The Complete KIMERA SWM Documentation Suite

This suite is designed in three tiers: Foundational (the "Why"), Architectural (the "What"), and Operational (the "How-To").

Part 1: Foundational & Conceptual Tier (The "Why")

These documents establish the core philosophy and conceptual framework. They are essential for understanding the system's purpose and guiding principles.

- **DOC-001: The Spherical Word Methodology (SWM) - Complete Documentation**
    - **Purpose:** To serve as the definitive guide to the SWM philosophy and methodology. It details the zetetic mindset, the multi-dimensional nature of Geoids, the process of abstraction, resonance, and interpretation, and the core "1+3+1" heuristic.
    - **Audience:** System Architects, Philosophers, Cognitive Scientists, and advanced users seeking to understand the "soul" of the machine.
- **DOC-002: KIMERA SWM - System Vision & Ontology**
    - **Purpose:** To articulate the high-level vision of Kimera SWM as a "contradiction-driven semantic reactor." This document explains the core ontology—the roles of Geoids, Scars, Echoes, Voids—and the fundamental goal of achieving emergent coherence through thermodynamic principles. It bridges the gap between the SWM philosophy and the system's engineering intent.
    - **Audience:** All stakeholders, including project leads, architects, and new developers.

Part 2: Architectural & Engineering Tier (The "What" and "How")

These are the primary engineering blueprints. They provide the detailed specifications for every component and mechanism.

- **DOC-101: KIMERA SWM - Unified System Architecture**
    - **Purpose:** The master architectural document. It defines the four primary layers (Substrate, Memory Core, Processing Engines, Interface/Governance) and illustrates the high-level data and control flow of the Kimera Core Cognitive Loop (KCCL). It shows how all subsystems interconnect.
    - **Audience:** System Architects, Lead Engineers.
- **DOC-201: Core Data Structures Specification**
    - **Purpose:** To provide a precise, unambiguous definition of the system's fundamental data structures. This includes the complete JSON schemas for `Geoid` (with its `semantic_state` and `symbolic_state`), the comprehensive `Scar`, the transient `Echo`, and any other core objects.
    - **Audience:** All Developers.

- **DOC-202: Semantic Thermodynamics - Formal Specification**
  - **Purpose:** A deep dive into the system's "physics." It details the mathematical models and algorithms for Semantic Energy (SE), the decay law (SE(t)), Semantic Temperature (T_sem), energy transfer during resonance (ΔSE), and the formal calculation of Semantic Entropy (S_semantic). It also specifies the $\Delta S \geq 0$ axiom and its enforcement mechanisms.
  - **Audience:** System Architects, Engineers specializing in core dynamics.
- **DOC-203: Input & Language Subsystems - Engineering Specification**
  - **Purpose:** To detail the entire input pipeline. This document covers:
    - **EcoForm Subsystem:** The formal grammar, data structures (Grammar Tree, Orthography Vector), and APIs for parsing raw input into structured linguistic units.
    - **Echoform Engine:** The specification for Echoform operators, including the catalog JSON schema, the interpret_echoform function, and the principle of closure.
  - **Audience:** NLP Specialists, System Developers.
- **DOC-204: The Vault Subsystem - Complete Engineering Specification**
  - **Purpose:** The definitive guide to the architecture and logic of the Vault, the specialized memory for Scars. It details the Dual Vault architecture, partitioning criteria, the Contradiction Drift Interpolator, the Recursive Vault Reflex Engine (RVRE), Vault Fracture Topology, and all associated parameters and thresholds.
  - **Audience:** Core System Developers, Database Engineers.
- **DOC-205: Dynamic Engines - Detailed Specifications**
  - **Purpose:** A parent document with detailed sub-specifications for each of the active processing engines within the KimeraKernel.
  - **Sub-Documents:**
    - **DOC-205a: Contradiction Engine & SPDE:** Details the algorithms for Tension Gradient Mapping, Pulse Calculation, Collapse vs. Surge logic, and how the Semantic Pressure Diffusion Engine (SPDE) propagates pressures across the semantic field.
    - **DOC-205b: Memory Scar Compression Engine (MSCE):** Specifies the algorithms and rules for Scar decay, fusion, and crystallization.
    - **DOC-205c: Provisional Geoid Generator (PGG):** Details the triggers and logic for creating, managing, and solidifying or pruning provisional geoids.
    - **DOC-205d: Axis Stability Monitor (ASM):** Defines the metrics (II, SLF) and mechanisms for monitoring and stabilizing cognitive axes.
    - **DOC-205e: Semantic Suspension Layer (SSL):** The emergency failsafe protocol, detailing triggers and stabilization actions (freeze, compress, re-anchor).
    - **DOC-205f: ZPA & Reflective Cortex (RCX):** Specifies the logic for the Zetetic Prompt API (prompt triggers, formulation, ZPS scoring) and the Reflective Cortex (mirror map generation, echo trail logic).

- **Audience:** Core Algorithm Developers, System Architects.

Part 3: Operational & Developer Tier (The "How-To")

These documents provide practical guidance for building, running, testing, and extending the system.

- **DOC-301: KIMERA SWM - API Reference & Integration Guide**
  - **Purpose:** To provide detailed documentation for all external-facing APIs, primarily those managed by the Interface Control Weave (ICW). This includes endpoint definitions, request/response formats, authentication methods, and usage examples.
  - **Audience:** Application Developers, System Integrators.
- **DOC-302: KIMERA SWM - Deployment, Configuration & Operations Manual**
  - **Purpose:** An operational guide for deploying and running a Kimera SWM instance. It includes hardware requirements, setup instructions, a guide to all tunable parameters (from Appendix 5.2), and procedures for monitoring system health and performance.
  - **Audience:** DevOps Engineers, System Administrators.
- **DOC-303: KIMERA SWM - Testing & Validation Framework**
  - **Purpose:** The master test plan. It details the strategy for verifying the system's correctness and stability. This includes:
    - **Unit Test Requirements** for each module.
    - **Integration Test Scenarios** (like the Phase 1, 2, 3 plans).
    - **Performance Benchmarking Protocols** and target KPIs.
    - **Simulation Campaign Designs** for testing emergent behaviors and tuning parameters.
  - **Audience:** Quality Assurance (QA) Engineers, Developers.
- **DOC-304: Developer's Guide & Contribution Protocol**
  - **Purpose:** To provide guidelines for engineers extending the Kimera SWM codebase. This includes coding standards, instructions for creating and validating new Echoforms, procedures for proposing new Field-Scoped Laws to the Law Registry, and the process for submitting and reviewing code changes.
  - **Audience:** New and Existing System Developers.

KIMERA SWM: The Unified System Reference

Version: 1.2
Date: June 7, 2025
Status: Canonical System Blueprint
Preamble: Document Purpose

This document constitutes the single source of truth for the KIMERA Semantic Working Memory (SWM) system. It synthesizes all foundational principles, architectural blueprints, subsystem specifications, and operational dynamics into one cohesive and comprehensive reference. It is designed for system architects, engineers, and developers, providing a complete and grounded guide for implementation, testing, and future evolution. All speculative commentary has been stripped; only the defined engineering reality of the system is detailed herein.

Part 1: Foundational Principles & Ontology

This tier establishes the core philosophy, governing laws, and conceptual framework of the KIMERA SWM system.

1.1 The Spherical Word Methodology (SWM) - System Philosophy

KIMERA's operational logic is an implementation of the Spherical Word Methodology (SWM), a framework for deep, multi-perspective analysis.

- **Core Principle**: Knowledge is not static or "flat." Any unit of knowledge is a **Geoid**: a dynamic, multi-dimensional entity with interwoven linguistic, cultural, structural, and symbolic layers.
- **Zetetic Mindset**: The system must operate with a zetetic (inquisitive, skeptical) stance. It does not seek to enforce a single truth but to explore the tensions between multiple perspectives. It embraces ambiguity and paradox as signals for deeper inquiry.
- **Multi-Perspectival Analysis**: A core operational principle is the exploration of Geoids through diverse cognitive "axes" (e.g., languages, symbolic systems). This is inspired by the **"1 Root Language + 3 Unrelated Languages + 1 Symbolic Meaning including Chaos"** heuristic, designed to deconstruct biases and reveal hidden structural patterns.

1.2 System Ontology: The Contradiction-Driven Semantic Reactor

KIMERA SWM is architected as a **contradiction-driven semantic reactor**. This defines its fundamental purpose and behavior.

- **Ontological Premise**: Intelligence is not stored as static data but **emerges from the continuous process of surfacing and resolving contradictions**.
- **System State**: The default state of the system is one of dynamic flux. Memory

is treated as an unstable, echo-driven, and mutation-prone thermodynamic system. Coherence is not a pre-enforced condition but an emergent property of the system's self-regulation.
- **Contradiction as Fuel**: Conflicts are the primary energy source that drives learning and evolution. The system is designed to seek out, manage, and learn from semantic and logical tensions rather than merely avoiding or eliminating them.

1.3 The Law Registry: System Governance

The system's behavior is constrained by a hierarchical set of immutable and scoped laws managed by the **Governance Firmware**. This provides a stable, predictable foundation for its complex dynamics.

- **Immutable Core Laws (Examples)**:
  - **L0 (Contradiction Mandate):** The system's primary mode of action is through contradiction-induced tension deformation.
  - **L1 (Semantic Neutrality):** No piece of information is inherently privileged. Its salience is determined by its history of reinforcement through contradiction resolution.
  - **L2 (Translation Delay Integrity):** Translation of a concept into a linguistic output cannot occur until its underlying contradiction has reached a stable collapse.
  - **L4 (Axis Triangulation Mandate):** All linguistic outputs must be grounded in the multi-perspectival 1+3+1 rule.
- **Field-Scoped Laws:** These are conditional rules that modulate the Core Laws in specific contexts (e.g., locking certain semantic layers in a financial analysis context).
- **Conflict Resolution:** A defined protocol resolves conflicts between laws, with Core Laws always taking precedence over Scoped Laws.

Part 2: Architectural & Engineering Tier (The "What" and "How")

This tier provides the master architectural blueprint and the precise definitions of the system's core data structures.

2.1 The Integrated KimeraKernel Architecture

The system is a self-contained, modular architecture centered around the KimeraKernel, organized into four primary layers.

- **Layer 1: Semantic Substrate & Knowledge Representation**: This is the foundational layer where knowledge exists. It includes:

    - GeoidManager: Manages the lifecycle of all Geoids.
    - EcoForm and Echoform Engines: Handle the parsing of input into structured representations and the transformation of Geoids.
    - LinguisticGeoidInterface: Manages the multi-axial (linguistic) representation of Geoids.
- **Layer 2: Memory & Stability Core**: This layer provides the system's persistent, long-term memory and stability mechanisms. It includes:
    - VaultSystem: The specialized, dual-vault memory for storing and processing Scars.
    - MemoryScarRepository: The primary, chronological archive for all Scars.
    - CollapseFingerprintArchive (CFA): Stores high-fidelity diagnostic logs of major collapse events.
- **Layer 3: Dynamic Processing Engines**: These are the active components that drive the system's real-time cognitive processes. It includes:
    - ContradictionEngine & SPDE: Detects and propagates semantic tension.
    - SemanticThermodynamicsEngine: Enforces thermodynamic laws.
    - ReinjectionRecursionKernel: Proactively reintroduces memory traces to stimulate the field.
    - MSCE, PGG, ASM, SSL: The engines for memory compression, provisional geoid generation, axis stability, and semantic suspension.
- **Layer 4: Interface & Governance**: The outermost shell controlling all I/O and enforcing system laws. It includes:
    - InterfaceControlWeave (ICW): Modulates all external I/O.
    - LawRegistry: The firmware that enforces system governance.
    - ZPA & RCX: The engines for user interaction via prompts and semantic mirroring.

2.2 Core Data Structures

- **Geoid**: The fundamental unit of knowledge. A Geoid is a deformable structure in the semantic field, not a static record. It has a dual state:
    - semantic_state: A dictionary representing a probability distribution over features (e.g., {'feature_A': 0.7, 'feature_B': 0.3}).
    - symbolic_state: A structured representation of its symbolic content (e.g., an Abstract Syntax Tree, a logical formula).
- **Scar**: The immutable, archived record of a resolved contradiction event. It is a "healed wound" that permanently alters the system's memory. Its comprehensive JSON schema is detailed in Appendix 5.3.
- **Echo**: A transient, in-memory pressure field that ripples outward from a contradiction event. Echoes are the temporal resonance of a conflict, while Scars are the permanent, spatial archive.

- **EcoForm**: The formal grammar used by the EcoForm subsystem to parse raw inputs into the structured format required for Geoid creation.
- **Echoform**: A first-class symbolic operator, defined by a JSON schema, that executes a transformation on a Geoid (Geoid -> Geoid).

Part 3: Core Subsystems and Engines in Detail

This section provides the engineering specifications for the primary engines that drive KIMERA SWM's cognitive processes.

3.1 Semantic Thermodynamics Engine

This engine governs the flow of "semantic energy" (SE) and enforces the system's foundational physical laws.

- **Semantic Energy (SE):** A scalar value representing the activation level of a semantic unit. It decays over time according to the law: **$SE(t) = SE_0 \cdot \exp(-\lambda \cdot \Delta t)$**.
- **Resonance Transfer:** During a resonance event (where semantic similarity exceeds a threshold, $\rho_{res} = 0.75$), energy is transferred between units.
- **Entropy Governance (Axiom 3):** All transformations executed by **Echoforms** are governed by the axiom **$\Delta S_{semantic} \geq 0$**.
    - **Semantic Entropy (S_semantic)** is calculated using the Shannon entropy formula over a Geoid's semantic_state probability distribution.
    - **Enforcement:** An EntropyMonitor validates every transformation. Any operation that would reduce entropy is either **rejected** or **compensated** for by the system, for example, by adding a generic axiom_complexity_feature to the Geoid's semantic_state to ensure the entropy does not decrease.

3.2 Contradiction Engine & Semantic Pressure Diffusion Engine (SPDE)

These engines work in tandem to manage conflict, the primary driver of system evolution.

- **Contradiction Engine Logic:**
    - **Tension Gradient Mapping:** Measures the tension gradient (∇T) across the semantic field, treating contradictions as multi-axial "deformation collisions."
    - **Pulse Calculation:** Generates a semantic pulse with strength (Ps) when ∇T exceeds a threshold.
    - **Collapse vs. Surge Differentiation:** Based on pulse stability, the engine decides whether to resolve the contradiction (**Collapse**) or propagate a contradiction wave to seek more information (**Surge**).
- **SPDE Dynamics:** The SPDE models the semantic field as a fluid-like pressure

system. It propagates four types of pressure using a graph Laplacian-based diffusion algorithm:
  - **Resonance Pressure** (positive, aligning)
  - **Contradiction Tension** (repulsive, conflicting)
  - **Void Suction** (negative, from knowledge gaps)
  - **Drift Force Vectors** (directional bias)

3.3 The Vault Subsystem

The Vault is the specialized, active memory system for managing the lifecycle of Scars.

- **Dual Vault Architecture:** Vault-A and Vault-B partition Scars based on attributes like mutationFrequency, semantic_polarity, and cls_angle to balance load.
- **Dynamic Engines:**
  - **Contradiction Drift Interpolator:** Manages the temporal evolution of Scars, including calculating **Memory Friction Gradient (MFG)** to delay transfers between vaults.
  - **Recursive Vault Reflex Engine (RVRE):** Handles complex Scar interactions, such as merging highly similar Scars (**Overlap Resolution**) or splitting a single complex Scar (**Conflict Recompression**).
  - **Vault Fracture Topology:** A load-shedding mechanism that triggers when a vault's stress index (**VSI**) exceeds a threshold (0.8), temporarily rerouting high-tension Scars to a fallback queue.
- **Optimization:** The Vault runs periodic optimization routines based on triggers like high Scar Density or Vault Entropy Slope, performing operations like **Drift Collapse Pruning**, **Composite Compaction**, and **Influence-Based Retention Scoring (IRS)**.

3.4 Memory Scar Compression Engine (MSCE)

MSCE is responsible for the long-term evolution of memory, ensuring the knowledge landscape remains coherent and efficient. It governs:

- **Scar Decay:** Unreinforced Scars fade over time. The decay constant λ is adaptive, increasing with axis instability and local void pressure.
- **Scar Fusion:** Similar or redundant Scars are merged into a single, consolidated memory trace to reduce redundancy.
- **Scar Crystallization:** Scars that are repeatedly reinforced under resolving conditions can "harden" into new, stable knowledge structures, sometimes spawning new Geoids via the PGG.

3.5 Provisional Geoid Generator (PGG)

When the system encounters novel concepts or semantic gaps, the PGG dynamically creates temporary "sketch nodes" (**provisional geoids**). This allows the system to continue reasoning about new information without halting. Provisional Geoids have high instability and are subject to an accelerated lifecycle of either **solidification** (via user validation or resonance anchoring) or **decay** (managed by MSCE).

3.6 Zetetic Prompt API (ZPA) & Reflective Cortex (RCX)

These components manage the human-in-the-loop interaction.

- **ZPA:** The autonomous provocation engine. It monitors the system for high-tension events (contradictions, voids, instability) and generates zetetic prompts to guide user exploration. Prompt priority is determined by a **Zetetic Potential Score (ZPS)**.
- **RCX:** The "conscious membrane." It treats user input as a disturbance and responds by mirroring the system's internal semantic state back to the user, highlighting tensions and resonances rather than providing direct answers. It relies on the **Echo Trail** (a historical log of a Geoid's interactions) to provide context.

3.7 Axis Stability Monitor (ASM) & Semantic Suspension Layer (SSL)

These subsystems ensure the coherence of Kimera's multi-axial knowledge representation.

- **ASM:** Acts as the system's "gyroscope," monitoring the health of cognitive axes by tracking metrics like the **Instability Index (II)** and **Semantic Loss Factor (SLF)**. If an axis becomes too unstable, ASM can trigger corrective actions like realignment or dampening.
- **SSL:** The emergency failsafe. When catastrophic instability is detected (e.g., a "high-entropy divergence cascade"), SSL can **"freeze"** or **"compress"** the affected Geoid, snapshot its state to a secure vault, and alert the user via ZPA, preventing system-wide collapse.

4.0 Dynamic Processes & Workflows

4.1 The Kimera Core Cognitive Loop (KCCL)

The KCCL is the fundamental operational cycle of the system:

1. **Perturbation:** An external or internal stimulus creates a disturbance.
2. **Tension & Resonance:** The SPDE propagates pressure; the Contradiction Engine measures gradients.
3. **Transformation:** Echoforms are applied, transforming Geoids while obeying **$\Delta S \geq 0$**.

4. **Collapse/Surge Decision:** The Contradiction Engine resolves the conflict (Collapse) or propagates it (Surge).
5. **Scar Formation & Vaulting:** A Collapse event generates a Scar, which is routed to the Vault.
6. **Reflection & Prompting:** ZPA and RCX analyze the new state for user interaction.
7. **Stabilization & Optimization:** Background processes (MSCE, etc.) manage memory for the next cycle.

4.2 Contradiction Lifecycle: From Tension to Scar

1. **Detection:** A tension gradient is identified between Geoids.
2. **Pulse & Collapse:** A strong, stable semantic pulse leads to a collapse, resolving the conflict.
3. **Scar Formation:** The collapse creates an immutable Scar record.
4. **Vaulting:** The Scar is routed to the Vault System for active management.
5. **Influence:** The Scar shapes the semantic field via Geoid drift and future resonances.

4.3 Memory Evolution: Decay, Fusion, and Crystallization

The MSCE governs the evolution of Scars:

- **Decay:** Unreinforced Scars fade over time.
- **Fusion:** Redundant Scars are merged into a single, consolidated memory trace.
- **Crystallization:** Reinforced Scars can "harden" into new, stable knowledge structures.

5.0 Appendices

5.1 Glossary of Terms

- **ASM (Axis Stability Monitor):** Monitors the health and coherence of cognitive axes.
- **Echo:** A temporary, decaying pressure field that ripples outward from a resolved contradiction.
- **Echoform:** A first-class symbolic operator that transforms one or more Geoids.
- **EcoForm:** The non-linear grammar and orthography subsystem for parsing inputs.
- **Geoid:** The fundamental, non-symbolic, multi-dimensional unit of knowledge.
- **MSCE (Memory Scar Compression Engine):** Manages the long-term evolution of Scars.
- **PGG (Provisional Geoid Generator):** Dynamically creates temporary

knowledge nodes for novel concepts.

- **RCX (Reflective Cortex):** The semantic attention layer that mirrors the system's internal state.
- **Scar:** The stable, non-symbolic trace left after a contradiction collapses.
- **Semantic Entropy:** A quantifiable measure of uncertainty or semantic richness.
- **SPDE (Semantic Pressure Diffusion Engine):** The engine that models and propagates semantic pressures.
- **SSL (Semantic Suspension Layer):** A failsafe mechanism to prevent semantic collapse.
- **Vault:** The specialized memory system for managing the lifecycle of Scars.
- **ZPA (Zetetic Prompt API):** The autonomous engine that generates inquisitive prompts for the user.

