This is the final polish. The resulting architecture is now robust against drift, immutability bugs, and schema confusion.

Here are the finalized documents. Hand these to the ck3raven-dev agent.

Document 1: The Architecture Specification

Save as: docs/arch/CANONICAL_REPLY_SYSTEM.md

Canonical Architecture: Centralized Reply System

> **Status:** APPROVED STANDARD
>
> **Target:** Global (MCP, Core Library, Policy)
>
> **Goal:** Eliminate "Context Poisoning" and "Silent Failures" via deterministic, registry-based replies.

1. Core Philosophy: "Replies, Not Errors"

We replace ad-hoc Python exceptions with a structured **Reply System**.

- **Exceptions** are for control flow *within* a function.
- **Replies** are for communication *across* boundaries (Tool $\to$ Agent).

The Razor-Sharp Separation of Concerns

We strictly define who owns which outcome to prevent logic leakage.

Component	Responsibility	Allowed Replies	Forbidden Replies
WorldAdapter	**Physics.** "Does this path exist in the lens?"	S, I	D (Denied)
Enforcement	**Governance.** "Is this permitted by contract?"	S, D	I (Invalid)

Infrastructure	**Transport.** "Did the system crash?"	E	D (Denied)

2. The Reply Taxonomy

The reply_type discriminator determines the Agent's **Next Action**.

Type	**Name**	**Owner**	**Semantics**	**Agent Reaction**
S	**Success**	All	Operation completed.	**Continue.**
I	**Invalid**	WorldAdapter / Tool Boundary	Request cannot be executed as stated (bad path, bad input, schema violation).	**Self-Correct:** Fix input format/path.
D	**Denied**	Enforcement	Valid request rejected by Policy/Governance.	**Escalate:** Request Token or Change Scope.
E	**Error**	Infrastructure	System failure, unhandled exception, timeout.	**Stop:** Report trace_id. Do not retry.

3. The Registry Contract (Anti-Drift)

To prevent message drift, we use a **Single Source of Truth**. Replies are defined by a code and a message_key.

Compatibility Contract: Agents must branch on reply_type and code. **NEVER**

branch on message text.

3.1 The Schema

Python

```python
@dataclass(frozen=True)
class Reply:
    reply_type: Literal["S", "I", "D", "E"]
    code: str                 # Canonical ID (e.g., WA-RES-I-001)
    message_key: str          # Registry key (e.g., PATH_NOT_FOUND)
    params: Dict[str, Any]      # Data for template rendering
    data: Optional[Dict] = None # Machine-readable payload (decisions, diffs)
    trace_id: Optional[str] = None

    @property
    def message(self) -> str:
        # Rendered from Registry (Convenience Only)
        return REGISTRY[self.code].format(**self.params)
```

3.2 Canonical Code Format

Format: LAYER-AREA-TYPE-NNN

- **Layer:** WA (World), EN (Enforcement), MCP (Transport), CT (Contract).
- **Area:** RES (Resolution), IO, READ, WRITE, LINT, GATE, SYS.
- **Type:** S, I, D, E.

Registry Examples:

- WA-RES-I-001: "Path '{path}' does not exist."
- EN-WRITE-D-002: "Write denied to '{path}'. Outside Contract Scope."
- MCP-SYS-S-900: "Legacy tool returned raw payload; wrapped as Success."

---

4. Implementation Architecture: The Safety Wrapper

The wrapper is a **pure safety net**. It guarantees the transport contract and immutability.

Responsibilities:

1. **Trace ID:** Generate unique ID for every request.

2. **Immutability:** Never mutate the result; return a new instance with the trace ID.
3. **Auto-Wrap:** If tool returns legacy data, wrap as Reply(S) using code MCP-SYS-S-900.
4. **Catch-All:** Catch Exception $\to$ Log Stack $\to$ Return Reply(E).

Pseudo-Code:

Python

```python
def mcp_safe_tool(func):
    def wrapper(*args, **kwargs):
        trace_id = generate_uuid()
        try:
            result = func(*args, **kwargs)

            # Legacy Support (Phase 1 Only)
            if not isinstance(result, Reply):
                return Reply(
                    reply_type="S",
                    code="MCP-SYS-S-900",
                    message_key="LEGACY_WRAP",
                    params={},
                    data=result,
                    trace_id=trace_id
                )

            # Immutability: Return new instance with trace_id
            return dataclasses.replace(result, trace_id=trace_id)

        except Exception as e:
            # 1. Capture Stack Trace to Disk
            log_exception_to_disk(trace_id, e)
            # 2. Return Clean 'E' Reply
            return Reply(
                reply_type="E",
                code="MCP-SYS-E-001",
                message_key="SYS_CRASH",
                params={"err": str(e)},
                trace_id=trace_id
            )
    return wrapper
```

Document 2: The Implementation Plan

Save as: docs/plans/REPLY_SYSTEM_IMPLEMENTATION.md

Canonical Reply System Implementation Plan

> **Status:** APPROVED
>
> **Goal:** Secure the transport immediately, then refactor core logic.

Phase 1: Infrastructure & Safety Net (Immediate)

Objective: Stop crashes and "silent" failures. Introduce the Registry.

1. **Create Core Infrastructure:**
   - src/ck3raven/core/replies.py: Define Reply dataclass (frozen).
   - src/ck3raven/core/reply_registry.py: Define REGISTRY dict.
     - *Must include:* MCP-SYS-S-900 ("Legacy tool returned raw payload").
2. **Implement Safety Wrapper:**
   - tools/ck3lens_mcp/safety.py: Implement @mcp_safe_tool.
   - *Logic:* Use dataclasses.replace for trace ID injection (Immutability).
3. **Wrap Tool Entrypoints:**
   - Decorate every function in tools/ck3lens_mcp/server.py.
   - *Verification:* Verify existing tools still work (wrapped with MCP-SYS-S-900).

Phase 2: Domain Adoption (The Refactor)

Objective: Replace ad-hoc exceptions with semantic I and D replies.

1. **Refactor Enforcement (The Guard):**
   - Target: src/ck3raven/policy/Enforcement.py.
   - Action: Change check_write_permission to return Reply(D) instead of raising PermissionError.
   - *Integration:* Update server.py to check if result.reply_type == 'D': return result.
2. **Refactor WorldAdapter (The Lens):**
   - Target: src/ck3raven/core/WorldAdapter.py.
   - Action: Return Reply(I) for missing paths.

    - *Constraint:* WorldAdapter **NEVER** returns D.

Phase 3: Strict Mode (Cleanup)

Objective: Enforce the standard.

1. **Disable Auto-Wrap:**
    - Update @mcp_safe_tool to return Reply(E) if the underlying tool returns a raw dict.
    - *Prerequisite:* All tools must be migrated to return Reply objects.
2. **Audit:**
    - Ensure no code branches on reply.message. All logic must use reply.code or reply.reply_type.