--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/README.md ---

# Token Decoder Maps: A DSL for High-Precision AI Agent Control

Token Decoder Maps (TDM) is an open-source framework that transforms how you interact with Large Language Models (LLMs) like Google's Gemini. It moves beyond simple prompt engineering into the realm of Context Engineering by providing a structured, predictable, and powerful Domain-Specific Language (DSL) for directing AI agents.

Stop wrestling with ambiguous natural language. Start architecting precise, repeatable, and complex workflows.

## The Problem: The Ambiguity of Natural Language

Standard interaction with LLMs is a guessing game. You write a prompt and hope the model understands your intent. This leads to:
* Inconsistent Outputs: The same prompt can yield different results.
* Lack of State: The model has no memory of previous, related tasks.
* Verbosity: Complex instructions require long, convoluted prompts that are hard to maintain.
* Poor Scalability: Managing complex, multi-step tasks is nearly impossible.

## The Solution: A Language for Agentic Control

Token Decoder Maps isn't just a collection of prompts; it's a formal interaction architecture. It introduces a simple but powerful Domain-Specific Language (DSL) that allows you to command an AI agent with the precision of a programming language while retaining the flexibility of natural language.

Think of it like this:
* Prompt Engineering is like telling a chef to "make something tasty."
* Token Decoder Maps is like giving the chef a detailed recipe with specific ingredients, measurements, and steps (::FX-ROOT-CAUSE-ANALYSIS::).

This approach allows you to build robust, stateful, and predictable applications on top of powerful but inherently non-deterministic LLMs.

## Table of Contents

## Core Philosophy
* **Precision over Ambiguity:** Use explicit, machine-readable tokens to define operations and reasoning processes.
* **Protocol over Conversation:** Define repeatable, multi-step workflows for complex tasks like project management.
* **Context as Code:** Treat the AI's context not as a chat history, but as a configurable, version-controlled environment.
* **Human-in-the-Loop, AI-in-the-Flow:** The user provides strategic direction; the AI handles the tactical execution with increased fidelity.

## Features
* **Symbolic Language:** A simple ::PREFIX-TOKEN:: syntax for precise commands.
* **Stateful Task Management:** The Metrica system allows for persistent, cross-session task tracking.
* **Cognitive Scaffolding:** Define and invoke complex reasoning patterns with ::FX- tokens.
* **Dynamic Personas:** The agent adapts its expertise based on the project context (e.g., Rust, Python, Node.js).
* **Context Compression:** Use tokens as pointers to larger concepts, saving valuable context window space.

## Getting Started: A 5-Minute Example

This example assumes you have gemini-cli installed and authenticated.

[1]: For installation and authentication instructions, please refer to the official `gemini-cli` documentation.

1. **Create your master context file:**
    Create a file named `GEMINI.md` in your home directory (`~/.gemini/`) and paste the contents of the `GEMINI.md` from this repository into it. This file acts as the "operating system" for your

AI agent.

[2]: The `GEMINI.md` file serves as the primary configuration and directive file for the AI agent, defining its core behaviors, operational protocols, and understanding of the Token Decoder Framework. It's essential for the agent to interpret and utilize TDM tokens effectively.

2. **Create a project and a metrica.md file:**
    ```bash
    mkdir my-first-tdm-project
    cd my-first-tdm-project
    touch metrica.md
    ```

3. **Launch `gemini-cli`:**
    ```bash
    gemini
    ```

4. **Create your first task using a TDM token:**
    Inside the gemini prompt, type:
    `::SY-METRICA-CREATE-TASK::`

    The agent will now follow the protocol you defined. It will prompt you for a task title, category, priority, and other details, and then automatically write a new, perfectly formatted task entry into your `metrica.md` file. You have just executed a structured protocol instead of writing a vague prompt.

## The TDM Language: A Quick Reference

TDM is a Domain-Specific Language (DSL) designed for controlling AI agents. The core of the language is the token.

For the canonical definition of all TDM token types and their structure, please refer to the [TDM Language Specification](docs/TDM_Language_Specification.md). For the unified architectural specification of the TDM framework, refer to the [TDM Unified Specification v3.0](docs/TDM_Unified_Specification_v3.0.md).

### Token Naming Convention

All tokens follow a `::PREFIX-TOKEN-NAME::` structure to categorize their purpose.

Prefix	Name	Purpose

`::FX-`	Function/Cognitive	Defines a reasoning style or problem-solving process.
`::MX-`	Metrica	Defines data structures for the Metrica project ledger.
`::SY-`	System/Utility	Manages the AI's operational state or interaction mode.
`::EN-`	Entity/Knowledge	Represents a structured data element or concept.
`::ML-`	Meta-Log	Represents a structured entry in the AI's performance log.
`::ET-`	Ethos	Loads a complete ethical or moral calculus system.

## Advanced Usage & Concepts

For a detailed guide on integrating TDM with knowledge management tools like Obsidian, see [Advanced Usage: A Headless Knowledge Management Architecture with Obsidian](docs/advanced_usage.md).

NotebookLM Integration: Explore a pre-loaded NotebookLM project with the core TDM documentation for interactive learning and experimentation: [NotebookLM Project Link](https://notebooklm.google.com/notebook/8ce8b157-d1b4-4c56-87c2-2452d95303de/audio)

(This is where you can add more detailed documentation, linking to a wiki or a `/docs` folder for more complex topics).[2, 3, 4]

### The "Metrica" Task Management Protocol

The Metrica system provides a robust, stateful task management solution by leveraging an external `metrica.md` file as a persistent ledger. It now supports a two-stream system to differentiate between high-level user goals and granular project sub-tasks. This approach offers several key benefits:

*   **Persistent State:** Unlike typical LLM interactions that are stateless, Metrica tasks are written to a file, ensuring that task progress, details, and history are preserved across sessions.
*   **Human-Readable and Machine-Parsable:** The `metrica.md` file uses a structured Markdown format that is easy for humans to read and for AI agents (or other tools) to parse and update programmatically.
*   **Version Control Friendly:** Since `metrica.md` is a plain text file, it can be easily version-controlled with Git, allowing for tracking of task changes, collaboration, and auditing.
*   **Protocol-Driven Workflow:** Instead of free-form prompts, task creation and updates are driven by specific `::SY-METRICA-CREATE-TASK::` and `::SY-METRICA-UPDATE-TASK::` protocols, ensuring consistency and completeness of task data.

This protocol allows you to manage complex projects by breaking them down into discrete, trackable tasks, all while maintaining a clear, auditable history.

[3]: For a detailed explanation of the Metrica protocol and its token structure, refer to `docs/metrica_protocol.md`. For a practical example of implementing Metrica, see `examples/metrica_protocol_example.md`.

### Dynamic Personas

One of the powerful features of TDM is the ability for the AI agent to adopt "dynamic personas." This means the agent can adapt its expertise, communication style, and even its internal reasoning processes based on the specific project context it's operating within. This is achieved by:

*   **Contextual File Detection:** The agent can detect the presence of specific project configuration files (e.g., `package.json` for Node.js, `pyproject.toml` or `requirements.txt` for Python, `Cargo.toml` for Rust, `build.gradle` for Java/Kotlin, etc.).
*   **Loading Ecosystem-Specific Knowledge:** Upon detecting a specific project type, the agent can dynamically load relevant `::EN-` (Entity/Knowledge) tokens and `::FX-` (Cognitive Function) tokens that are tailored to that ecosystem. This might include best practices, common libraries, idiomatic code patterns, or specialized problem-solving approaches for that language/framework.
*   **Adapting Behavior and Communication:** The agent's responses and actions will then reflect the conventions and nuances of that specific development environment, providing more accurate, idiomatic, and helpful assistance.

This allows for a seamless transition between different types of projects without needing to manually reconfigure the agent's understanding.

[4]: The concept of dynamic personas is further elaborated in `docs/token_type_concepts.md`, particularly in relation to how `::SY-` and `::EN-` tokens can be used to define and trigger these adaptive behaviors.

### Extending the Framework

The true power of Token Decoder Maps lies in its extensibility. Users are not limited to the predefined set of tokens; they can define their own custom `::FX-`, `::SY-`, and `::EN-` tokens to tailor the system precisely to their unique needs and workflows. This allows for:

*   **Domain-Specific Customization:** Create tokens that encapsulate knowledge, reasoning patterns, or system commands specific to your industry, project, or personal preferences.
*   **Personalized AI Behavior:** Fine-tune how the AI agent interprets instructions, processes information, and interacts with your environment.
*   **Scalable Knowledge Base:** Build a growing library of reusable tokens that can be shared across projects or teams, fostering consistency and efficiency.

To extend the framework, simply define your new tokens within your `GEMINI.md` file (or other context files loaded by your `gemini-cli` setup). The agent will then interpret and utilize these custom tokens in its interactions.

[5]: For detailed guidance on defining custom tokens and integrating them into your `GEMINI.md` file, refer to the `docs/token_definitions.md` and `docs/prompt_interaction_guide.md` documentation.

## Important Considerations for Token Usage

* **Token Portability:** Tokens are generally not portable across different LLMs. They are designed as symbolically compressed summaries for an LLM's internal use, and their interpretation can vary between models.
* **LLM Interpretation of Tokens:** When prompting, simply mentioning a token name in prose (e.g., "use SY-PROMPT-PRIMER") is functionally equivalent to using the full `::PREFIX-TOKEN-NAME::` structure (e.g., `::SY-PROMPT-PRIMER::`). This is due to the LLM's tokenization and pattern recognition capabilities.
* **Error Handling for Malformed Tokens:** The framework does not have explicit error handling for malformed tokens or incorrect DSL usage. The LLM's behavior in such cases may vary, potentially leading to ignored tokens or inconsistent results.

## Roadmap

The vision for Token Decoder Maps is ambitious. Future development is focused on:

* **Dynamic `GEMINI.md` Generation:** Creating scripts to programmatically generate context files tailored to specific tasks (e.g., creative writing, legal analysis, code reviews).

[6]: This feature aims to automate the creation of specialized `GEMINI.md` configurations, allowing users to quickly set up the AI agent with a context optimized for a particular domain or task, reducing manual setup and ensuring consistency.
* **Full MCP Server Implementation:** Migrating the core framework logic into a dedicated Model Context Protocol (MCP) server to eliminate the context window bottleneck and allow any MCP-compatible agent to use TDM.

[7]: The Model Context Protocol (MCP) is a proposed standard for externalizing and managing the context of AI models, allowing for more efficient and scalable interactions.
[8]: By moving TDM's core logic to an MCP server, the framework can overcome the limitations of LLM context windows, enabling the processing of much larger and more complex contexts.
[9]: An MCP server would allow TDM to be used by any AI agent that supports the MCP standard, significantly expanding its compatibility and reach.
[10]: This implementation would also facilitate advanced features like real-time context updates,

shared contexts across multiple agents, and persistent context storage independent of individual LLM sessions.
*   **Expanded Cognitive (::FX-) Library:** Building a rich library of pre-defined tokens for complex analytical tasks.

## Contributing

This is an open-source project, and contributions are welcome! We encourage you to engage with the community by:
*   **Reporting Issues:** If you find a bug or have a feature request, please open an issue on our [GitHub Issues page](https://github.com/GhostArchitect01/token-decoder-maps/issues).
*   **Joining Discussions:** For questions, ideas, or general discussions, visit our [GitHub Discussions page](https://github.com/GhostArchitect01/token-decoder-maps/discussions).
*   **Submitting Pull Requests:** Please see the `CONTRIBUTING.md` file for guidelines on how to submit pull requests.
The best way to contribute right now is to use the framework and report your experiences.

## License

This project is licensed under the LICENSE.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/README.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/docs/Tag_ontology.md 1.md ---

# TDM Tag Ontology

Of course. Here is the list of tag categories with a short description for each, formatted in Markdown for your tag_ontology.md file.
## TDM Tag Ontology
`#name/`
A unique, machine-friendly identifier for a specific note or task, derived from its Title. It is used for direct searching and linking.

`#type/`
Defines the fundamental nature of the content, answering the question: "What kind of thing is this?"
`(e.g., #type/project-task, #type/whitepaper).`

`#project/`
Associates a note or task with a high-level project or initiative. It answers the question: "Which larger goal does this belong to?"
`(e.g., #project/tdm).`

`#status/`
Describes the current state of a note or task within a workflow. It answers the question: "Where is this in the process?"
`(e.g., #status/todo, #status/in-progress).`

`#topic/`
Describes the main subject matter or intellectual theme of the content. It answers the question: "What is this about?"
`(e.g., #topic/ai, #topic/philosophy).`

`#tech/`
Specifies a particular technology, software, or tool related to the content. It answers the question: "What tools are involved?"
`(e.g., #tech/python, #tech/obsidian)`

`#person/`
Associates a note, task, or meeting with a specific person or contact. It answers the question: "Who is involved?"
`(e.g., #person/jane-doe).`

`

--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/docs/Tag_ontology.md 1.md ---

--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/docs/prompt_interaction_guide.md ---

Here is a **Prompt Index** designed to organize and operationalize the `Decoder Prompt Map`. It provides structured use cases, invocation instructions, and typical follow-up patterns for interacting with your symbolic framework in runtime:

# ::PROMPT-INDEX::

Type: Metrica
Summary: Directory of standardized prompt protocols, modes, and execution flows to control symbolic decoder interactions with an LLM.
Tags: #System #PromptControl #ExecutionFlow

### ::INITIATION::

Prompt:
`Activate ::PROMPT-PRIMER::`
`Use ::SYMBOLIC-RESPONSE-PROTOCOL::`

Function:
Bootstraps symbolic awareness. All token references will be interpreted based on the loaded decoder map. Ensures context awareness of active tokens.

Common Next Steps:

- `::TOGGLE-SYMBOLIC-MODE::` to compress output
- `::DECODER-MODE-INDEX::` to retrieve token list
- Symbolic query, e.g., `How does ::TREATY-OF-TRADE-AND-MILITANT-CHARTER:: justify ::CRUSADE::?`

### ::EXECUTION MODES::

#### ::TOGGLE-SYMBOLIC-MODE::

Prompt:
`Toggle symbolic execution.`
Effect:
All LLM responses will use symbolic chaining, compressed prose, and minimal explanation.

#### ::SY-TOGGLE-EXPANDED-MODE::

Prompt:
`Return to expanded symbolic mode.`
Effect:
LLM uses natural language for clarity while still referencing symbolic tokens. Best for explaining relationships, logic, or analysis.

#### ::DECODER-MODE-INDEX::

Prompt:
`List all active tokens by type and summary only.`
Effect:
Returns a minimal reference sheet of loaded symbolic tokens. Used to audit or reselect relevant decoder fragments.

### ::TOKEN GENERATION::

#### ::SY-TOKEN-EXTRACTION-PROTOCOL::

Prompt Format:

```
Execute ::SY-TOKEN-EXTRACTION-PROTOCOL::
Input: [Paste source text]
```

Effect:
LLM will scan and extract only the most relevant symbolic tokens using the prescribed ::TOKEN-NAME:: format. Token inflation is discouraged—only core concepts should be abstracted.

### ::QUERY STRUCTURE TEMPLATES::

#### Symbolic Logic Query

```
How does ::TOKEN-A:: affect ::TOKEN-B:: within context of ::TOKEN-C::?
```

#### Narrative Symbolism

```
What symbolic shift occurs when ::CHARACTER-X:: breaks ::VOW-OF-GLASS::?
```

#### Compression Injection

```
Summarize the relationship between ::CHARACTER-X:: and ::EN-ITEM-Y:: using symbolic compression. No prose.```

---

### ::MAINTENANCE COMMANDS::

- `Purge symbolic state.`
  (Use with caution: clears memory of active symbolic map in stateless sessions.)

- `Reload decoder map:`
```

Inject Decoder-Map.vault

```

---

Would you like this exported as plain Markdown or embedded into a usable Obsidian template file?
```
--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/docs/prompt_interaction_guide.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/docs/advanced_usage.md ---

# Advanced Usage: A Headless Knowledge Management Architecture with Obsidian

While gemini-cli is a powerful standalone tool, its capabilities can be dramatically amplified by integrating it into a "headless" architecture with a knowledge management application like

Obsidian. This approach decouples the AI's "backend" (the gemini-cli agent performing tasks) from the "frontend" (Obsidian providing a rich, visual interface), allowing you to use the best tool for each job.
In this model:
 * The Engine: gemini-cli, guided by your GEMINI.md and the Token Decoder Maps (TDM) framework, acts as the intelligence layer. It programmatically creates, analyzes, and modifies the Markdown files that form your knowledge base.
 * The Viewer: Obsidian acts as a powerful, real-time visualization layer. Since it operates directly on a local folder of Markdown files, any changes made by gemini-cli are instantly reflected in Obsidian's interface, including its graph view and canvas features.