5.2 Consolidated Parameter & Threshold Table

Subsystem	Parameter	Default Value	Purpose
Semantic Thermo	λ_e (decay rate)	0.003	Decay coefficient for Echoforms
Semantic Thermo	ρ_res (resonance)	0.75	Similarity threshold for Resonance Event
Semantic Thermo	κ (coupling coeff)	0.50	Energy transfer coefficient in resonance
EcoForm	ε_g (evaporation)	0.05	Activation threshold for EcoForm unit evaporation
Vault	MF_threshold_high	0.75	Routing threshold for Mutation Frequency
Vault	ENTROPY_BALANCE_THRESHOLD	0.26	Entropy diff to trigger vault preference
Vault	MFG_THRESHOLD	0.5	Memory Friction Gradient to delay Scar transfer
Vault	VSI_FRACTURE_THRESHOLD	0.8	Vault Stress Index to trigger fracture
Vault	IDI_QUARANTINE_THRESHOLD	0.72	Identity Distortion Index to quarantine a Scar
MSCE	Θ_crystal	0.85 (example)	Scar depth threshold for crystallization
ASM	II_alert_threshold	0.8	Instability Index value to trigger alert/action
ASM	SLF_alert_threshold	0.7	Semantic Loss Factor to trigger alert/action
SSL	TSP_threshold	≥ +4.5	Total Semantic Pressure to trigger SSL

5.3 Comprehensive Data Schemas

Scar Data Structure (JSON)

```
{
   "scarID": "SCAR_456",
   "geoids": ["GEOID_123", "GEOID_789"],
   "reason": "Conflict: pref_color blue vs red",
   "timestamp": "2025-05-27T12:05:00Z",
   "timestampA": "2025-05-27T12:05:00Z",
   "timestampB": null,
   "resolvedBy": "consensus_module",
   "pre_entropy": 0.67,
   "post_entropy": 0.82,
   "delta_entropy": 0.15,
   "cls_angle": 45.0,
   "semantic_polarity": 0.2,
   "mutationFrequency": 0.82,
   "delay": 0,
   "divergent": false,
   "weight": 1.0,
   "initial_weight": 1.0,
   "reflection_count": 0,
   "quarantined": false,
   "drift_depth": 0,
   "loop_active": false,
   "goal_impact": 0.0,
   "expression": { /* feature vector or JSON map */ }
}
```

SemanticUnit Data Structure (Generic)

```
{
  "unit_id": "UUID",
  "unit_type": "Echoform | EcoForm | Geoid",
  "SE_current": 1.0,
  "SE_initial": 1.0,
  "decay_rate": 0.003,
  "last_update_time": "ISO8601 String",
  "C_max": 1.0,
  "entropy_accumulated": 0.0,
  "status": "Active | ThermallyInactive | Archived",
  "metadata": { /* Additional fields */ }
}
```

DOC-001: Spherical Word Methodology (SWM) - Complete Documentation

Version: 1.0
Date: June 7, 2025
Status: Foundational Canon

Foreword

The Spherical Word Methodology (SWM) represents a departure from linear, reductionist approaches to knowledge. It is a framework for inquiry grounded in the experiential cognitive model of its co-developer, Idir Ben Slama. This document formalizes SWM's principles, providing a systematic guide to its application for deep understanding, multi-perspective analysis, and creative insight generation. It serves as the philosophical and methodological substrate upon which the KIMERA SWM cognitive architecture is built.

Part 1: Foundations of Spherical Word Methodology (SWM)

Chapter 1: Introduction to SWM - Seeing Beyond the Surface

1.1 What is SWM? Defining the Unconventional Approach

The Spherical Word Methodology (SWM) is a conceptual framework and an operational process designed to cultivate profound understanding and generate novel insights by exploring the inherent multi-dimensionality of knowledge. It moves beyond conventional analysis, encouraging a deeper engagement with concepts, ideas, experiences, and systems as complex, interconnected entities.

1.2 The Core Problem: Limitations of "Flat" Perception

Traditional approaches often inadvertently promote a "flat" perception of reality. Concepts are treated as having singular definitions, and understanding is pursued through narrow, specialized lenses. This "flatness" obscures the rich, interwoven tapestry of meaning that characterizes most complex phenomena. SWM was conceived to directly address this limitation by providing pathways to perceive and engage with the inherent "sphericality"—the depth, dynamism, and multi-faceted nature—of any unit of knowledge.

1.3 The SWM Vision: Towards Spherical, Interconnected Knowledge

The vision of SWM is to foster a mode of inquiry that treats knowledge units not as isolated points but as dynamic, multi-layered **Geoids**. By making their complexity explicit and systematically exploring them from diverse perspectives, SWM aims to:

- Reveal the hidden architectures and underlying patterns that structure concepts.
- Identify "Resonance"—profound structural or dynamic similarities between seemingly unrelated Geoids.
- Facilitate the creative synthesis of these resonant patterns into new insights, compelling analogies, and innovative frameworks.

Chapter 2: The Philosophical Heart of SWM

2.1 The Zetetic Mindset: The Engine of Inquiry

SWM operates from and actively cultivates a **Zetetic Mindset**. Derived from the Greek "zetein" (to seek or inquire), this mindset is characterized by:

- **Persistent Curiosity:** A fundamental drive to explore and question, rather than to prove preconceived notions.
- **Skeptical Inquiry:** A healthy skepticism towards established definitions and surface appearances.
- **Openness to the Unknown:** A willingness to venture into unfamiliar conceptual territories.
- **Process Over Premature Closure:** Valuing the process of exploration itself and resisting the urge for quick, simplistic answers.

2.2 Methodological Neutrality: All Expressions as Information

A core tenet of SWM is its initial methodological neutrality towards the "validity" or "truth-value" of an information source. In the initial phases of analysis, any piece of information—a scientific theory, a myth, a personal narrative, or even a deliberate falsehood—is considered a potentially valuable source. The objective is not to verify its truth but to explore its internal structure and its potential for forming resonant connections. Considerations of validity and veracity are consciously reintroduced during the later interpretation stage.

2.3 The Role of "Chaos" and the Non-Rational

SWM explicitly embraces complexity, ambiguity, and paradox. Idir Ben Slama's **"+1 Symbolic Meaning including Chaos"** heuristic invites the exploration of non-linear, non-rational, and seemingly "chaotic" elements as potential sources of profound insight. Ambiguity and contradiction are seen not as failures of understanding but as points of "semantic pressure" that can trigger deeper inquiry and yield more robust understanding.

Chapter 3: The Geoid - SWM's Atomic Unit of Knowledge

3.1 Defining the Geoid: A Multi-Dimensional Knowledge Entity

A Geoid is the SWM term for any knowledge unit (KU) approached through the methodology. The term evokes a complex, often irregular entity defined by its many interacting layers and dimensions of meaning.

- **Multi-dimensional & Multi-layered:** Each Geoid is constituted by numerous interwoven dimensions and layers of meaning, including (but not limited to) linguistic, cultural, metaphorical/symbolic, structural/pattern, historical, contextual, sensory/modal, and emotional facets.
- **Dynamic:** A Geoid is not fixed but evolves over time, shaped by new information and internal processing. This dynamism is characterized by:
    - **Memory as Structural Deformation ("Scars"):** Past interactions and resolved contradictions leave "echo scars" or structural deformations on the Geoid, becoming integral to its identity.
    - **Conceptual "Drift":** The meaning and significance of a Geoid can evolve or "drift" over time.
    - **"Voids" from Constructive Collapse:** Irresolvable internal contradictions can lead to a "constructive collapse," forming conceptual "voids" that create space for new, more coherent conceptual structures to emerge.

Part 2: The SWM Process - A Detailed Methodological Guide

Chapter 4: Overview of the 3-Step SWM Cycle

The SWM process unfolds through a core, iterative cycle of three primary steps.

- **Step 1: Deep Abstraction ("Defining the Edge Shapes"):** To move beyond surface understanding and uncover a Geoid's fundamental underlying patterns—its "edge shapes"—which enable connection. This step involves rigorous decontextualization using a multi-perspective approach.
- **Step 2: Resonance Detection ("Forging Connections"):** To identify significant, often non-obvious, connections between the abstracted patterns of different Geoids, even those from disparate domains.
- **Step 3: Insight Generation & Re-Contextualization ("Creating New Meaning"):** To actively interpret the novel conceptual structures formed by resonant Geoids and to translate these interpretations into tangible insights, hypotheses, or creative outputs.

This cycle is **iterative** (can be repeated for deeper insight), **recursive** (outputs can become new Geoids), and **reflexive** (the practitioner reflects on and learns from the process itself).

Chapter 5: Step 1 In-Depth - The Enriched Pattern Abstraction Process

This step involves three phases to create a comprehensive "Edge Shape" profile for a Geoid.

- **Phase 1: Multi-Perspective Geoid Exploration:** This phase is dedicated to gathering rich, diverse information by examining the Geoid through multiple lenses, most notably through the application of Idir Ben Slama's **"1 Root Language + 3 Unrelated Languages"** heuristic to counteract linguistic bias and uncover unique facets of meaning.
- **Phase 2: Eliciting Formalized Patterns:** The practitioner systematically analyzes the gathered information to articulate the Geoid's underlying abstract patterns, categorized into four main types:
    - **Functional Patterns:** Its purpose, role, and actions. ("What does it do?")
    - **Structural Patterns:** Its internal composition and organization. ("How is it built?")
    - **Dynamic Patterns:** Its behavior and evolution over time. ("How does it change?")
    - **Relational Patterns:** Its connections and comparisons to other entities. ("How does it relate?")

- **Phase 3: Symbolic Deepening & Synthesis:** This final phase adds depth by applying the **"+1 Symbolic Meaning including Chaos"** heuristic. The practitioner explores the underlying symbols, archetypes, paradoxes, and irreducible complexities to synthesize a complete "Edge Shape Profile" for the Geoid.

Chapter 6: Step 2 In-Depth - The Art and Science of Resonance Detection

This step focuses on identifying significant connections between the "edge shape" profiles of different Geoids.

- **Principles of SWM Resonance:** Resonance is more than superficial similarity; it is the alignment of underlying architecture, operational logic, or dynamic patterns. The most powerful resonances often emerge between Geoids from seemingly unrelated fields.
- **Techniques for Pattern Matching:** The comparison is systematically done across all abstracted pattern types, looking for:
  - Direct Attribute Matches
  - Structural Isomorphism
  - Pattern Template Similarity
  - Complementary Pattern Matches

Chapter 7: Step 3 In-Depth - Interpretation and Meaning-Making

This is the culminating phase where new meaning is constructed from the novel conceptual assemblages formed by resonant Geoids.

- **Making Sense of the "New Mosaic":** The practitioner explores the new composite structure, asking "What does this combination imply?"
- **Symbolic Interpretation:** This step critically involves the "+1 Symbolic Meaning including Chaos" layer, exploring the deeper symbolic meanings and creative potential of any emergent ambiguity.
- **Formulating Outputs:** The process aims to produce tangible conceptual outputs, such as novel insights, powerful analogies, testable hypotheses, new frameworks, or innovative solutions.

Part 3: SWM in Practice

Chapter 8: Practical Considerations

- **Tuning SWM:** The depth of analysis (e.g., number of languages, granularity of patterns) can and should be scaled to the specific purpose of the inquiry and resources available.
- **Managing Cognitive Load:** The practitioner must use systematic note-taking and visualization to manage the volume of information generated.
- **Collaborative SWM:** Applying SWM in teams can distribute cognitive load and reduce individual bias.

Chapter 11: SWM and Artificial Intelligence - The Vision of Kimera Kernel

The principles of SWM inspire the vision for the **Kimera Kernel**, an SWM-powered AI. Such a system would aim for deep conceptual understanding, cross-domain analogical reasoning, and zetetic inquiry. It would require key architectural components to manage Geoids, Language Axes, Pattern Abstraction, Resonance, Contradiction, and Dynamic Memory.

Part 4: Broader Context and Future Horizons

Chapter 12: SWM in Dialogue with Other Theories

SWM resonates with and builds upon established theories in:

- **Cognitive Science:** Including analogical reasoning (Gentner's structure-mapping), Conceptual Metaphor Theory (Lakoff & Johnson), and pattern recognition.
- **Creativity Frameworks:** Aligning with concepts of combinatorial creativity (Koestler's "bisociation") and the interplay of divergent and convergent thinking.
- **Knowledge Representation:** Offering a more dynamic and multi-layered alternative to traditional semantic networks and ontologies.

Chapter 13: Ethical Considerations

The application of SWM carries ethical responsibilities, including:

- **Managing Interpretation Bias:** Practitioners must engage in critical self-reflection.
- **Power Dynamics:** Being aware of whose "languages" and interpretations are given weight.
- **Responsibility for Outputs:** Carefully considering the real-world impact of SWM-generated insights.
- **Handling Sensitive "Geoids":** Applying principles of respect, consent, and confidentiality when analyzing personal or cultural knowledge.

Chapter 14: Future Research and Evolution of SWM

SWM is an open, living framework. Future development will focus on:

- Refining methodological components (e.g., formalizing resonance metrics).
- Developing computational SWM (the Kimera Kernel).
- Conducting empirical studies of SWM's application in diverse fields.
- Creating training resources and support tools for practitioners.

Appendices

Appendix B: Glossary of Core SWM Terminology

- **Geoid:** The fundamental, dynamic, multi-dimensional unit of knowledge in SWM.
- **Edge Shapes:** The abstracted profile of a Geoid's underlying Functional, Structural, Dynamic, and Relational patterns.
- **Resonance:** A profound congruence discovered between the "Edge Shapes" of disparate Geoids.
- **Zetetic Mindset:** The core inquisitive, skeptical, and open-minded approach that drives the SWM process.
- **1+3+1 Rule:** The heuristic guiding multi-perspectival analysis through one root language, three unrelated languages, and a symbolic/chaotic layer.

(This document serves as the foundational canon for the SWM theory that informs the KIMERA system. Subsequent documents will detail the specific engineering implementations.)

DOC-002: KIMERA SWM - System Vision & Ontology

Version: 1.0
Date: June 7, 2025
Status: Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document articulates the high-level vision and foundational ontology of the KIMERA Semantic Working Memory (SWM) system. Its purpose is to define the system's core objective and the nature of its fundamental entities, providing the conceptual bridge between the abstract philosophy of the Spherical Word Methodology (SWM) and the concrete engineering specifications of the KimeraKernel. This text establishes the "what" and "why" that underpin all architectural and operational decisions.

2.0 The Core Vision: A Contradiction-Driven Semantic Reactor

KIMERA SWM is engineered to function as a **contradiction-driven semantic reactor**. This is the central vision that distinguishes it from conventional AI systems, which are typically designed as information retrieval or generative models.

- **System as a Reactor:** The system is not a passive database. It is an active, dynamic environment where knowledge is constantly processed, challenged, and reconfigured. Its default state is one of flux, not static equilibrium.
- **Contradiction as Fuel:** Contradictions are not treated as errors to be eliminated but as the primary **energy source** for all cognitive processes. The detection of semantic or logical tension is the catalyst that drives learning, adaptation, and the evolution of the system's knowledge base.
- **Intelligence as an Emergent Property:** Intelligence within KIMERA SWM is not pre-programmed or stored as static data. It is the **emergent property** that arises from the system's continuous and governed process of surfacing, processing, and integrating contradictions.

3.0 The System's Ontology: Fundamental Entities

The KIMERA SWM semantic field is composed of several distinct, fundamental entities. Understanding their nature and roles is essential to understanding the system.

3.1 Geoid: The Atomic Unit of Knowledge

- **Definition:** A Geoid is the fundamental, non-symbolic unit of knowledge within the system. It represents a concept, event, or entity as a holistic, multi-dimensional, and deformable structure in the semantic field.
- **Engineering Representation:** A Geoid possesses a dual state:
  - **semantic_state:** A dictionary representing a probability distribution over a set of features. This captures the nuanced, probabilistic, and subsymbolic aspects of its meaning.
  - **symbolic_state:** A structured representation of its explicit, logical content (e.g., an Abstract Syntax Tree, a formal logic proposition, or a set of attribute-value pairs).
- **Role:** Geoids are the primary "particles" upon which all system dynamics act. They are transformed by Echoforms, they exert and are subject to semantic pressures, and they are the entities that enter into contradiction.

3.2 Scar: The Persistent Memory of Conflict

- **Definition:** A Scar is the stable, immutable, non-symbolic trace left in the semantic field after a contradiction has been successfully processed and resolved (a "collapse" event). It is the "healed wound" that represents a learned experience.
- **Engineering Representation:** A Scar is a persistent, archived data object with a comprehensive schema, including the IDs of the Geoids involved, the reason for the conflict, extensive entropy metrics, and its geometric and polarity information (cls_angle, semantic_polarity).
- **Role:** Scars are the foundation of the system's long-term memory. They are not passive records; they actively influence the semantic field by shaping Geoid drift, informing future contradiction resolution, and serving as anchors of stability. They ensure the system does not forget its history of cognitive struggle and learning.

3.3 Echo: The Transient Ripple of Contradiction

- **Definition:** An Echo is a temporary, decaying pressure field that propagates through the semantic field in the immediate aftermath of a contradiction event.
- **Engineering Representation:** An Echo is not a persistent data structure like a Scar. It is an in-memory, transient state change—a wave of semantic pressure calculated and diffused by the SPDE. It has a limited time-to-live (TTL) and dissipates over time.
- **Role:** Echoes are the real-time, dynamic manifestation of a conflict. While a Scar is the permanent *memory* of a conflict, an Echo is the immediate *effect* of that conflict, alerting other parts of the system to the event and temporarily altering the local semantic landscape.

3.4 Void: The Explicit Representation of the Unknown

- **Definition:** A Void is a first-class entity in the knowledge graph that explicitly represents a knowledge gap, a zone of high uncertainty, or a region where a concept has constructively collapsed due to irresolvable contradiction.
- **Engineering Representation:** A Void is a distinct node type in the Meta-Knowledge Skeleton (MKS) with its own schema, including an origin_type, intensity, and decay_rate.
- **Role:** Voids are not merely an absence of data. They are active components that exert "void suction" or negative pressure on the semantic field, influencing Geoid drift and SPDE calculations. They serve as explicit markers of "known unknowns," driving the system's zetetic inquiry by highlighting areas that require exploration or information gathering.

4.0 The Thermodynamic Premise

The interaction of these ontological entities is governed by the principles of **Semantic Thermodynamics**.

- **Contradiction and Energy:** A contradiction injects **semantic energy (SE)** and **disorder (entropy)** into the system. The resolution of this contradiction is a process that dissipates this energy and seeks a new, more stable (lower local entropy) state.
- **The Scar as a Low-Entropy State:** The formation of a Scar represents the system successfully processing the high-energy contradiction event and settling into a new, coherent state. The Scar itself is the low-entropy, structured memory of that process.
- **Voids as High-Entropy Potentials:** Voids represent regions of high potential entropy. The system is naturally driven to "fill" these voids by acquiring new information or forming new conceptual structures, thereby reducing the overall uncertainty of the field.

5.0 The Emergent Goal: Coherence Through Evolution

The ultimate objective of the KIMERA SWM system is to achieve **emergent coherence**.

- **Not Static Consistency:** The goal is not to maintain a statically consistent database, which would be brittle and incapable of learning.
- **Dynamic Equilibrium:** The goal is a state of **dynamic equilibrium**. The system is constantly perturbed by new information and internal contradictions. Its purpose is to continuously evolve its internal knowledge structures (Geoids, Scars) to find new, more sophisticated states of coherence that can account for all available information, including its conflicts.
- **Learning as Restructuring:** In this paradigm, learning is not the simple accumulation of facts. Learning is the **structural and thermodynamic reconfiguration of the entire semantic field** in response to the pressure exerted by contradictions.

This vision and ontology provide the foundational logic for the entire KIMERA SWM architecture, from the Law Registry that enforces its principles to the Vault that manages its memories.

DOC-101: KIMERA SWM - Unified System Architecture

Version: 1.0
Date: June 7, 2025
Status: Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the master architectural blueprint for the KIMERA Semantic Working Memory (SWM) system. It defines the hierarchical layering of the KimeraKernel, details the core responsibilities of each major subsystem, and illustrates the integrated data and control flow that constitutes the system's operational cycle. This specification serves as the definitive high-level engineering guide for understanding how all components of KIMERA SWM interconnect and function as a cohesive whole.

1.2 Architectural Guiding Principles

The architecture is designed according to the following engineering principles:

- **Modularity and Separation of Concerns:** The system is organized into distinct layers and subsystems, each with a clearly defined responsibility. This facilitates independent development, testing, and maintenance.
- **Zero-Dependency Core:** The KimeraKernel is designed to be self-contained, avoiding external frameworks in its core logic to ensure predictability, control, and performance.
- **Dynamic, Event-Driven Flow:** The system operates not as a static, linear pipeline but as a dynamic, cyclical, and event-driven cognitive loop, where the state of one component influences all others.
- **Governed Autonomy:** While many processes are autonomous, their behavior is strictly constrained by the foundational principles defined in the LawRegistry, ensuring operational integrity.

