Example of a Metrica Decoder Map

::MX-TASK-ID::
- **Title:** [Short, clear title of the task]
- **Category:** [Optional grouping label, e.g., #ProjectName, #Admin, #Research]
- **Status:** [Planned | In Progress | Completed | Blocked | Cancelled]
- **Priority:** [Low | Medium | High | Critical]
- **Children:** [Related ::MX-PROJECT-TASK-ID::]
- **Dependencies:** [Other ::MX-TASK-ID:: tokens if any]
- **Deadline:** [Optional: YYYY-MM-DD format]
- **Created:** [Auto-assigned or manual date YYYY-MM-DD]
- **Tags:** [#Flexible #Tag #List]
- **Notes:** [Freeform details or sub-steps]
- **Related Files:

::MX-USER-001::
- **Title:** White Paper: Symbolic Token Decoder Map System
- **Category:** #SystemDesign
- **Status:** Completed
- **Priority:** High
- **Dependencies:** None
- **Children:**
- **Deadline:**
- **Created:** 2025-07-02
- **Updated:** 2025-07-19
- **Tags:** #LLM, #ContextWindow, #PersistentMemory, #MCP
- **Notes:** The foundational white paper describing the system to enhance LLM interaction by externalizing knowledge into decoder maps referenced by symbolic tokens. The document itself is complete.

::MX-USER-002::
- **Title:** Implement Minimalist Model Context Protocol (MCP) Server
- **Category:** #Implementation
- **Status:** Planned
- **Priority:** Critical
- **Dependencies:** ::MX-USER-001::
- **Children:**
- **Deadline:**
- **Created:** 2025-07-02
- **Updated:** 2025-07-19
- **Tags:** #MCP, #Server, #LLM, #ProofOfConcept
- **Notes:** Develop a lightweight intermediary server to test the Symbolic Token Decoder Map system. Core functions include prompt interception, token identification, decoder map lookup, and context injection for the LLM.

::MX-USER-003::
- **Title:** Research Synergy of Symbolic Tokens and OCTAVE Protocol
- **Category:** #Research
- **Status:** Planned
- **Priority:** Medium
- **Dependencies:** ::MX-USER-001::, ::MX-USER-002::
- **Children:**
- **Deadline:**
- **Created:** 2025-07-02
- **Updated:** 2025-07-19
- **Tags:** #OCTAVE, #SemanticCompression, #SystemDesign
- **Notes:** Explore the benefits of combining the Symbolic Token system with the OCTAVE protocol. The idea is to use OCTAVE-formatted content within the Decoder Maps to make the injected context more structured and semantically precise.

::MX-USER-004::
- **Title:** Implement Interactive Logging in zsh-ai
- **Category:** #Feature
- **Status:** Planned
- **Priority:** High
- **Dependencies:** None
- **Children:**
- **Deadline:**
- **Created:** 2025-07-02
- **Updated:** 2025-07-19
- **Tags:** #zsh-ai, #Logging, #UpstreamContribution
- **Notes:** Develop and integrate a configurable logging feature into the `zsh-ai` project. The plan includes adding config flags, creating a core logging function, integrating it at all interaction points, and adding documentation and tests for a potential upstream pull request.

::MX-USER-005::
- **Title:** Align `gemini-cli` Gemma 3 Plan with Multi-Provider Implementation
- **Category:** #ProjectManagement
- **Status:** Planned
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Deadline:**
- **Created:** 2025-07-02
- **Updated:** 2025-07-19
- **Tags:** #gemini-cli, #Gemma3, #Refactor
- **Notes:** Revise the development plan in `gemini-cli/GEMINI.md` to acknowledge and leverage the existing `TextToolCallParser` from the `multi-provider` branch. Future work should

focus on verification and refinement rather than building from scratch.