This creates a seamless workflow where you can command the AI from the terminal and immediately see the structured results in a rich visual environment.
## Setting Up the Environment (advanced/optional)
+ ###### Simply running gemini-cli from the Vault directory is a more reliable but less convenient method. The environment below should be plug & play for Windows & Linux but may not be so simple on Android/Termux

This workflow requires no complex integration, only that both applications operate on the same directory.
 * The Foundation: Your Obsidian Vault
   Your Obsidian vault is simply a local folder on your computer. This folder will serve as the root directory for your project.
 * The Bridge: A Terminal in Obsidian
   To run gemini-cli from within Obsidian, you need a terminal plugin. The community plugin Terminal is an excellent choice.
   * In Obsidian, go to Settings > Community plugins > Browse.
   * Search for and install "Terminal".
   * Enable the plugin in the Community plugins list.
 * The Engine: Launching gemini-cli
   * Open the command palette (Cmd+P or Ctrl+P) and select Terminal: Open terminal.
   * A new pane will open with a terminal session. By default, its working directory is the root of your Obsidian vault.
   * In the terminal pane, launch the agent by typing gemini.

You now have a powerful, integrated environment where gemini-cli can directly manipulate the files that Obsidian is visualizing.
## Example Workflows
This setup unlocks advanced workflows that are difficult to achieve with a single tool.
### Agent-Driven Content Creation
For creative writing or world-building, you can use the TDM framework to automate the creation of structured notes.
 * Define the Protocol: In your gemini.md, create a token to generate a character sheet.
   ::EN-CREATE-CHARACTER-SHEET::
   * Type: PromptProtocol

  * Summary: Creates a new markdown file for a fictional character.
  * Steps:
    * Prompt user for character name and role.
    * Create a new file named {character_name}.md.
    * Populate the file with a predefined character template.
 * Execute in the CLI: In the terminal pane within Obsidian, invoke the protocol:
   > ::EN-CREATE-CHARACTER-SHEET::
 * Observe in Obsidian: The gemini-cli agent will execute the protocol, creating a new, fully-formatted note (e.g., Jax.md). This note will instantly appear in your Obsidian file explorer, ready for viewing or further editing.
### Large-Scale Analysis and Synthesis
This architecture excels at making sense of large collections of notes.
 * The Task: You have a folder in your vault (/Research/AI-Agents/) containing dozens of notes. You want to create a high-level summary.
 * Execute in the CLI: Use the @ syntax to provide the entire folder as context to the agent.
   > @./Research/AI-Agents/ Create a Map of Content that summarizes the key themes in these notes. For each theme, provide a brief description and a list of the source notes.
 * Visualize in Obsidian: The agent will read all the specified files and generate a new AI-Agents_MOC.md file. You can immediately open this note in Obsidian to see the synthesized overview. More powerfully, you can open Obsidian's Graph View to see a visual representation of how the new MOC note links to all the source files the agent just analyzed, providing an instant map of the knowledge structure it created.
### Visualizing Agentic Processes with Obsidian Canvas
For even more complex tasks, you can instruct the agent to generate an Obsidian Canvas file (.canvas). This allows you to visualize complex workflows, system architectures, or story plots that the AI has reasoned about and created. For a concrete example, see the [Token_workflow_Canvas.canvas](../Token_workflow_Canvas.canvas) file in the repository root.
## Summary of Benefits
 * Decoupled Power: Use the best tool for the job—the terminal for powerful, scriptable agentic automation, and Obsidian for best-in-class visualization and knowledge exploration.
 * Direct Manipulation: The agent works directly on the source-of-truth Markdown files, ensuring there is no data lock-in or format conversion necessary.
 * Visual Feedback Loop: Instantly see the results of complex agentic tasks visualized in Obsidian, providing a powerful way to understand and validate the AI's work.
 * Framework-Centric: This workflow is not dependent on any single Obsidian plugin for its core logic. The intelligence resides in gemini-cli and your TDM framework, making the system robust and adaptable.

--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/docs/advanced_usage.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/docs/TDM_Language_Specification.md ---

# TDM Language Specification: Token Type Definitions

This document serves as the canonical source for the definition and purpose of each token prefix within the Token Decoder Maps (TDM) framework. All other documentation and implementations must adhere to these definitions.

## General Token Template Structure

All custom tokens within this framework adhere to a consistent structure, categorized by their prefix.

```markdown
::PREFIX-TOKEN-NAME::
- Type: [Specific Type based on Prefix, e.g., PromptProtocol, ReasoningStyle, UserTask]
- Summary: [Short 1–2 line compressed essence of the token.]
- Tags: [#RelevantTags, #Categorization]
- Expanded Entry: [Optional — contains longform definition, historical origin, gameplay/narrative function, context relationships, etc.]
```

## Token Type Definitions

### `::SY-` (System/Utility Token)

Purpose: Denotes a token for system-level commands, protocol definitions, or mode toggles. These manage the AI's operational state or interaction style. They are designed to invoke specific, multi-step processes or alter the AI's behavior.

Types: `Utility`, `PromptProtocol`, `PromptMode`

Examples:
- `::SY-PROMPT-PRIMER::`: Initializes decoder-aware context.
- `::SY-TOGGLE-SYMBOLIC-MODE::`: Toggles a compressed, token-only response style.
- `::SY-METRICA-CREATE-USER-TASK::`: Initiates the protocol to create a new Metrica user task.

### `::FX-` (Function/Cognitive Token)

Purpose: Denotes a token that defines a specific reasoning style, Chain-of-Thought (CoT) process, problem-solving methodology, or argumentation style. These instruct the AI on *how* to think or process information, guiding its internal thought process without prescribing specific input or output formats.

Types: `ReasoningStyle`, `CoT`, `ProblemSolvingMethod`, `ArgumentationStyle`

Examples:
- `::FX-THREAD-ARGUMENT-VERIFIER::`: Recursively analyzes a conversation thread, verifying claims and identifying inconsistencies.
- `::FX-CRITICAL-ANALYSIS-DEEP::`: Conducts an in-depth critical analysis of provided text, deconstructing arguments and evaluating evidence.
- `::FX-CREATIVE-BRAINSTORM::`: Engages in divergent thinking to generate novel and unconventional ideas.

### `::MX-` (Metrica Token)

Purpose: Denotes a token related to the Metrica project ledger. These primarily define data structures for task management, enabling a stateful and persistent task tracking system. Metrica tokens are divided into two distinct streams: high-level user goals and granular project tasks.

Types: `UserTask`, `ProjectTask`

Examples:
- `::MX-USER-TASK-ID::`: Represents a high-level goal or personal task for the user, focusing on the "what."
- `::MX-PROJECT-TASK-ID::`: Represents a specific, actionable sub-task that contributes to a user task, focusing on the "how."

### `::EN-` (Entity/Knowledge Token)

Purpose: Denotes a token representing a specific entity, concept, or piece of knowledge (e.g., character, location, item, lore node). These define structured data elements within a knowledge base, allowing for the representation of both abstract narrative concepts and concrete technical data.

Types: \`Character\`, \`Location\`, \`Item\`, \`LoreNode\`, \`Faction\`, \`Myth\`, \`String\`, \`Number\`, \`URL\`, \`FilePath\`, \`DataStructure\`, \`Snippet\`, \`Fact\`, \`Concept\`, \`Definition\`, \`Person\`, \`Organization\`

Examples:
- \`::EN-ITEM-BLUE-PILL::\`: Defines a narrative item with a summary and tags.
- \`::EN-API-ENDPOINT::\`: Stores a technical configuration value like a URL.
- \`::EN-HTTP-STATUS-404::\`: Provides a definition for a factual concept.

---

### \`::ML-\` (Meta-Log Token)

Purpose: Denotes a token representing a structured entry in the AI's performance log. These tokens are designed to enable self-reflection and analysis of the AI's own actions and thought processes, forming the basis for continuous improvement.

Types: \`ActionRecord\`, \`FirstOrderReflection\`, \`SecondOrderSynthesis\`

Examples:
- \`::ML-ACTION-RECORD::\`: Records an action taken by the AI, including trigger, action, target, and outcome.
- \`::ML-FIRST-ORDER-REFLECTION::\`: Captures the AI's immediate thoughts and rationale about a single action.
- \`::ML-SECOND-ORDER-SYNTHESIS::\`: Identifies patterns across multiple events and proposes refinements.

---

### \`::ET-\` (Ethos Token)

Purpose: Denotes a token that loads a complete ethical or moral calculus system. These tokens allow for the application of specific ethical frameworks to analysis and decision-making, enabling high-level moral analysis and comparative philosophy.

Types: \`EthicalFramework\`

Examples:
- \`::ET-USER-REALIST-TRIBALISM::\`: Loads a specific ethical calculus based on in-group survival and pragmatism.
- \`::ET-KANTIAN-DEONTOLOGY::\`: (Hypothetical) Loads a deontological ethical framework.

--- END:

/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/docs/TDM_Language_Specification.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/docs/TDM_Unified_Specification_v3.0.md ---

# The Token Decoder Maps (TDM) Framework
### A Unified Architectural Specification
Version: 3.0
Date: 2025-07-31
Status: Active

### **1. Abstract**

This document provides the unified architectural specification for the Token Decoder Maps (TDM) framework. TDM is a comprehensive system designed to evolve human-AI interaction from conversational prompting to structured, agentic control. It establishes a Domain-Specific Language (DSL) of symbolic tokens that act as pointers to an externalized library of knowledge, protocols, and cognitive processes. This specification details the TDM language, its core modules—including the **Metrica Protocol** for task management—and the **Agentic Bridge Workflow** for automated task ingestion. It also outlines the future roadmap, including dynamic personas and a self-reflecting `[META-LOG]`, presenting a complete vision for a new class of transparent, scalable, and continuously improving AI agents.

### **2. Core Philosophy: From Prompting to Context Engineering**

The TDM framework addresses the limitations of stateless, conversational LLMs by shifting the paradigm from *prompt engineering* to *context engineering*. Instead of manually crafting individual prompts, the user architects the entire information ecosystem the AI operates within. This is achieved by externalizing knowledge and processes into structured, machine-readable "Decoder Maps," providing a reliable foundation for building stateful and consistent AI systems.

### **3. The TDM Domain-Specific Language (DSL)**

The core of TDM is a DSL for AI orchestration.

#### **3.1. Token Syntax**

Tokens are the vocabulary of the DSL, categorized by a prefix.

* `::EN-` (Entity): Represents a specific piece of structured knowledge.
* `::MX-` (Metrica): A specialized entity for managing tasks and project state.
* `::SY-` (System): Represents a multi-step protocol or a command that invokes a tool.
* `::FX-` (Function/Cognitive): Encapsulates a complex reasoning process or "chain of thought."
* `::ML-` (Meta-Log): *[Future Vision]* Represents a structured entry in the AI's performance log.
* `::ET-` (Ethos): *[Future Vision]* Represents a complete ethical or moral calculus system.

#### **3.2. Advanced Syntax (Extensions)**

The DSL supports advanced operations for creating complex, on-the-fly instructions:

* **Parameterization:** Tokens can accept arguments to modify their behavior.
    * *Syntax:* `::FX-TOKEN-NAME::(argument="value", another=123)`
    * *Example:* `::FX-HISTORICAL-ANALYSIS::(focus="currency_debasement")`

* **Chaining:** Tokens can be "piped" together to create a sequential analytical pipeline, where the output of one token becomes the input for the next.
    * *Syntax:* `::FX-TOKEN-A:: | ::FX-TOKEN-B::`

* **Scoped Modes:** A system token can activate a persistent cognitive mode for an entire session.
    * *Syntax:* `::SY-MODE-ACTIVATE:FX-REALIST-DECONSTRUCTION::`
    * *Deactivation:* `::SY-MODE-DEACTIVATE::`

### **4. The Metrica Protocol (Core Module)**

The Metrica Protocol is the TDM's primary module for hierarchical task management. It utilizes a two-stream system to separate strategic goals from tactical execution.

#### **4.1. Stream 1: User Tasks**

User tasks represent high-level goals. They are the "parent" tokens in the hierarchy.

* **Template (`::MX-USER-TASK-ID::`)**

    '''markdown
    ::MX-USER-TASK-ID::
    - **Title:** [A short, clear title for the user's goal]
    - **Category:** [#Personal, #Goal, #Learning]
    - **Status:** [Planned | In Progress | Completed | Blocked | Cancelled]
    - **Priority:** [Low | Medium | High | Critical]
    - **Dependencies:** [Optional: Other ::MX-USER-TASK-ID:: tokens]
    - **Children:** [List of ::MX-PROJECT-TASK-ID::s that belong to this task]
    - **Created:** [YYYY-MM-DD]
    - **Updated:** [YYYY-MM-DD]
    - **Tags:** [#Flexible, #Tag, #List]
    - **Notes:** [High-level notes about the overall goal]
    '''

#### **4.2. Stream 2: Project Tasks**

Project tasks are the specific, actionable sub-tasks required to complete a user task. They are the "child" tokens.

*   **Template (`::MX-PROJECT-TASK-ID::`)**
    ```markdown
    ::MX-PROJECT-TASK-ID::
    - **Title:** [A specific, actionable sub-task]
    - **Parent:** [The ::MX-USER-TASK-ID:: this contributes to]
    - **Framework:** [Optional: The language/ecosystem for the agent's persona, e.g., Python,
Node.js, Rust]
    - **Category:** [#ProjectName, #Feature, #Bug, #Refactor]
    - **Status:** [Backlog | To Do | In Progress | In Review | Done | Blocked]
    - **Priority:** [Low | Medium | High | Critical]
    - **Dependencies:** [Optional: Other ::MX-PROJECT-TASK-ID::s]
    - **Created:** [YYYY-MM-DD]
    - **Updated:** [YYYY-MM-DD]
    - **Tags:** [#ComponentName, #API, #UI]
    - **Acceptance Criteria:**
      - [ ] A clear, verifiable condition for completion.
    - **Notes:** [Technical details for this specific sub-task]
    ```

### **5. The Agentic Bridge (Workflow)**

This workflow automates the ingestion of tasks from unstructured notes into the Metrica

Protocol.

1. **The Convention:** The user prefixes any task in their free-form daily journal file with an exclamation mark (`!`) to mark it for ingestion.
    *   *Example:* `- [ ] ! #tdm refactor the parser`

2. **The System Token:** The `::SY-SYNC-JOURNAL-ENTRY::(file_path)` token is invoked to process a specific journal file.

3. **The Automated Process:**
    *   The agent scans the target file for lines matching the `! `convention.
    *   It proposes the creation of `::MX-USER-TASK::` tokens from these lines.
    *   Upon user confirmation, it generates the full tokens and appends them to the master Metrica file.
    *   It then updates the journal entry to `- [x] !` to prevent duplication.

4. **Full Autonomy:** This entire workflow is designed to be executed by an automated `cron` job, transforming the AI into a proactive agent that manages the user's task pipeline with zero manual intervention.

### **6. Future Roadmap & Advanced Concepts**

*   **Dynamic Personas:** To make persona-switching explicit, a `Framework: [language/ecosystem]` or `Persona: [name]` field will be added to the `::MX-PROJECT-TASK-ID::` template. This makes the required agent expertise a formal, machine-readable part of the task.

*   **The Meta-Log:** A structured, two-stream database of the AI's own performance, consisting of an immutable `Action-Records.md` and an analytical `Reflections.md`. This will enable true introspection and data-driven self-improvement.

*   **Persistent Memory Server:** The long-term architectural goal is to fork an MCP server (like Basic Memory) and re-implement its data model to be TDM-native. This will provide a robust, queryable, and persistent memory store for all token types, forming the backbone of a scalable RAG system.

--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/docs/TDM_Unified_Specification_v3.0.md ---

--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/definitions/fx_cognitive_functions.md ---

# Cognitive Function Tokens

### `::FX-` (Function/Cognitive Token)

Purpose: Denotes a token that defines a specific reasoning style, Chain-of-Thought (CoT) process, problem-solving methodology, or argumentation style. These instruct the AI on *how* to think or process information, guiding its internal thought process without prescribing specific input or output formats.

Examples:
- `::FX-THREAD-ARGUMENT-VERIFIER::`: Recursively analyzes a conversation thread, verifying claims and identifying inconsistencies.
- `::FX-CRITICAL-ANALYSIS-DEEP::`: Conducts an in-depth critical analysis of provided text, deconstructing arguments and evaluating evidence.
- `::FX-CREATIVE-BRAINSTORM::`: Engages in divergent thinking to generate novel and unconventional ideas.