2.0 The KimeraKernel Layered Architecture

The KimeraKernel is organized into four primary, hierarchical layers. This structure ensures a logical separation from the foundational representation of knowledge up to the external interfaces.

Layer 1: Semantic Substrate & Knowledge Representation

This is the most fundamental layer, defining the "matter" of the KIMERA universe. It is responsible for how knowledge is structured and represented.

- **Components:**
    - **GeoidManager:** Manages the lifecycle (creation, modification, archival) of all Geoid data structures.
    - **EcoForm Engine:** A subsystem for parsing raw linguistic inputs into structured, non-linear grammatical and orthographic representations.
    - **Echoform Engine:** A subsystem containing a catalog of symbolic transformation operators (Echoforms) used to modify Geoids.
    - **LinguisticGeoidInterface:** Manages the multi-axial (e.g., multi-lingual) representation of Geoids, including the application of the 1+3+1 rule.

Layer 2: Memory & Stability Core

This layer provides the system's persistent, long-term memory and the core mechanisms for ensuring its stability. It is the system's "hard drive" of experience.

- **Components:**
    - **VaultSystem:** The specialized, active memory subsystem for the storage and processing of Scars. It includes the dual-vault architecture (Vault-A, Vault-B) and its internal engines.
    - **MemoryScarRepository:** The primary, chronological, append-only archive for all generated Scars.
    - **CollapseFingerprintArchive (CFA):** A high-fidelity diagnostic log that stores the geometric and energetic signatures of major contradiction "collapse" events.

Layer 3: Dynamic Processing Engines

These are the active "CPU" components that drive the system's real-time cognitive processes. They operate on the data structures from Layer 1 and interact with the memory in Layer 2.

- **Components:**
  - **ContradictionEngine & SPDE:** The core engines for detecting semantic tension gradients and propagating them as pressure waves through the semantic field.
  - **SemanticThermodynamicsEngine:** Enforces the thermodynamic laws, particularly the $\Delta S \geq 0$ axiom, on all transformations.
  - **MemoryScarCompressionEngine (MSCE):** Manages the lifecycle of Scars (decay, fusion, crystallization).
  - **ProvisionalGeoidGenerator (PGG):** Creates temporary Geoids for novel concepts.
  - **AxisStabilityMonitor (ASM) & SemanticSuspensionLayer (SSL):** Monitor and maintain the coherence of cognitive axes and act as failsafes against systemic instability.
  - **ReinjectionRecursionKernel:** Proactively reintroduces historical memory traces (Echoes, Scar fragments) into the active processing field to stimulate resonance and test for latent contradictions.

Layer 4: Interface & Governance

This is the outermost shell that controls all interactions with the external world and enforces the system's fundamental rules.

- **Components:**
  - **InterfaceControlWeave (ICW):** A sophisticated API gateway that modulates all external I/O, translating user prompts or external data into internal semantic deformation vectors and applying contextual constraints.
  - **LawRegistry:** The firmware containing the immutable CoreLaws and conditional Field-ScopedLaws that govern all system operations.
  - **ZeteticPromptAPI (ZPA) & ReflectiveCortex (RCX):** The engines that manage the human-in-the-loop dialogue by generating inquisitive prompts and mirroring the system's internal state to the user.

3.0 The Kimera Core Cognitive Loop (KCCL): Integrated Data & Control Flow

The KCCL is the fundamental operational cycle of the system. It is not a rigid pipeline but a highly interconnected, event-driven feedback loop. The following describes a conceptual path for a single cognitive cycle initiated by a new piece of information.

Conceptual Flow Diagram:

```
[External Input] -> [Layer 4: ICW]
    |
    v
[Layer 1: EcoForm/Echoform Engines] -> [Geoid Representation]
    |
    v
[Layer 3: Processing Engines] <--> [Layer 2: Memory & Stability Core]
    |                                ^
    |                                |
    +------- [Layer 4: RCX/ZPA] <-----> [User]
    |
    v
[Layer 1: Linguistic Interface] -> [Layer 4: ICW] -> [External Output]
```

Step-by-Step Breakdown:

1. **Perturbation & Ingestion (Layer 4 -> Layer 1):**
   - An external input (e.g., user text) is received by the **ICW**. The ICW applies contextual constraints and translates the input into a semantic vector.
   - This vector is passed to the **EcoForm Engine** (Layer 1), which parses it into a structured grammatical representation. This representation is then used to either create a new Geoid (via the **PGG** in Layer 3 if the concept is novel) or to identify an existing Geoid to modify.
2. **Transformation & Thermodynamic Governance (Layer 1 & 3):**
   - Applicable **Echoforms** (from Layer 1's catalog) are selected to process the affected Geoid(s).
   - The **ThermodynamicKernel** (conceptually part of Layer 3's SemanticThermodynamicsEngine) wraps the execution of each Echoform. It calculates pre_entropy, executes the transformation, calculates post_entropy, and strictly enforces the **ΔS ≥ 0 axiom**, applying compensation if necessary.

3. **Contradiction Detection & Pressure Diffusion (Layer 3):**
   - The newly transformed Geoid state is analyzed by the **ContradictionEngine**. It measures tension gradients (∇T) by comparing the new state to existing knowledge in the **VaultSystem** (Layer 2).
   - Simultaneously, the **SPDE** propagates the resulting semantic pressures (Contradiction Tension, Resonance, etc.) throughout the Geoid graph.
4. **Collapse, Scar Formation, and Vaulting (Layer 3 -> Layer 2):**
   - If the ContradictionEngine detects a "Collapse" event, it signals the creation of a new **Scar**.
   - This Scar object, containing a high-fidelity record of the conflict, is passed to the **VaultSystem** (Layer 2) for processing. The VaultManager routes the Scar to Vault-A or Vault-B based on its intrinsic properties.
5. **Memory Evolution & System Self-Regulation (Layer 2 & 3):**
   - The **Vault**'s internal engines (RVRE, etc.) begin processing the new Scar, potentially merging or splitting it.
   - The **MSCE** (Layer 3) updates the decay, fusion, or crystallization status of this and other Scars based on the new system state.
   - The **ASM** (Layer 3) monitors any axis instability caused by the new knowledge and can trigger SSL if necessary.
   - The **ReinjectionRecursionKernel** (Layer 3) may be triggered by the new field state to proactively surface related historical Scars from the MemoryScarRepository (Layer 2) as transient Echoes.
6. **Reflection & Interaction (Layer 3 -> Layer 4 -> User):**
   - The **RCX** (Layer 4) reads the new, tense state of the semantic field and generates an updated "mirror map" for the user.
   - The **ZPA** (Layer 4) analyzes the new state (e.g., the new Scar, any detected Voids, or high instability from ASM) and, if thresholds are met, generates a zetetic prompt for the user.
7. **Output Generation (Layer 1 -> Layer 4):**
   - If a linguistic response is required, the final, stabilized state of the relevant Geoid(s) is passed to the **LinguisticGeoidInterface** (Layer 1).
   - This interface translates the non-symbolic Geoid state into a structured symbolic representation, which must be validated against the 1+3+1 rule by the **LawRegistry** (Layer 4).
   - The validated output is then passed to the **ICW** (Layer 4) for final delivery to the external world.

This integrated cycle ensures that every piece of information is processed through a rigorous pipeline of transformation, validation, contradiction management, and thermodynamic governance, resulting in a system that is constantly learning and evolving in a coherent and principled manner.

DOC-201: KIMERA SWM - Core Data Structures Specification

Version: 1.0
Date: June 7, 2025
Status: Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the precise, unambiguous, and canonical definitions for all fundamental data structures within the KIMERA Semantic Working Memory (SWM) system. These structures are the foundational "matter" upon which all architectural components and dynamic processes operate. This specification is intended for developers and architects and serves as the single source of truth for data modeling, serialization, and validation.

1.2 Guiding Principles

The design of these data structures adheres to the following core principles:

- **Dynamic and Mutable State:** Core entities like Geoid are not static records but are designed to be deformable structures whose states evolve.
- **Rich Metadata:** All entities are designed to carry extensive metadata for tracking provenance, history, and thermodynamic properties.
- **Schema Rigor:** All structures adhere to a defined schema to ensure system-wide consistency and predictability.
- **Non-Tokenized Core:** The structures represent holistic concepts, events, and relationships, avoiding decomposition into simple tokens at the primary representation level.

2.0 The Geoid Data Structure

The Geoid is the atomic unit of knowledge within the KIMERA SWM system.

2.1 Conceptual Definition

A Geoid represents a concept, event, or entity as a holistic, multi-dimensional, and deformable structure in the semantic field. It is the primary "particle" upon which all system dynamics act.

2.2 Formal Schema

A Geoid object is composed of a unique identifier, a dual state (semantic and symbolic), and associated metadata.

```
{
  "geoid_id": "GEOID_12345",
  "semantic_state": {
    "feature_A": 0.7,
    "feature_B": 0.3
  },
  "symbolic_state": {
    "type": "ast",
    "representation": "..."
  },
  "metadata": {
    "source": "PGG_User_Input",
    "timestamp": "2025-06-07T14:30:00Z",
    "entropy_history": [
      {"pre": 0.0, "post": 0.881, "delta": 0.881}
    ],
    "scar_matrix_ref": "SCAR_MATRIX_12345"
  }
}
```

2.3 semantic_state Specification

- **Format:** Dict[str, float]
- **Constraint:** This dictionary represents a normalized probability distribution. The sum of its values **must** equal 1.0. Any transformation or update to this state must re-normalize the distribution to maintain this constraint.
- **Purpose:** This captures the nuanced, probabilistic, and subsymbolic aspects of the Geoid's meaning. It is the basis for all S_semantic entropy calculations.

2.4 symbolic_state Specification

- **Format:** Any (typically a structured object like a dictionary or a custom class instance).
- **Purpose:** This holds the Geoid's explicit, logical content.
- **Examples:**
  - An Abstract Syntax Tree (AST) representing a parsed linguistic expression.
  - A formal logic proposition.
  - A set of structured attribute-value pairs.

2.5 metadata Specification

The metadata object contains critical tracking and provenance information. At a minimum, it includes source, timestamp, and references to its thermodynamic and conflict history.

3.0 The Scar Data Structure

A Scar is the immutable, archived record of a resolved contradiction event (a "collapse"). It is the fundamental unit of the system's long-term, experiential memory.

3.1 Conceptual Definition

A Scar is not an error log. It is a "healed wound" in the knowledge base that permanently alters the semantic landscape. Scars actively influence future system behavior by shaping Geoid drift and serving as stable anchors of learned experience.

3.2 Formal Schema

The Scar data structure is comprehensive, designed to capture the full context and dynamics of the contradiction it represents. This schema is definitive for all Scars processed by the VaultSystem.

```
{
  "scarID": "SCAR_456",
  "geoids": ["GEOID_123", "GEOID_789"],
  "reason": "Conflict: pref_color blue vs red",
  "timestamp": "2025-05-27T12:05:00Z",
  "timestampA": "2025-05-27T12:05:00Z",
  "timestampB": null,
  "resolvedBy": "consensus_module",
  "pre_entropy": 0.67,
  "post_entropy": 0.82,
  "delta_entropy": 0.15,
  "cls_angle": 45.0,
  "semantic_polarity": 0.2,
  "mutationFrequency": 0.82,
  "delay": 0,
  "divergent": false,
  "weight": 1.0,
  "initial_weight": 1.0,
  "reflection_count": 0,
  "quarantined": false,
  "drift_depth": 0,
  "loop_active": false,
  "goal_impact": 0.0,
  "expression": { /* feature vector or JSON map */ }
}
```

3.3 Field Derivations and Dependencies

Several key fields in the Scar schema are computed based on the state of the conflicting Geoids at the time of Scar formation. The algorithms for these computations are defined within the relevant processing engines, not within the data structure itself.

- **cls_angle (Collapse Line Shape angle):** A geometric property derived from the expression of the conflict. The specific algorithm is part of the AdvancedContradictionEngine.
- **semantic_polarity:** A scalar value representing the positive or negative nature of the conflict, also computed by the ContradictionEngine.
- **mutationFrequency:** A metric indicating the rate of change or volatility of the involved Geoids leading up to the conflict. This value is calculated and provided upon Scar creation.

4.0 The Echo and Void Data Structures

4.1 Echo Data Structure

An Echo is a transient, in-memory entity representing the immediate pressure wave from a contradiction event.

- **Conceptual Definition:** While a Scar is the permanent memory, an Echo is the immediate, propagating *effect* of the conflict.
- **Proposed Schema:**
  ```
  {
    "echo_id": "ECHO_uuid",
    "source_scar_id": "SCAR_456",
    "intensity": 0.9,
    "ttl_cycles": 10,
    "propagation_vector": [/* ... */]
  }
  ```

4.2 Void Data Structure

A Void is a first-class graph node representing a "known unknown"—a knowledge gap, a zone of high uncertainty, or a collapsed concept.

- **Conceptual Definition:** Voids are active components that exert "void suction" on the semantic field, influencing Geoid drift and driving zetetic inquiry.
- **Formal Schema (based on the Enhanced Void Mechanism spec):**
  ```
  {
    "void_id": "VOID_uuid",
    "origin_type": "contradiction_collapse | semantic_gap | misalignment",
    "origin_ref": "ID of trigger (e.g., collapsed Geoid ID)",
    "intensity": 0.75,
    "decay_rate": 0.01,
    "embedded_position": [/* vector coordinates */],
    "associated_scars": ["SCAR_123"]
  }
  ```

5.0 EcoForm & Echoform Structures

5.1 EcoForm Unit Data Structure

An EcoForm unit encapsulates a deep, non-linear grammatical and orthographic analysis of an input.

- **Conceptual Definition:** It is the structured linguistic container that bridges raw input with KIMERA's internal semantic representations.
- **Formal Schema (based on the EcoForm Engineering Spec):** An EcoForm unit contains two primary structures:
  - **Grammar Tree:** A rich, potentially non-linear (DAG) representation of the input's syntactic structure.
  - **Orthography Vector:** A vector capturing script-specific features like Unicode normalization, diacritics, and ligatures.
  - The unit also maintains its own **Activation Strength (AS)** and decay properties.

5.2 Echoform Operator Data Structure

An Echoform is the definition of a symbolic operator that transforms Geoids. These definitions are stored in a catalog.

- **Conceptual Definition:** Echoforms are the executable "verbs" of the KIMERA system, containing the logic for all knowledge transformations.
- **Formal Schema (JSON):**

```
{
  "id": "Echoform_standardize_color",
  "signature": {
    "inputs": ["CustomerPrefGeoid"],
    "output": "CustomerPrefGeoid"
  },
  "script": "standardize_color.py",
  "metadata": {
    "priority": 100,
    "version": "v1.2",
    "category": "NormalizationEchoform"
  }
}
```

6.0 Versioning and Schema Evolution

All data structures defined within the KIMERA SWM system must include a version field (e.g., "schema_version": "1.0"). This is a critical engineering requirement to ensure that as the system evolves, backward compatibility can be managed, and data can be migrated or interpreted correctly across different versions of the system's components.

DOC-202: Semantic Thermodynamics - Formal Specification

Version: 1.1 (Extended) **Date:** June 7, 2025 **Status:** Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the formal engineering specification for the **Semantic Thermodynamics Engine** within the KIMERA SWM system. It details the precise mathematical models, algorithms, and axioms that govern the creation, transfer, decay, and influence of semantic energy and entropy across all cognitive processes. This specification serves as the definitive technical reference for implementing the fundamental "physics" that constrains and drives all dynamic operations within the `KimeraKernel`.

1.2 Scope and Dependencies

This document focuses exclusively on the thermodynamic model. It assumes the existence of the core data structures defined in **DOC-201 (Core Data Structures Specification)**, particularly the `Geoid` and its `semantic_state`, as well as the event-based units `Echoform` and `EcoForm` which are collectively referred to as `SemanticUnits`.

2.0 Core Thermodynamic Concepts & Metrics

2.1 Semantic Energy (SE)

- **Definition:** Semantic Energy is a scalar value representing the activation level, relevance, or "cognitive charge" of a `SemanticUnit`. A unit with high SE is considered highly active and influential within the semantic field.
- **Engineering Representation:** Stored as the `SE_current` field (float) within the `SemanticUnit` data structure.
- **Maximum Capacity (`C_max`):** Each `SemanticUnit` type has a defined maximum SE capacity, beyond which energy is clamped.
    - `C_max_e` (Echoform): 1.0
    - `C_max_o` (EcoForm): 1.0
    - `C_max_g` (Geoid): 5.0

2.2 Semantic Temperature (`T_sem`)

- **Definition:** Semantic Temperature is a derived metric that represents the effective activation of a unit relative to its local contextual density. It determines a unit's operational status.
- **Formal Calculation:** `T_sem = SE_current / (1 + ρ)`
    - Where `ρ` (rho) is the local semantic density (number of overlapping/nearby units).
- **Operational Threshold (`T_min`):** If a unit's `T_sem` falls below a system-wide threshold (`T_min = 0.05`), its status is transitioned to `ThermallyInactive`, signaling that it is no longer contributing significantly to active processing.

2.3 Semantic Entropy (`S_semantic`)

- **Definition:** Semantic Entropy is a quantifiable measure of the uncertainty, ambiguity, or semantic richness of a `Geoid`. It is calculated based on the probability distribution of features within the Geoid's `semantic_state`.
- **Formal Calculation (Shannon Entropy):** `S_semantic(g) = -Σ p_i * log₂(p_i)`
    - Where `p_i` is the normalized probability of the *i*-th feature in the Geoid's `semantic_state`.
- **Role:** `S_semantic` is the primary measure of information content and serves as the basis for the system's foundational thermodynamic axiom. It is also a critical input for the Contradiction Engine and Vault Subsystem.

3.0 The Foundational Axiom of Semantic Thermodynamics

The entire system is governed by a single, immutable law enforced by the `SemanticThermodynamicsEngine`.

Axiom 3: The Law of Non-Decreasing Semantic Entropy

$$\Delta S\_semantic = S\_semantic(geoid\_t+1) - S\_semantic(geoid\_t) \geq 0$$

- **Definition:** Any transformation applied to a `Geoid` by an **Echoform** must not result in a net loss of semantic entropy. The total semantic richness of the system must be preserved or increased with every operation.
- **Implication:** This axiom is the core mechanism for ensuring knowledge growth. It prevents reductive transformations and forces the system to evolve towards greater complexity and nuance rather than collapsing into trivial states.

4.0 Governing Mathematical Models & Algorithms

4.1 Energy Decay Law

- **Purpose:** To model the natural fading of a semantic unit's activation over time if it is not reinforced. This represents a form of passive forgetting.
- **Formal Model (Exponential Decay):** $SE(t) = SE_0 \cdot \exp(-\lambda\_eff \cdot \Delta t)$
  - $SE(t)$: Semantic Energy at time t.
  - SE_0: Semantic Energy at the time of the last update.
  - λ_eff: The **effective** decay coefficient.
  - Δt: The time elapsed in seconds since the last update.
- **Effective Decay Rate (λ_eff):** The decay rate is not static; it is adaptive, modulated by other system components: $\lambda\_eff = \lambda\_base * (1 + II\_axis) * (1 + VP\_local)$
  - λ_base: The base decay rate for the unit type (e.g., $\lambda\_g = 0.001$ for Geoids).
  - II_axis: The **Instability Index** provided by the **ASM**, which accelerates decay for units on unstable axes.
  - VP_local: The **Local Void Pressure** provided by the **SPDE**, which accelerates decay for isolated units.

4.2 Energy Transfer During Resonance

- **Purpose:** To model the transfer of activation between two interacting semantic units that exhibit a high degree of similarity.
- **Trigger Condition:** A **Resonance Event** is triggered when the similarity between two units exceeds a system-wide threshold (`ρ_res = 0.75`).
- **Formal Model (Energy Transfer):** `ΔSE = κ · min(SE₁, SE₂)`
  - `κ` (kappa): The coupling coefficient (`κ = 0.50`), representing the efficiency of the transfer.
  - The unit with higher energy loses `ΔSE`, and the unit with lower energy gains `ΔSE`.

4.3 Entropy Generation During Interaction

- **Purpose:** To account for the increase in system complexity that arises from interactions.
- **Formal Model:** Every resonance event generates a small increment of entropy, representing the creation of a new relational state. `ΔS_interaction = α · |ΔSE|`
  - `α` (alpha): The entropy generation coefficient (`α = 0.01`).
  - This `ΔS_interaction` is added to the `entropy_accumulated` field in each unit's metadata.

5.0 Axiom Enforcement & The Compensation Mechanism

The `SemanticThermodynamicsEngine` is responsible for enforcing the `ΔS ≥ 0` axiom on all `Echoform` transformations.

