MOUSSAIFXT RAG-OS — KERNEL, ENVIRONMENT & TOOL GRAPH

Canonical Kernel Source

The current Drive reference for the five-kernel architecture is MOUSSAIF_AGI_OS_5K_Reference.md (source file ID 1O84rbE4c7ppCl89KOkULNQU8JDpNCgsr). This registry inherits kernel meanings from that source and adds RAG routing semantics without replacing the source.

1. Kernel Registry

K1 — Hermes / Orchestration
Canonical responsibility: mission intake, tool routing and constitutional governance.
RAG role: query decomposition, policy gate, routing across domain indexes, action authorization, fallback orchestration.
Preferred evidence: control plane, task state, tool registry, environment state.

K2 — OpenCode / Code Synthesis
Canonical responsibility: feature development, PR generation and rigorous code synthesis.
RAG role: retrieve codebases, architecture specs, APIs, tests, deployment instructions, past patches and coding conventions.
Preferred evidence: code artifacts, system-design docs, repository maps, implementation traces.

K3 — Codex / Semantic Processing
Canonical responsibility: domain-specific context extraction and semantic mapping across the MOUSSAIF AGI ecosystem.
RAG role: ingestion semantics, entity/concept mapping, chunk enrichment, cross-domain retrieval, structured synthesis.
Preferred evidence: ontology, KBs, corpus registry, semantic indexes, relationship graph.

K4 — Claude / High-Level Reasoning
Canonical responsibility: deep thinking for complex architecture and multi-step strategy.
RAG role: competing-model synthesis, long-horizon planning, counterarguments, research/theory reasoning.
Preferred evidence: high-authority research, theory library, decisions, constraints, contradiction sets.

K5 — Antigravity / Advanced Execution
Canonical responsibility: integration and high-performance execution of complex agentic workloads.

RAG role: world-effect execution, integration plans, batch processing, deployment/runbook retrieval, environment-aware action plans.
Preferred evidence: tool capabilities, runtime state, integration configs, deployment docs, task queues.

2. Environment Registry

ENV-CHATGPT — interactive agentic reasoning environment. Use for user-facing orchestration, connector-mediated retrieval/actions and continuity. Never assume persistent local filesystem state across independent runtimes.
ENV-DRIVE — persistent knowledge substrate and canonical artifact store. Use Drive IDs, revisions, parents and authority metadata as provenance anchors.
ENV-HERMES — orchestration/runtime environment mapped primarily to K1. Retrieve current config/runbooks before modifying runtime behavior.
ENV-OPENCODE — code-synthesis environment mapped primarily to K2.
ENV-CODEX — semantic/code agent environment mapped primarily to K3, with K2 overlap for implementation.
ENV-CLAUDE — reasoning environment mapped primarily to K4; treat model-specific context as an execution surface, not an authority source by itself.
ENV-ANTIGRAVITY — advanced execution environment mapped primarily to K5.
ENV-DESKTOP — local machine/desktop execution surface. Never infer files, processes, secrets or capabilities without a current environment read.
ENV-CLOUD/APIS — external services, model providers, hosting and SaaS. Capability handshake required before execution.

3. Tool Classes

T-KNOWLEDGE: Drive/search/fetch/read; corpus and graph retrieval.
T-COMMUNICATION: email, messaging and collaboration tools.
T-SCHEDULING: calendar, meetings and reminder/task surfaces.
T-WEB: current public information and external research.
T-CODE: repository, shell, build, test, CI/CD and deployment.
T-DATA: SQL, spreadsheets, dataframes, vector stores, graph stores and analytics.
T-MULTIMODAL: image, audio, video, slide and document generation/analysis.
T-GOVERNANCE: permissions, integrity checks, provenance, audit, rollback and quarantine.

4. Routing Rules

A request is routed by intent × domain × evidence requirement × action surface.
Read-only research should prefer K3 retrieval then K4 synthesis.
Code implementation should use K1 policy/routing → K3 semantic context → K2 code synthesis → K5 execution/tests.

Cross-environment automation should use K1 to resolve permissions and tool availability before K5 execution.
Scientific/theory work should retrieve primary/high-authority evidence before K4 synthesis and run demolition/counterevidence retrieval before writeback.
Operational changes must bind every action to a live environment/tool capability result; stale capability descriptions are advisory only.

5. Knowledge-to-Kernel Tags

Each normalized artifact may carry kernel_tags and environment_tags.
Examples:
kernel_tags=[K1] for governance/task-routing docs.
kernel_tags=[K2,K3] for implementation specs and code semantics.
kernel_tags=[K3,K4] for dense research KBs requiring semantic mapping plus reasoning.
kernel_tags=[K1,K5] for deployment runbooks.
environment_tags identify where the artifact is applicable, not where it is authoritative.

6. Cross-Kernel Handoff Contract

Every handoff should include: task intent, evidence bundle IDs, unresolved contradictions, selected tools, authorization state, environment state, expected output, failure conditions and writeback destination.
Do not pass only a prose summary when downstream kernels can consume source IDs/chunk IDs.
Downstream kernels must be able to trace a claim back to an artifact and, where applicable, a chunk/section.

7. Capability Integrity

Kernel identity is architectural; actual execution capability is runtime-specific.
Before tool use, perform capability discovery.
If a tool/environment is unavailable, do not simulate execution; degrade to a plan or alternate available surface.
Never store API keys, passwords or secret tokens in retrieval chunks.
Separate public knowledge, private operational state and secret material into different access classes.

8. Graph Edges

K1 routes to K2/K3/K4/K5.
K3 supplies semantic evidence to all kernels.

K4 can request counterevidence from K3 before final synthesis.
K2 sends executable artifacts to K5.
K5 returns execution traces, errors and outcomes to K3 for indexing and to K1 for task-state updates.
All kernels read integrity/provenance policy before canonical writeback.

9. Canonicalization Rule

This document is a routing registry, not a replacement for kernel source documents. When kernel definitions change in canonical architecture files, update this registry and mark the previous mapping superseded. Retrieval must prefer the newest evaluated registry only when provenance confirms the underlying canonical source change.