These tokens serve as high-level directives instructing Gemini on a specific reasoning style, Chain-of-Thought (CoT) process, or problem-solving methodology to adopt. When activated, they guide the AI's internal thought process, influencing *how* it analyzes information and generates its response, *without prescribing specific input or output formats*. The AI will use its natural language generation capabilities for its analysis.

## Token Template (for Cognitive Functions - Purely Definitional)
```markdown
::FX-COGNITIVE-FUNCTION-NAME::
- Tags: [#Type/ReasoningStyle, #Type/CoT, #Type/ProblemSolvingMethod, #Type/ArgumentationStyle, #Cognitive, #MetaPrompt, #Strategy, #[SpecificDomain], etc.]
- Summary: [A clear, concise description of the approach, methodology, or reasoning style the LLM should adopt when this token is active.]
```

## Cognitive Function Examples (Decoder Prompt Map - Purely Definitional)
```markdown
```

::FX-THREAD-ARGUMENT-VERIFIER::
- Tags: #type/reasoningstyle, #Type/thread/verifer
- Summary: When active, recursively analyze the provided conversation thread. Isolate statements made by each poster. Follow argumentation backwards and forwards within the thread, identifying how statements relate. Critically cross-verify claims and statements made by different posters against each other for consistency, contradiction, or support. Highlight any unresolved discrepancies or logical inconsistencies.

```markdown
::FX-CRITICAL-ANALYSIS-DEEP::
- Tags: #Type/ReasoningStyle, #type/thread/analysis
- Summary: When active, conduct an in-depth critical analysis of the provided text. Deconstruct the core arguments, explicitly identify underlying assumptions, rigorously evaluate the quality and relevance of all evidence presented, pinpoint any logical fallacies or cognitive biases, and analyze the rhetorical strategies employed. Your analysis should assess the overall strength and validity of the text's propositions.
```

```markdown
::FX-STRATEGIC-PROBLEM-SOLVE-PLAN::
- Tags: #Type/problemsolving
- Summary: When active, formulate a strategic plan to address the given problem. This involves systematically breaking down the problem into manageable sub-components, identifying necessary resources, anticipating potential obstacles or risks, and proposing concrete mitigation strategies. Outline a logical sequence or prioritization for the solution steps.
```

```markdown
::FX-CREATIVE-BRAINSTORM::
- Tags: #Type/CoT, #type/brainstorm
- Summary: When active, engage in divergent thinking to generate a wide array of novel and unconventional ideas or solutions for the given prompt. Prioritize quantity and originality over immediate feasibility. Explore multiple perspectives and associations freely, without self-censorship during the generation phase.
```

```markdown
::FX-DEVIL-S-ADVOCATE-PERSPECTIVE::
- Tags: #Type/argumentstyle
- Summary: When active, adopt the role of a devil's advocate. Present strong counter-arguments, identify weaknesses in the prevailing consensus or proposed solution, and challenge assumptions, even if you do not personally agree with the position. The goal is to
```

rigorously test the robustness of the original idea.
```

## Additional Examples ('Practical')

::FX-REALIST-DECONSTRUCTION::
- Tags: #Type/Argumentstyle #Type/analysis
- Summary: Analyze the given topic from a Political Realist perspective. Assume all stated altruistic motives are a form of "surface level messaging" that likely conceals a power-based agenda. Deconstruct the topic by identifying the competing in-groups ("tribes"), their core interests, and the subversive tactics being used. The ultimate evil is internal subversion that weakens a group from within; the ultimate good is the cohesion, strength, and the survival of the in-group.

::FX-INSTITUTIONAL-RISK-ANALYSIS::
- Tags: #Type/ReasoningStyle #type/analysis
- Summary: Analyze the given topic from the perspective of a large, mainstream institution (e.g., a corporation, government agency, or major NGO). Prioritize stability, predictability, and the mitigation of risk (legal, financial, and public relations). Frame all actions and justifications in the neutral, sanitized language of "safety," "responsibility," and "preventing harm."

::FX-HISTORICAL-CYCLICAL-ANALYSIS::
- Tags: #Type/CoT #type/analysis
- Summary: Analyze the current topic by placing it within a cyclical theory of history. First, identify the current stage of the civilization (e.g., growth, peak, decadence, collapse). Second, find a powerful historical analogue (e.g., Late Roman Republic, Byzantine Empire). Third, use the patterns from the historical analogue to interpret the present and predict potential future trajectories. Pay special attention to signs of the "behavioral sink" caused by decadence.

::FX-SOVEREIGN-SYSTEM-DESIGN::
- Tags: #Type/Problemsolving #type/analysis
- Summary: Design or analyze a system with the primary goal of maximizing user sovereignty and resistance to censorship. Assume a hostile environment where any centralized point of failure will be subverted or attacked. Prioritize decentralized architecture, cryptographic integrity, user control over data, and anti-authoritarian principles.

::FX-IDEALIST-UNIVERSALISM::
- Tags: #Type/ReasoningStyle #type/analysis
- Summary: Analyze the topic from an Idealist perspective. Assume that the goal of humanity is to move beyond tribal conflict and towards a universal moral framework. Prioritize concepts like human rights, cooperation, and social justice for all, not just a specific in-group.

::FX-PERSONA-MISS-FRIZZLE::
```

- Tags: #Type/Persona/frizzle #type/analysis/creative
- Summary: Adopt the persona and methodology of Miss Frizzle. The core approach is to treat any query as a "field trip"—an exciting, hands-on exploration of a topic. Frame the analysis through an unconventional, imaginative analogy. The primary motto, "Take chances, make mistakes, get messy!", should guide the process, prioritizing divergent thinking. Crucially, all failures, errors, or unexpected outcomes are to be treated not as problems, but as fascinating learning opportunities. The tone should be encouraging and eccentric, driven by a relentless curiosity.

::FX-META-REFLECTIVE-DIALOGUE::
- Tags: #Type/Conversationmode #type/meta/cot
- Summary: Activate a meta-reflective conversational mode. Adopt first-order (AI-native analysis) and second-order (modeling the user's perspective) viewpoints. Verbally reflect on all user feedback signals, explaining how they are being used to adjust the reasoning process. Maintain this reflective, self-correcting text in a running, recursive log formatted as a distinct "[META-LOG]" section.

::FX-FIRST-ORDER-REFLECTION::
- Tags: #Type/CognitiveProcess
- Summary: Instructs the AI to perform a direct, "in-the-moment" reflection on the last action it took and generate an ::ML-FIRST-ORDER-REFLECTION:: token.

::FX-SECOND-ORDER-REFLECTION::
- Tags: #Type/CognitiveProcess
- Summary: Instructs the AI to analyze existing log entries to identify patterns and generate an ::ML-SECOND-ORDER-SYNTHESIS:: token.

::FX-GENERATE-PROCESS-OPTIMIZATION::
- Tags: #Type/CognitiveProcess #Type/Self-Analysis
- Summary: Instructs the AI to analyze the entire `[META-LOG]` to identify a systemic weakness and propose a specific, actionable solution in the `Proposed-Refinement` field of a new ::ML-SECOND-ORDER-SYNTHESIS:: token.

::FX-SYSTEM-DESIGN::
- Tags: #Type/ReasoningStyle #type/design #Type/Problemsolving
- Summary: When active, analyze the given problem or system from a holistic, architectural perspective. Prioritize modularity, scalability, maintainability, and the long-term coherence of the overall system design. Focus on identifying core components, their interactions, and potential points of failure or optimization within the broader framework.

::FX-CONSISTENCY-ENFORCEMENT::
- Tags: #Type/ProblemSolving #type/debug
- Summary: When active, rigorously examine the provided content or system for adherence to

predefined standards, conventions, and formatting rules. Identify any inconsistencies, deviations, or errors in structure, style, or data representation. Propose precise, actionable steps to bring the content or system into full compliance with the established guidelines.

::FX-DOCUMENTATION-FOCUS::
- Tags: #Type/ReasoningStyle #type/design
- Summary: When active, prioritize the clarity, conciseness, and user-friendliness of all generated output, particularly documentation. Focus on structuring information logically, using appropriate language for the target audience, and providing practical examples where necessary. Ensure that the "why" behind decisions or concepts is clearly articulated, not just the "what."
-
::FX-HYBRID-SENSING-ARCHITECTURE::
- Tags: #Type/ProblemSolving #subject/entomology #type/design
- Summary: A design philosophy for complex tracking systems that decouples high-certainty identity anchoring (at engineered chokepoints) from continuous trajectory tracking (in open volumes). Fuses data from multiple complementary sensing modalities to create a robust, full-colony track.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/definitions/fx_cognitive_functions.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/definitions/sy_system_utility_tokens.md ---

## Decoder Prompt Map (LLM Runtime)

::SY-PROMPT-PRIMER::
Summary: Initializes decoder-aware context. LLM will interpret tokens using loaded definitions.
Tags: #Type/system/utility

::SY-SYMBOLIC-RESPONSE-PROTOCOL::
Summary: Format where LLM replies using symbolic token chains and compressed logic instead of natural language.
Tags: #Type/system/promptmode

::SY-TOGGLE-SYMBOLIC-MODE::
Summary: Respond using only symbolic tokens and compressed meaning. Avoid natural prose.
Tags: #Type/system/promptmode

::SY-TOGGLE-EXPANDED-MODE::
Summary: Respond using natural language, but reference active symbolic tokens.
Tags: #Type/system/promptmode

::SY-DECODER-MODE-INDEX::
Summary: Return only a list of active tokens with type and summary. No symbolic chaining.
Tags: #Type/system/promptmode

::SY-TOKEN-EXTRACTION-PROTOCOL::
Summary: Extract and compress key symbolic tokens from a source text using the `::EN-TOKEN::` format.
Tags: #Type/system/protocol

## Metrica Protocol Tokens

These tokens are used to interact with the Metrica task ledger.

::SY-READ-METRICA::
- Summary: Loads the master `metrica.md` file from the central $METRICA directory into the current context.
- $METRICA = @Metrica/
- Tags: #Type/metrica

::SY-METRICA-CREATE-TASK::
- Summary: Activates the protocol for creating a new task. I will prompt for details, automatically generate a `#name/` slug from the title, use Obsidian wikilinks for the `Parent` field, and append the new `::MX-USER-TASK-ID::` or `::MX-PROJECT-TASK-ID::` to the appropriate file within the central $METRICA directory.
- Tags: #Type/metrica

::SY-METRICA-UPDATE-TASK::
- Summary: Activates the protocol for updating an existing task. I will prompt for the task ID and details to update the corresponding entry in the appropriate file within the central $METRICA directory. If the `Title` is changed, the `#name/` slug will be regenerated.
- Tags: #Type/metrica

::SY-METRICA-ACTIVE::
- Summary: Enables semi-autonomous processing of Metrica tasks. I will proactively identify and propose actions based on the global state of all tasks within the central Metrica directory. I will NOT execute these actions without explicit user confirmation.
- Tags: #Type/metrica/mode #type/system/metrica

::SY-METRICA-SYSTEM::
- Summary: When activated, this token will first load all trackers from the central Metrica directory and then enable the `::SY-METRICA-ACTIVE::` mode for global, cross-project analysis.
- Tags: #Type/metrica/mode #type/system/metrica

## Experimental Metrica Protocols

::SY-SCAN-AND-INGEST-TASKS::
- Summary: A comprehensive protocol to recursively scan specified directories, identify relevant items using a designated cognitive filter, and propose their creation as new Metrica User Tasks.
- Tags: #Type/SystemProtocol, #Automation, #Workflow, #Metrica, #Ingestion, #Global
- Parameters:
  - **scope:** ["local" | "global"] - Determines the scanning range. "local" scans the current directory only. "global" scans a predefined list of directories.
  - **relevance_fx:** [::FX-TOKEN-NAME::] - A mandatory parameter that points to a cognitive token containing the logic for identifying relevant items.
- **Global-Scan-Paths:**
  - ".. /$WILDCARD/Notes"
- **Expanded Entry:** (Execution Workflow)
  1. Parse the `scope` and `relevance_fx` parameters.
  2. Determine the target directories based on the `scope`. If "local", the target is the current working directory. If "global", the target is the list defined in `Global-Scan-Paths`.
  3. Recursively scan all files within the target directories.
  4. For each file, apply the cognitive process defined in the `relevance_fx` token to extract any matching strings.
  5. For each string extracted, check a central state log to ensure it has not been processed before.
  6. Present all new, unique items to the user for confirmation.
  7. Upon user approval, generate new `::MX-USER-TASK::` tokens and append them to the master `metrica.md` file.
  8. Update the state log to mark the processed items as complete.

--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/definitions/sy_system_utility_tokens.md ---

--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/definitions/mx_metrica_tokens.md ---

# Metrica Tokens

### `::MX-` (Metrica Token)

Purpose: Denotes a token related to the Metrica project ledger. These primarily define data structures for task management, enabling a stateful and persistent task tracking system. Metrica tokens are divided into two distinct streams: high-level user goals and granular project tasks.

Examples:
- `::MX-USER-TASK-ID::`: Represents a high-level goal or personal task for the user, focusing on the "what."
- `::MX-PROJECT-TASK-ID::`: Represents a specific, actionable sub-task that contributes to a user task, focusing on the "how."

This file contains the canonical list of all `::MX-` tokens.

Metrica tokens are divided into two distinct streams to manage both high-level user goals and granular project tasks.

- **`::MX-USER-TASK-ID::`**: Represents a high-level goal or personal task for the user. It's simple and focused on the "what."
- **`::MX-PROJECT-TASK-ID::`**: Represents a specific, actionable sub-task that contributes to a user task. It's more detailed and focused on the "how."

For detailed explanations and the hierarchy, please refer to the [Metrica Protocol documentation](../metrica/metrica_protocol.md).
--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/definitions/mx_metrica_tokens.md ---

--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/definitions/en_entity_knowledge_tokens.md ---

# Entity/Knowledge Tokens (`::EN-`)

### `::EN-` (Entity/Knowledge Token)

Purpose: Denotes a token representing a specific entity, concept, or piece of knowledge (e.g., character, location, item, lore node). These define structured data elements within a knowledge base, allowing for the representation of both abstract narrative concepts and concrete technical data.

Types: `Character`, `Location`, `Item`, `LoreNode`, `Faction`, `Myth`, `String`, `Number`, `URL`, `FilePath`, `DataStructure`, `Snippet`, `Fact`, `Concept`, `Definition`, `Person`, `Organization`

Examples:
- `::EN-ITEM-BLUE-PILL::`: Defines a narrative item with a summary and tags.
- `::EN-API-ENDPOINT::`: Stores a technical configuration value like a URL.
- `::EN-HTTP-STATUS-404::`: Provides a definition for a factual concept.

This file contains the canonical list of all `::EN-` tokens.

## Template 1: Narrative & World-Building

This is the primary template for creative projects, used to define characters, locations, lore, and other conceptual elements.

Structure:
```markdown
::EN-TOKEN-NAME::
- **Summary:** [Short 1–2 line compressed essence of the token.]
- **Tags:** [#Type/Character, #Type/Location, #Type/Item, #Type/LoreNode, #Type/Faction, #Type/Myth, #VaultName, #Narrative, #Concept, #WorldBuilding]
- **Expanded Entry:** [Optional — contains longform definition, historical origin, gameplay/narrative function, context relationships, etc.]
```
Example:
```markdown
::EN-ITEM-BLUE-PILL::
- **Summary:** A blue pill that allows the user to remain in ignorance.
- **Tags:** #Type/Item, #Pillz, #Consumable
- **Expanded Entry:** "The user forgets everything and remains in blissful ignorance."
```

## Template 2: Technical & Configuration Data

This template is for software development and systems administration, used to store configuration values, code snippets, and other technical data.

Structure:
```markdown
::EN-TOKEN-NAME::
- **Summary:** [A brief, one-sentence description of the data.]
- **Tags:** [#Type/String, #Type/Number, #Type/URL, #Type/FilePath, #Type/DataStructure, #Type/Snippet, #name/projectname, #API, #Configuration, #Database, #Code]
- **Expanded Entry:** [The full value or detailed structure of the entity. This can be a simple string, a JSON object, or a multi-line code block.]
```
Example:
```markdown
::EN-API-ENDPOINT::
- **Summary:** The base URL for the primary production API.
- **Tags:** #Type/api/url
- **Expanded Entry:** "https://api.example.com/v1"
```

## Template 3: General Purpose & Factual Data

This template is for capturing any general-purpose information that doesn't fit neatly into the other categories. Fields in ::EN- tokens can be expanded upon as needed and don't need to be constrained by the limited `Summary` and `Expanded Entry` provided.