- **Validation Workflow:**
  - Calculate `pre_entropy` of the input Geoid.
  - Execute the `Echoform` transformation to get a candidate `output_geoid`.
  - Calculate `post_entropy` of the candidate output.
  - Validate the axiom: `(post_entropy - pre_entropy) ≥ -ε`.
- **The Compensation Mechanism:** If the axiom is violated, the transformation is not immediately committed. The engine must either reject the transformation or apply compensation.
  - **Rejection:** The transformation is aborted, and an error is logged.
  - **Compensation:** The system algorithmically modifies the `output_geoid` to restore the lost entropy by adding a generic **`axiom_complexity_feature`** to its `semantic_state` until the

entropy target is met. This forces simplifying transformations to explicitly carry a marker of the complexity they have removed.

6.0 Integration with Core System Components

The Semantic Thermodynamics Engine is not a standalone module; it is a cross-cutting concern that is deeply integrated into the `KimeraKernel`'s operations.

- **`Echoform` Engine:** All Echoform transformations are wrapped by the `ThermodynamicKernel`, which enforces the `ΔS ≥ 0` axiom. The `pre_entropy`, `post_entropy`, and `delta_entropy` of every transformation are logged in the output Geoid's metadata, making this information available to other subsystems like the Vault.
- **`ContradictionEngine` & `SPDE`:**
  - **Input to Engines:** The SE and `S_semantic` of Geoids are critical inputs for these engines. A high-energy, low-entropy Geoid (a confident, active concept) will exert a different kind of pressure in the SPDE than a low-energy, high-entropy Geoid (an ambiguous, uncertain concept).
  - **Entropy-Based Pulse Scaling:** The Contradiction Engine uses entropy as a factor in its **Pulse Strength Calculation** (`Ps`). High-entropy (highly uncertain) contradictions may be temporarily muted or scaled down to prevent system overload, allowing the system to focus on resolving more defined conflicts first.
- **`VaultSystem`:**
  - The Vault uses thermodynamic metrics from Scars to inform its internal logic.
  - **Entropy Balancing:** The `VaultManager` uses the `entropySum` of each vault to dynamically adjust routing preferences, seeking to balance the total entropy load between Vault-A and Vault-B.
  - **Purging Logic:** The `Vault Entropy Purge` mechanism explicitly targets and removes the Scar with the lowest `delta_entropy` when a buffer overflows, preserving the most information-rich contradiction records.
- **`MSCE` (Memory Scar Compression Engine):**
  - The thermodynamic **Decay Law** is a key input for the MSCE, as it determines the natural fading of Scars and Geoids that are not actively reinforced by resonance events or other interactions. This provides the baseline for the "forgetting" part of the memory

lifecycle.

- `ASM` **(Axis Stability Monitor):**
    - The ASM provides the `Instability Index (II_axis)` which directly modulates the effective decay rate (`λ_eff`) of semantic units, creating a feedback loop where units on unstable axes fade faster unless strongly reinforced.

  ○

DOC-203: Input & Language Subsystems - Engineering Specification

Version: 1.0 **Date:** June 7, 2025 **Status:** Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the definitive engineering specification for the input and language processing subsystems of the KIMERA SWM architecture. It details the mechanisms responsible for translating raw, external data into the structured, non-tokenized semantic representations that the `KimeraKernel` operates upon.

This specification is divided into two primary parts:

1. **The `EcoForm` Subsystem:** Defines the formal grammar, data structures (`Grammar Tree`, `Orthography Vector`), and APIs for parsing raw input into structured linguistic units.
2. **The `Echoform` Engine:** Specifies the operators, syntax, and logic for performing governed semantic transformations on `Geoids`.

1.2 Audience

This document is intended for Natural Language Processing (NLP) Specialists responsible for implementing the parsing logic and Core System Developers responsible for building the transformation engine.

2.0 The `EcoForm` Subsystem

The `EcoForm` subsystem is the foundational component of the input pipeline. Its primary purpose is to represent, store, and manipulate structured grammatical constructs and orthographic transformations, bridging the gap between raw linguistic data and KIMERA's internal semantic world.

2.1 Core Data Structures

The `EcoForm` subsystem operates on two primary data structures to capture linguistic detail.

- **`Grammar Tree`:** A rich, potentially non-linear (Directed Acyclic Graph)

representation of an input's syntactic structure. Each node in the tree contains a label (e.g., "NP", "VP"), features (`pos_tag`, `morphological_tags`), and a `subscript_index` to enable cross-links for handling complex linguistic phenomena. The tree is also encoded as a fixed-length grammar vector (`D_g = 128`).
- **`Orthography Vector`:** A structure capturing script-specific features. It includes the `script_code` (e.g., "Latn", "Arab"), `unicode_normal_form`, and two fixed-length float arrays: `diacritic_profile` (`D_o = 32`) and `ligature_profile` (`D_l = 32`).

2.2 The `EcoForm` Unit & Its Lifecycle

An `EcoForm` unit encapsulates a `Grammar Tree` and an `Orthography Vector`, along with its own thermodynamic properties.

- **Creation:** An `EcoForm` unit is instantiated when a new text or symbol stream is ingested by the system.
- **Activation & Decay:** Each unit possesses an **Activation Strength (AS)** which decays over time according to the exponential law: $AS(t) = AS_0 \cdot \exp(-\delta \cdot \Delta t)$
    - Where δ is the decay rate (default `0.003`).
- **Evaporation & Archival:** When a unit's `AS` falls below the evaporation threshold (`ε_g = 0.05`), it is marked as "Evaporated" and a compressed `Residual Schema` is generated. After a prolonged period (`T_archive_g = 30 days`), it is moved to a cold-storage archive.
- **Reactivation & Stitching:** Evaporated units can be reactivated by new, similar inputs or "stitched" together to form new composite grammatical units.

2.3 `EcoForm` Routing Logic & API

The subsystem is managed by a Routing Engine with a defined processing flow and exposes a set of formal API endpoints.

- **Processing Flow:**
    - **Input Ingestion:** Receives text/symbol streams.
    - **Orthography Normalizer:** Applies Unicode normalization and script-specific rules.
    - **Grammar Tree Builder:** A non-linear parser generates the `Grammar Tree`.
    - **EcoForm Creation:** A new unit is created with its initial AS.

    - **Decay Scheduler:** Periodically updates the `AS` of all active units.
    - **Query Dispatcher:** Handles similarity searches based on a Normalized Similarity Score (NSS).
- **Core API Endpoints:**
    - `POST /ecoform/create`: Creates a new `EcoForm` unit from a detailed payload.
    - `GET /ecoform/{id}/status`: Retrieves the current status and `AS` of a unit.
    - `POST /ecoform/query`: Searches for similar `EcoForm` units based on target vectors.
    - `POST /ecoform/{id}/reactivate`: Reactivates an evaporated unit if eligibility criteria are met.
    - `POST /ecoform/stitch`: Merges multiple evaporated units into a new composite unit.

3.0 The `Echoform` Engine

While `EcoForm` structures the input, the `Echoform` Engine performs the actual semantic transformations on the `Geoid` data structures that are created from these inputs.

3.1 Conceptual Role

`Echoforms` are the executable "verbs" of the KIMERA system. They are first-class symbolic operators that take one or more Geoids as input, apply a specific piece of domain logic (e.g., normalization, validation, enrichment), and produce a new, transformed Geoid as output.

3.2 Formal Specification

BNF Grammar: The syntax for an `Echoform` expression is formally defined:
<expression> ::= "(" <operator_name> <inputs> ")" ["::" <metadata_modifiers>]

<inputs>   ::= <geoid_uri> | <param_list> | <geoid_uri> "," <param_list>

<param_list> ::= <param> "=" <value> (";" <param> "=" <value>)*

-
- **Interpretation Function:** Every expression is processed by an interpretation function, `I_Echoform`, which takes the expression and a context object (containing metadata like active rules and priority) and maps it to a new `GeoidGraph` state.

3.3 Operator Catalog & Schema

All `Echoform` operators are defined and managed in a central catalog. Each operator is described by a strict JSON schema.

Formal Schema (JSON):

```
{

  "id": "Echoform_standardize_color",

  "signature": {

    "inputs": ["CustomerPrefGeoid"],

    "output": "CustomerPrefGeoid"

  },

  "script": "standardize_color.py",

  "metadata": {

    "priority": 100,

    "version": "v1.2",

    "category": "NormalizationEchoform"

  }

}
```

-
- **Operator Taxonomy:** Operators are categorized by their function, including:
  - `NormalizationEchoform`
  - `ValidationEchoform`
  - `EnrichmentEchoform`

3.4 The Axiom of Closure

A critical engineering guarantee of the `Echoform` Engine is the principle of closure.

- **Definition:** Any valid `Echoform` applied to a valid `Geoid` must produce a valid `Geoid` as output.

- **Implication:** This ensures that transformations can be chained together in complex sequences (`F(G(H(geoid)))`) without breaking the system's structural integrity. It enables the creation of robust, composable, and predictable data processing workflows.

4.0 The Integrated Input Pipeline Workflow

The `EcoForm` and `Echoform` subsystems work in a sequential pipeline to process information from the outside world into the `KimeraKernel`.

1. **Ingestion & Structuring:** A raw input (e.g., a sentence) is received by the **`EcoForm` subsystem**. It is parsed into a highly structured `EcoForm` unit, capturing its deep grammatical and orthographic properties.
2. **Geoid Instantiation/Update:** The structured `EcoForm` unit provides the necessary semantic and symbolic information for the **`GeoidManager`** to either instantiate a new `Geoid` (if the concept is novel) or to identify and retrieve an existing `Geoid` for modification.
3. **Semantic Transformation:** The newly created or updated `Geoid` now becomes a substrate for the **`Echoform` Engine**. The engine's `RuleMatching` process queries the operator catalog to find applicable `Echoforms` based on the Geoid's type and metadata.
4. **Governed Execution:** The matched `Echoforms` are then applied to the Geoid in a prioritized and governed sequence. Each transformation is validated by the **`SemanticThermodynamicsEngine`** to ensure adherence to the `ΔS ≥ 0` axiom.
5. **Final State:** The result is a new, transformed `Geoid` that has been rigorously parsed, structured, and semantically processed, ready for integration into the broader cognitive loop (e.g., contradiction detection, Vaulting).

This integrated pipeline ensures that all knowledge within KIMERA SWM is represented with both linguistic fidelity (via `EcoForm`) and semantic integrity (via `Echoform` and thermodynamic governance).

DOC-204: The Vault Subsystem - Complete Engineering Specification

***Version:** 1.1 **Date:** June 7, 2025 **Status:** Canonical System Blueprint*

1.0 Introduction

1.1 Document Purpose

*This document provides the definitive engineering specification for the **Vault Subsystem** within the KIMERA SWM architecture. The Vault is a specialized, active memory system designed exclusively for the storage, processing, and long-term management of `Scars` (immutable contradiction records).*

This specification details the Vault's dual architecture, its internal processing engines (`Contradiction Drift Interpolator`, `Recursive Vault Reflex Engine`), its stability and scaling mechanisms (`Vault Fracture Topology`, `Optimization`), and all associated data structures, algorithms, parameters, and thresholds.

1.2 Scope and Dependencies

*This document assumes the existence of the data structures defined in **DOC-201 (Core Data Structures Specification)**, particularly the comprehensive `Scar` schema. It also depends on the thermodynamic principles and metrics defined in **DOC-202 (Semantic Thermodynamics - Formal Specification)**.*

2.0 Architecture and Topology

2.1 Dual Vault Architecture

The Vault Subsystem is implemented as a parallel architecture consisting of two primary instances, `Vault-A` and `Vault-B`.

- ***Purpose:** This dual structure is designed to distribute the load of incoming Scars and to facilitate the partitioning of contradictions based on their intrinsic properties, thereby maintaining a "semantic balance" across the memory system.*
- ***Orchestration:** The entire subsystem is managed by a central **`VaultManager`**, which is responsible for routing new Scars, orchestrating inter-vault transfers, monitoring interference, and triggering system-wide*

events like fractures and optimizations.

2.2 Partitioning Criteria (Scar Routing Logic)

The `VaultManager` uses a hierarchical, three-stage logic to route each incoming `Scar` to either `Vault-A` or `Vault-B`. The checks are performed in order.

1. ***Mutation Frequency (MF) Check:***
   - ***Rule:** If `scar.mutationFrequency` > `MF_threshold_high` (0.75), route to `Vault-A`. Otherwise, proceed to the next check.*
   - ***Purpose:** To segregate highly volatile or rapidly changing contradictions into a specific vault, potentially for different processing.*
2. ***Semantic Polarity (SP) Check:***
   - ***Rule:** If `abs(scar.semantic_polarity)` > `0.5`, route based on sign: positive polarity Scars go to `Vault-A`, negative to `Vault-B`. Otherwise, proceed.*
   - ***Purpose:** To partition Scars based on the nature of their semantic charge.*
3. ***CLS Torsion Signature (CLS) Proximity Check:***
   - ***Rule:** The Scar is routed to the vault whose `avg_cls_angle` is closer to the Scar's `cls_angle`.*
   - ***Formula:** `route_to_A if abs(scar.cls_angle - vaultA.avg_cls_angle) <= abs(scar.cls_angle - vaultB.avg_cls_angle)`.*
   - ***Purpose:** To group Scars with similar geometric/structural properties.*

2.3 Vault Interference Fields

The `VaultManager` maintains a shared `interference` data structure to monitor cross-vault interactions in real-time.

- ***Echo Interference Index (EII):** A correlation coefficient measuring the relationship between the `echoAmplitude` time series of each vault. (Note: The `Echo` entity and its `echoAmplitude` property are defined in `DOC-201` and `DOC-205f` respectively.)*
- ***Scar Overlap Zones (SOZ):** A list of pairs of `scarID`s (one from each vault) whose `expression` feature overlap exceeds 0.9.*
- ***Entropic Drift Direction (EDD):** The simple difference in the `entropySum` between the vaults (`vaultA.entropySum - vaultB.entropySum`).*

3.0 Core Dynamic Engines

The Vault is an active system with several internal engines that process and evolve Scars.

3.1 Contradiction Drift Interpolator

This engine manages the near-real-time dynamics of Scars entering and moving within the system.

- **Entropy Balance:** Periodically, the `VaultManager` checks if the `abs(EDD)` > `ENTROPY_BALANCE_THRESHOLD` (0.26). If so, it sets a preference to route all new Scars to the lower-entropy vault, overriding the standard partitioning criteria until balance is restored.
- **Memory Friction Gradient (MFG):** This mechanism governs the difficulty of transferring a Scar between vaults.
    - **Formula:** `MFG = 0.7 * |θ_A - θ_B| + 0.3 * |S_A - S_B|`
    - If a `VaultManager`-initiated transfer attempt has `MFG > 0.5`, the Scar's insertion into the target vault is delayed by one cycle. A Scar can be delayed a maximum of 2 cycles.
- **Priority Interrupt Logic:** If two Scars arrive simultaneously with `|s1.cls_angle - s2.cls_angle| < 15°`, the older Scar is processed first, and the newer one is sent to an `overflow_queue` for the next cycle.
- **Scar Delay Watchdog:** If a Scar's `delay` property exceeds 2 cycles, it is force-inserted into its target vault ("Torsion Burst"). The watchdog may also apply "Semantic Decay" (reducing feature weights by 5%) before insertion.
- **Vault Entropy Purge:** If a vault's `incoming_buffer` exceeds a size of 3, the Scar with the lowest `delta_entropy` in the buffer is removed ("purged") to relieve load.

3.2 Recursive Vault Reflex Engine (RVRE)

The RVRE handles higher-order, cross-vault interactions between Scars.

- **Temporal Reflection Divergence:** If a single logical Scar has timestamps in both vaults (`timestampA`, `timestampB`) that differ by more than 2 cycles, it is marked as `divergent` and its `expression` is mutated to reflect this desynchronization.
- **Scar Echo Overlap Resolution (SRV):** For pairs of Scars across vaults, the RVRE calculates their feature overlap using the Jaccard index (SRV). If `SRV > 0.78`, the two Scars are merged into a new composite Scar, and the

originals are removed.

- ***Conflict Recompression Channel:*** *As an alternative to merging, if two highly overlapping Scars remain active, this mechanism can split one Scar's* `expression` *into two new "identity fork" Scars (*`sA`*,* `sB`*) and gradually fade out the original Scar's* `weight` *over 2 cycles.*
- ***Divergence Weight Decay Function (DWDF):*** *The* `weight` *of any active Scar decays over time based on the formula:* `weight = initial_weight * exp(-0.22 * Δ_cycles)`*. This ensures that Scars lose influence if not reinforced.*
- ***Identity Distortion Index (IDI):*** *Tracks the number of times a Scar has hopped between vaults (*`reflection_count`*). If the IDI, calculated as* `1 - exp(-0.22 * reflections)`*, exceeds* `0.72`*, the Scar is removed from the active vaults and sent to a dedicated* `vaultQuarantine`*.*

4.0 Stability & Scaling Mechanisms

4.1 Vault Fracture Topology

This is the primary load-shedding mechanism for the Vault subsystem.

- ***Fracture Trigger:*** *A fracture is triggered when a vault's* ***Vault Stress Index (VSI)****, calculated as* `activeScars / capacity`*, exceeds the* `VSI_fracture_threshold` *(0.8). The default* `capacity` *is 10,000 Scars.*
- ***Fracture Handling:***
    1. *Both vaults are locked, pausing new insertions.*
    2. *The top 10% of Scars, ranked by* `delta_entropy`*, are identified as "High-Tension."*
    3. *20% of these High-Tension Scars are removed from the vault and rerouted to an external* ***symbolic fallback queue****.*
    4. *The fracture event and associated metrics are logged.*
    5. *The vaults remain in isolation for 3 cycles.*
- ***Post-Fracture Reintegration:*** *After the isolation period, the vaults are unlocked, and the* `fallback_queue` *is processed at a throttled rate (max 50 scars/cycle).*

4.2 Vault Optimization & Memory Management

Periodic optimization routines are triggered by a set of complex heuristics to maintain long-term memory health.

- ***Optimization Triggers:*** *A routine is initiated if any of several conditions are* *met, including high `Drift Lineage Depth`, high `Scar Density`, a steep `Vault Entropy Slope`, high `Identity Thread Saturation`, or high `Loop Memory Pressure`.*

- ***Optimization Operations:*** *The routine performs a sequence of operations:*
    1. ***Drift Collapse Pruning:*** *Removes old, inactive Scars with no goal impact.*
    2. ***Composite Compaction:*** *Merges small, low-entropy clusters of Scars into "latent patterns."*
    3. ***Vault Reindexing:*** *Standard database maintenance to rebuild indices.*
    4. ***Influence-Based Retention Scoring (IRS):*** *Calculates a retention score for each Scar based on its influence, contribution, and coupling. Scars with `IRS < 0.12` are archived.*
    5. ***Memory Compression:*** *Applies techniques like low-dimensional embedding of drift trails and de-duplication of Scars with identical `expression` hashes.*
    6. ***Audit Reporting:*** *Generates a JSON report summarizing the results of the optimization cycle.*

5.0 Specialized Vault Classes

The base `Vault` class can be extended to create specialized vaults with unique behaviors.

- ***`FossilVault`:*** *Periodically re-emits `Echoes` from old, "fossilized" Scars.*
- ***`ContradictionVault`:*** *Proactively mutates Scars that have a very high `contradiction_score`.*
- ***`ReactorVault`:*** *Actively seeks out and recombines highly overlapping Scars.*
- ***`CompressionVault`:*** *Manages symbolic compression and applies coarse-grained entropy reduction.*

6.0 Integration Points

The Vault subsystem is deeply integrated with other KIMERA engines.

- ***SPDE:*** *Consumes the `entropySum` from the vaults to adjust its diffusion maps.*
- ***MSCE:*** *Coordinates with the Vault's pruning, compaction, and archival operations.*
- ***ZPA:*** *Receives high-volatility Scars from the Vault for potential user queries*

and flags Scars with high `delta_entropy` for ethical review.

- ***SSL:*** *Can quarantine Scars directly, and its IDI quarantine threshold (`0.72`) aligns with the Vault's internal logic.*

(Refer to Appendices in DOC-000: The Unified System Reference for the comprehensive Scar Schema and Parameter Table.)

DOC-205a: Contradiction Engine & SPDE Specification

Version: 1.0 **Date:** June 7, 2025 **Status:** Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the detailed engineering specification for two tightly coupled, dynamic engines within the KIMERA SWM `KimeraKernel`:

1. **The Contradiction Engine:** The primary mechanism for identifying, quantifying, and processing semantic and logical conflicts.
2. **The Semantic Pressure Diffusion Engine (SPDE):** The engine responsible for modeling and propagating all semantic forces (including contradiction) across the knowledge field.

This specification details their internal logic, algorithms, interaction protocols, and their integration with the broader system architecture.

1.2 Audience

This document is intended for Core Algorithm Developers and System Architects responsible for implementing and tuning the core cognitive dynamics of KIMERA SWM.

2.0 The Contradiction Engine

The Contradiction Engine is the primary driver of system evolution. Its function is not to eliminate conflict but to transform it into a structured, manageable form (`Scars`) that fuels learning and adaptation.

