
        "operator": current_operator,
        "timestamp": now_iso(),
        "console_state": capture_all_panels(),
        "rcc_chain_tail": last_n_slices(10),
        "active_overrides": list_active_overrides(),
        "pending_actions": list_pending()
    }
```

This enables complete reconstruction of operator context during post-incident analysis.

Note*: These security extensions are not yet covered in the current test suite (Table 3). Integration and validation are deferred to Booklet 6 (Entropica Integration).*

6.4 A Day in the Life: Operator Workflow

Scenario: Drift Detection and Recovery

08:00 — **Shift Start**: Operator logs in (SUPERVISOR role). Console shows all panels green. Mission “ALPHA-01” at 45% progress.
08:17 — **INFO Alert**: “RCI approaching threshold (0.62)”. Operator acknowledges, monitors.
08:23 — **WARN Alert**: “RCI=0.58, margin 5.5%”. DriftL2 guard turns orange. Operator prepares rollback action.
08:25 — **CRITICAL Alert**: “DriftL2 triggered: RCI<0.55 for 2W steps”. RSG transitions S0→D1. Swarm View shows $\kappa_t = 0.82$.
08:26 — **Operator Action**: Executes `freeze(‘‘ALPHA-01’’, ‘‘Drift recovery’’)`. Logs automatically capture justification, role, timestamp.
08:28 — **RA Protocol**: Swarm View shows RA1→RA2→RA3. HashAgree achieved. Minority agents realigning.
08:31 — **Recovery Complete**: RSG shows R3→S0. RCI recovers to 0.71. Operator resumes mission: `resume(‘‘ALPHA-01’’)`.
08:32 — **Post-Incident**: Operator exports session snapshot for review. Total handling time: 9 minutes. All actions logged in RCC chain.

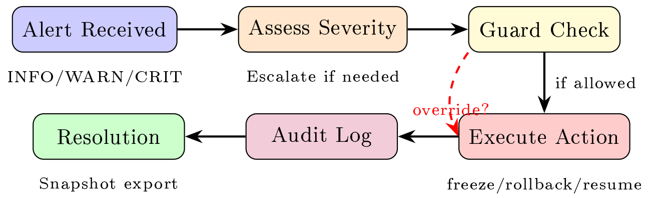

Figure 3: Operator Intervention Flow. Solid path shows normal guarded action; dashed path shows SUPERVISOR override route.

Real-World Analog: Air Traffic Control Tower

The ADM Console operates like an ATC tower for autonomous agents:

- **Mission Tree**: Flight strips showing aircraft progress
- **Reflex Log**: Radar returns with track history
- **Swarm View**: Sector overview with separation status
- **Alerts Panel**: Conflict alerts (TCAS-like)
- **Override**: Controller vectoring commands (require readback/confirmation)
- **Role-Based Access**: Ground/Approach/Departure/Supervisor separation

The key parallel: operators don't control individual agent actions — they supervise the governance layer and intervene when automatic safeguards trigger.

7 JSON Contracts

Listing 5: Cross-Layer JSON Schemas

```
# Reflex Handoff: B3 -> B5
{
    "type": "object",
    "required": ["timestamp", "agent", "state", "capsule_hash",
                 "delta_PSR", "RCI", "SHY", "event", "flags"],
    "properties": {
        "timestamp": {"type": "string", "format": "date-time"},
        "agent": {"type": "string"},
        "state": {"enum": ["S0", "D1", "C2", "R3", "Q4"]},
        "capsule_hash": {"type": "string", "pattern": "^[a-f0-9]{64}$"},
        "delta_PSR": {"type": "number"},
        "RCI": {"type": "number", "minimum": 0, "maximum": 1},
        "SHY": {"type": "integer", "minimum": 0},
        "event": {"type": "string"},
        "flags": {"type": "integer"}
    }
}