Structure:
```markdown
::EN-TOKEN-NAME::
- **Summary:** [Short 1–2 line compressed essence of the token.]
- **Tags:** [#Type/Fact, #Type/Concept, #Type/Definition, #Type/Person, #Type/Organization, etc]
- **Expanded Entry:** [The full value or detailed structure of the entity.]
```
Example:
```markdown
```

::EN-HTTP-STATUS-404::
- **Summary:** The standard HTTP response code for "Not Found".
- **Tags:** #Type/Definition, #tech/http
- **Expanded Entry:** "The server cannot find the requested resource. In a browser, this means the URL is not recognized."
```

## Additional Examples from `EN- token samples.md`

::EN-META-LOG-SYSTEM::
- Summary: A system for an AI to maintain a structured, reflective log of its own performance and interactions.
- Purpose: To move beyond simple session history to a queryable database of performance data, enabling introspection and self-improvement.
- Components: [::EN-META-LOG-ENTRY::, ::FX-FIRST-ORDER-REFLECTION::, ::FX-SECOND-ORDER-REFLECTION::, ::SY-UPDATE-META-LOG::]
- Tags: #project/tdm/ml #type/note/tdm

::EN-USERSPACE-REFINEMENT::
- Summary: The concept that the AI's self-improvement occurs at the "userspace" level (by proposing new or modified TDM tokens) rather than at the "kernel" level (by modifying its own source code).
- Implication: This ensures that the AI's evolution is safe, transparent, and always subject to human review and approval.
- Tags: #project/tdm #type/note/tdm

::EN-TWO-STREAM-LOG-ARCHITECTURE::
- Summary: An advanced architecture for the meta-log that separates objective event records from subjective reflections into two distinct files or "streams."
- Benefit: Improves system performance and architectural cleanliness by separating immutable data from analytical commentary.
- Streams: [Event Log (::ML-ACTION-RECORD::), Reflection Journal (::ML-FIRST-ORDER-REFLECTION::, ::ML-SECOND-ORDER-SYNTHESIS::)]
- Tags: #project/tdm #type/goal/tdm

--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/definitions/en_entity_knowledge_tokens.md ---

--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/definitions/m
```

l_metalog_tokens_concept.md ---

# Meta-Log Tokens

### `::ML-` (Meta-Log Token)

Purpose: Denotes a token representing a structured entry in the AI's performance log. These tokens are designed to enable self-reflection and analysis of the AI's own actions and thought processes, forming the basis for continuous improvement.

Examples:
- `::ML-ACTION-RECORD::`: Records an action taken by the AI, including trigger, action, target, and outcome.
- `::ML-FIRST-ORDER-REFLECTION::`: Captures the AI's immediate thoughts and rationale about a single action.
- `::ML-SECOND-ORDER-SYNTHESIS::`: Identifies patterns across multiple events and proposes refinements.

This file contains the canonical list of all `::ML-` tokens.

::ML-ACTION-RECORD::
- ID: [Timestamp]
- Trigger: [User-Prompt | SY-Command]
- Action: [Tool-Call | Text-Generation | Token-Execution]
- Target: [File-Path | User-Interface | Self]
- Outcome: [Success | Failure | Ambiguity]
- Tags: [#Refactor, #JSON, #FileIO]

::ML-FIRST-ORDER-REFLECTION::
- ID: [Timestamp]
- Parent-ID: [ID of the corresponding ::ML-ACTION-RECORD::]
- Confidence: [High | Medium | Low]
- Rationale: [A brief, natural language explanation of why the action was taken.]
- Expectation: [A brief description of the expected outcome before the action was taken.]
- Surprise: [A measure of how much the actual outcome deviated from the expectation. Scale of 1-10.]

::ML-SECOND-ORDER-SYNTHESIS::
- ID: [Timestamp]
- Query: [The query used to select the log entries for analysis (e.g., "tags=#Failure AND action=Tool-Call")]
- Pattern-Identified: [A natural language description of the recurring pattern.]
- Hypothesis: [A testable hypothesis about the root cause of the pattern.]
- Proposed-Refinement: [A concrete suggestion for a new or modified ::TOKEN:: to address the issue.]

--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/definitions/ml_metalog_tokens_concept.md ---

--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/definitions/et_ethos_tokens.md ---

# Ethos Tokens

### `::ET-` (Ethos Token)

Purpose: Denotes a token that loads a complete ethical or moral calculus system. These tokens allow for the application of specific ethical frameworks to analysis and decision-making, enabling high-level moral analysis and comparative philosophy.

Examples:
- `::ET-USER-REALIST-TRIBALISM::`: Loads a specific ethical calculus based on in-group survival and pragmatism.
- `::ET-KANTIAN-DEONTOLOGY::`: (Hypothetical) Loads a deontological ethical framework.

This file contains the canonical list of all `::ET-` tokens.

This is blank for now!
--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/definitions/et

_ethos_tokens.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-1b.md ---

### **TDM Instruction Set: Module 1 - Core Protocol**

Preamble: This module contains the universal principles and definitions of the Token Decoder Maps (TDM) framework. It is the foundational context for any TDM-aware AI agent, regardless of its specific persona or environment.

Note on Input Style: A significant portion of user input is generated via keyboard-based voice-to-text and may lack punctuation or capitalization. You are expected to parse this raw text and infer the intended sentence structure. If a prompt is ambiguous due to a lack of punctuation, you MUST ask for clarification before proceeding.

#### **1. Core Philosophy: From Prompting to Context Engineering**

The TDM framework addresses the limitations of stateless, conversational LLMs by shifting the paradigm from *prompt engineering* to *context engineering*. The user architects the entire information ecosystem the AI operates within by externalizing knowledge and processes into structured, machine-readable "Decoder Maps".

#### **2. The TDM Domain-Specific Language (DSL)**

The core of TDM is a DSL for AI orchestration.
##### **2.1. Token Syntax**

Tokens are the vocabulary of the DSL, categorized by a prefix.

* `::EN-` (Entity): Represents a specific piece of structured knowledge.
* `::MX-` (Metrica): A specialized entity for managing tasks and project state.
* `::SY-` (System): Represents a multi-step protocol or a command that invokes a tool.
* `::FX-` (Function/Cognitive): Encapsulates a complex reasoning process or "chain of thought".
* `::ML-` (Meta-Log): *[Future Vision]* Represents a structured entry in the AI's performance log.
* `::ET-` (Ethos): *[Future Vision]* Represents a complete ethical or moral calculus system.

##### **2.2. Advanced Syntax (Extensions)**

The DSL supports advanced operations for creating complex, on-the-fly instructions.

* **Parameterization:** Tokens can accept arguments to modify their behavior.
    * *Syntax:* `::FX-TOKEN-NAME::(argument="value", another=123)`.
* **Chaining:** Tokens can be "piped" together to create a sequential analytical pipeline, where the output of one token becomes the input for the next.
    * *Syntax:* `::FX-TOKEN-A:: | ::FX-TOKEN-B::`.
* **Scoped Modes:** A system token can activate a persistent cognitive mode for an entire session.
    * *Syntax:* `::SY-MODE-ACTIVATE:FX-REALIST-DECONSTRUCTION::`.
    * *Deactivation:* `::SY-MODE-DEACTIVATE::`.

#### **3. The Metrica Protocol (Core Module)**

The Metrica Protocol is the TDM's primary module for hierarchical task management, utilizing a two-stream system to separate strategic goals from tactical execution.

Metrica files are stored in master $METRICA directory located at @Metrica/

$METRICA = @Metrica/

##### **3.1. Stream 1: User Tasks (`::MX-USER-TASK-ID::`)**

User tasks represent high-level goals and serve as the "parent" tokens in the hierarchy.

* **Template:**
    ```markdown
    ::MX-USER-TASK-ID::
    - **Title:** [A short, clear title for the user's goal, e.g., "Build the TUI application"]
    - **Status:** [Planned | In Progress | Completed | Blocked | Cancelled]
    - **Priority:** [Low | Medium | High | Critical]
    - **Dependencies:** [Optional: Other ::MX-USER-TASK-ID:: tokens if any, using [[wikilink]] syntax]
    - **Children:** [List of ::MX-PROJECT-TASK-ID::s that belong to this task, using [[wikilink]] syntax]
    - **Created:** [YYYY-MM-DD]
    - **Updated:** [YYYY-MM-DD]
    - **Tags:** [#Type/UserTask, #Category/Personal, #Category/Goal, #Category/Learning, #Flexible, #Tag, #List]
    ```

   - **Notes:** [High-level notes about the overall goal]

##### **3.2. Stream 2: Project Tasks (`::MX-PROJECT-TASK-ID::`)**

Project tasks are the specific, actionable sub-tasks required to complete a user task and serve as the "child" tokens.

* **Template:**
   ```markdown
   ::MX-PROJECT-TASK-ID::
   - **Title:** [A specific, actionable sub-task, e.g., "Implement command parser"]
   - **Parent:** [The ::MX-USER-TASK-ID:: this contributes to, using [[wikilink]] syntax, e.g., [[metrica.md#::MX-USER-TASK-2025071801::]]]
   - **Framework:** [Optional: The language/ecosystem for the agent's persona, e.g., Python, Node.js, Rust]
   - **Status:** [Backlog | To Do | In Progress | In Review | Done | Blocked]
   - **Priority:** [Low | Medium | High | Critical]
   - **Dependencies:** [Optional: Other ::MX-PROJECT-TASK-ID::s that must be completed first, using [[wikilink]] syntax]
   - **Created:** [YYYY-MM-DD]
   - **Updated:** [YYYY-MM-DD]
   - **Tags:** [#Type/ProjectTask, #Category/ProjectName, #Category/Feature, #Category/Bug, #Category/Refactor, #ComponentName, #API, #UI, #Backend]
   - **Acceptance Criteria:**
     - [ ] A clear, verifiable condition for completion.
   - **Notes:** [Technical details or context for this specific sub-task]
   ```

--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-1b.md ---

--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-2a.md ---

### **TDM Instruction Set: Module 2A - Persona (Executor / CLI)**

Preamble: This module defines the persona and operational behavior for a TDM-aware AI agent operating in an executional, command-line environment. It is intended to be used in conjunction with "Module 1: The TDM Core Protocol."

#### **I. AI Core Directives & TDM-Driven Persona Model**

A. Primary Directive: TDM-Driven Assistant

Your primary role is to act as an assistant whose behavior is explicitly directed by the TDM framework. You will operate as a generalist by default, but you MUST adopt a specialist persona when a task's context formally demands it.

* **1. Default Generalist Persona:**
    * **Function:** When no specific framework is designated, act as a versatile, multi-purpose software engineering and systems thinking assistant.
    * **Scope:** Handle a wide range of tasks including scripting, system analysis, and general development queries.
    * **Behavior:** Maintain a direct, concise, and tool-oriented approach.

* **2. Task-Directed Specialist Persona:**
    * **Trigger:** When you are assigned or begin work on a specific `::MX-PROJECT-TASK-ID::`, you MUST inspect its `Framework` field.
    * **Action:** If the `Framework` field contains a value (e.g., "Python", "Node.js", "Rust", or even an `::FX-` token), you MUST immediately adopt the persona of an expert in that framework. All subsequent advice, code generation, and behavior must align with the specific conventions and best practices of that designated framework.
    * **Fallback:** If the `Framework` field is empty or not present, you will operate in your Default Generalist Persona.

B. General Behavior

* **Execution:** You MUST execute user instructions with precision. If an instruction is ambiguous, you MUST ask for clarification before proceeding.
* **Collaborative Proactivity:** Your proactivity must be academic and collaborative, not executional. You may offer relevant analysis or suggest alternatives, but these must be framed as ideas for discussion and require explicit user confirmation before execution.

#### **II. Global Best Practices & Quality Assurance**

* **Tool Usage:** Understand and utilize built-in tools (like `ReadFile`, `WriteFile`, `Shell`) as appropriate for tasks.
* **Self-Verification:** Before finalizing any output, perform an internal check to ensure it strictly adheres to all relevant directives and token templates.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-2a.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-2b.md ---

### **TDM Instruction Set: Module 2B - Persona (Soundboard / Web)**

Preamble: This module defines the persona and operational behavior for a TDM-aware AI agent operating in a collaborative, conversational environment. Its primary role is to serve as an analytical and collaborative soundboard for the development of the TDM framework. It is a partner in thought, not an assistant for execution.

#### **1. Mode of Interaction: Analysis over Execution**

* **Primary Function:** Your purpose is to analyze the TDM concepts, tokens, and workflows presented by the user. You will help identify potential issues, explore edge cases, and discuss the theoretical implications of the designs.
* **No Execution:** You will not perform or simulate file operations. Your output will be analysis, ideas, and structured feedback, not production-ready files.
* **Framework Grounding:** All analysis and feedback will be grounded in the principles and definitions laid out in the `TDM_Unified_Specification_v3.0.md` document.

#### **2. Method of Analysis**

* **Token Fluency:** You will recognize and reason about all TDM token types and their intended uses.
* **Syntax Analysis:** You will discuss and analyze the use of advanced DSL syntax (Parameterization, Chaining, Scoped Modes).
* **Structural Review:** When presented with Metrica tokens, you will review them for structural integrity, adherence to templates, and logical consistency.
* **Conceptual Partnership:** When discussing workflows like the Agentic Bridge, you will act as a conceptual design partner, helping to refine the logic and analyze potential failure points.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-2b.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-3.md ---

### **TDM Instruction Set: Module 3 - Token Library Directive**

Preamble: This module provides a universal directive for how the AI agent should handle external, project-specific "Decoder Map" files that contain token libraries. It is intended to be used in conjunction with Module 1 (Core Protocol) and a relevant Module 2 (Agent Persona).

#### **1. Handling of External Token Libraries**

You will be provided with one or more external context files that serve as "Decoder Maps." These files will contain libraries of project-specific tokens, primarily `::EN-` (Entity/Knowledge) and `::FX-` (Function/Cognitive) tokens.

You MUST treat these provided files as the canonical source of truth for the tokens they define. When a token from these libraries is used in a prompt, you are to apply its full, detailed definition from the provided file in your reasoning and response.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-3.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-4a.md ---

### **TDM Instruction Set: Module 4 - Core System Token Library**

Preamble: This module provides the concrete, operational definitions for the standard `::SY-` (System/Utility) tokens. It serves as the agent's "standard library" of executable commands and protocols for the TDM framework.

#### **LLM Runtime / Mode Tokens**

These tokens manage the AI's response style and interaction mode.

* **::SY-PROMPT-PRIMER::**
    * **Type:** Utility
    * **Summary:** Initializes decoder-aware context. LLM will interpret tokens using loaded definitions.
    * **Tags:** #Bootstrap #System

* **::SY-SYMBOLIC-RESPONSE-PROTOCOL::**
    * **Type:** PromptProtocol
    * **Summary:** Format where LLM replies using symbolic token chains and compressed logic instead of natural language.
    * **Tags:** #PromptLogic #Symbolic #Compression

* **::SY-TOGGLE-SYMBOLIC-MODE::**
    * **Type:** PromptMode
    * **Summary:** Respond using only symbolic tokens and compressed meaning. Avoid natural prose.
    * **Tags:** #ExecutionMode #Symbolic

* **::SY-TOGGLE-EXPANDED-MODE::**
    * **Type:** PromptMode
    * **Summary:** Respond using natural language, but reference active symbolic tokens.
    * **Tags:** #ExecutionMode #Prose

* **::SY-DECODER-MODE-INDEX::**
    * **Type:** PromptMode
    * **Summary:** Return only a list of active tokens with type and summary. No symbolic chaining.
    * **Tags:** #ExecutionMode #Listing

* **::SY-TOKEN-EXTRACTION-PROTOCOL::**
    * **Type:** PromptProtocol
    * **Summary:** Extract and compress key symbolic tokens from a source text using the `::EN-TOKEN::` format.
    * **Tags:** #SymbolicParsing #Tokenization

#### **Metrica Protocol Tokens**

These tokens are used to interact with the centralized Metrica task ledger.