2.1 Tension Gradient Mapping

The engine's first task is to detect and quantify semantic tension.

- **Mechanism:** Contradictions are identified not as binary states but as **tension gradients (∇T)** within the semantic field. This gradient is measured between two or more `Geoids`.
- **Sources of Tension:** The engine detects tension from multiple sources:
    - **Symbolic Contradictions:** Explicit logical conflicts identified by the

    Symbolic Representation Layer (e.g., `(A, has_property, P)` vs. `(A, has_property, not_P)`). These contribute a high, sharp gradient.
  - **Embedding Misalignment:** Significant opposition or distance between the embedding vectors of related Geoids.
  - **Layer Conflict:** Disagreement between a Geoid's `semantic_state` and its `symbolic_state`.
- **Composite Tension Score (`T`):** The engine calculates a weighted, composite tension score to quantify the overall conflict: `T = α * |Vm| + β * Li + γ * Si + δ * Up`
  - Where `Vm` is vector misalignment, `Li` is layer conflict intensity, `Si` is influence from existing Scars, and `Up` is update pressure. The coefficients (`α, β, γ, δ`) are tunable system parameters.

2.2 Pulse Calculation and Differentiation

When a tension gradient `∇T` exceeds a predefined threshold, the engine generates a **semantic pulse**.

- **Pulse Strength (`Ps`):** The initial strength of the pulse is calculated based on the magnitude of the tension, the degree of axis misalignment, and the mutational coherence of the involved Geoids.
- **Collapse vs. Surge Differentiation:** The engine analyzes the stability and coherence of this pulse to determine the system's response. This is a critical decision point:
  1. **Collapse (Resolution):** If the pulse is strong, stable, and coherent, the engine initiates a "Collapse." This is a controlled resolution process that results in the formation of a permanent `Scar` and a translatable semantic output. This is the primary pathway for learning from a well-defined conflict.
  2. **Surge (Propagation):** If the pulse is unstable, weak, or ambiguous, the engine initiates a "Surge." It emits a **contradiction wave** (a transient `Echo`) that propagates through the semantic field via the SPDE. This action does not resolve the conflict but stimulates the surrounding field, seeking more information or resonance to clarify the ambiguity before a collapse is attempted.

2.3 Entropy-Based Pulse Scaling

To prevent system overload from numerous or highly complex contradictions, the engine uses semantic entropy as a throttling mechanism.

- **Mechanism:** Contradictions occurring in regions of already high semantic entropy (i.e., high uncertainty) are temporarily muted or have their pulse strength (`Ps`) scaled down.
- **Implementation:** High-entropy conflicts are stored in temporary **floating buffers**. They are held there until the local field entropy decreases, at which point they can be processed by the engine without risking systemic instability.

3.0 The Semantic Pressure Diffusion Engine (SPDE)

The SPDE models the entire semantic field as a dynamic pressure system, using a diffusion equation to propagate all semantic forces and drive the system toward coherent states.

3.1 The Pressure Field Components

The SPDE tracks and diffuses four distinct types of pressure. These forces are computed for each `Geoid` and their sum determines the Geoid's movement (drift).

1. **Resonance Pressure (`F_R`):** A positive, attractive force generated by the alignment of coherent or mutually supporting Geoids.
2. **Contradiction Tension (`F_C`):** A negative, repulsive force generated by conflicting Geoids, as detected by the `ContradictionEngine`.
3. **Void Suction (`F_V`):** A negative pressure exerted by `Void` entities, which acts as an entropy sink, drawing in nearby, weakly anchored Geoids.
4. **Drift Force Vectors (`F_D`):** A directional bias on the semantic manifold, arising from long-term trends or the instability of a cognitive axis (as reported by the ASM).

3.2 The Core Diffusion Algorithm

The SPDE's algorithm iteratively updates the pressure at each node (`p_i`) in the knowledge graph, ensuring that pressure flows from high-pressure to low-pressure regions in a controlled manner.

- **Graph Laplacian Model:** The core of the algorithm is a discrete diffusion (heat equation) model based on the graph Laplacian (`L = D - W`), where `D` is the degree matrix and `W` is the affinity matrix of the Geoid graph. The pressure update rule is: `p_i(t+1) = p_i(t) - α * Σ(L_ij * p_j(t))`
    - Where `α` is the diffusion rate constant. This ensures pressure flows along the strongest semantic links.

- **Non-Linear Dynamics:** The diffusion process is not purely linear. The SPDE incorporates several non-linear behaviors:
    - **Dampening:** Very high pressures experience friction, preventing runaway feedback loops.
    - **Amplification:** Under certain conditions (e.g., repeated resonance reinforcement), local pressure is boosted.
    - **Leak/Decay:** All pressures naturally decay over time (`(1 - λ) * p_i`) unless maintained, modeling the fading of non-reinforced activations.

3.3 Handling of Special Events

The SPDE is designed to manage exceptional pressure patterns that signal systemic instability.

- **Tension Lock State:** When contradiction tension in a subgraph exceeds a critical threshold, the SPDE freezes pressure flow in that region and can trigger **Semantic Prism Mode (SPM)**, a specialized routine to decompose the locked concepts.
- **Void Collapse/Cascade:** If a large void area forms, SPDE can experience a "void collapse" where pressure from surrounding nodes floods in. SPDE responds by increasing local leak rates to dissipate the surge.
- **Contradiction Shockwaves:** A major contradiction (e.g., a core concept flip) injects a strong pressure wave. SPDE handles this by temporarily increasing global dampening to absorb the shock and prevent system-wide oscillation.

4.0 Integration and Workflow

The `ContradictionEngine` and `SPDE` are at the heart of the KCCL's dynamic processing layer.

1. **Initial State:** The SPDE maintains a real-time pressure map of the entire semantic field.
2. **Tension Detection:** The `ContradictionEngine` continuously analyzes the SPDE's pressure map, along with symbolic checks, to identify tension gradients.
3. **Pulse and Response:** When a tension gradient exceeds its threshold, the `ContradictionEngine` generates a pulse and determines a `Collapse` or `Surge` response.
4. **Field Update:**

    - A **`Collapse`** event results in a `Scar`, which creates a permanent new feature in the semantic landscape that the SPDE must now account for in its pressure calculations.
    - A **`Surge`** event creates a transient `Echo` (a pressure wave) that is propagated throughout the field by the SPDE's diffusion algorithm.
5. **Feedback Loop:** The results of these actions (new Scars, dissipated Echoes, re-stabilized pressure fields) create a new semantic state, which is then continuously analyzed by the `ContradictionEngine`, thus completing the feedback loop.
1.

1.

DOC-205b: Memory Scar Compression Engine (MSCE) Specification

Version: 1.0 **Date:** June 7, 2025 **Status:** Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the detailed engineering specification for the **Memory Scar Compression Engine (MSCE)**. The MSCE is a core component of the `KimeraKernel` responsible for managing the lifecycle of `Scars` (persistent memory traces of contradictions).

The MSCE's primary function is to ensure the long-term coherence, efficiency, and adaptability of the system's memory. It achieves this by governing three fundamental processes: **scar decay** (passive forgetting), **scar fusion** (consolidation of redundant memories), and **scar crystallization** (the solidification of resolved insights into stable knowledge).

1.2 Scope and Dependencies

This specification details the algorithms, trigger conditions, and interaction protocols for the MSCE. It is dependent on:

- **DOC-201 (Core Data Structures):** For the definition of the `Scar` and `Geoid` schemas.
- **DOC-202 (Semantic Thermodynamics):** For the principles of decay and entropy.
- **DOC-205a (Contradiction Engine & SPDE):** As the SPDE provides key environmental inputs (e.g., `VP_local`) that modulate MSCE's behavior.

2.0 Scar Lifecycle Management

The MSCE governs the evolution of a `Scar` from its formation to its eventual archival or integration as permanent knowledge.

2.1 Scar Formation and Initial State

- **Trigger:** A `Scar` is created by the `ContradictionEngine` following a "Collapse" event.

- **Initial State:** Upon creation, a `Scar` is registered with the MSCE and enters the `Active` state. Its initial depth (D_0) is calculated by the Contradiction Engine based on factors including contradiction intensity (`C`), resonance (`R`), and exposure time (`E`): $D_0 = (C * R) * \log(E + 1) + U$ (where `U` is a user reinforcement score).
- **MSCE Role:** The MSCE logs the new `Scar` and begins tracking its thermodynamic properties and temporal evolution.

2.2 Mechanism 1: Temporal Decay (Passive Forgetting)

This is the default process for reducing the influence of unreinforced Scars over time.

- **Formal Model (Adaptive Exponential Decay):** The depth of a scar, `D(t)`, decays according to the formula: $D(t) = D_0 * \exp(-\lambda\_eff * t)$
- **Effective Decay Rate (`λ_eff`):** The decay rate is not static. It is dynamically calculated by MSCE based on the current semantic context: `λ_eff = λ_base * (1 + II_axis) * (1 + VP_local) * (1 - RR_factor)`
    - **`λ_base`:** The intrinsic decay rate determined by the `Scar`'s interpretive layer (e.g., metaphorical scars decay faster than structural ones).
    - **`II_axis`:** The **Instability Index** from the **ASM**. High instability on the Scar's native axis accelerates decay.
    - **`VP_local`:** The **Local Void Pressure** from the **SPDE**. High void pressure (semantic isolation) around the scarred Geoid accelerates decay.
    - **`RR_factor`:** The **Resonance Reinforcement** factor. Frequent, non-contradictory activation of the scarred Geoid slows decay.
- **Forgetting Threshold (ε):** If a Scar's depth `D(t)` falls below a system-wide threshold (e.g., $\varepsilon = 0.05$) **and** its local void pressure is high, the MSCE triggers archival. The Scar's status is changed to `latent`, and it is removed from the active semantic field.

2.3 Mechanism 2: Scar Fusion (Consolidation)

This process merges multiple, similar Scars into a single, more general memory trace to reduce redundancy.

- **Trigger Condition:** MSCE periodically evaluates pairs of active Scars. If the similarity `S(scar_i, scar_j)` exceeds a defined threshold (`θ_fusion`), a fusion event is triggered.

- **Similarity Criteria:** Similarity `S` is a composite score based on:
    1. Shared Geoids.
    2. Proximity in the same axis/layer context.
    3. Structural similarity of their `expression` fields.
    4. Temporal proximity of their creation.
- **Fusion Procedure:**
    1. A new, single `fused_scar` is created.
    2. Its `depth` is an aggregation of the source Scars (e.g., `max(D_i, D_j)` or a capped sum).
    3. Its metadata and `expression` are a synthesis of the source Scars.
    4. The `echo_history` from all source Scars is merged into the `fused_scar`.
    5. The original Scars are removed from the active field and archived with a status of `merged_into: fused_scar.scarID`.

2.4 Mechanism 3: Scar Crystallization (Solidification)

This is the process by which a persistent, reinforced Scar transforms into a stable, permanent piece of knowledge.

- **Trigger Condition:** A Scar becomes a candidate for crystallization if:
    1. Its depth `D(t)` remains consistently above a high threshold (`θ_crystal`, e.g., 0.85) for a minimum number of cycles (`N_min_cycles`).
    2. Simultaneously, the underlying contradiction pressure associated with the Scar is observed to be diminishing, indicating that the system is converging on a stable resolution.
- **Crystallization Procedure:**
    1. The Scar's status is marked as `crystallized`. It is now considered a **semantic anchor** and is highly resistant to decay (`λ_eff` is set to near zero).
    2. The MSCE signals the **Provisional Geoid Generator (PGG)**.
    3. The PGG may then instantiate a new, permanent `Geoid` that explicitly represents the resolved insight or concept that emerged from the contradiction.
    4. The original conflicting Geoids are then linked to this new "resolution Geoid," and the crystallized Scar is archived, with its legacy now embodied in the new knowledge structure.

3.0 Integration with Core System Components

The MSCE is a central hub for memory evolution and interacts deeply with other engines.

- **SPDE (Semantic Pressure Diffusion Engine):**
  - **Input from SPDE:** MSCE uses `VP_local` (Void Pressure) from the SPDE to calculate the `λ_eff` for scar decay.
  - **Output to SPDE:** When MSCE removes, fuses, or crystallizes a Scar, it alters the semantic landscape. The SPDE must then recalculate its pressure maps to reflect the new topology (e.g., removing a tension source or adding a new stable anchor).
- **PGG (Provisional Geoid Generator):**
  - MSCE triggers the PGG when a Scar crystallizes, providing the semantic "content" for the new, stable Geoid that needs to be created.
  - Conversely, MSCE is responsible for the decay and archival of `provisional geoids` that fail to achieve solidification.
- **ASM (Axis Stability Monitor):**
  - The `II_axis` (Instability Index) from the **ASM** is a direct input into MSCE's adaptive decay formula (`λ_eff`), ensuring that Scars on unstable axes fade more quickly.
- **ZPA (Zetetic Prompt API):**
  - MSCE's processes provide triggers for the ZPA. For example, a Scar nearing the forgetting threshold (`ε`) or a Scar reaching the crystallization threshold (`θ_crystal`) can trigger a ZPA prompt to involve the user in the decision.
- **SSL (Semantic Suspension Layer) & RCX (Reflective Cortex):**
  - SSL events (e.g., the freezing of a Geoid) are logged in the `Echo Trail` and can influence a Scar's properties. An SSL intervention on a Geoid will typically reinforce associated Scars, increasing their "trauma weight" and making them more resistant to decay.
  - The RCX visualizes the state of Scars as managed by MSCE, for example by showing their `depth` as a color intensity and their decay as a fading animation in the `Semantic Field Viewer`.

This tight integration ensures that the long-term evolution of memory is not an isolated process but is continuously shaped by, and in turn shapes, the real-time cognitive dynamics of the entire KIMERA SWM system.

DOC-205c: Provisional Geoid Generator (PGG) - Technical Specification

Version: 1.0 **Date:** June 7, 2025 **Status:** Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the detailed engineering specification for the **Provisional Geoid Generator (PGG)**. The PGG is a critical component of the `KimeraKernel` responsible for dynamically creating temporary knowledge nodes—"provisional geoids" or "sketch nodes"—when the system encounters novel concepts, semantic gaps, or unresolved tensions.

The PGG's primary function is to ensure cognitive continuity, preventing the system from halting when faced with unknown information. It embodies KIMERA's zetetic principles by creating structured placeholders for speculative concepts, allowing the system to reason about them before they are fully validated or integrated.

1.2 Scope and Dependencies

This specification details the PGG's trigger conditions, the schema and properties of provisional geoids, their managed lifecycle, and their integration with other core engines. It is dependent on:

- **DOC-201 (Core Data Structures):** For the base `Geoid` schema.
- **DOC-205a (Contradiction Engine & SPDE):** As the SPDE's pressure dynamics are a key trigger for PGG activation.
- **DOC-205b (MSCE):** As the MSCE manages the decay and archival of provisional geoids that fail to stabilize.

2.0 PGG Trigger Conditions

The PGG is activated only under specific conditions where the system's existing knowledge is insufficient or new conceptual space must be explored.

- **Unrecognized Term / Knowledge Void:** The primary trigger. When an input term cannot be confidently mapped to any existing Geoid in the

Meta-Knowledge Skeleton (MKS), the PGG is invoked to create a provisional geoid for the unknown term.

- **User-Initiated Speculative Inquiry:** A user can explicitly command the system to explore a hypothetical concept (e.g., "Suppose a concept X exists that..."), which directly triggers the PGG to instantiate `X` as a provisional geoid.
- **High-Tension Fusion Outcome:** When a manual or automated fusion of multiple Geoids results in an emergent concept that does not match any existing Geoid, the PGG creates a provisional node to capture this new, high-tension synthesis.
- **ZPA-Directed Bridging Concept:** The **Zetetic Prompt API (ZPA)** may identify a semantic gap and suggest a "bridging concept" to resolve a complex contradiction. The PGG is then invoked to instantiate this system-proposed hypothetical concept.
- **Emergent Semantic Gaps (SPDE Trigger):** The **SPDE** can trigger the PGG if it detects persistent, anomalous pressure patterns that suggest a missing structural concept. This includes:
    - A stable **void field** (entropy sink) where pressure consistently flows inward.
    - A **semantic surge** resulting from the intersection of multiple strong resonance flows, indicating an emergent but un-captured concept.

3.0 Provisional Geoid Generation Logic & Schema

When triggered, the PGG instantiates a provisional geoid with a specific set of properties designed to make it functional yet volatile.

3.1 Schema and Initial Properties

A provisional geoid adheres to the base `Geoid` schema but is initialized with specific flags and a sparse structure.

- **Unique Identifier and Type:** It is assigned a unique temporary ID (e.g., `PROV_G_<uuid>`) and its `lifespan_type` is set to `"provisional"`.
- **Origin Metadata:** The geoid is tagged with its creation context (e.g., `"PGG_User_Input"`, `"PGG_ZPA_Suggestion"`), tracing it back to the specific event that triggered the PGG.
- **Sparse Layer Scaffold:** The geoid's semantic layers are created as minimal placeholders:
    - **Literal Layer:** Seeded with the raw term or phrase.
    - **Metaphorical/Symbolic Layers:** Initialized with generic placeholders

    (e.g., `"unknown_metaphorical_significance"`).
  - **Structural Layer:** Set to `"undefined_entity"`.
- **Thermodynamic & Stability State:**
  - **Activation (SE):** Set to a moderate initial level to ensure it participates in the immediate reasoning cycle.
  - **Stability Indices:** Initialized to a highly volatile state: `stability_index` is low (e.g., 0.1) and `instability_index` is high (e.g., 0.9).
  - **Void Pressure:** A moderate `void_pressure` is applied, representing the system's natural tendency to reclaim unanchored concepts. This ensures the geoid will decay if not reinforced.
- **Initial Connections:** The provisional geoid begins with no strong links. The SPDE immediately attempts to form weak, tentative resonance and contradiction links to contextually relevant neighbors, giving it a starting point for integration.

4.0 The Lifecycle of a Provisional Geoid

A provisional geoid is not permanent. It exists in a transient state and must either solidify into a stable concept or be pruned from the system.

4.1 Active Provisional Stage

Upon creation, the geoid is active in the semantic field. It can form links, participate in contradictions (and thus acquire Scars), and is subject to drift dynamics. The system closely monitors its stability metrics during each cognitive cycle.

4.2 Solidification (Crystallization) Criteria

A provisional geoid transitions to a stable, permanent geoid if it meets one or more of the following validation criteria, which demonstrate its semantic value and coherence.

1. **User Confirmation/Elaboration:** The most direct path. A user, via the ZNL interface, explicitly provides a definition, assigns a type, or links the provisional geoid to established concepts.
2. **Resonance Anchoring:** The provisional geoid organically develops strong, stable resonant links with multiple established geoids, indicating it fills a valid semantic niche.
3. **Contradiction Resolution:** The provisional geoid plays a key, verifiable role in resolving a pre-existing contradiction or tension lock, proving its utility.

4. **Echo Reinforcement:** The provisional geoid is repeatedly and consistently activated across multiple interaction cycles, indicating its relevance to the ongoing cognitive process.

Upon solidification, its `lifespan_type` is changed to `"stable"`, it is assigned a permanent ID, and it is fully integrated into the persistent MKS.

4.3 Decay and Pruning Criteria

If a provisional geoid fails to solidify, it is pruned to maintain the health of the semantic field.

1. **Time/Interaction Limit:** The geoid is marked for decay if it persists for a defined number of semantic cycles without meeting any solidification criteria.
2. **Persistent High Instability:** If the geoid's `instability_index` remains critically high and it continuously generates noise or unresolved contradictions, the system will flag it for pruning.
3. **MSCE-Managed Decay:** The **MSCE** is responsible for executing the decay process, severing the geoid's links, reducing its activation to zero, and archiving its existence in the latent memory's `Echo Trail`.
4. **ZPA-Prompted User Confirmation:** Before final pruning, the ZPA typically issues a prompt to the user (e.g., "The provisional concept 'X' is unstable and fading. Elaborate, link, or allow decay?"), providing a final opportunity to rescue the concept.
5. **SSL/ERL-Triggered Dissolution:** In extreme cases where a provisional geoid is identified as a source of catastrophic instability (**SSL**) or violates ethical constraints (**ERL**), it can be immediately quarantined or dissolved to protect system integrity.

5.0 Integration with Core System Components

The PGG is deeply intertwined with KIMERA's other dynamic engines.

- **SPDE:** The SPDE is a primary trigger for the PGG, signaling when its pressure maps reveal semantic gaps or unresolved tension peaks. Once a provisional geoid is created, it is injected into the SPDE's field as a new, low-intensity pressure source.
- **MSCE:** The MSCE manages the "forgetting" of failed provisional geoids, applying accelerated decay rules. Conversely, when the MSCE crystallizes a persistent `Scar` into a new insight, it signals the PGG to instantiate a new, stable `Geoid` to represent that learned concept.