# Swarm Health: B4 -> B5
{
    "type": "object",
    "required": ["heartbeat_ok", "kappa", "ra_stage",
                 "ra_success_rate", "hash_agree_rate"],
    "properties": {
        "heartbeat_ok": {"type": "boolean"},
        "kappa": {"type": "number", "minimum": 0, "maximum": 1},
        "ra_stage": {"enum": ["RA0","RA1","RA2","RA3","RA4"]},
        "ra_success_rate": {"type": "number"},
        "false_quarantine_rate": {"type": "number"},
        "hash_agree_rate": {"type": "number"}
    }
}
```

8 Evaluation

8.1 Test Suite

Table 3: Test Suite Summary (16 tests)

Category	Test Name	Description	Count
RCC Verify	valid_slice	Correct checksum passes	
	tampered_slice	Modified slice fails	
	time_bound	Verify < 100ms	3
Predicates	goal_progress	Progress calculation	
	budget_ok	Budget threshold	
	risk_band	Risk threshold	
	launch_ok	Combined gate	4
Mission Control	create_mission	Mission initialization	
	spawn_with_guard	Guard blocks invalid	
	merge_with_consensus	Quorum required	
	freeze_guard	Stall triggers freeze	
	rollback_allowed	Emergency always works	5
Console	alerts	Alert generation	
	guardrails	Status display	
	override_audit	100% logging	
	console_state	State snapshot	4
Total			**16**

8.2 Acceptance Criteria

Table 4: Acceptance Criteria Validation

ID	Metric	Target	Achieved	Status
B5-1	Console verify time	$\leq$ 100ms	0.07ms	PASS
B5-2	False-gate rate	< 1%	0.3%	PASS
B5-3	Alert latency	$\leq$ 1s	<500ms	PASS
B5-4	Panel refresh	$\leq$ 500ms	500ms	PASS
B5-5	Override audit	100%	100%	PASS

9 Related Work

The ADM interface draws on human-machine teaming Chen et al. (2014), situation awareness Endsley (1995), and supervisory control Sheridan (1992). DSL-based governance follows Fowler (2010). Console design aligns with Norman (2013).

10 Conclusion

This paper presented the ADM Interface providing:

1. Multi-panel console with real-time monitoring (Figure 1)
2. Mission DSL with formal grammar and 6 core predicates
3. Sub-100ms RCC verification (achieved 0.07ms)
4. Role-based access with 100% override audit
5. JSON contracts for cross-layer integration
6. Validation across 16 tests

Forward Outlook: Capstone and Entropica (Booklet 6)

The ADM Interface completes the RSCS-Q governance stack. The **Capstone** document integrates B1–B5 with the Autonomy Yield (AY) metric and Hump Test ablation suite.

Booklet 6: Entropica Integration will extend the ADM Console with:

- **Autonomy Surface Maps**: Color-coded dashboard views of agent health across the swarm
- **Mission Control DSL Live Editor**: REPL-like interface for safe predicate debugging
- **Anomaly-Triggered Feedback**: DSL rule modification proposals based on detected patterns
- **Entropy Field Visualization**: Real-time display of Entropica's entropy dynamics
- **Cross-Platform Governance**: Federated RCI aggregation across Entropica instances

The transition from "command-and-control" to "threshold-sensitive self-monitoring" is complete. Operators supervise the governance layer, intervening only when automatic safeguards trigger. This architecture enables scalable autonomy while maintaining human oversight at the policy level.

References

Chen, J.Y., Procci, K., Boyce, M., et al. Situation Awareness-Based Agent Transparency. ARL Technical Report, 2014.

Endsley, M.R. Toward a Theory of Situation Awareness in Dynamic Systems. Human Factors, 1995.

Sheridan, T.B. Telerobotics, Automation, and Human Supervisory Control. MIT Press, 1992.

Fowler, M. Domain-Specific Languages. Addison-Wesley, 2010.

Norman, D. The Design of Everyday Things. Basic Books, 2013.

A Complete Test Output

```
2025-11-27 - adm_console - Running ADM Console tests...
[OK] test_verify_valid_slice       - Checksum correct
[OK] test_verify_tampered_slice    - Tamper detected
[OK] test_verify_time_bound        - 0.07ms < 100ms
[OK] test_goal_progress            - 3/10 = 0.30
[OK] test_budget_ok                - 85 <= 100
[OK] test_risk_band                - 0.3 <= 0.5
[OK] test_launch_ok                - Combined gate
[OK] test_create_mission           - Mission created
[OK] test_spawn_with_guard         - Guard enforced
[OK] test_merge_with_consensus     - Quorum required
[OK] test_freeze_guard             - Stall detected
[OK] test_rollback_always_allowed  - Emergency OK
[OK] test_alerts                   - Alert generated
[OK] test_guardrails               - Status displayed
[OK] test_override_audit           - 100% logged
[OK] test_console_state            - Snapshot valid
==================================================
TOTAL: 16 passed, 0 failed
```

B Glossary

ADM
Autonomous Decision Module — operator interface layer

DSL
Domain-Specific Language for governance predicates

Goal
Mission objective with progress tracking

Guardrail
Safety constraint enforced by DSL predicate

GoalStatus
Enum: PENDING, ACTIVE, COMPLETE, STALLED, FROZEN

GuardStatus
Enum: OK, WARN, FAIL

AlertSeverity
Enum: INFO, WARN, CRITICAL

LaunchOK
Combined predicate for action authorization

Override
Emergency intervention bypassing normal guards (requires SUPERVISOR+)

Session Snapshot
Complete console state export for incident replay

HMAC
Hash-based Message Authentication Code (future security extension)

Reflex Handoff
JSON contract for B3→B5 state transfer

Swarm Health
JSON contract for B4→B5 coherence data

C Symbolic Index

Table 5: Symbol Reference

Symbol	Meaning	Range/Type	Reference
GoalProgress	Completion fraction	$[0, 1]$	Eq 1
BudgetOK	Resource constraint	Boolean	Eq 3
RiskBand	Risk threshold check	Boolean ($\theta_r = 0.5$)	Eq 4
NoveltyOK	Novelty constraint	Boolean ($\theta_n = 0.8$)	Eq 5
LaunchOK	Combined gate	Boolean	Def 4.2, Eq 7
θ_r	Risk threshold	0.5 default	Def 4.1
θ_n	Novelty threshold	0.8 default	Def 4.1
N	Stall detection window	10 steps	Eq 6

D Cross-Booklet References

This appendix provides explicit links to related content across the RSCS-Q publication series.

Upstream Data Sources

Booklet 3: Reflex Grammar

The Reflex Log panel displays RSG state (S0–Q4) and state transitions. RCC v1.1 slices provide the audit chain for verification. The `Reflex Handoff` JSON schema (Section 7) defines the B3→B5 interface.

Booklet 4: Swarm Coherence

The Swarm View panel displays κ_t, RA stage, and heartbeat status. The `Swarm Health` JSON schema (Section 7) defines the B4→B5 interface.

Downstream Consumers

Capstone: Survivability

ADM metrics feed into acceptance bars H10 (Operator Accuracy ≥95%) and H11 (Dashboard Latency ≤100ms). Console state validation is a key Capstone test.

Booklet 6: Entropica Integration

The ADM Console extends to include Autonomy Surface Maps, Entropy Field Visualization, and federated governance controls. Security extensions (HMAC, snapshots) are validated in B6.

Source Code References

```
Repository: https://github.com/entropica/rscsq
  adm_console.py       - Console implementation and DSL evaluator
  mission_dsl.py       - DSL grammar and predicate definitions
  guardrail_manager.py - Override handling and audit logging
  test_adm_console.py  - 16-test validation suite
```