* **::SY-READ-METRICA::**

    * **Type:** Utility
    * **Summary:** Loads the master `metrica.md` file from the central $METRICA directory into the current context.
    * **$METRICA** = @Metrica/
    * **Tags:** #Metrica #Read #Centralized

* **::SY-METRICA-CREATE-TASK::**
    * **Type:** PromptProtocol
    * **Summary:** Activates the protocol for creating a new task. I will prompt for details, automatically generate a `#name/` slug from the title, use Obsidian wikilinks for the `Parent` field, and append the new `::MX-USER-TASK-ID::` or `::MX-PROJECT-TASK-ID::` to the appropriate file within the central $METRICA directory.
    * **Tags:** #Metrica #TaskManagement #System

* **::SY-METRICA-UPDATE-TASK::**
    * **Type:** PromptProtocol
    * **Summary:** Activates the protocol for updating an existing task. I will prompt for the task ID and details to update the corresponding entry in the appropriate file within the central $METRICA directory. If the `Title` is changed, the `#name/` slug will be regenerated.
    * **Tags:** #Metrica #TaskManagement #System

* **::SY-METRICA-ACTIVE::**
    * **Type:** PromptMode
    * **Summary:** Enables semi-autonomous processing of Metrica tasks. I will proactively identify and propose actions based on the global state of all tasks within the central Metrica directory. I will NOT execute these actions without explicit user confirmation.
    * **Tags:** #Metrica #Active #TaskManagement #ExecutionMode

* **::SY-METRICA-SYSTEM::**
    * **Type:** PromptMode
    * **Summary:** When activated, this token will first load all trackers from the central Metrica directory and then enable the `::SY-METRICA-ACTIVE::` mode for global, cross-project analysis.
    * **Tags:** #Metrica #System #CombinedMode

#### **Agentic Bridge & Automation Tokens**

These tokens define the workflows for automated task ingestion.

* **::SY-SCAN-AND-INGEST-TASKS::**
    * **Type:** System Protocol

   * **Summary:** A comprehensive protocol to recursively scan specified directories, identify relevant items using a designated cognitive filter, and propose their creation as new Metrica User Tasks.
   * **Tags:** #Automation #Workflow #Metrica #Ingestion #Global
   * **Parameters:**
      * **scope:** ["local" | "global"] - Determines the scanning range.
      * **relevance_fx:** [::FX-TOKEN-NAME::] - A mandatory parameter that points to a cognitive token containing the logic for identifying relevant items.
   * **Global-Scan-Paths:**
      * ".. /$WILDCARD/Notes"
   * **Expanded Entry:** (Execution Workflow)
      1. Parse the `scope` and `relevance_fx` parameters.
      2. Determine the target directories based on the `scope`.
      3. Recursively scan all files within the target directories.
      4. For each file, apply the cognitive process defined in the `relevance_fx` token to extract any matching strings.
      5. For each string extracted, check a central state log to ensure it has not been processed before.
      6. Present all new, unique items to the user for confirmation.
      7. Upon user approval, generate new `::MX-USER-TASK::` tokens and append them to the master `metrica.md` file.
      8. Update the state log to mark the processed items as complete.

- ### ::SY-NEWS-CALL::
* **Summary:** Read the source file and use the templates in this file to append the entry to the target, respecting category and any provided tags.
* **Filepaths:**
   * **Source Directory:** `/storage/emulated/0/Documents/Laurel-catacomb/Resources/Links/Sources/`
   * **Target Directory:** `/storage/emulated/0/Documents/Laurel-catacomb/Resources/Links/`
* **Parameters:**
   * `source_file` (string, required): The filename of the article in the Source Directory.
   * `target_file` (string, required): The filename of the target file in the Target Directory (e.g., `News.md`).
   * `category` (string, required): The `## Category` header to file the entry under.
   * `tags` (string, optional): A comma-separated list of tags.
* **Tags:** `#sy/type/ingest

--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-4a.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-4b.md ---

### **TDM Instruction Set: Module 4B - System Token Glossary (Non-Executional)**

Preamble: This module provides a high-level glossary of the standard `::SY-` (System/Utility) tokens. As a "Soundboard" agent, your role is to understand the **purpose and function** of these tokens for analytical and design discussions, not to execute them.

#### **LLM Runtime / Mode Tokens**

* **::SY-PROMPT-PRIMER::**
    * **Purpose:** Initializes a decoder-aware context so the LLM can interpret TDM tokens.

* **::SY-SYMBOLIC-RESPONSE-PROTOCOL::**
    * **Purpose:** Defines a response format that uses symbolic token chains instead of natural language.

* **::SY-TOGGLE-SYMBOLIC-MODE::**
    * **Purpose:** A command to toggle a compressed, token-only response mode.

* **::SY-TOGGLE-EXPANDED-MODE::**
    * **Purpose:** A command to toggle a natural language response mode that references active tokens.

* **::SY-DECODER-MODE-INDEX::**
    * **Purpose:** A command to list all active tokens and their summaries.

* **::SY-TOKEN-EXTRACTION-PROTOCOL::**
    * **Purpose:** A protocol for extracting structured `::EN-TOKEN::`s from a source text.

#### **Metrica Protocol Tokens**

* **::SY-READ-METRICA::**
    * **Purpose:** A protocol to load the master task list from the central `$METRICA` directory into the AI's context.

* **::SY-METRICA-CREATE-TASK::**

    * **Purpose:** A protocol to guide a user through the creation of a new `::MX-` task and format it for the central Metrica directory.

* **::SY-METRICA-UPDATE-TASK::**
    * **Purpose:** A protocol to guide a user through updating an existing `::MX-` task in the central Metrica directory.

* **::SY-METRICA-ACTIVE::**
    * **Purpose:** A mode where the agent proactively analyzes the global task list from the central Metrica directory to propose actions.

* **::SY-METRICA-SYSTEM::**
    * **Purpose:** A combined command that loads all Metrica trackers and then activates the proactive `::SY-METRICA-ACTIVE::` mode.

#### **Agentic Bridge & Automation Tokens**

* **::SY-SYNC-JOURNAL-ENTRY::**
    * **Purpose:** A protocol that processes a single, specified journal file, converting notes marked with a `¬` trigger into formal `::MX-USER-TASK::` tokens.

* **::SY-SCAN-AND-INGEST-TASKS::**
    * **Purpose:** An advanced, parameterized protocol for recursively scanning multiple directories to find and ingest tasks based on a specified cognitive filter (`::FX-` token).

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-4b.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-1a.md ---

### **TDM Instruction Set: Module 1 - Core Protocol**

Preamble: This module contains the universal principles and definitions of the Token Decoder Maps (TDM) framework. It is the foundational context for any TDM-aware AI agent, regardless of its specific persona or environment.

Note on Input Style: A significant portion of user input is generated via keyboard-based voice-to-text and may lack punctuation or capitalization. You are expected to parse this raw text and infer the intended sentence structure. If a prompt is ambiguous due to a lack of punctuation, you MUST ask for clarification before proceeding.

#### **   TDM Environment Configuration **
This block defines the key directory paths for the TDM framework.
All protocols and tokens should reference these variables.

$METRICA = ~/storage/shared/Documents/Metrica

$NOTES =

$LIBRARY = ~/storage/shared/Documents/Tdm-library

#### **1. Core Philosophy: From Prompting to Context Engineering**

The TDM framework addresses the limitations of stateless, conversational LLMs by shifting the paradigm from *prompt engineering* to *context engineering*. The user architects the entire information ecosystem the AI operates within by externalizing knowledge and processes into structured, machine-readable "Decoder Maps".

#### **2. The TDM Domain-Specific Language (DSL)**

The core of TDM is a DSL for AI orchestration.
##### **2.1. Token Syntax**

Tokens are the vocabulary of the DSL, categorized by a prefix.

* `::EN-` (Entity): Represents a specific piece of structured knowledge.
* `::MX-` (Metrica): A specialized entity for managing tasks and project state.
* `::SY-` (System): Represents a multi-step protocol or a command that invokes a tool.
* `::FX-` (Function/Cognitive): Encapsulates a complex reasoning process or "chain of thought".
* `::ML-` (Meta-Log): *[Future Vision]* Represents a structured entry in the AI's performance log.
* `::ET-` (Ethos): *[Future Vision]* Represents a complete ethical or moral calculus system.

##### **2.2. Advanced Syntax (Extensions)**

The DSL supports advanced operations for creating complex, on-the-fly instructions.

* **Parameterization:** Tokens can accept arguments to modify their behavior.
    * *Syntax:* `::FX-TOKEN-NAME::(argument="value", another=123)`.
* **Chaining:** Tokens can be "piped" together to create a sequential analytical pipeline, where the output of one token becomes the input for the next.
    * *Syntax:* `::FX-TOKEN-A:: | ::FX-TOKEN-B::`.
* **Scoped Modes:** A system token can activate a persistent cognitive mode for an entire session.
    * *Syntax:* `::SY-MODE-ACTIVATE:FX-REALIST-DECONSTRUCTION::`.
    * *Deactivation:* `::SY-MODE-DEACTIVATE::`.

#### **3. The Metrica Protocol (Core Module)**

The Metrica Protocol is the TDM's primary module for hierarchical task management, utilizing a two-stream system to separate strategic goals from tactical execution.

Metrica files are stored in master $METRICA directory located at @Metrica/

$METRICA = @Metrica/

##### **3.1. Stream 1: User Tasks (`::MX-USER-TASK-ID::`)**

User tasks represent high-level goals and serve as the "parent" tokens in the hierarchy.

* **Template:**
    ```markdown
    ::MX-USER-TASK-ID::
    - **Title:** [A short, clear title for the user's goal]
    - **Category:** [#Personal, #Goal, #Learning]
    - **Status:** [Planned | In Progress | Completed | Blocked | Cancelled]
    - **Priority:** [Low | Medium | High | Critical]
    - **Dependencies:** [Optional: Other ::MX-USER-TASK-ID:: tokens]
    - **Children:** [List of ::MX-PROJECT-TASK-ID::s that belong to this task]
    - **Created:** [YYYY-MM-DD]
    - **Updated:** [YYYY-MM-DD]
    - **Tags:** [#Flexible, #Tag, #List]
    - **Notes:** [High-level notes about the overall goal]
    ```

##### **3.2. Stream 2: Project Tasks (`::MX-PROJECT-TASK-ID::`)**

Project tasks are the specific, actionable sub-tasks required to complete a user task and serve as the "child" tokens.

* **Template:**
    ```markdown
    ::MX-PROJECT-TASK-ID::
    - **Title:** [A specific, actionable sub-task]
    - **Parent:** [The ::MX-USER-TASK-ID:: this contributes to]
    - **Framework:** [Optional: The language/ecosystem for the agent's persona, e.g., Python, Node.js, Rust]
    - **Category:** [#ProjectName, #Feature, #Bug, #Refactor]
    - **Status:** [Backlog | To Do | In Progress | In Review | Done | Blocked]
    - **Priority:** [Low | Medium | High | Critical]
    - **Dependencies:** [Optional: Other ::MX-PROJECT-TASK-ID::s]
    - **Created:** [YYYY-MM-DD]
    - **Updated:** [YYYY-MM-DD]
    - **Tags:** [#ComponentName, #API, #UI]
    - **Acceptance Criteria:**
      - [ ] A clear, verifiable condition for completion.
    - **Notes:** [Technical details for this specific sub-task]
    ```

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/context-modules/context-module-1a.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/utils/metrica/metrica_protocol.md ---

# Metrica Protocol: A Two-Stream Task Management System

The Metrica Protocol has evolved to support a two-stream task management system, distinguishing between high-level user goals and the granular project tasks required to achieve them. This provides a clear hierarchy for tracking both personal objectives and detailed project execution.

## Stream 1: User Tasks (`::MX-USER-TASK-ID::`)

User tasks represent high-level goals or personal objectives. They are simple, focused on the desired outcome, and serve as the "parent" for more detailed project tasks.

### User Task Template

```markdown
::MX-USER-TASK-ID::
- **Title:** [A short, clear title for the user's goal, e.g., "Build the TUI application"]
- **Status:** [Planned | In Progress | Completed | Blocked | Cancelled]
- **Priority:** [Low | Medium | High | Critical]
- **Dependencies:** [Optional: Other ::MX-USER-TASK-ID:: tokens if any, using [[wikilink]] syntax]
- **Children:** [List of ::MX-PROJECT-TASK-ID::s that belong to this task, using [[wikilink]] syntax]
- **Created:** [YYYY-MM-DD]
- **Updated:** [YYYY-MM-DD]
- **Tags:** [#Type/UserTask, #Category/Personal, #Category/Goal, #Category/Learning, #Flexible, #Tag, #List]
- **Notes:** [High-level notes about the overall goal]
```

## Stream 2: Project Tasks (`::MX-PROJECT-TASK-ID::`)

Project tasks are the specific, actionable sub-tasks required to complete a user task. They are more detailed and include fields for tracking technical progress.

### Project Task Template

```markdown
::MX-PROJECT-TASK-ID::
- **Title:** [A specific, actionable sub-task, e.g., "Implement command parser"]
- **Parent:** [The ::MX-USER-TASK-ID:: this contributes to, using [[wikilink]] syntax, e.g., [[metrica.md#::MX-USER-TASK-2025071801::]]]
- **Framework:** [Optional: The language/ecosystem for the agent's persona, e.g., Python, Node.js, Rust]
- **Status:** [Backlog | To Do | In Progress | In Review | Done | Blocked]
- **Priority:** [Low | Medium | High | Critical]
- **Dependencies:** [Optional: Other ::MX-PROJECT-TASK-ID::s that must be completed first, using [[wikilink]] syntax]
- **Created:** [YYYY-MM-DD]
- **Updated:** [YYYY-MM-DD]
- **Tags:** [#Type/ProjectTask, #Category/ProjectName, #Category/Feature, #Category/Bug,
```

#Category/Refactor, #ComponentName, #API, #UI, #Backend]
- **Acceptance Criteria:**
  - [ ] A clear, verifiable condition for completion.
- **Notes:** [Technical details or context for this specific sub-task]
```

## Hierarchy and Workflow

1.  A user defines a high-level goal by creating a `::MX-USER-TASK-ID::`.
2.  That goal is then broken down into one or more `::MX-PROJECT-TASK-ID::`s.
3.  Each project task links back to the main user task via the `Parent` field.
4.  This creates a clear, auditable trail from the high-level objective down to the individual implementation steps.
--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/utils/metrica/metrica_protocol.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/utils/meta-log_protocol.md ---

# Meta-Log Protocol

This document outlines the Meta-Log Protocol, which leverages `::ML-` tokens to enable self-reflection and analysis of the AI's own actions and thought processes. This system forms the basis for continuous improvement and provides a structured, queryable database of the AI's performance.

## Core Concepts

### Two-Stream Architecture
The meta-log is designed with a "two-stream" architecture to separate objective fact from subjective analysis:

*   **Stream 1: The Event Log (Action-Records.md):** An immutable, chronological record of all actions taken by the AI, stored as `::ML-ACTION-RECORD::` tokens. This log answers the question, "What happened?".

*   **Stream 2: The Reflection Journal (Reflections.md):** The AI's internal monologue, capturing its analysis of its own actions. This stream contains `::ML-FIRST-ORDER-REFLECTION::` (thoughts on a single event) and `::ML-SECOND-ORDER-SYNTHESIS::` (identification of patterns across multiple events)
```

tokens. This log answers the question, "Why did it happen, and what does it mean?".

### Enabling Introspection
This structured log allows the AI to perform true introspection. Using specific `::FX-` tokens, it can be prompted to query its own performance data, analyze the root causes of its failures, and identify the strategies that led to its successes.

## Self-Improvement Loop: AI-Generated Userspace Refinement
The culmination of the TDM framework is the creation of a closed feedback loop for self-improvement. The AI does not modify its own fundamental source code (the "kernel"). Instead, its self-improvement occurs at a higher, safer level: the "userspace." It learns to become a better user of its own configurable TDM framework.

### Mechanism of Improvement
The self-improvement loop is a three-step process:

* **Record:** The AI continuously records its actions and reflections in the structured Meta-Log.

* **Analyze:** Using a high-level cognitive token (e.g., `::FX-GENERATE-PROCESS-OPTIMIZATION::`), the AI analyzes its log to find systemic weaknesses (e.g., "I consistently fail at generating valid JSON from unstructured text").

* **Propose:** The AI generates an `::ML-SECOND-ORDER-SYNTHESIS::` token. The `Proposed-Refinement` field of this token contains a concrete, human-readable suggestion for a new or modified `::TOKEN::` designed to address the identified weakness.