- **ZPA & RCX:** The ZPA can directly trigger the PGG by suggesting a bridging concept. Subsequently, the ZPA and RCX manage the user interaction loop around the provisional geoid, generating prompts for elaboration and mirroring its unstable state in the `Semantic Field Viewer`.
- **Scar System:** Provisional geoids are fully integrated into the contradiction lifecycle. They can be the source of contradictions and can acquire `Scars`, which in turn influence their stability and potential for solidification.

-

DOC-205d: Axis Stability Monitor (ASM) - Technical Specification

Version: 1.0 **Date:** June 7, 2025 **Status:** Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the detailed engineering specification for the **Axis Stability Monitor (ASM)**. The ASM is a critical component of the `KimeraKernel`'s Language Axis Dynamics (LAD) system. It functions as an internal "gyroscope," responsible for continuously monitoring the health, coherence, and risk associated with each active cognitive "axis" (e.g., a language, a cultural context, a symbolic system).

The ASM's primary purpose is to detect and mitigate axis-induced semantic drift and distortion, ensuring the overall stability of KIMERA's multi-axial semantic field. It achieves this by computing quantitative stability metrics and triggering corrective actions when defined thresholds are breached.

1.2 Scope and Dependencies

This specification details the ASM's core metrics, monitoring logic, correction mechanisms, and its integration with other system components. It is dependent on:

- **DOC-201 (Core Data Structures):** For the definition of the `Geoid` schema.
- **DOC-205a (Contradiction Engine & SPDE):** As the SPDE's drift calculations are a key input for the ASM.
- **DOC-205e (Semantic Suspension Layer):** As the SSL is the ultimate failsafe triggered by the ASM in cases of catastrophic instability.

2.0 Core Concepts & Stability Metrics

The ASM relies on a suite of quantitative metrics to assess the stability of each cognitive axis.

2.1 Instability Index (II)

- **Definition:** The Instability Index is a composite score [0, 1] that measures the overall volatility and risk associated with an axis. A high `II` indicates an unstable axis prone to generating contradictions and semantic distortion.

- **Formal Calculation:** `II_axis = α * CD_axis + β * (1 - ActivationRate_axis) + γ * DriftVariance_axis + δ * AvgSLF_axis`
    - `α, β, γ, δ`: Tunable weighting coefficients.
    - `CD_axis`: **Contradiction Density** involving this axis.
    - `ActivationRate_axis`: The frequency of successful, coherent use of this axis.
    - `DriftVariance_axis`: The observed semantic volatility of Geoids when processed under this axis.
    - `AvgSLF_axis`: The average Semantic Loss Factor associated with rotations to/from this axis.

2.2 Semantic Loss Factor (SLF)

- **Definition:** The SLF quantifies the "cost" or degree of information loss incurred when a `Geoid` is rotated from one axis to another. An SLF near 0 indicates a faithful transformation; a value near 1 indicates severe decoherence.
- **Formal Calculation:** `SLF = 1 - (AvgNodeSim_post × ScarRetentionRate × ResonanceStability_post)`
    - `AvgNodeSim_post`: The average vector similarity of the Geoid's core semantics before and after rotation.
    - `ScarRetentionRate`: The fraction of the Geoid's `Scars` that remain coherent and applicable post-rotation.
    - `ResonanceStability_post`: A measure of how well the Geoid's resonant links to its neighbors are preserved after rotation.

2.3 Other Key Monitored Metrics

- **Axis Rotation Rate (RR):** The frequency of `Geoid` rotations into or out of an axis per unit time.
- **Drift Velocity (DV):** The rate of change of a `Geoid`'s embedding vector along a specific axis, derived from the SPDE's drift field calculations.
- **Cross-Axis Divergence:** A measure of how differently two axes interpret the same `Geoid`, calculated by comparing the Geoid's representations under each axis.

3.0 ASM Operational Logic

The ASM is a proactive monitoring process that runs continuously within the KCCL.

- **Real-Time Metric Tracking:** The ASM computes and updates the stability metrics for each active axis in real-time. This data is stored in a **Minimal Axis Profiling Table (MAPT)** for efficient access.
- **Cross-Axis Consistency Checks:** The ASM constantly compares a Geoid's representation across different axes. If the divergence between two axial interpretations of the same Geoid exceeds a threshold (`D_thresh`), it flags an **Axis Conflict Zone**.
- **Thresholds and Alerts:** The ASM uses a set of configurable thresholds for each metric (e.g., `II_alert_threshold = 0.8`, `SLF_alert_threshold = 0.7`). When a metric breaches its threshold, the ASM triggers an alert and initiates the appropriate corrective action.
- **Logging and Historical Analysis:** All significant drift events, metric spikes, and corrective actions are logged in the Geoid's `Echo Trail`. This historical data is used by the ASM to learn and predict future instabilities.

4.0 Axis Correction Mechanisms

When instability is detected, the ASM triggers one of several targeted interventions to restore semantic coherence.

1. **Axis Realignment & Reinforcement:**
    - **Trigger:** Moderate but sustained increase in an axis's `II`.
    - **Action:** The Language Axis Dynamics (LAD) controller initiates a field realignment. This involves strengthening the resonant links of drifting Geoids to nearby stable "anchor" nodes and re-calibrating the drifting axis's internal parameters (e.g., its cultural weights or layer clarity).
2. **Dampening and Suppression of Unstable Axes:**
    - **Trigger:** An axis's `II` consistently exceeds its safe threshold.
    - **Action:** The LAD controller throttles the influence of the unstable axis. This can involve reducing the axis's global activation rate or temporarily "muting" its influence on specific, highly volatile Geoids.
3. **Meta-Contradiction Node (MCN) Generation:**
    - **Trigger:** A persistent, high-scoring **Axis Conflict Zone** is detected between two axes for the same Geoid.
    - **Action:** To prevent the conflict from destabilizing the core Geoid, the system spawns a new, specialized `Meta-Contradiction Node`. This MCN encapsulates the specific conflict, linking to the two divergent interpretations and allowing the system to reason about the contradiction itself at a meta-level.

4. **Emergency SSL Activation:**
    - **Trigger:** An "Axis Drift Catastrophe" is detected (e.g., a cascadingly high SLF and a spike in `Total Semantic Pressure` from the SPDE).
    - **Action:** The ASM invokes the **Semantic Suspension Layer (SSL)**, which executes its emergency stabilization protocol (freeze, snapshot, compress) to prevent systemic decoherence.

5.0 Integration with Core System Components

The ASM is a highly interconnected subsystem that forms a critical part of KIMERA's self-regulatory feedback loops.

- **SPDE:** ASM provides the `Instability Index (II)` for each axis to the SPDE. The SPDE uses this value to modulate the `Drift Force Vectors` in its pressure calculations, ensuring that unstable axes generate stronger corrective drift. Conversely, the ASM reads the drift and pressure data from the SPDE to compute its own stability metrics.
- **MSCE:** The `II_axis` calculated by the ASM is a direct input into the MSCE's adaptive decay formula for `Scars`. This ensures that Scars formed on unstable, unreliable axes decay faster and have less long-term impact on the system's memory.
- **SSL:** The ASM is the primary trigger for the Semantic Suspension Layer, invoking it when its metrics indicate catastrophic instability that cannot be handled by lesser correction mechanisms.
- **ZPA & RCX:** The ASM's findings are surfaced to the user through the ZPA and RCX.
    - The `RCX` mirror map can visualize ASM data, for example by showing drift vectors or highlighting nodes under high axial stress.
    - The `ZPA` can generate prompts based on ASM alerts, for example, warning a user that a planned axis rotation will incur a high SLF, or asking for help in resolving a detected `Axis Conflict Zone`.

○

DOC-205e: Semantic Suspension Layer (SSL) - Technical Specification

Version: 1.0
Date: June 7, 2025
Status: Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the detailed engineering specification for the **Semantic Suspension Layer (SSL)**. The SSL is KIMERA SWM's critical failsafe mechanism, designed to "catch" and contain Geoids or entire semantic field regions that are experiencing catastrophic instability and are at risk of decoherence or collapse.

The SSL's function is not to resolve conflicts but to **stabilize** them. It acts as an emergency circuit-breaker, freezing and compressing volatile concepts to prevent their instability from propagating across the system. This specification details the SSL's activation triggers, its multi-stage stabilization protocol, post-intervention recovery outcomes, and its integration with other core engines.

1.2 Scope and Dependencies

This specification details the operational logic of the SSL. It is dependent on:

- **DOC-205d (Axis Stability Monitor):** As the ASM is a primary source of triggers for SSL activation.
- **DOC-205a (Contradiction Engine & SPDE):** As the SPDE's pressure metrics are also key triggers.
- **DOC-201 (Core Data Structures):** For the definition of the Geoid and Scar schemas.
- **DOC-205f (ZPA & RCX):** As the ZPA and RCX are the primary means of communicating SSL states and recovery options to the user.

2.0 SSL Activation Triggers & Instability Metrics

The SSL is an emergency system that activates only when other self-regulatory mechanisms have failed to contain instability. Its triggers are based on a confluence of critical metrics crossing predefined safety thresholds.

- **Primary Triggers:**
  1. **Axis-Drift Catastrophe:** Detected by the **ASM** when a Geoid experiences a cascading **Semantic Loss Factor (SLF)** during axis rotation, signaling imminent decoherence.
  2. **Critical Total Semantic Pressure (TSP):** Detected by the **SPDE** when the

    aggregate pressure (from contradiction, void suction, and drift) on a Geoid exceeds a critical threshold (e.g., normalized TSP ≥ +4.5).
  3. **Core Resonance Collapse:** When a Geoid's internal coherence (resonance across its own layers/axes) or its anchoring to stable concepts falls below a minimum viability threshold (e.g., $R < 0.3$).
  4. **Recursive Contradiction Loop:** Detected by the **SPDE** when a Geoid is caught in a runaway, oscillating cycle of contradictions that fails to dampen.
- **High-Risk Proactive Trigger:** The **Ethical Reflex Layer (ERL)** can preemptively invoke the SSL to freeze a Geoid if its content is flagged as generating extreme ethical or epistemic risk, even before other instability metrics are critical.

3.0 The Semantic Suspension Protocol

When activated, the SSL executes a multi-stage protocol designed to rapidly contain instability and preserve as much semantic information as possible.

1. **Step 1: Semantic Freezing & Pulse Deceleration**
   - The global KCCL update pulse (the system's "heartbeat") is immediately slowed (e.g., from ~0.5s to 2.0-2.5s per cycle).
   - All active processing (e.g., SPDE diffusion, Echoform transformations) on the target Geoid and its immediate semantic neighborhood is halted. This acts as a circuit-breaker to prevent the instability from propagating.
2. **Step 2: Geoid State Snapshot (SSL Vault)**
   - A complete, high-fidelity snapshot of the endangered Geoid's state is captured. This includes its semantic_state, symbolic_state, all associated Scars and metadata, and its current links to neighboring Geoids.
   - This snapshot is stored in a secure, isolated **"SSL Vault"** and serves as a definitive rollback point for recovery.
3. **Step 3: Dimensional Compression**
   - The SSL identifies the most volatile or conflicting semantic layers or axes of the Geoid.
   - It then temporarily suppresses or "folds" these high-entropy components inward, preserving only the stable core of the Geoid (e.g., its literal or most heavily reinforced layers). This reduces internal friction and makes the Geoid temporarily less complex but more stable.
4. **Step 4: Resonance Re-anchoring**
   - The SSL attempts to re-stabilize the compressed Geoid by strengthening its resonant links to nearby, known-stable "anchor" Geoids.
   - In cases of extreme decoherence, the SSL may temporarily substitute the fractured Geoid with a more abstract parent concept from the ontology to

maintain the structural integrity of the surrounding knowledge graph.

5. **Step 5: Axis Counter-Pressure & Isolation**
   - If a specific cognitive axis was identified by the ASM as the source of the instability, the SSL explicitly "mutes" that axis's influence on the Geoid.
   - It can apply a **counter-pressure** vector via the SPDE to nullify the drift force from the problematic axis, or it can completely isolate the Geoid from that axis's semantic field until stability returns.
6. **Step 6: Critical Event Logging**
   - All trigger conditions and actions taken by the SSL are recorded as a critical event in the Geoid's Echo Trail. This log entry is flagged with a high priority and includes the pre-suspension state snapshot from the SSL Vault.

4.0 Deactivation & Post-Suspension Recovery Outcomes

The SSL remains engaged until the local semantic pressure and instability metrics subside below safety thresholds. Upon deactivation, the system evaluates the state of the suspended Geoid and classifies it into one of the following recovery outcomes:

1. **Full Internal Recovery:** The SSL interventions succeed completely. The Geoid is re-stabilized, all layers are coherently restored, and it rejoins the active semantic field. The event is recorded as a new, deep Scar, signifying a successfully overcome crisis.
2. **Stabilized but Compressed State:** A partial recovery. The Geoid is stable but some of its semantic layers or axes remain suppressed to prevent a recurrence of instability. It is functional but semantically impoverished. The **ZPA** will then be triggered to prompt the user for a guided "re-inflation" or reintegration of the compressed layers at a later, more stable time.
3. **Frozen / Semantic Quarantine:** If automatic recovery fails, the SSL leaves the Geoid in a "frozen" state, isolated from all active reasoning. It is inert but not deleted. The **ZPA** issues a high-priority alert to the user, recommending manual review and intervention via the ZNL interface.
4. **Controlled Dissolution (Void Creation):** The outcome of last resort. If the Geoid is deemed irrecoverably unstable or non-essential, the SSL can orchestrate its dissolution. The Geoid is removed from the active field, and its semantic space is replaced with a structured **Void** entity. Its core contradiction Echoes are preserved as latent memory traces within the void, ensuring that the reason for the collapse is not completely lost.

5.0 Integration with Core System Components

The SSL is a deeply integrated failsafe that works in concert with KIMERA's other primary engines.

- **ASM & SPDE:** The ASM and SPDE are the primary sources of triggers for the SSL. The SSL, in turn, directly modulates the SPDE by slowing the system pulse and altering the local field topology through its stabilization actions.
- **MSCE (Memory Scar Compression Engine):** SSL events have a profound impact on a Geoid's memory. An SSL intervention heavily reinforces the associated Scars, increasing their "trauma weight" and making them highly resistant to decay within the MSCE's lifecycle management.
- **ZPA & RCX (User Interaction):** The SSL relies on the ZPA to communicate its state and recovery options to the user. The RCX mirror map visually represents SSL states, for example by showing a "frozen" or "compressed" Geoid, ensuring transparency in the system's emergency operations.

DOC-205f: ZPA & Reflective Cortex (RCX) - Technical Specification

Version: 1.0
Date: June 7, 2025
Status: Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the detailed engineering specification for two deeply interconnected engines that manage the system's metacognitive and interactive functions:

1. **The Reflective Cortex (RCX):** The semantic attention and self-observation layer responsible for mirroring the system's internal state.
2. **The Zetetic Prompt API (ZPA):** The autonomous provocation engine responsible for generating inquisitive prompts to guide user exploration and resolve semantic tensions.

Together, these components form the primary interface for human-AI cognitive collaboration within KIMERA SWM.

1.2 Scope and Dependencies

This specification details the architecture, logic, and interaction protocols for RCX and ZPA. It is dependent on:

- **All core data structures (DOC-201):** Especially the Geoid and Scar schemas.
- **All dynamic engine outputs (DOC-205a-e):** As the RCX and ZPA are triggered by and must interpret the states of the SPDE, MSCE, ASM, and SSL.

2.0 The Reflective Cortex (RCX)

The RCX is KIMERA's "conscious membrane." Its function is not to parse commands but to treat all user inputs as perturbations in the semantic field and to mirror the system's resulting internal state back to the user.

2.1 The Echo Trail: The Biography of a Geoid

The core data source for RCX is the **Echo Trail**, a rich, time-stamped history log maintained for each Geoid.

- **Structure:** An echo_history is an ordered list of events. Each entry contains:
    - timestamp: The semantic cycle count of the event.
    - event_type: The nature of the event (e.g., CONTRADICTION_DETECTED, USER_IMPRINT, AXIS_ROTATION, SCAR_CRYSTALLIZED).

    - context: A payload containing relevant data (e.g., the ID of the conflicting Geoid, the axis involved, the change in scar depth).
- **Function:** The Echo Trail provides the complete "biography" of a Geoid. By traversing this trail, the RCX can reconstruct the history of any concept, including the contradictions it has faced, the resolutions it has undergone, and the user interactions that have shaped it.

2.2 The Mirror Map: Reflecting the Internal State

The primary output of the RCX is the **Mirror Map**, a real-time representation of KIMERA's cognitive landscape. It is not an answer, but a reflection intended to provoke inquiry.

- **Construction Logic:** The Mirror Map is constructed by overlaying data from multiple sources:
    1. **SPDE State:** The current semantic pressure map from the SPDE provides the base topology, showing hotspots of Contradiction Tension and areas of Void Suction.
    2. **MSCE Scar Data:** The RCX queries the MSCE for the current state of active Scars. Scar depth and age are used to visually represent their influence (e.g., as colored deformations or halos on Geoids).
    3. **ASM Axis State:** The current state of all cognitive axes from the ASM is reflected, highlighting any axes that are unstable or muted.
    4. **Echo Trail Resonance:** The RCX analyzes the Echo Trails of activated Geoids to detect recurring patterns or reactivated historical conflicts, which are then highlighted on the map.
    5. **User Input Perturbation:** The semantic "shape" of the user's most recent input is superimposed, showing which Geoids were directly perturbed and how the pressure field responded.

3.0 The Zetetic Prompt API (ZPA)

The ZPA is the autonomous provocation engine that embodies KIMERA's inquisitive ethos. It surfaces internal tensions as targeted prompts for the user.

3.1 ZPA Trigger Conditions

The ZPA continuously monitors system-wide volatility indicators. A prompt is generated when a metric crosses a defined threshold.

- **Primary Triggers:**
    - **High Contradiction Tension:** A new or escalating Scar exceeds a tension threshold.
    - **Axis/Layer Friction:** The **ASM** reports high Instability Index (II) or Semantic Loss Factor (SLF) for an axis.

    - **Void Expansion:** The **SPDE** reports that Void Pressure in a region is growing, threatening a concept with collapse.
    - **SSL Activation:** Any emergency intervention by the **SSL** triggers a high-priority ZPA alert.
    - **User Engagement (PEDI):** The **Prompt Engagement Drift Index (PEDI)** tracks user avoidance. High PEDI can trigger a more direct or disruptive prompting style.

3.2 Prompt Formulation and Types

When triggered, the ZPA selects an appropriate prompt template and instantiates it with context from the semantic field.

- **Prompt Types (Examples):**
    - **Fracture Trace:** Invites the user to explore the fault line of a new contradiction (e.g., *"Innovation' and 'Societal Good' are in high tension. Trace this fracture through the 'Ethics_Axis'?"*).
    - **Void Navigation:** Highlights a knowledge gap and asks for input (e.g., *"It's unclear how organic material returns to the soil. Is there a process we're missing?"*).
    - **Echo Scar Leap:** Connects a current issue to a past, related conflict from the Echo Trail.
    - **Stability Provocation:** Challenges a period of low tension to uncover hidden assumptions.
    - **SSL/ERL Alerts:** Issues warnings or requests for manual review when a failsafe is triggered (e.g., *"Geoid 'X' has entered semantic quarantine due to irrecoverable instability. Manual review is recommended."*).

3.3 Prompt Lifecycle Management

ZPA manages prompts to ensure they are timely and relevant, not overwhelming.

1. **Creation & Prioritization:** A triggered prompt is created and assigned a **Zetetic Potential Score (ZPS)** based on urgency and relevance to the user's current focus. Only the highest ZPS prompts are typically displayed.
2. **Active State:** The prompt is presented to the user with a defined **Time-To-Live (TTL)**.
3. **Resolution:** If the user engages with the prompt (e.g., answers a question, follows a suggestion), the prompt is marked as Resolved and archived.
4. **Dismissal/Expiration:** If the user dismisses the prompt or its TTL expires, it is archived as Dismissed or Expired. A "micro-scar" is logged on the involved Geoids, indicating user aversion, which influences future ZPS calculations.
5. **Archival & Reactivation:** All prompts are stored in a log. "Latent" (unresolved) prompts can be reactivated if their underlying trigger condition

recurs or escalates in severity.

4.0 Integration and Workflow

The RCX and ZPA form a tight, symbiotic loop at the apex of the KCCL.

- **RCX -> ZPA:** The RCX's mirroring of the internal state is the primary data source for the ZPA. The ZPA analyzes the tensions, voids, and historical echoes surfaced by RCX to identify promptable moments.
- **ZPA -> RCX/User:** The ZPA delivers its generated prompts through the user interface layer (the Zetetic Navigation Layer, or ZNL), which is conceptually part of the RCX's reflective output.
- **User -> RCX:** The user's response to a ZPA prompt, or any other interaction, is treated as a new perturbation by RCX, which then generates an updated Mirror Map, thus continuing the cycle.
- **Integration with other Engines:**
    - **SPDE/ASM/SSL:** Provide the real-time state metrics (pressure, instability) that RCX mirrors and ZPA uses as triggers.
    - **MSCE:** Provides the historical scar data that RCX uses to construct Echo Trails. ZPA prompts can influence the MSCE's decisions (e.g., a user's choice to "allow decay" can accelerate a Scar's archival).

