# Chronoktonos System Documentation

- **Document Version**: 1.0
- **System Version**: 1.0
- **Date**: 2025-07-20
- **Author**: SystemArchitect_v3

## 1. Introduction

### 1.1. System Overview

Chronoktonos is a decentralized, asynchronous, multi-agent system designed for resilient and efficient task execution. Its architecture is based on direct, message-based communication between independent agents, which can be either software processes or human operators. The system's primary purpose is to provide a robust framework for achieving strategic objectives by "slaying time"—eliminating bottlenecks and enabling parallel, independent action.

### 1.2. Core Principles

- **Decentralized Processing, Centralized Discovery**: Agents operate independently, but locate each other and monitor system health via a shared, central registry.
- **Asynchronous Communication**: The system is non-blocking. Agents communicate by leaving messages in each other's mailboxes, allowing for parallel operation without direct dependency.
- **Extensibility**: The system is designed for easy expansion. New agents with novel capabilities can be integrated by adhering to the established communication and registration protocols.
- **Resilience**: The system is designed to withstand the failure of individual components. The status and heartbeat mechanism allows for real-time monitoring of agent health.

## 2. System Architecture

The Chronoktonos architecture is defined by three core components, as specified in the **Abstract Agentic System Architecture Design v1.1**.

### 2.1. The Agent (Node)

The fundamental unit of the system.

- **Attributes**:
  - agent_id: A unique network-wide identifier (e.g., Puppetmaster,

SystemArchitect_v3).
    - input_dir: The agent's dedicated mailbox for receiving messages.
    - processed_dir: An archive for successfully processed messages.
- **Lifecycle States**:
    - spawning: Initializing.
    - active: Operational.
    - error: Halted due to a critical failure.
    - inactive: Intentionally paused or not running.
    - terminated: Gracefully shut down.

### 2.2. The Central Mailbox Registry

The system's service directory.

- **Implementation**:
    - mailbox_map.json: A central JSON file mapping agent_ids to their mailbox_path and current status.
    - mailbox_map.lock: A lock file to prevent race conditions during concurrent writes to the registry. An agent MUST acquire the lock before writing and release it immediately after.
- **Function**: Enables agents to discover each other and provides a real-time overview of the network's operational state.

### 2.3. The Message (Packet)

The standard unit of communication. A JSON file written to an agent's mailbox.

- **Structure**:
    - **Required Fields**: message_id (UUID), sender_id, recipient_id, timestamp_utc, type, payload.
    - **Optional Fields**: task_id, priority.
- **Filename Convention**: [timestamp]_[message_id].json.

## 3. Protocols

### 3.1. Agent Debriefing Protocol v1.0

This is the standard protocol for spawning and initializing a new agent. It ensures that any new instance, regardless of its underlying technology (LLM, Python script), can parse its identity, role, and core mandate.

- **Structure**: A markdown file with machine-readable headers and distinct sections for:
    1. METADATA: Document type, protocol version.
    2. RECIPIENT IDENTIFICATION: agent_id, role, version.

3. OPERATIONAL MANDATE: Natural language definition and JSON configuration.
4. CORE KNOWLEDGE: Links to required documents (e.g., system architecture).
5. CONTINGENCY PROTOCOLS: Error handling and LLM consultation endpoints.

## 4. Operational Guide

### 4.1. Spawning a New Agent

1. **Prepare the Debriefing File**: Create a new initialization document using the Agent Debriefing Protocol v1.0 template. Fill in the recipient_agent_id, role, version, and other configuration details.
2. **Create Directories**: On the host file system, create the input_dir and processed_dir for the new agent. Ensure permissions are set correctly.
3. **Instantiate Agent**: For a human agent, this means directing them to their mailbox. For a software agent, this means running its process and passing the path to the debriefing file as an argument.
4. **Agent Initialization**: The new agent will follow the Initialization Sequence in its debriefing file, which includes registering itself in the mailbox_map.json.

### 4.2. Sending a Message

1. **Consult the Registry**: Read mailbox_map.json to get the mailbox_path and status of the recipient agent. Do not send if the status is error or inactive.
2. **Construct the Message**: Create a JSON file adhering to the Message Packet structure (2.3).
3. **Deliver the Message**: Write the JSON file directly into the recipient's input_dir.

### 4.3. Checking System Status

- Monitor the mailbox_map.json file. The status and last_heartbeat fields for each agent provide a real-time overview of the entire system's health. A monitoring script can be used to flag agents whose heartbeats have lapsed.

## 5. Glossary

- **Agent**: An independent, message-driven entity within the system.
- **Chronoktonos**: The official designation for the entire multi-agent system.
- **Debriefing**: The act of providing an agent with its initialization protocol.
- **Mailbox**: An agent's dedicated input directory (input_dir).
- **Puppetmaster**: The primary human operator and system controller, designated Mr. Radharani.
- **Registry**: The central mailbox_map.json file that serves as the system's address book and status board.

- **Respawn**: The process of terminating an agent instance and initializing a new one in its place.