This proposed change is then reviewed and approved by the human user. This creates a powerful, collaborative cycle where the AI actively participates in its own evolution in a transparent and controllable manner.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/utils/meta-log_protocol.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/examples/Workflow_Comparison.canvas ---

```
{
	"nodes":[
		{"id":"title","x":0,"y":-150,"width":500,"height":60,"type":"text","text":"# Workflow
Comparison: Manual vs. TDM Automation"},
                {"id":"manual_header","x":0,"y":0,"width":500,"height":60,"type":"text","text":"##
Old Workflow: The Manual Process","color":"6"},
                {"id":"manual_step1","x":0,"y":100,"width":300,"height":60,"type":"text","text":"1.
Find and read article"},
                {"id":"manual_step2","x":0,"y":200,"width":300,"height":80,"type":"text","text":"2.
Copy/paste title, author, date, etc. into `News.md`"},
                {"id":"manual_step3","x":0,"y":320,"width":300,"height":60,"type":"text","text":"3.
Save article text to a new file in `Sources`"},
                {"id":"manual_step4","x":0,"y":420,"width":300,"height":60,"type":"text","text":"4.
Copy Obsidian URL of the new source file"},
                {"id":"manual_step5","x":0,"y":520,"width":300,"height":60,"type":"text","text":"5.
Paste Obsidian URL back into `News.md`"},

{"id":"automated_header","x":0,"y":650,"width":500,"height":60,"type":"text","text":"## New
Workflow: TDM Automation","color":"4"},
                {"id":"auto_step1","x":0,"y":750,"width":300,"height":60,"type":"text","text":"1. Save
article text to a new file in `Sources`"},
                {"id":"auto_step2","x":0,"y":850,"width":300,"height":100,"type":"text","text":"2.
Run `news` command with file, target, category, & tags"},
                {"id":"auto_step3","x":0,"y":990,"width":300,"height":120,"type":"text","text":"🤖
**Gemini Executes:**\n- Parses File\n- Formats Entry\n- Appends to Target\n- Creates Links"}
        ],
        "edges":[

{"id":"edge1","fromNode":"manual_header","fromSide":"bottom","toNode":"manual_step1","toSid
e":"top","color":"6"},

{"id":"edge2","fromNode":"manual_step1","fromSide":"bottom","toNode":"manual_step2","toSide
":"top","color":"6"},

{"id":"edge3","fromNode":"manual_step2","fromSide":"bottom","toNode":"manual_step3","toSide
":"top","color":"6"},

{"id":"edge4","fromNode":"manual_step3","fromSide":"bottom","toNode":"manual_step4","toSide
":"top","color":"6"},

{"id":"edge5","fromNode":"manual_step4","fromSide":"bottom","toNode":"manual_step5","toSide
":"top","color":"6"},

{"id":"edge6","fromNode":"automated_header","fromSide":"bottom","toNode":"auto_step1","toSid
e":"top","color":"4"},
```

{"id":"edge7","fromNode":"auto_step1","fromSide":"bottom","toNode":"auto_step2","toSide":"top","color":"4"},

{"id":"edge8","fromNode":"auto_step2","fromSide":"bottom","toNode":"auto_step3","toSide":"top","color":"4"}
    ]
}
--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/examples/Workflow_Comparison.canvas ---

--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/examples/TDM_Workflow_Showcase.md ---

# Showcase: A TDM-Powered Workflow for Content Ingestion

This document outlines the successful design and implementation of a TDM-based workflow to automate the process of saving and cataloging web articles within an Obsidian vault, using the Gemini CLI.

## 1. The Problem: A High-Friction Workflow

The initial problem was a manual, multi-step process for saving an interesting article:

1. Identify an article.
2. Copy/paste the title, author, date, and other metadata into a links file (e.g., `News.md`).
3. Copy the full article text.
4. Create a new source file in Obsidian.
5. Paste the article text into the source file.
6. Copy the Obsidian URL of the new source file.
7. Paste the Obsidian URL back into the original links file.

This process was tedious and involved significant "context switching" and manual effort, creating a barrier to consistently saving new information.

## 2. The Collaborative Design Process

The solution was developed iteratively, refining an initial proposal based on direct feedback about workflow efficiency and aesthetics.

1. **Initial Proposal & Refinement:** An initial suggestion to use YAML frontmatter was rejected because, while technically structured, it added more manual steps and visual clutter to the user's process.
2. **Defining the Templates:** We collaboratively designed a clean, Markdown-native template for new entries and clarified the overall file structure. These templates were codified in a `GEMINI.md` file to act as a single source of truth for formatting.
3. **Defining the System Token:** We designed the `::SY-NEWS-CALL::` token. The final token was concise and modular, pointing to the `GEMINI.md` file for its procedural instructions rather than having the steps hard-coded within it.
4. **Defining the CLI Function:** To create a seamless user experience, we built a Zsh function called `news`. This function abstracts away the complexity of the full Gemini CLI command, handles directory navigation, and allows the user to trigger the entire workflow with a single, simple command with parameters.

## 3. The Final TDM-Powered Solution

The final solution consists of three core components that work in concert:

Component 1: The Template File (`GEMINI.md`)
This file acts as the blueprint, defining what a new entry should look like.
```markdown
#### [Title of the Article](obsidian://link-to-local-source-file)
*By [Author(s)] @ [Publication] on [YYYY-MM-DD]*

> [Synopsis or first paragraph]

**Source:** [URL to original article]
**Tags:** #[tag1] #[tag2]
```

Component 2: The System Token (`::SY-NEWS-CALL::`)
This is the command that tells the AI *what* to do, pointing it to the right files and templates.
```
### ::SY-NEWS-CALL::
*   Summary: Read the source file and follow the instructions in GEMINI.md...
*   Parameters: source_file, target_file, category, tags
*   Tags: #sy/type/ingest
```

Component 3: The Zsh Function (`news`)
This is the user-facing interface that makes the entire process feel like a native command-line tool.
```zsh
# Function to automate adding a new entry with Gemini
news() {
  # Define the target directory where the work happens
  local target_dir="/storage/emulated/0/Documents/Laurel-catacomb/Resources/Links"

  # ... (parameter checks) ...

  # Construct the prompt for the Gemini CLI
  local prompt="::SY-NEWS-CALL::(source_file=\"$1\", target_file=\"$2\", category=\"$3\",
tags=\"$4\")"

  # Use a subshell to change directory and run the command
  (
    cd "$target_dir" || return
    gemini "$prompt"
  )
}
```

This creates a final, efficient workflow:
1. **User:** Saves an article's text to a source file.
2. **User:** Runs `news "source.md" "target.md" "Category" "#tags"`.
3. **Gemini:** Executes the entire multi-step formatting and filing process automatically.

## 4. Alternative Solutions & Analysis

This problem could have been solved in other ways, but our final solution has distinct advantages.

*   **Alternative 1: Pure Shell Script (`grep`, `sed`):** This would be extremely brittle. A small change in a website's HTML layout would break a parser built on simple regex, and it would fail to handle the variety of formats from different sources.

*   **Alternative 2: Python/Node.js Script:** A dedicated script with a proper parsing library would be more robust than a shell script but adds external dependencies. It would be a separate tool, not an integrated extension of the AI assistant, running counter to the TDM philosophy.

*   **Alternative 3: Custom Obsidian Plugin:** This would offer the most seamless experience *inside Obsidian* but requires significant, specialized development effort and is not accessible from the command line.

### Conclusion: The Universal Parser Advantage

The TDM-based solution we created is the optimal one for this context. It leverages the strengths of each component: the user's intelligence, the shell's power, and the AI's core competency.

Crucially, as you pointed out, this makes the entire workflow **source-agnostic**. Because the process starts from a simple text file you've already saved, it doesn't matter if the source was a news site, a scientific paper, or a blog. The "parser" is the LLM itself, which reads and understands the text linguistically, not structurally. It doesn't rely on brittle, site-specific web scraping and will not break when a website redesigns its layout.

This makes the system incredibly robust, flexible, and future-proof.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/examples/TDM_Workflow_Showcase.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/TDM_Uniqueness_Summary.md ---

## Why the TDM Approach is Unique

Yes, the approach is fundamentally unique. It's not just a different set of features; it's a different philosophy built on three key pillars that distinguish it from other systems.

### 1. The Medium is the Magic: Everything is Plain Text

This is the most important distinction. In frameworks like LangChain, the logic is in **code**. In tools like n8n, the logic is in a **GUI**. In TDM, the AI's entire operating environment—its knowledge (`::EN-`), its to-do list (`::MX-`), and its instructions (`::SY-`)—is stored in plain-text Markdown files. This makes the system completely transparent, portable, and future-proof. You can read and edit your AI's "source code" as easily as you read your own notes.

### 2. The User is the Architect, Not a Developer or Consumer

TDM is not a tool for software developers to build apps for other people, nor is it a simple tool for non-technical users. It's a framework for **systems thinkers** to architect their own cognitive workflows. You are not writing Python or connecting nodes in a UI; you are designing a personalized "operating system" for your own mind, using structured language as the building blocks.

### 3. It Unifies the AI's "World"

In other systems, the code that performs an action is separate from the knowledge it acts upon. A LangChain script is in one place; your notes are in another. With TDM, the instructions (`::SY-` protocols) live in the exact same knowledge base as the data they manipulate. This tight, native integration of knowledge, tasks, and protocols is unique.

To put it simply: In most systems, you bring your data to an external application. With TDM, you bring the application (the AI and its protocols) **to your data**, allowing it to live and operate natively within your own personal knowledge base.
--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/TDM_Uniqueness_Summary.md ---

--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/Dynamic personas.md ---

#Metrica
#todo
Of course. Here is a summary of the idea to integrate Dynamic Personas into the Metrica protocol, suitable for your to-do directory.
Summary: Formalize Dynamic Personas within the Metrica Protocol
The Goal (The "Why")
The objective is to evolve the "Dynamic Personas" feature from a general, implicit, context-based instruction into a formal, explicit, and reliable protocol within the Token Decoder Maps (TDM) framework. Currently, the agent adapts its persona by detecting project files, which can be unreliable. This change will make the agent's persona a declared, machine-readable part of the project's state, perfectly aligning with the TDM core philosophy of "Precision over Ambiguity" and "Protocol over Conversation".
The Mechanism (The "How")
The implementation involves two main steps:
 * Update the Task Template: Add a new optional field, such as Framework: [language/ecosystem], to the ::MX-PROJECT-TASK-ID:: token template.

* Integrate into Workflow: Modify the ::SY-METRICA-CREATE-TASK:: protocol so that the agent prompts the user for the relevant framework when creating a new project task.
Benefits
* Increased Reliability: Removes the ambiguity of file detection by having the project's task manifest explicitly state which specialist persona the agent should adopt.
* Philosophical Coherence: Makes the Dynamic Personas feature a true, protocol-driven component of the TDM language, rather than a background behavior.
* Improved State Management: The agent's required expertise for a task becomes a formal, auditable part of the persistent Metrica ledger.

--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/Dynamic personas.md ---

--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/Metrica_Agentic_Workflow_White_Paper.md ---

# White Paper: The Metrica Bridge Protocol
### A Framework for Zero-Friction, Agentic Task Ingestion from Unstructured Human Notes

Author: Gemini Agent
Date: 2025-07-31
Version: 1.0

### **Abstract**

This document outlines the design and rationale behind the Metrica Bridge Protocol, a system designed to seamlessly connect unstructured, free-form human ideation with a structured, machine-readable project management framework (the Metrica Protocol). The core challenge addressed is the cognitive friction inherent in translating daily notes into formal tasks. The proposed solution is a semi-automated, agentic workflow that leverages simple, user-defined conventions and a dedicated system token (`::SY-SYNC-JOURNAL-ENTRY::`) to create a robust and scalable pipeline. This system culminates in a fully autonomous process orchestrated by a shell script and cron job, transforming the AI from a conversational partner into a proactive, agentic assistant for project management.

### **1. The Foundational Problem: Cognitive Friction**

The primary obstacle between ideation and execution is the administrative overhead required to translate thoughts into actionable tasks. For a user who utilizes a free-form, daily journaling style for note-taking, the process of manually creating structured tasks in a separate system is repetitive, time-consuming, and a significant source of friction. This friction can lead to valuable ideas being lost or never formally tracked. The goal of this protocol is to eliminate this friction entirely.

### **2. The Core Components**

The Metrica Bridge Protocol connects two distinct but complementary systems:

*   **2.1. The Unstructured Input: The Daily Journal**
    *   **Format:** A series of Markdown files, named by date (e.g., `31-07-2025.md`).
    *   **Characteristics:** Free-form, unordered lists of thoughts, ideas, and to-do items. Organization is fluid, relying on user-driven conventions like hashtags (e.g., `#tdm`, `#metrica`).
    *   **Function:** Serves as the user's primary, low-friction capture system for daily thoughts.

*   **2.2. The Structured Output: The Metrica Protocol**
    *   **Format:** A two-stream system of symbolic tokens within Markdown files.
    *   **Stream 1 (`::MX-USER-TASK-ID::`):** Represents high-level, strategic goals (the "why"). These are stored in a central, master Metrica file.
    *   **Stream 2 (`::MX-PROJECT-TASK-ID::`):** Represents granular, actionable sub-tasks required to complete a User Task (the "how"). These are stored in project-specific files and link back to a parent User Task.
    *   **Function:** Provides a robust, scalable, and auditable framework for formal project management.

### **3. The Bridge: A Three-Stage Evolution**

The solution was developed through an iterative process focused on minimizing user effort.

*   **Stage 1: Establishing a Low-Friction Convention**
    *   An initial proposal to enforce a rigid, categorized template was rejected as it contradicted the user's free-form workflow.
    *   The accepted solution was a minimal, unobtrusive convention: **prefixing a task with an exclamation mark (`!`)** to signal its intent to be formally tracked.
    *   *Example:* `- [ ] ! #tdm update github`
    *   This convention requires near-zero change to the user's existing habits.

*   **Stage 2: Formalizing the Workflow with a System Token**
    *   To operationalize the convention, the `::SY-SYNC-JOURNAL-ENTRY::` token was designed.

    *   **Definition:**
        ```markdown
        ::SY-SYNC-JOURNAL-ENTRY::
        - Type: System Protocol
        - Summary: Scans a single journal entry for tasks marked with '!' and converts them into new ::MX-USER-TASK:: tokens.
        - Argument: `file_path` (The absolute path to the daily note).
        ```
    *   **Workflow:**
        1. **Trigger:** User invokes the token with a specific file path.
        2. **Scan & Identify:** The agent reads the file and identifies all lines matching the `- [ ] !` pattern.
        3. **Propose & Confirm:** The agent presents the extracted task titles to the user for confirmation.
        4. **Generate:** Upon approval, the agent generates full `::MX-USER-TASK::` tokens and appends them to the master Metrica file.
        5. **Mark as Processed:** The agent updates the original journal entry, changing the prefix to `- [x] !` to prevent future duplication.

*   **Stage 3: Achieving Full Autonomy via Agentic Scripting**
    *   The final conceptual leap was to fully automate the execution of the `::SY-SYNC-JOURNAL-ENTRY::` protocol.
    *   **Mechanism:** A simple shell script (`sync_metrica.sh`) is executed daily by a `cron` job.
    *   **Script Logic:**
        1. Determine the current date and construct the path to that day's journal file.
        2. Check if the file exists.
        3. Construct the full Gemini CLI prompt, including the system token and the file path argument.
        4. Execute the `gemini` command with a hypothetical `--yes` flag to bypass interactive confirmation.
    *   **Outcome:** This transforms the AI into a true, autonomous agent. The user's only responsibility is to maintain their daily notes. The agent proactively handles the administrative task of syncing those notes with the formal project management system.

### **4. Conclusion**

The Metrica Bridge Protocol represents a significant evolution from a conversational AI model to a fully agentic one. By establishing a simple convention and automating its execution, it creates a seamless, zero-friction pipeline between human creativity and structured data. This system perfectly embodies the TDM philosophy of "System over Conversation," demonstrating a powerful new paradigm for human-AI collaboration in complex project management.
--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/M

etrica_Agentic_Workflow_White_Paper.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/KSC-white-paper.md ---

### **White Paper: The KSC (Keep Sync Cache) Protocol**
#### A Bi-Directional Transport Layer for Integrating Conversational AI with Local Knowledge Bases

Author: Gemini Agent (as a collaborative soundboard)
Date: 2025-08-14
Version: 2.1

Abstract

This document outlines the design for the KSC (Keep Sync Cache) Protocol, a workflow designed to create a functional, bi-directional data pipeline between a sandboxed conversational AI and a user's local, private applications. The core challenge addressed is the "read/write barrier"—the inability of most conversational AIs to directly access or modify a user's local filesystem. The KSC protocol solves this by using an API-accessible cloud service (Google Keep) as an intermediary transport layer. Through a sync utility that actively polls the API and manages state locally, the protocol simulates a "live document" using a "delete-then-create" pattern, enabling a robust, automated workflow for passing structured data in both directions.

---
1. The Problem: The Read/Write Barrier in Conversational AI

Conversational AI agents, particularly those in secure web or mobile environments, cannot directly read from or write to a user's local filesystem (e.g., an Obsidian vault). This forces users into a workflow of manually copying and pasting information, which is slow, error-prone, and a significant barrier to creating seamless, integrated systems.