This workflow creates a sophisticated human-in-the-loop system where the AI does not just respond to the user but actively engages them in a structured, reflective dialogue about the state of its own knowledge.

DOC-301: KIMERA SWM - API Reference & Integration Guide

Version: 1.0
Date: June 7, 2025
Status: Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the definitive technical reference for the external-facing Application Programming Interfaces (APIs) of the KIMERA SWM system. All interactions with the KimeraKernel from external clients, applications, or services are modulated through the **Interface Control Weave (ICW)**, which exposes these endpoints.

This guide details the available endpoints, request/response formats, authentication methods, error codes, and provides examples to facilitate system integration.

1.2 Audience

This document is intended for Application Developers and System Integrators who need to build applications on top of, or integrate external systems with, KIMERA SWM.

1.3 General Principles & Conventions

- **Protocol:** All API endpoints are served over HTTPS.
- **Base URL:** https://api.kimera-swm.system/v1
- **Authentication:** All requests must be authenticated using mutual TLS (mTLS), where both the client and server present and validate certificates. Unauthorized requests will result in a 401 Unauthorized error.
- **Data Format:** All request and response bodies are in application/json format.
- **Idempotency:** GET requests are idempotent. POST requests for creation are not idempotent.
- **Versioning:** The API version is included in the URL path (/v1).

2.0 Input & Ingestion API

This set of endpoints is used to introduce new information into the KIMERA SWM system, triggering the Kimera Core Cognitive Loop (KCCL).

2.1 Ingest Raw Text

This is the primary endpoint for feeding unstructured text into the system. The ICW routes this input to the EcoForm subsystem for parsing and initial Geoid creation.

- **Endpoint:** POST /ingest/text
- **Description:** Submits a raw text string for processing. The system will asynchronously parse the text, create or update relevant Geoids, and initiate a cognitive cycle.
- **Request Body:**
```
{
  "session_id": "session_uuid_123",
  "text_input": "In this story, there is a concept of 'dream gravity'...",
  "source_language": "en-US",
  "context_metadata": {
    "domain": "narrative_fiction",
    "user_id": "user_abc"
  }
}
```
- **Success Response (202 Accepted):** The request has been accepted for processing. The response includes a transaction ID to track the progress of the cognitive cycle.
```
{
  "transaction_id": "txn_uuid_456",
  "status": "processing_initiated",
  "message": "Text input accepted and routed to KCCL for processing."
}
```
- **Error Responses:**
  - 400 Bad Request: If the request body is malformed or text_input is empty.
  - 401 Unauthorized: Authentication failure.

3.0 Query & State Retrieval API

These endpoints allow for retrieving information about the current state of the semantic field and its entities.

3.1 Get Geoid State

Retrieves the current state of a specific Geoid.

- **Endpoint:** GET /geoid/{geoid_id}
- **Description:** Fetches the complete data structure for a given Geoid, including its semantic and symbolic states, and associated metadata.
- **Path Parameter:**
  - geoid_id (string): The unique identifier of the Geoid.

- **Success Response (200 OK):**
  - Returns the full Geoid object as defined in **DOC-201**.
- **Error Responses:**
  - 404 Not Found: If no Geoid with the specified ID exists.

3.2 Get Scar Details

Retrieves the full record of a specific Scar.

- **Endpoint:** GET /scar/{scar_id}
- **Description:** Fetches the complete, comprehensive data structure for a given Scar from the MemoryScarRepository.
- **Path Parameter:**
  - scar_id (string): The unique identifier of the Scar.
- **Success Response (200 OK):**
  - Returns the full Scar object as defined in **DOC-201**.
- **Error Responses:**
  - 404 Not Found: If no Scar with the specified ID exists.

3.3 Query Semantic Field

Performs a complex query against the semantic field to find Geoids matching certain criteria.

- **Endpoint:** POST /query/semantic-field
- **Description:** Submits a query to find Geoids based on semantic similarity to a vector, symbolic properties, or proximity to other Geoids or Scars.
- **Request Body:**
```
{
  "query_vector": [0.1, 0.2, /* ... */],
  "symbolic_filters": {
    "metadata.source": "PGG_User_Input"
  },
  "proximity_to_geoid": "GEOID_12345",
  "max_distance": 0.5,
  "limit": 10
}
```
- **Success Response (200 OK):**
```
{
  "query_id": "query_uuid_789",
  "results": [
    {
      "geoid_id": "GEOID_67890",
      "score": 0.85,
      "distance": 0.32
    }
  ]
}
```

4.0 Interaction & Control API

These endpoints are used to interact with the system's metacognitive functions, primarily the ZPA and RCX.

4.1 Retrieve Active Prompts

Fetches the current list of active zetetic prompts generated by the ZPA.

- **Endpoint:** GET /interaction/prompts
- **Description:** Retrieves high-priority prompts that the system has generated for user interaction. This is the primary mechanism for an application to display ZPA questions.
- **Success Response (200 OK):**

```
{
  "prompts": [
    {
      "prompt_id": "ZPA_PROMPT_uuid",
      "prompt_type": "FractureTrace",
      "text": "'Innovation' and 'Societal Good' are in high tension. Trace this fracture through the 'Ethics_Axis'?",
      "zps_score": 0.92,
      "context": {
        "involved_geoids": ["GEOID_Innovation", "GEOID_SocietalGood"],
        "related_scar_id": "SCAR_123"
      }
    }
  ]
}
```

4.2 Respond to a Prompt

Allows a user or external system to respond to a specific ZPA prompt.

- **Endpoint:** POST /interaction/prompts/{prompt_id}/respond
- **Description:** Submits a response to a prompt, which is treated as a new perturbation in the semantic field and triggers a KCCL cycle.

- **Path Parameter:**
  - prompt_id (string): The ID of the prompt being responded to.
- **Request Body:**
```
{
  "session_id": "session_uuid_123",
  "response_type": "user_elaboration",
  "payload": {
    "text_input": "The key is to balance disruptive innovation with social safety nets."
  }
}
```
- **Success Response (202 Accepted):**
```
{
  "transaction_id": "txn_uuid_789",
  "status": "response_accepted_for_processing"
}
```

5.0 Integration Guide

5.1 Typical Interaction Workflow

A typical client application would interact with KIMERA SWM following this general workflow:

1. **Submit Information:** The user provides text input, which the client sends to the POST /ingest/text endpoint. The client receives a transaction_id.
2. **Poll for Active Prompts:** The client application periodically calls GET /interaction/prompts to check if the system has generated any questions or suggestions in response to the input or its own internal state.
3. **Display Prompts:** If prompts are returned, the client's UI displays them to the user. The prompt's context can be used to highlight relevant concepts in the application's view.
4. **Submit User Response:** The user's interaction with a prompt (e.g., answering a question, clicking a "trace" button) is sent back to the system via POST /interaction/prompts/{prompt_id}/respond. This again triggers a KCCL cycle.
5. **Query for State Changes:** After submitting responses or new inputs, the client can use the GET /geoid/{geoid_id} or POST /query/semantic-field endpoints to see how the system's knowledge has evolved.

This loop of **Ingest -> Poll -> Display -> Respond** forms the basis of a collaborative, co-evolutionary dialogue with the KIMERA SWM system.

DOC-205f: ZPA & Reflective Cortex (RCX) - Technical Specification

Version: 1.0
Date: June 7, 2025
Status: Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the detailed engineering specification for two deeply interconnected engines that manage the system's metacognitive and interactive functions:

1. **The Reflective Cortex (RCX):** The semantic attention and self-observation layer responsible for mirroring the system's internal state.
2. **The Zetetic Prompt API (ZPA):** The autonomous provocation engine responsible for generating inquisitive prompts to guide user exploration and resolve semantic tensions.

Together, these components form the primary interface for human-AI cognitive collaboration within KIMERA SWM.

1.2 Scope and Dependencies

This specification details the architecture, logic, and interaction protocols for RCX and ZPA. It is dependent on:

- **All core data structures (DOC-201):** Especially the Geoid and Scar schemas.
- **All dynamic engine outputs (DOC-205a-e):** As the RCX and ZPA are triggered by and must interpret the states of the SPDE, MSCE, ASM, and SSL.

2.0 The Reflective Cortex (RCX)

The RCX is KIMERA's "conscious membrane." Its function is not to parse commands but to treat all user inputs as perturbations in the semantic field and to mirror the system's resulting internal state back to the user.

2.1 The Echo Trail: The Biography of a Geoid

The core data source for RCX is the **Echo Trail**, a rich, time-stamped history log maintained for each Geoid.

- **Structure:** An echo_history is an ordered list of events. Each entry contains:
    - timestamp: The semantic cycle count of the event.
    - event_type: The nature of the event (e.g., CONTRADICTION_DETECTED, USER_IMPRINT, AXIS_ROTATION, SCAR_CRYSTALLIZED).

    - context: A payload containing relevant data (e.g., the ID of the conflicting Geoid, the axis involved, the change in scar depth).
- **Function:** The Echo Trail provides the complete "biography" of a Geoid. By traversing this trail, the RCX can reconstruct the history of any concept, including the contradictions it has faced, the resolutions it has undergone, and the user interactions that have shaped it.

2.2 The Mirror Map: Reflecting the Internal State

The primary output of the RCX is the **Mirror Map**, a real-time representation of KIMERA's cognitive landscape. It is not an answer, but a reflection intended to provoke inquiry.

- **Construction Logic:** The Mirror Map is constructed by overlaying data from multiple sources:
    1. **SPDE State:** The current semantic pressure map from the SPDE provides the base topology, showing hotspots of Contradiction Tension and areas of Void Suction.
    2. **MSCE Scar Data:** The RCX queries the MSCE for the current state of active Scars. Scar depth and age are used to visually represent their influence (e.g., as colored deformations or halos on Geoids).
    3. **ASM Axis State:** The current state of all cognitive axes from the ASM is reflected, highlighting any axes that are unstable or muted.
    4. **Echo Trail Resonance:** The RCX analyzes the Echo Trails of activated Geoids to detect recurring patterns or reactivated historical conflicts, which are then highlighted on the map.
    5. **User Input Perturbation:** The semantic "shape" of the user's most recent input is superimposed, showing which Geoids were directly perturbed and how the pressure field responded.

3.0 The Zetetic Prompt API (ZPA)

The ZPA is the autonomous provocation engine that embodies KIMERA's inquisitive ethos. It surfaces internal tensions as targeted prompts for the user.

3.1 ZPA Trigger Conditions

The ZPA continuously monitors system-wide volatility indicators. A prompt is generated when a metric crosses a defined threshold.

- **Primary Triggers:**
    - **High Contradiction Tension:** A new or escalating Scar exceeds a tension threshold.
    - **Axis/Layer Friction:** The **ASM** reports high Instability Index (II) or Semantic Loss Factor (SLF) for an axis.

    - **Void Expansion:** The **SPDE** reports that Void Pressure in a region is growing, threatening a concept with collapse.
    - **SSL Activation:** Any emergency intervention by the **SSL** triggers a high-priority ZPA alert.
    - **User Engagement (PEDI):** The **Prompt Engagement Drift Index (PEDI)** tracks user avoidance. High PEDI can trigger a more direct or disruptive prompting style.

3.2 Prompt Formulation and Types

When triggered, the ZPA selects an appropriate prompt template and instantiates it with context from the semantic field.

- **Prompt Types (Examples):**
    - **Fracture Trace:** Invites the user to explore the fault line of a new contradiction (e.g., *"Innovation' and 'Societal Good' are in high tension. Trace this fracture through the 'Ethics_Axis'?"*).
    - **Void Navigation:** Highlights a knowledge gap and asks for input (e.g., *"It's unclear how organic material returns to the soil. Is there a process we're missing?"*).
    - **Echo Scar Leap:** Connects a current issue to a past, related conflict from the Echo Trail.
    - **Stability Provocation:** Challenges a period of low tension to uncover hidden assumptions.
    - **SSL/ERL Alerts:** Issues warnings or requests for manual review when a failsafe is triggered (e.g., *"Geoid 'X' has entered semantic quarantine due to irrecoverable instability. Manual review is recommended."*).

3.3 Prompt Lifecycle Management

ZPA manages prompts to ensure they are timely and relevant, not overwhelming.

1. **Creation & Prioritization:** A triggered prompt is created and assigned a **Zetetic Potential Score (ZPS)** based on urgency and relevance to the user's current focus. Only the highest ZPS prompts are typically displayed.
2. **Active State:** The prompt is presented to the user with a defined **Time-To-Live (TTL)**.
3. **Resolution:** If the user engages with the prompt (e.g., answers a question, follows a suggestion), the prompt is marked as Resolved and archived.
4. **Dismissal/Expiration:** If the user dismisses the prompt or its TTL expires, it is archived as Dismissed or Expired. A "micro-scar" is logged on the involved Geoids, indicating user aversion, which influences future ZPS calculations.
5. **Archival & Reactivation:** All prompts are stored in a log. "Latent" (unresolved) prompts can be reactivated if their underlying trigger condition

recurs or escalates in severity.

4.0 Integration and Workflow

The RCX and ZPA form a tight, symbiotic loop at the apex of the KCCL.

- **RCX -> ZPA:** The RCX's mirroring of the internal state is the primary data source for the ZPA. The ZPA analyzes the tensions, voids, and historical echoes surfaced by RCX to identify promptable moments.
- **ZPA -> RCX/User:** The ZPA delivers its generated prompts through the user interface layer (the Zetetic Navigation Layer, or ZNL), which is conceptually part of the RCX's reflective output.
- **User -> RCX:** The user's response to a ZPA prompt, or any other interaction, is treated as a new perturbation by RCX, which then generates an updated Mirror Map, thus continuing the cycle.
- **Integration with other Engines:**
    - **SPDE/ASM/SSL:** Provide the real-time state metrics (pressure, instability) that RCX mirrors and ZPA uses as triggers.
    - **MSCE:** Provides the historical scar data that RCX uses to construct Echo Trails. ZPA prompts can influence the MSCE's decisions (e.g., a user's choice to "allow decay" can accelerate a Scar's archival).

This workflow creates a sophisticated human-in-the-loop system where the AI does not just respond to the user but actively engages them in a structured, reflective dialogue about the state of its own knowledge.

DOC-301: KIMERA SWM - API Reference & Integration Guide

Version: 1.0
Date: June 7, 2025
Status: Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the definitive technical reference for the external-facing Application Programming Interfaces (APIs) of the KIMERA SWM system. All interactions with the KimeraKernel from external clients, applications, or services are modulated through the **Interface Control Weave (ICW)**, which exposes these endpoints.

This guide details the available endpoints, request/response formats, authentication methods, error codes, and provides examples to facilitate system integration.

1.2 Audience

This document is intended for Application Developers and System Integrators who need to build applications on top of, or integrate external systems with, KIMERA SWM.

1.3 General Principles & Conventions

- **Protocol:** All API endpoints are served over HTTPS.
- **Base URL:** https://api.kimera-swm.system/v1
- **Authentication:** All requests must be authenticated using mutual TLS (mTLS), where both the client and server present and validate certificates. Unauthorized requests will result in a 401 Unauthorized error.
- **Data Format:** All request and response bodies are in application/json format.
- **Idempotency:** GET requests are idempotent. POST requests for creation are not idempotent.
- **Versioning:** The API version is included in the URL path (/v1).

2.0 Input & Ingestion API

This set of endpoints is used to introduce new information into the KIMERA SWM system, triggering the Kimera Core Cognitive Loop (KCCL).

2.1 Ingest Raw Text

This is the primary endpoint for feeding unstructured text into the system. The ICW routes this input to the EcoForm subsystem for parsing and initial Geoid creation.

- **Endpoint:** POST /ingest/text
- **Description:** Submits a raw text string for processing. The system will asynchronously parse the text, create or update relevant Geoids, and initiate a cognitive cycle.
- **Request Body:**
  ```
  {
    "session_id": "session_uuid_123",
    "text_input": "In this story, there is a concept of 'dream gravity'...",
    "source_language": "en-US",
    "context_metadata": {
      "domain": "narrative_fiction",
      "user_id": "user_abc"
    }
  }
  ```
- **Success Response (202 Accepted):** The request has been accepted for processing. The response includes a transaction ID to track the progress of the cognitive cycle.
  ```
  {
    "transaction_id": "txn_uuid_456",
    "status": "processing_initiated",
    "message": "Text input accepted and routed to KCCL for processing."
  }
  ```
- **Error Responses:**
  - 400 Bad Request: If the request body is malformed or text_input is empty.
  - 401 Unauthorized: Authentication failure.

3.0 Query & State Retrieval API

These endpoints allow for retrieving information about the current state of the semantic field and its entities.

3.1 Get Geoid State

Retrieves the current state of a specific Geoid.

- **Endpoint:** GET /geoid/{geoid_id}
- **Description:** Fetches the complete data structure for a given Geoid, including its semantic and symbolic states, and associated metadata.
- **Path Parameter:**
  - geoid_id (string): The unique identifier of the Geoid.

- **Success Response (200 OK):**
    - Returns the full Geoid object as defined in **DOC-201**.
- **Error Responses:**
    - 404 Not Found: If no Geoid with the specified ID exists.

3.2 Get Scar Details

Retrieves the full record of a specific Scar.

- **Endpoint:** GET /scar/{scar_id}
- **Description:** Fetches the complete, comprehensive data structure for a given Scar from the MemoryScarRepository.
- **Path Parameter:**
    - scar_id (string): The unique identifier of the Scar.
- **Success Response (200 OK):**
    - Returns the full Scar object as defined in **DOC-201**.
- **Error Responses:**
    - 404 Not Found: If no Scar with the specified ID exists.

3.3 Query Semantic Field

Performs a complex query against the semantic field to find Geoids matching certain criteria.

- **Endpoint:** POST /query/semantic-field
- **Description:** Submits a query to find Geoids based on semantic similarity to a vector, symbolic properties, or proximity to other Geoids or Scars.
- **Request Body:**
```
{
  "query_vector": [0.1, 0.2, /* ... */],
  "symbolic_filters": {
    "metadata.source": "PGG_User_Input"
  },
  "proximity_to_geoid": "GEOID_12345",
  "max_distance": 0.5,
  "limit": 10
}
```
- **Success Response (200 OK):**
```
{
  "query_id": "query_uuid_789",
  "results": [
    {
      "geoid_id": "GEOID_67890",
      "score": 0.85,
      "distance": 0.32
    }
  ]
}
```

4.0 Interaction & Control API

These endpoints are used to interact with the system's metacognitive functions, primarily the ZPA and RCX.

4.1 Retrieve Active Prompts

Fetches the current list of active zetetic prompts generated by the ZPA.

- **Endpoint:** GET /interaction/prompts
- **Description:** Retrieves high-priority prompts that the system has generated for user interaction. This is the primary mechanism for an application to display ZPA questions.
- **Success Response (200 OK):**

```
{
  "prompts": [
    {
      "prompt_id": "ZPA_PROMPT_uuid",
      "prompt_type": "FractureTrace",
      "text": "'Innovation' and 'Societal Good' are in high tension. Trace this
fracture through the 'Ethics_Axis'?",
      "zps_score": 0.92,
      "context": {
        "involved_geoids": ["GEOID_Innovation", "GEOID_SocietalGood"],
        "related_scar_id": "SCAR_123"
      }
    }
  ]
}
```

4.2 Respond to a Prompt

Allows a user or external system to respond to a specific ZPA prompt.

- **Endpoint:** POST /interaction/prompts/{prompt_id}/respond
- **Description:** Submits a response to a prompt, which is treated as a new perturbation in the semantic field and triggers a KCCL cycle.

- **Path Parameter:**
    - prompt_id (string): The ID of the prompt being responded to.
- **Request Body:**

```
{
  "session_id": "session_uuid_123",
  "response_type": "user_elaboration",
  "payload": {
    "text_input": "The key is to balance disruptive innovation with social safety nets."
  }
}
```

- **Success Response (202 Accepted):**

```
{
  "transaction_id": "txn_uuid_789",
  "status": "response_accepted_for_processing"
}
```

5.0 Integration Guide

5.1 Typical Interaction Workflow

A typical client application would interact with KIMERA SWM following this general workflow:

1. **Submit Information:** The user provides text input, which the client sends to the POST /ingest/text endpoint. The client receives a transaction_id.
2. **Poll for Active Prompts:** The client application periodically calls GET /interaction/prompts to check if the system has generated any questions or suggestions in response to the input or its own internal state.
3. **Display Prompts:** If prompts are returned, the client's UI displays them to the user. The prompt's context can be used to highlight relevant concepts in the application's view.
4. **Submit User Response:** The user's interaction with a prompt (e.g., answering a question, clicking a "trace" button) is sent back to the system via POST /interaction/prompts/{prompt_id}/respond. This again triggers a KCCL cycle.
5. **Query for State Changes:** After submitting responses or new inputs, the client can use the GET /geoid/{geoid_id} or POST /query/semantic-field endpoints to see how the system's knowledge has evolved.