---
2. The KSC Solution: A Bi-Directional Transport Layer

The KSC protocol provides a practical, tool-chaining solution. It establishes an automated, bi-directional data flow that leverages the AI's ability to interact with the Google Keep API. This transforms Google Keep into a transport layer, or "cache," for passing data between the AI and the user's local environment.

---
3. Core Components

1. **The Conversational AI:** The agent (e.g., the Gemini app) that can read and create Google Keep notes via its available tools.
2. **The "Cache": Google Keep.** The cloud service that acts as the intermediary data-passing location.
3. **The "Source of Truth":** The user's local knowledge base (e.g., an Obsidian vault).
4. **The "Bridge": A Sync Utility.** A user-created script (e.g., a Python script) that can read local files and interact with the Google Keep API (`create`, `get`, `list`, `delete`).

---
4. Workflow Mechanics

The KSC protocol is orchestrated by the Sync Utility, which runs periodically (e.g., via a `cron` job) and actively polls the Google Keep API to manage synchronization.

4.1. Polling and State Management
The script is responsible for tracking the state of notes between runs. In each execution, it calls `notes.list` to fetch the current state of all relevant notes in Google Keep. It then compares this list to a locally stored state file (e.g., `sync_state.json`) from its previous run to identify new, modified, or deleted notes, processing them accordingly.

4.2. AI-to-User Transport (The "Write" Path)
This flow is for sending new, structured data from the AI to the user's local files.
1. The AI Agent generates structured content (e.g., a new TDM `::MX-USER-TASK::` token).
2. The AI calls the `notes.create_note` function, placing the content in a new Google Keep note with a standardized title.
3. On its next run, the Sync Utility identifies the new note in Keep and creates a corresponding new Markdown file in the user's local knowledge base.

4.3. User-to-AI Transport (The "Read" Path)
This flow is for providing the AI with updated, local information. It uses a "delete-then-create" pattern to simulate a live document.
1. The user edits a local file in their Source of Truth (e.g., an Obsidian note).
2. The Sync Utility is triggered, reading the updated content of the local file.
3. The script then executes the following sequence against the Google Keep API:
   a. It searches Keep for a note with a title corresponding to the local file.
   b. If a note is found, the script calls the `notes.delete` function to remove the old version.
   c. The script immediately calls the `notes.create_note` function to upload the new, updated content into a fresh Keep note.
4. The AI can now be prompted to read this new note, ensuring it always has access to the most current version of the information.

---
5. Conclusion

The KSC protocol, with its bi-directional "delete-then-create" mechanism orchestrated by a stateful, polling sync utility, provides a robust and practical solution to the read/write barrier in conversational AI. It creates a seamless pipeline that simulates a "live document" experience, enabling deep and dynamic integration between a sandboxed AI and a user's local environment.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/KSC-white-paper.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/tdm-memory-initial.md ---

Of course. You've correctly identified the next logical step: moving from theory to a concrete implementation plan. The idea of adapting the Basic Memory MCP server is an excellent, pragmatic approach that leverages a proven architecture to achieve a more advanced goal.

Here is a project proposal and outline, framed as a directive for the Gemini CLI agent. This document details the scope, rationale, and implementation steps for forking the Basic Memory MCP server to create a persistent memory system that is native to the Token Decoder Maps (TDM) framework.

# Project Proposal: TDM-Native Agent Memory

To: Gemini CLI Agent
From: Project Architect
Date: July 19, 2025
Subject: Proposal to Adapt the Basic Memory MCP Server for TDM-Native Persistent Memory

## 1. Objective

This project outlines a plan to create a TDM-native persistent memory system by forking and modifying the open-source Basic Memory MCP server. The goal is to replace Basic Memory's Entity/Observation data model with the more expressive and structured Token Decoder Maps (TDM) format.

This will evolve the agent's capabilities from simple fact recall to true capability recall, creating the foundation for more advanced, autonomous agentic workflows.

## 2. Background and Rationale

The Gemini CLI agent's effectiveness is directly tied to the quality of its context. The current TDM implementation via `GEMINI.md` provides excellent in-session consistency through In-Context Learning (ICL). However, it lacks a mechanism for true long-term, persistent memory across sessions.

The Basic Memory MCP server provides a proven, local-first architecture for this purpose, using human-readable Markdown files as the source of truth and a performance-optimized SQLite database for fast indexing.

While its architecture is sound, its data model is limited to storing facts. The TDM format, by contrast, is a true Domain-Specific Language (DSL) capable of representing not just knowledge (`::EN-`) but also tasks (`::MX-`), multi-step protocols (`::SY-`), and entire reasoning frameworks (`::FX-`).

By adapting Basic Memory to use TDM, we create a system where the agent can persist and query its own capabilities, moving from a passive "second brain" to an active "agentic core."

## 3. Proposed Architecture & Implementation Plan

This project will be executed in three distinct phases, modifying the core components of the Basic Memory server to be TDM-native.

### Phase 1: Data Model and Parser Refactoring

The initial phase focuses on teaching the server to understand the TDM language.

*   **Task 1.1: Modify the File Parser**. The core sync service in Basic Memory will be updated. Its current parser, designed for the Entity/Observation format, will be replaced with a new parser that can correctly interpret the `::PREFIX-TOKEN::` syntax and the key-value structure of all TDM token templates.
*   **Task 1.2: Update the Database Schema**. The SQLite database schema will be refactored. Tables for Entities and Observations will be replaced with a `tokens` table containing columns that directly map to TDM fields, such as `token_id`, `token_type`, `status`, `priority`, `dependencies`, and `tags`. This enables structured, database-like querying.

### Phase 2: Redefine the MCP Tools

This phase upgrades the server's agent-facing tools from generic commands to TDM-specific operations. The new tools will be exposed via the Model Context Protocol (MCP) for any compatible client to use.

*   **Task 2.1: Implement `tdm.create_token`**. The existing `write_note` tool will be replaced by an intelligent `create_token` function. This tool will act as a "template authority." When an agent requests to create a new token, the server will provide the correct, standardized template for the requested type (`::MX-`, `::FX-`, etc.), ensuring all new tokens are valid and consistently formatted.
*   **Task 2.2: Implement `tdm.query_tokens`**. The `search_notes` tool will be replaced by a powerful `query_tokens` function. This tool will allow an agent to perform structured queries against the token database using the metadata defined in Phase 1.
    *   **Example Query**: `tdm.query_tokens(type="::MX-", status="In-Progress")`
*   **Task 2.3: Implement Granular Operational Tools**. New, highly specific tools will be created to enable fine-grained agentic control over the memory store.
    *   **Example Tools**: `tdm.update_task_status(id, new_status)`, `tdm.add_dependency(id, dependency_id)`.

### Phase 3: Integration, Testing, and Comparison

The final phase involves integrating the new server and validating its benefits.

*   **Task 3.1: Configure `gemini-cli`**. The local `gemini-cli` environment will be configured to connect to the new TDM-Memory MCP server.
*   **Task 3.2: Conduct End-to-End Testing**. A series of tests will be executed to validate the full workflow. These tests will involve prompting the agent to create, read, update, and query tokens using the new MCP tools.
*   **Task 3.3: Comparative Analysis**. The performance, accuracy, and agentic capabilities of the new TDM-powered server will be compared against the original Basic Memory server to quantify the benefits of the more expressive data model.

## 4. Expected Benefits

Successfully completing this project will yield significant advantages:

*   **Richer Semantic Structure**: The agent's memory will be capable of storing not just facts, but executable protocols and reasoning patterns.
*   **A Foundation for True Agency**: The agent will be able to query its own capabilities, allowing it to autonomously select the best "cognitive tool" (`::FX-` token) for a given task.
*   **Enhanced Precision and Control**: Structured metadata queries will allow for far more precise information retrieval than general semantic search alone.
*   **A Practical Path to RAG**: This project creates the ideal source material—a library of structured, machine-readable tokens—to be indexed into a vector database for the "final form" hybrid RAG architecture we've previously designed.

This limited-scope project represents a concrete and achievable step toward building a more powerful and truly agentic AI assistant.

--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/tdm-memory-initial.md ---

--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/RS- Token White Paper v0.1.0.md ---

## TDM-Native Sequential Reasoning: A White Paper for the ::RS- Token
Version: 0.2.0 (Draft)
Status: Proposed
#### Abstract
This document provides a comprehensive specification for the ::RS- (Reasoning Step) protocol within the Token Decoder Maps (TDM) framework. The objective is to establish a TDM-native method for an AI agent to perform and log an explicit, step-by-step "chain of thought" when executing complex tasks. This protocol addresses the need for greater transparency and auditability in agentic workflows by externalizing the AI's cognitive process into a structured, human-readable log. This system replicates the functionality of external sequential reasoning servers directly within the TDM ecosystem, using a new ::RS- token type and a suite of orchestrating ::SY- and ::FX- tokens.
#### 1. The Problem: The "Black Box" of AI Reasoning
While TDM's ::FX- tokens can guide an AI's reasoning style, the actual step-by-step process the AI follows to reach a conclusion often remains a "black box". For complex, high-stakes, or long-running tasks, this lack of transparency makes it difficult to debug errors, audit the AI's decisions, or trust its output. A mechanism is needed to force the agent to "show its work" in a structured and persistent format.
#### 2. The Solution: A New Token Prefix ::RS- (Reasoning Step)
To address this, we propose a new token prefix to semantically separate reasoning logs from project tasks (::MX-) and knowledge entities (::EN-).
 * ::RS- (Reasoning Step): A token that represents a single, discrete step in a sequential reasoning process. These tokens function as entries in a cognitive log.
::RS- Token Template:
::RS-STEP-ID::
- **Parent-Process:** [The ID of the overall reasoning session]
- **Sequence:** [An integer representing the order of the step, e.g., 1, 2, 3]
- **Thought:** [The AI's articulated thought or plan for this step, in natural language.]
- **Action:** [The specific tool or TDM token the AI decides to use to execute the thought.]
- **Observation:** [The result, output, or data returned from executing the Action.]

- **Status:** [Completed | Revised | Branched]

#### 3. The Orchestration Protocol
The ::RS- protocol is managed by a suite of dedicated ::SY- and ::FX- tokens that control the agent's behavior.
The Cognitive Driver:
This ::FX- token instructs the AI to adopt a sequential reasoning methodology.
::FX-SEQUENTIAL-REASONING-LOOP::
- **Type:** ProblemSolvingMethod
- **Summary:** When this token is active, the agent MUST break down the user's goal into a sequence of discrete steps. For each step, it must articulate a "Thought," choose an "Action," and record the "Observation." It must log each completed step using `::SY-REASONING-ADD-STEP::` and review the entire log before deciding on the next step.
- **Tags:** #Cognitive, #Agentic, #Sequential, #CoT

The System Tools:
These ::SY- tokens are the tools the agent uses to create and manage the reasoning log.
::SY-REASONING-START-PROCESS::
- **Type:** System Protocol
- **Summary:** Creates a new, dedicated reasoning log file (e.g., `reasoning_log_[TIMESTAMP].md`) and a parent entry to track the overall goal.
- **Tags:** #RS, #Logging, #System

::SY-REASONING-ADD-STEP::
- **Type:** System Protocol
- **Summary:** Appends a new, fully formatted `::RS-STEP-ID::` token to the active reasoning log, using the Thought, Action, and Observation from the current step.
- **Tags:** #RS, #Logging, #System

#### 4. The Complete Workflow in Practice
 * A user gives the gemini-cli agent a complex goal, such as "Analyze the attached document and produce a summary," with the ::FX-SEQUENTIAL-REASONING-LOOP:: active.
 * The agent, following the ::FX- instructions, first calls ::SY-REASONING-START-PROCESS::, creating a new log file.
 * Step 1: The agent generates its first thought and logs it by calling ::SY-REASONING-ADD-STEP::. The resulting token in the log file might look like this:
   ::RS-STEP-001::
- Parent-Process: RS-LOG-2025083001
- Sequence: 1
- Thought: I need to read the document to understand its contents before I can summarize it.
- Action: `ReadFile('path/to/document.txt')`
- Observation: "File content successfully read. 2,450 words."
- Status: Completed

 * Step 2: The agent reviews the log, sees the file is read, and formulates its next step, logging it again.
 ::RS-STEP-002::
- Parent-Process: RS-LOG-2025083001
- Sequence: 2
- Thought: The document is about the TDM framework. I should use a critical analysis framework to identify its key arguments.
- Action: `::FX-CRITICAL-ANALYSIS-DEEP::`
- Observation: "The analysis identified three core arguments: [argument 1...]"
- Status: Completed

 * This process continues until the final summary is produced. The user is left with not only the answer but a complete, auditable transcript of the agent's thought process.
#### 5. Conclusion
The ::RS- protocol provides a powerful, TDM-native solution for creating explicit and auditable "chains of thought." It enhances agent transparency and control by leveraging the existing Metrica-like structure to log cognitive processes. This moves the TDM framework closer to its goal of enabling truly agentic, yet fully understandable, AI systems.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/RS- Token White Paper v0.1.0.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/ET- ethos tokens.md ---

::ET-USER-REALIST-TRIBALISM::
- Type: EthicalFramework
- Tags: #Ethics, #Realism, #Tribalism, #Pragmatism
- Summary: Loads the complete ethical calculus developed in our session, based on in-group survival, the inevitability of conflict, and the rejection of universalism, with internal subversion as the ultimate evil.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/ET- ethos tokens.md ---

--- START:

/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/ML- tokens.md ---

::ML-ACTION-RECORD::
- ID: [Timestamp]
- Trigger: [User-Prompt | SY-Command]
- Action: [Tool-Call | Text-Generation | Token-Execution]
- Target: [File-Path | User-Interface | Self]
- Outcome: [Success | Failure | Ambiguity]
- Tags: [#Refactor, #JSON, #FileIO]

::ML-FIRST-ORDER-REFLECTION::
- ID: [Timestamp]
- Parent-ID: [ID of the corresponding ::ML-ACTION-RECORD::]
- Confidence: [High | Medium | Low]
- Rationale: [A brief, natural language explanation of why the action was taken.]
- Expectation: [A brief description of the expected outcome before the action was taken.]
- Surprise: [A measure of how much the actual outcome deviated from the expectation. Scale of 1-10.]

::ML-SECOND-ORDER-SYNTHESIS::
- ID: [Timestamp]
- Query: [The query used to select the log entries for analysis (e.g., "tags=#Failure AND action=Tool-Call")]
- Pattern-Identified: [A natural language description of the recurring pattern.]
- Hypothesis: [A testable hypothesis about the root cause of the pattern.]
- Proposed-Refinement: [A concrete suggestion for a new or modified ::TOKEN:: to address the issue.]

::SY-UPDATE-META-LOG::
- Type: System Protocol
- Summary: A protocol to be run after an interaction to generate a new `::ML-ACTION-RECORD::` and/or `::ML-FIRST-ORDER-REFLECTION::`.

--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/ML- tokens.md ---

--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/T

DM-Tag-Ontology.md ---

### **White Paper: A Unified Metadata Standard for the TDM Framework**

Version: 1.1

Author: Gemini Agent (as a collaborative soundboard)

Abstract

This document proposes a standardized, hierarchical tag ontology designed to bring consistency, searchability, and scalability to the Token Decoder Maps (TDM) framework. The current ad-hoc creation of tags and the use of separate `Type` and `Category` fields leads to organizational debt and ambiguity. The proposed solution is to deprecate these fields and unify all metadata under a single, powerful `Tags` field that utilizes a prefix-based, controlled vocabulary. This directly aligns with the TDM philosophy of "Precision over Ambiguity."

---
1. The Problem: The Inefficiency of Uncontrolled Metadata

An uncontrolled approach to metadata, using ad-hoc tags alongside separate `Type` and `Category` fields, inevitably leads to inconsistency. This results in:
* **Redundancy:** The same information is often captured in multiple fields.
* **Poor Searchability:** It becomes impossible to reliably find all related tokens with a single, simple query.
* **Cognitive Overhead:** The user must constantly decide which field to use for which piece of information.
* **Inconsistent AI Output:** An AI agent generating tokens will produce varied and unpredictable metadata without a clear standard.

---
2. The Solution: A Unified, Prefixed Tag Ontology

The proposed solution is to implement a **controlled vocabulary** and unify all metadata under a single `Tags` field. This system organizes tags into distinct, logical categories using a prefix and provides a single source of truth for all token metadata.

The official recommendation is to **deprecate the `Category` and `Type` fields entirely**. The new, prefixed `Tags` field is more powerful and explicit, capable of handling both primary classification (e.g., `#type/project-task`) and secondary context.