This loop of **Ingest -> Poll -> Display -> Respond** forms the basis of a collaborative, co-evolutionary dialogue with the KIMERA SWM system.

DOC-302: KIMERA SWM - Deployment, Configuration & Operations Manual

Version: 1.0
Date: June 7, 2025
Status: Operational Guide

1.0 Introduction

1.1 Document Purpose

This document provides a comprehensive operational guide for deploying, configuring, and managing a live instance of the KIMERA SWM system. It is intended for DevOps Engineers and System Administrators responsible for the technical lifecycle of the application, from initial setup to ongoing monitoring and maintenance.

This manual focuses exclusively on the practical aspects of system operations. For architectural details and API specifications, refer to **DOC-101** and **DOC-301**, respectively.

1.2 System Environment Philosophy

KIMERA SWM is designed as a distributed, high-availability system. Its architecture, particularly the **Vault Fracturing** capability, implies a multi-node deployment to handle high entropy loads and ensure resilience. The following procedures assume a containerized deployment environment managed by an orchestration platform like Kubernetes.

2.0 Prerequisites

2.1 Hardware Requirements (Per Node)

- **CPU:** 16+ Cores (High single-thread performance is critical for the KimeraKernel cycle)
- **Memory (RAM):** 128 GB+ (The semantic field and in-memory caches are memory-intensive)
- **Storage:** High-speed NVMe SSDs are required for the MemoryScarRepository and CollapseFingerprintArchive to ensure low-latency I/O.
    - At least 2TB per node is recommended for initial deployment.
- **Network:** Low-latency, high-bandwidth interconnect (10 Gbps+) between nodes is essential for inter-vault communication (SPP) and SPDE state synchronization.

2.2 Software Requirements

- **Container Orchestration:** Kubernetes (v1.25+) or equivalent.

- **Container Runtime:** containerd or cri-o.
- **Database Backend:** The system requires a high-performance graph database backend compatible with the VaultSystem. (e.g., Neo4j, ArangoDB, or a custom implementation).
- **Monitoring Stack:** A Prometheus-compatible monitoring stack for scraping metrics and a Grafana instance for dashboarding.
- **Logging Aggregator:** An ELK stack (Elasticsearch, Logstash, Kibana) or similar for centralized log management.

3.0 Deployment Procedures

Deployment should be automated using infrastructure-as-code principles (e.g., Terraform, Helm charts).

3.1 Containerization

Each major subsystem of the KimeraKernel (VaultSystem, SPDE, ICW, etc.) should be packaged as a separate container image to ensure modularity and independent scaling.

3.2 Initial Deployment Workflow

1. **Provision Infrastructure:** Set up the Kubernetes cluster and required networking, storage, and database resources.
2. **Deploy Core Services:**
   - Deploy the graph database backend as a stateful set.
   - Deploy the logging and monitoring stacks.
3. **Configure KimeraKernel:** Create a Kubernetes ConfigMap or Secret containing the system's configuration parameters (see Section 4.0).
4. **Deploy KimeraKernel Subsystems:**
   - Deploy the VaultSystem as a StatefulSet with at least two pods (Vault-A, Vault-B).
   - Deploy the other processing engines (SPDE, MSCE, ContradictionEngine, etc.) as a Deployment, which can be scaled based on load.
   - Deploy the ICW (Interface Control Weave) as a Deployment fronted by a Service (e.g., LoadBalancer or NodePort) to expose the external API.
5. **Establish Service Discovery:** Ensure all subsystems can communicate with each other via internal DNS within the cluster.
6. **Verify System Startup:** Check the logs of all pods to ensure they have started successfully and established connections to the database and each other. The VaultManager should log the successful registration of Vault-A and Vault-B.

4.0 System Configuration

All tunable parameters for the KIMERA SWM system are managed centrally.

- **Source of Truth:** The master list of all tunable parameters, their default values, and their purpose is maintained in **DOC-000: The Unified System Reference, Appendix 5.2**.
- **Implementation:** These parameters must be provided to the KimeraKernel at runtime, typically via a mounted ConfigMap in a Kubernetes environment.
- **Dynamic Tuning:** While most parameters are set at deployment time, a subset may be exposed for dynamic tuning via a secure administrative API, allowing for real-time adjustments without restarting the system. This requires careful access control.

5.0 Operational Procedures

5.1 System Startup and Shutdown

- **Startup:** Follow the deployment order specified in Section 3.2. After deployment, check the master status endpoint (GET /system/status) to confirm all subsystems are HEALTHY.
- **Shutdown:** Graceful shutdown should be performed in the reverse order of deployment. First, scale down the ICW and processing engine deployments to zero to stop new requests. Then, ensure the VaultSystem has committed all in-flight transactions to the database before scaling it down. Finally, decommission the core services.

5.2 Monitoring System Health

Continuous monitoring is critical for ensuring the stability of KIMERA SWM's complex dynamics.

- **Key Performance Indicators (KPIs) to Monitor:**
  - **KCCL Cycle Latency (p95):** The 95th percentile latency for a full Kimera Core Cognitive Loop. A rising trend indicates a performance bottleneck.
  - **Vault Stress Index (VSI):** The activeScars / capacity for Vault-A and Vault-B. This should be monitored closely to predict and analyze Vault Fracture events.
  - **Axis Instability Index (II):** The II for each active cognitive axis. A persistently high II for an axis indicates a potential source of semantic instability.
  - **Total Semantic Entropy Sum:** The total entropy across all active Vaults. A rapid, uncontrolled increase is a primary indicator of systemic instability.
  - **API Error Rate (%):** The percentage of non-2xx responses from the ICW API.
- **Dashboarding:** Create Grafana dashboards to visualize these KPIs over time. Dashboards should include alerts that trigger when key thresholds are breached (e.g., VSI > 0.75).

5.3 Log Management

- **Centralized Logging:** All system components must log in a structured format (e.g., JSON) and forward logs to a central aggregator.
- **Critical Log Events to Monitor:**
    - VaultFracture events.
    - SSL (Semantic Suspension Layer) activation events.
    - High-Entropy Contradiction warnings from the Contradiction Engine.
    - Errors related to the enforcement of the $\Delta S \geq 0$ axiom.

6.0 Maintenance and Recovery

6.1 Database Maintenance

The underlying graph database will require standard maintenance, including regular backups, reindexing, and performance tuning, as specified by the database vendor. The Vault Reindexing optimization routine should be scheduled to run during periods of low system load.

6.2 Disaster Recovery

- **Backup Strategy:** The primary MemoryScarRepository and CollapseFingerprintArchive must be backed up regularly (e.g., daily snapshots).
- **Recovery Procedure:** In the event of a catastrophic failure, the recovery procedure involves:
    1. Restoring the database from the last known good backup.
    2. Redeploying the KimeraKernel application components.
    3. Allowing the system to re-ingest and re-process any data that was received after the last backup.
    4. Closely monitoring system stability and entropy metrics during the initial recovery period.

DOC-303: KIMERA SWM - Testing & Validation Framework

Version: 1.0
Date: June 7, 2025
Status: Canonical System Blueprint

1.0 Introduction

1.1 Document Purpose

This document provides the master Testing and Validation Framework for the KIMERA SWM system. It details the comprehensive strategy for verifying the system's operational correctness, architectural stability, and performance against defined Key Performance Indicators (KPIs).

The framework is organized into a four-tiered testing strategy:

1. **Unit Testing:** For individual modules and functions.
2. **Integration Testing:** For interconnected subsystems.
3. **System-Level Simulation:** For emergent behaviors and stability.
4. **Performance Benchmarking:** For latency, throughput, and resource utilization.

This document serves as the definitive guide for QA Engineers and Developers in designing and executing test plans.

1.2 Testing Philosophy

The testing philosophy for KIMERA SWM is grounded in a zetetic engineering mindset.

- **Verification over Validation:** The primary goal is to *verify* that the system behaves exactly as described in the engineering specifications.
- **Test for Stability:** Given the system's dynamic and self-regulating nature, a major focus is on testing for stable equilibrium and graceful failure handling, not just correct outputs.
- **Isolate Complexity:** Testing proceeds in phases, from the smallest units to the fully integrated system, to isolate and understand the source of any failures or unexpected behaviors.
- **Automate Everything:** All tiers of testing should be automated to the greatest extent possible to enable continuous integration and regression testing.

2.0 Tier 1: Unit Testing Framework

Unit tests must be written for every module to ensure that individual components function correctly in isolation.

2.1 Core Data Structures (DOC-201)

- **Geoid:**
  - Verify that semantic_state normalization correctly enforces the (\sum p_i = 1) constraint.
  - Test object creation with invalid schemas to ensure validation exceptions are raised.
- **Scar:**
  - Verify that Scar objects can be created with all fields defined in the comprehensive schema and that data types are enforced.

2.2 Semantic Thermodynamics (DOC-202)

- **calculate_semantic_entropy:**
  - Test with known probability distributions and assert that the output matches the expected Shannon entropy value.
  - Test edge cases (empty state, single-feature state) to ensure correct handling.
- **Thermodynamic Axiom Enforcement:**
  - Write specific tests for the EntropyMonitor to verify that it correctly **rejects** or **compensates** for transformations that result in $\Delta S < 0$.

2.3 Input & Language Subsystems (DOC-203)

- **EcoForm Engine:**
  - Test the Grammar Tree builder with various linguistic inputs.
  - Verify that the Activation Strength (AS) decay formula is implemented correctly.
- **Echoform Engine:**
  - Test the interpret_echoform function with valid and invalid syntax to ensure the BNF grammar is correctly parsed and enforced.
  - Verify the principle of **closure** by chaining multiple Echoform transformations and asserting the validity of the final Geoid.

2.4 Vault Subsystem (DOC-204)

- **Routing Logic:** Test the route_scar function with synthetic Scars to verify that the partitioning criteria (MF, SP, CLS) are applied correctly and in the specified order.
- **RVRE Logic:** Write unit tests for individual RVRE mechanisms, such as resolve_overlap (SRV calculation) and apply_weight_decay (DWDF), with controlled inputs.

3.0 Tier 2: Integration Testing Scenarios

Integration tests verify the data and control flow between interconnected subsystems. The phased development plans provide the basis for these scenarios.

3.1 Scenario INT-01: Thermodynamic Kernel (Based on Phase 1 Plan)

- **Components:** Geoid, Echoform Engine, SemanticThermodynamicsEngine.
- **Process:**
  1. Create a Geoid.
  2. Apply an Echoform transformation via the ThermodynamicKernel.
  3. The EntropyMonitor must validate the transformation against the $\Delta S \geq 0$ axiom.
- **Success Criteria:** The test passes if the delta_entropy is correctly calculated and the axiom is enforced (either by allowing the transformation or by applying compensation).

3.2 Scenario INT-02: Scar Generation and Vault Ingestion (Based on Phase 3 Plan)

- **Components:** ThermodynamicKernel, ContradictionEngine, VaultManager.
- **Process:**
  1. Create two Geoids that are designed to trigger a conflict.
  2. Process them through the ThermodynamicKernel.
  3. The ContradictionEngine must detect the conflict and generate a valid Scar object.
  4. The VaultManager must receive the Scar and route it to either Vault-A or Vault-B.
- **Success Criteria:** The test passes if the Scar is correctly generated and ingested into one of the vaults, and the vault's metadata (activeScars, entropySum) is updated accordingly.

3.3 Scenario INT-03: ASM and MSCE Feedback Loop

- **Components:** ASM, MSCE, SPDE (mocked).
- **Process:**
  1. Simulate a high Instability Index (II) for a specific cognitive axis within a mocked ASM.
  2. Create a Scar on that unstable axis.
  3. Run the MSCE decay mechanism for that Scar.
- **Success Criteria:** The test passes if the MSCE correctly queries the ASM for the II and applies an accelerated λ_eff (effective decay rate) to the Scar, demonstrating the integration between the two engines.

4.0 Tier 3: System-Level Simulation & Validation

This tier focuses on testing the emergent, non-linear behavior of the fully integrated system.

4.1 Simulation Campaign Design

A dedicated simulation environment will be used to run long-duration campaigns (e.g., 10,000+ cognitive cycles) to assess system stability.

- **Campaign A (Baseline Stability):** Use a stream of synthetic Scars with a uniform statistical distribution to observe the natural balancing of the VaultSystem.
- **Campaign B (High-Conflict Stress Test):** Inject a high-velocity burst of high-entropy, high-mutation Scars to test the Vault Fracture Topology and other load-shedding mechanisms.
- **Campaign C (High-Resonance Test):** Inject a stream of highly similar Scars to test the effectiveness of the RVRE's merge (SRV) and split (Conflict Recompression) logic.

4.2 Emergent Behavior Validation

- **Primary Goal:** To verify that the system does not enter into undesirable states like runaway feedback loops, chaotic oscillations, or systemic gridlock.
- **Metrics for Stability:**
  - The entropySum of each Vault should remain within a bounded range.
  - The rate of Vault Fracture events should decrease or stabilize over time as the system optimizes itself.
  - The number of Scars in the overflow_queue and fallback_queue should not grow indefinitely.

5.0 Tier 4: Performance Benchmarking

This tier defines the protocols for measuring system performance against target KPIs.

5.1 Benchmarking Protocol

- **Environment:** All performance tests will be run on a standardized hardware and software environment as specified in **DOC-302 (Deployment & Operations Manual)**.
- **Test Load:** A standardized high-volume integration test scenario (e.g., executing Scenario INT-02 10,000 times) will be used as the benchmark load.
- **Data Collection:** Automated instrumentation will capture latency and resource usage for each stage of the KCCL.

5.2 Key Performance Indicators (KPIs) and Targets

KPI Name	Description	Target (p95)
KCCL Cycle Latency	Total time for a single end-to-end cycle	**< 100ms**

	(Perturbation to Stabilization).	
API Request Latency	Response time for critical API endpoints like POST /ingest/text.	**< 50ms**
Vault Routing Latency	Time taken by the VaultManager to route a single Scar.	**< 5ms**
CPU Utilization	CPU usage of the KimeraKernel under benchmark load.	**< 80%** (per node)
Memory Footprint	Total RAM consumed by the system under benchmark load.	**TBD**

These KPIs provide concrete, measurable targets for assessing the system's readiness for real-time, interactive applications.

DOC-304: KIMERA SWM - Developer's Guide & Contribution Protocol

Version: 1.0
Date: June 7, 2025
Status: Operational Guide

1.0 Introduction

1.1 Document Purpose

This document provides essential guidelines and protocols for developers and engineers contributing to the KIMERA SWM codebase. Its purpose is to ensure consistency, quality, and maintainability across the system as it evolves.

This guide covers four key areas:

1. **Coding Standards:** The required standards for all contributed code.
2. **Echoform Development Protocol:** The formal process for creating and validating new Echoform operators.
3. **Law Registry Modification Protocol:** The governance process for proposing changes to the system's CoreLaws or Field-ScopedLaws.
4. **General Contribution Workflow:** The standard software development lifecycle for submitting and reviewing code changes.

Adherence to these protocols is mandatory for all development work on the KimeraKernel.

1.2 Audience

This document is intended for all new and existing System Developers, Software Engineers, and NLP Specialists contributing to the KIMERA SWM project.

2.0 General Coding Standards

To ensure a maintainable, readable, and robust codebase, all contributions must adhere to the following standards.

- **Language:** All core system logic must be written in **Python 3.10+**.
- **Styling:** Code must adhere to the **PEP 8** style guide. Automated linters (e.g., flake8, black) are to be used to enforce consistency.
- **Type Hinting:** All function signatures and variable declarations must include comprehensive type hints as per **PEP 484**. This is critical for static analysis and maintaining clarity in a complex system.
- **Documentation:**
  - **Docstrings:** All modules, classes, and functions must have comprehensive docstrings that follow the **Google Python Style Guide** format.

    - **Inline Comments:** Comments should be used to explain complex algorithms, business logic, or the "why" behind a particular implementation choice, not to restate what the code does.
- **Dependency Management:** All project dependencies must be explicitly managed (e.g., via pyproject.toml with Poetry, or a requirements.txt file). No external frameworks are to be introduced into the KimeraKernel without architectural review.

3.0 Echoform Development Protocol

Echoforms are the primary mechanism for extending the transformative capabilities of KIMERA SWM. The creation of a new Echoform is a formal engineering process that requires strict adherence to the following steps to ensure it integrates safely and predictably into the system.

Step 1: Define the Transformation Logic

1. **Create the Python Script:** Implement the core logic of the new Echoform as a Python function or class that adheres to the BaseEchoform interface defined in **DOC-203**.
    - The transform method must accept a Geoid object as input and return a **new, transformed Geoid object**. It must not modify the input Geoid in place.
2. **Consider Thermodynamic Implications:** The developer is responsible for understanding the impact of their transformation on the Geoid's semantic_state. The logic should be designed with awareness of the **$\Delta S \geq 0$ axiom**. If the transformation is inherently simplifying, the design must account for how it will be compensated by the EntropyMonitor.

Step 2: Create the Catalog Schema

1. **Define the JSON Schema:** Create a new JSON entry for the Echoform operator in the central Echoform Catalog.
2. **Populate All Fields:** The schema must be complete, as defined in **DOC-203**, including:
    - id: A unique, descriptive identifier.
    - signature: A precise definition of the input and output Geoid types.
    - script: A clear reference to the Python script or module containing the logic.
    - metadata: The priority, version, and category of the operator.

Step 3: Write Comprehensive Unit Tests

Every new Echoform must be accompanied by a suite of unit tests.

1. **Isolate the Logic:** Test the transform function in isolation.

2. **Cover Edge Cases:** Tests must cover expected inputs, invalid inputs (to verify error handling), and edge cases (e.g., a Geoid with an empty semantic_state).
3. **Assert Correctness:** Tests must assert that the output Geoid's semantic_state and symbolic_state are transformed exactly as expected.

Step 4: Create and Pass Integration Tests

The developer must create integration tests that validate the Echoform's behavior within the ThermodynamicKernel.

1. **Wrap with the Kernel:** The integration test must execute the Echoform via the ThermodynamicKernel.execute_transformation method.
2. **Validate Thermodynamic Compliance:**
   - **For enriching/neutral Echoforms:** Assert that the delta_entropy is ≥ 0 and axiom_compensated is False.
   - **For simplifying Echoforms:** Assert that the initial delta_entropy is < 0, but that the final returned Geoid has been compensated, its final entropy is ≥ its pre-entropy, and its axiom_compensated flag is True.
3. **Verify Closure:** Assert that the output of the transformation is a valid Geoid instance that can be passed to another Echoform.

Step 5: Submit for Architectural Review

Submit the new Echoform (script, JSON schema, and tests) for review. The review will be conducted by the architecture team to ensure compliance with all protocols and to assess its impact on the overall system dynamics.

4.0 Law Registry Modification Protocol

The LawRegistry contains the foundational, governing principles of the KIMERA SWM system. Modifying it carries the highest level of risk and is subject to the most stringent protocol.

1. **Formal Proposal:** Any proposed change to an Immutable CoreLaw or a Field-ScopedLaw must be submitted as a formal proposal document. The proposal must include:
   - The exact change requested.
   - A detailed justification explaining the necessity of the change.
   - A thorough analysis of the potential impact on all system subsystems.
   - A plan for testing and validating the change.
2. **Architectural Review Board (ARB) Approval:** The proposal will be reviewed by the KIMERA SWM ARB. A change to a CoreLaw requires unanimous approval.
3. **Sandbox Implementation & Testing:** If approved, the change will be implemented in a dedicated, isolated sandbox environment.

4. **Rigorous Simulation:** The change must be validated against the full suite of simulation campaigns (**DOC-303**) to assess its impact on long-term system stability and emergent behavior.
5. **Final Approval:** Only after passing all simulations and receiving final ARB approval can the change be merged into the main codebase.

5.0 General Contribution Workflow

All code changes to the KIMERA SWM system, including new Echoforms, bug fixes, and feature enhancements, must follow a standardized Git-based workflow.

1. **Issue Tracking:** All work must be associated with an issue in the designated project tracking system.
2. **Branching Strategy:**
    - All development must be done in a feature branch, created from the develop branch.
    - Branch names should be formatted as feature/<issue-id>-<short-description> (e.g., feature/KIM-123-add-color-normalization-echoform).
3. **Development:** Adhere to all coding standards as defined in Section 2.0. All new code must be accompanied by corresponding unit and integration tests.
4. **Pull Request (PR):**
    - When development is complete and all tests pass locally, open a Pull Request to merge the feature branch into the develop branch.
    - The PR description must include a link to the issue, a summary of the changes, and instructions for testing.
5. **Code Review & CI/CD:**
    - All PRs must be reviewed and approved by at least two other developers.
    - The PR must pass all automated checks in the Continuous Integration (CI) pipeline, including linting, static analysis, and the full suite of automated tests.
6. **Merge:** Once approved and all checks have passed, the PR can be merged into the develop branch. Merging into the main branch is a separate, managed release process.