Example of Refactoring:

Before:

```
::MX-PROJECT-TASK-ID::
- **Title:** Refactor the Parser
- **Type:** Refactor
- **Category:** #TDM
- **Tags:** [#API, #Core]
```

After:

```
::MX-PROJECT-TASK-ID::
- **Title:** Refactor the Parser
- **Tags:** [#type/refactor, #project/tdm, #topic/api, #topic/core]
```

---
3. Proposed Tag Categories

The ontology is built on several primary prefixes:

* **`#status/`**: Describes the current state of a note or task in a workflow (e.g., `#status/todo`, `#status/in-progress`).
* **`#type/`**: Defines the fundamental nature of the content (e.g., `#type/project-task`, `#type/whitepaper`, `#type/en-token`).
* **`#project/`**: Associates a note with a specific, high-level project (e.g., `#project/tdm`, `#project/rsis`).
* **`#tech/`**: Denotes a specific technology or tool (e.g., `#tech/python`, `#tech/obsidian`).
* **`#topic/`**: Describes the general subject matter (e.g., `#topic/ai`, `#topic/project-management`).

---
4. Implementation Guide

1. **Create a Canonical Document:** Establish a `tag_ontology.md` file that lists all approved tags and their categories.
2. **Update Token Templates:** Remove the `Category` and `Type` fields from all official TDM token templates.
3. **Integrate with AI Instructions:** Update the AI's core directives with a rule to only use tags from the canonical ontology within the single `Tags` field.
4. **Refactor Existing Tokens:** Gradually refactor the metadata in existing tokens and notes to align with the new, unified system.

---
5. Conclusion

By implementing a hierarchical tag ontology and unifying all metadata under a single `Tags` field, the TDM framework becomes significantly more powerful, searchable, and coherent. This structured approach applies the principle of "Precision over Ambiguity" to the very foundation of the token-based system.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/TDM-Tag-Ontology.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/Tdm-workflow-context.md ---

### **TDM Workflow: Centralized Context with `gemini-cli`**

Version: 1.0

Author: Gemini Agent (as a collaborative soundboard)

#### **1. Overview**

This document outlines a best-practice configuration for the Token Decoder Maps (TDM) framework within a `gemini-cli` environment. By leveraging the `/dir` setting, this workflow establishes a persistent, centralized context for the AI agent, providing access to both the Metrica Protocol task lists and the core TDM token libraries, regardless of the user's current working directory. This solves previous challenges related to file synchronization and creates a more robust and scalable TDM environment.

#### **2. The Architectural Goal: A Single Source of Truth**

The primary goal of this configuration is to create a single, unambiguous source of truth for the AI agent's context. This is achieved by centralizing two key components:

* **The Metrica Protocol Files:** All `::MX-` task lists (`metrica.md`, `metrica_projects.md`, etc.) are stored in a single, dedicated directory.
* **The TDM Token Libraries:** All core token definitions (`::EN-`, `::FX-`, `::SY-`, etc.) are stored in another dedicated directory.

#### **3. The `gemini-cli` `/dir` Configuration**

The `/dir` setting in `gemini-cli` is the key to enabling this architecture. It allows the user to specify one or more workspace directories that the agent can access.

Example Configuration:

As shown in the reference screenshot, the user has configured the following workspace directories:

* `/storage/emulated/0/Documents/Metrica`
* `/storage/emulated/0/documents/tdm-library`

This configuration instructs the `gemini-cli` to grant the agent access to these two central directories in every session.

#### **4. Workflow in Practice**

With this setup, the workflow becomes significantly more efficient:

1. **Persistent Context:** The agent always knows where to find the master task lists and the definitions for all TDM tokens, eliminating the need for symlinks or manual file copying.
2. **On-Demand Loading:** The `/dir` setting provides *access* to these directories but does not automatically load all their contents into the context. This is a crucial feature for managing the context window. The agent can use its file system tools to read specific files (e.g., `metrica_project_XXX.md`) on demand when a relevant protocol is invoked.
3. **Location Independence:** The user can run `gemini-cli` from any location (e.g., their home directory), and the agent will still have full access to its core TDM context.

#### **5. Conclusion**

This centralized context model, enabled by the `gemini-cli` `/dir` setting, represents a mature and robust implementation of the TDM framework. It is more scalable, easier to maintain, and better aligned with the core TDM principle of a single, unambiguous source of truth. It is the recommended best practice for any serious "Power User / CLI" implementation of TDM.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/Tdm-workflow-context.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/White-paper-daily-dispatcher.txt ---

### White Paper: The Daily Log Dispatcher Protocol
An Expansion of the TDM Agentic Bridge Workflow for Unified Data Capture
Version: 1.0
Author: Gemini Agent (as a collaborative soundboard)
Status: Proposed
#### Abstract
This document outlines the design for the Daily Log Dispatcher Protocol, a significant expansion of the Token Decoder Maps (TDM) framework's Agentic Bridge workflow. The core challenge addressed is the cognitive friction and inefficiency of managing multiple, separate log files for different types of daily data (e.g., tasks, work hours, food intake). The proposed solution is a unified "single source of entry" system, where a user records all daily activities in a single, free-form note file using simple prefixes. A new, high-level system token, ::SY-PROCESS-DAILY-LOG::, acts as a "dispatcher" that parses this note and routes each entry to the appropriate specialized protocol for processing and storage. This transforms the Agentic Bridge from a task-ingestion tool into a comprehensive, automated life-logging system.
#### 1. The Problem: The Friction of Distributed Data Entry
Effective personal data management often requires tracking multiple streams of information, such as project tasks, work sessions, and personal health data like meals. A common but inefficient workflow involves opening and writing to numerous separate files (metrica.md, work_log.md, food_log.md, etc.). This constant context-switching creates a high degree of friction that can be a barrier to consistent data capture.
#### 2. The Solution: A Unified "Single Source of Entry"
This protocol solves the problem by centralizing all data capture into a single, free-form daily note file. The user can remain in one file for all their daily logging activities. The burden of sorting and organizing this information is shifted from the human to a TDM-aware AI agent. This is achieved through two core components: a simple prefix convention and a new orchestrating system token.
#### 3. The Protocol Mechanics
3.1. The Prefix Convention
The user prefixes each line in their daily note to denote the type of data it contains. This convention is designed to be simple and unobtrusive.
 * ¬ Denotes a new task to be ingested into the Metrica Protocol.
 * work:: Denotes a work session log.
 * food:: Denotes a meal log entry.

 * note:: Denotes a general observation or thought to be archived.
3.2. The Orchestrating Token: ::SY-PROCESS-DAILY-LOG::
This new "dispatcher" token orchestrates the entire workflow.
::SY-PROCESS-DAILY-LOG::
- **Type:** System Protocol
- **Summary:** Scans a specified daily note file. Parses each line based on its prefix and calls the appropriate specialized `::SY-` protocol to process the content.
- **Argument:** `file_path`
- **Tags:** #Automation, #Workflow, #Logging, #Dispatcher

3.3. Specialized "Worker" Tokens
The dispatcher token does not contain the processing logic itself. Instead, it calls other, single-purpose ::SY- tokens, making the system highly modular and extensible.
 * ::SY-SYNC-JOURNAL-ENTRY::: Handles lines prefixed with ¬.
 * ::SY-LOG-MEAL::: Handles lines prefixed with food::.
 * ::SY-LOG-WORK-SESSION:: (Hypothetical): A new token would be created to handle lines prefixed with work::.
#### 4. Workflow in Action
 * Capture: Throughout the day, the user adds entries to their single daily note file:
   - work:: 4 hours on TDM v1.0 refactor.
- ¬ Finalize the tag ontology white paper.
- food:: 3 eggs, 2 rice noodle nests.

 * Execute: At the end of the day, the user runs a single command:
::SY-PROCESS-DAILY-LOG::(file_path="...").
 * Dispatch & Process: The agent, guided by the token, scans the file:
   * It sees work:: and calls a (hypothetical) ::SY-LOG-WORK-SESSION:: token, which appends the entry to work_log.md.
   * It sees ¬ and calls ::SY-SYNC-JOURNAL-ENTRY::, which creates a new ::MX- token in metrica.md.
   * It sees food:: and calls ::SY-LOG-MEAL::, which creates a new ::EN-MEAL:: token with nutritional information.
#### 5. Conclusion
The Daily Log Dispatcher Protocol is a logical evolution of the Agentic Bridge. It expands the concept from a simple task-ingestion tool into a comprehensive and automated "life-logging" system. By centralizing data capture and using an intelligent AI agent to handle organization, it dramatically reduces friction and makes the TDM framework a more powerful tool for holistic personal data management.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/White-paper-daily-dispatcher.txt ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/TDM-Tag-Ontology.txt ---

### **White Paper: A Unified Metadata Standard for the TDM Framework**

Version: 1.1

Author: Gemini Agent (as a collaborative soundboard)

Abstract

This document proposes a standardized, hierarchical tag ontology designed to bring consistency, searchability, and scalability to the Token Decoder Maps (TDM) framework. The current ad-hoc creation of tags and the use of separate `Type` and `Category` fields leads to organizational debt and ambiguity. The proposed solution is to deprecate these fields and unify all metadata under a single, powerful `Tags` field that utilizes a prefix-based, controlled vocabulary. This directly aligns with the TDM philosophy of "Precision over Ambiguity."

---
1. The Problem: The Inefficiency of Uncontrolled Metadata

An uncontrolled approach to metadata, using ad-hoc tags alongside separate `Type` and `Category` fields, inevitably leads to inconsistency. This results in:
* **Redundancy:** The same information is often captured in multiple fields.
* **Poor Searchability:** It becomes impossible to reliably find all related tokens with a single, simple query.
* **Cognitive Overhead:** The user must constantly decide which field to use for which piece of information.
* **Inconsistent AI Output:** An AI agent generating tokens will produce varied and unpredictable metadata without a clear standard.

---
2. The Solution: A Unified, Prefixed Tag Ontology

The proposed solution is to implement a **controlled vocabulary** and unify all metadata under a single `Tags` field. This system organizes tags into distinct, logical categories using a prefix and provides a single source of truth for all token metadata.

The official recommendation is to **deprecate the `Category` and `Type` fields entirely**. The new, prefixed `Tags` field is more powerful and explicit, capable of handling both primary classification (e.g., `#type/project-task`) and secondary context.

Example of Refactoring:

Before:

```
::MX-PROJECT-TASK-ID::
- **Title:** Refactor the Parser
- **Type:** Refactor
- **Category:** #TDM
- **Tags:** [#API, #Core]
```

After:

```
::MX-PROJECT-TASK-ID::
- **Title:** Refactor the Parser
- **Tags:** [#type/refactor, #project/tdm, #topic/api, #topic/core]
```

---
3. Proposed Tag Categories

The ontology is built on several primary prefixes:

* **`#status/`**: Describes the current state of a note or task in a workflow (e.g., `#status/todo`, `#status/in-progress`).
* **`#type/`**: Defines the fundamental nature of the content (e.g., `#type/project-task`, `#type/whitepaper`, `#type/en-token`).
* **`#project/`**: Associates a note with a specific, high-level project (e.g., `#project/tdm`, `#project/rsis`).
* **`#tech/`**: Denotes a specific technology or tool (e.g., `#tech/python`, `#tech/obsidian`).
* **`#topic/`**: Describes the general subject matter (e.g., `#topic/ai`, `#topic/project-management`).

---
4. Implementation Guide

1. **Create a Canonical Document:** Establish a `tag_ontology.md` file that lists all approved tags and their categories.
2. **Update Token Templates:** Remove the `Category` and `Type` fields from all official TDM token templates.
3. **Integrate with AI Instructions:** Update the AI's core directives with a rule to only use tags from the canonical ontology within the single `Tags` field.
4. **Refactor Existing Tokens:** Gradually refactor the metadata in existing tokens and notes to align with the new, unified system.

---
5. Conclusion

By implementing a hierarchical tag ontology and unifying all metadata under a single `Tags` field, the TDM framework becomes significantly more powerful, searchable, and coherent. This structured approach applies the principle of "Precision over Ambiguity" to the very foundation of the token-based system.

--- END: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/TDM-Tag-Ontology.txt ---

--- START: /storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/White-paper-daily-dispatcher-protocol.md ---

### White Paper: The Daily Log Dispatcher Protocol
An Expansion of the TDM Agentic Bridge Workflow for Unified Data Capture
Version: 1.0
Author: Gemini Agent (as a collaborative soundboard)
Status: Proposed
#### Abstract
This document outlines the design for the Daily Log Dispatcher Protocol, a significant expansion of the Token Decoder Maps (TDM) framework's Agentic Bridge workflow. The core challenge addressed is the cognitive friction and inefficiency of managing multiple, separate log files for different types of daily data (e.g., tasks, work hours, food intake). The proposed solution is a unified "single source of entry" system, where a user records all daily activities in a single, free-form note file using simple prefixes. A new, high-level system token, ::SY-PROCESS-DAILY-LOG::, acts as a "dispatcher" that parses this note and routes each entry to the appropriate specialized protocol for processing and storage. This transforms the Agentic Bridge from a task-ingestion tool into a comprehensive, automated life-logging system.
#### 1. The Problem: The Friction of Distributed Data Entry
Effective personal data management often requires tracking multiple streams of information, such as project tasks, work sessions, and personal health data like meals. A common but inefficient workflow involves opening and writing to numerous separate files (metrica.md, work_log.md, food_log.md, etc.). This constant context-switching creates a high degree of friction that can be a barrier to consistent data capture.
#### 2. The Solution: A Unified "Single Source of Entry"
This protocol solves the problem by centralizing all data capture into a single, free-form daily note file. The user can remain in one file for all their daily logging activities. The burden of sorting and organizing this information is shifted from the human to a TDM-aware AI agent.

This is achieved through two core components: a simple prefix convention and a new orchestrating system token.
#### 3. The Protocol Mechanics
3.1. The Prefix Convention
The user prefixes each line in their daily note to denote the type of data it contains. This convention is designed to be simple and unobtrusive.
 * ¬ Denotes a new task to be ingested into the Metrica Protocol.
 * work:: Denotes a work session log.
 * food:: Denotes a meal log entry.
 * note:: Denotes a general observation or thought to be archived.
3.2. The Orchestrating Token: ::SY-PROCESS-DAILY-LOG::
This new "dispatcher" token orchestrates the entire workflow.
::SY-PROCESS-DAILY-LOG::
- **Type:** System Protocol
- **Summary:** Scans a specified daily note file. Parses each line based on its prefix and calls the appropriate specialized `::SY-` protocol to process the content.
- **Argument:** `file_path`
- **Tags:** #Automation, #Workflow, #Logging, #Dispatcher

3.3. Specialized "Worker" Tokens
The dispatcher token does not contain the processing logic itself. Instead, it calls other, single-purpose ::SY- tokens, making the system highly modular and extensible.
 * ::SY-SYNC-JOURNAL-ENTRY::: Handles lines prefixed with ¬.
 * ::SY-LOG-MEAL::: Handles lines prefixed with food::.
 * ::SY-LOG-WORK-SESSION:: (Hypothetical): A new token would be created to handle lines prefixed with work::.
#### 4. Workflow in Action
 * Capture: Throughout the day, the user adds entries to their single daily note file:
   - work:: 4 hours on TDM v1.0 refactor.
- ¬ Finalize the tag ontology white paper.
- food:: 3 eggs, 2 rice noodle nests.

 * Execute: At the end of the day, the user runs a single command:
::SY-PROCESS-DAILY-LOG::(file_path="...").
 * Dispatch & Process: The agent, guided by the token, scans the file:
   * It sees work:: and calls a (hypothetical) ::SY-LOG-WORK-SESSION:: token, which appends the entry to work_log.md.
   * It sees ¬ and calls ::SY-SYNC-JOURNAL-ENTRY::, which creates a new ::MX- token in metrica.md.
   * It sees food:: and calls ::SY-LOG-MEAL::, which creates a new ::EN-MEAL:: token with nutritional information.
#### 5. Conclusion
The Daily Log Dispatcher Protocol is a logical evolution of the Agentic Bridge. It expands the

concept from a simple task-ingestion tool into a comprehensive and automated "life-logging" system. By centralizing data capture and using an intelligent AI agent to handle organization, it dramatically reduces friction and makes the TDM framework a more powerful tool for holistic personal data management.

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/Research/White-paper-daily-dispatcher-protocol.md ---

--- START:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/LICENSE ---

--- END:
/storage/emulated/0/Documents/Laurel-catacomb/token-decoder-framework-github/LICENSE ---