::MX-USER-006::
- **Title:** Token Decoder GitHub
- **Category:** #Implementation #Documentation
- **Status:** In Progress
- **Priority:** High
- **Dependencies:** ::MX-USER-001::, ::MX-USER-TDM-001::
- **Children:**
- **Deadline:**
- **Created:** 2025-07-04
- **Updated:** 2025-07-19
- **Tags:** #TokenDecoder, #GitHub, #Review, #Testing
- **Notes:** Repository pushed to GitHub. Testing and examples are still pending. Working directory: [[/data/data/com.termux/files/home/projects/token-decoder-framework-github]]. Tool calls via tokens is a success. Testing is good but uncertain.

::MX-USER-007::
- **Title:** Art Tokens
- **Category:** #Research #Implementation
- **Status:** Planned
- **Priority:** Medium
- **Dependencies:** ::MX-USER-001::, ::MX-USER-006::
- **Children:**
- **Deadline:**
- **Created:** 2025-07-04
- **Updated:** 2025-07-19
- **Tags:** #Art, #Tokens, #DecoderMap
- **Notes:** Investigate and prototype the concept of "art tokens" within the Symbolic Token Decoder Map system. This includes defining their structure, potential use cases, and integration with existing token types.

::MX-USER-008::
- **Title:** Tui_app
- **Category:** #Implementation
- **Status:** In Progress
- **Priority:** High
- **Dependencies:** ::MX-USER-009::
- **Children:**
- **Deadline:**
- **Created:** 2025-07-04
- **Updated:** 2025-07-19
- **Tags:** #TUI, #SQLAI, #Timestamps
- **Notes:** Missing Tui wrapper. Need to verrify that timestamps are backwards compatible with

the Forum update. Integrate with the SQL AI. Database schema has been unified, with `db_inserter.py` as the source of truth.

::MX-USER-009::
- **Title:** SQL-AI (Replicate zsh-ai for SQLite)
- **Category:** #Implementation
- **Status:** In Progress
- **Priority:** High
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-04
- **Updated:** 2025-07-19
- **Tags:** #SQL, #AI, #SQLite, #zsh-ai
- **Notes:** Replicate the functionality of zsh-ai for SQLite databases, allowing AI-powered interaction and query generation for SQL operations. Plan outlined in sql-ai/GEMINI.md

::MX-USER-010::
- **Title:** Implement Gemma compatibility in zsh-ai
- **Category:** #zsh-ai
- **Status:** Completed
- **Priority:** High
- **Dependencies:**
- **Children:**
- **Created:** 2025-07-05
- **Updated:** 2025-07-19
- **Tags:** #Development #Gemma #zsh-ai
- **Notes:** Need to refer to zsh-ai-gemlite for solutions

::MX-USER-011::
- **Title:** Fork Logseq for Obsidian Features
- **Category:** #Obsidian #logseq
- **Status:** Planned
- **Priority:** Low
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-19
- **Updated:** 2025-07-19
- **Tags:** #Development
- **Notes:** Examine the source code of Logseq and compile a feature list from Obsidian.

::MX-USER-TDM-001::
- **Title:** Token Decoder Framework Project
- **Category:** #Project
- **Status:** In Progress

- **Priority:** High
- **Dependencies:** None
- **Children:** ::MX-PROJECT-001::, ::MX-PROJECT-008::, ::MX-PROJECT-016::, ::MX-PROJECT-017::, ::MX-PROJECT-018::, ::MX-PROJECT-019::, ::MX-PROJECT-020::, ::MX-PROJECT-021::, ::MX-PROJECT-022::, ::MX-PROJECT-023::, ::MX-PROJECT-024::, ::MX-PROJECT-025::, ::MX-PROJECT-026::, ::MX-PROJECT-027::, ::MX-PROJECT-028::, ::MX-PROJECT-029::, ::MX-PROJECT-030::, ::MX-PROJECT-033::, ::MX-PROJECT-037::, ::MX-PROJECT-038::, ::MX-PROJECT-039::, ::MX-PROJECT-040::, ::MX-PROJECT-041::, ::MX-PROJECT-042::, ::MX-PROJECT-045::, ::MX-PROJECT-046::, ::MX-PROJECT-047::
- **Created:** 2025-07-31
- **Updated:** 2025-07-31
- **Tags:** #TDM, #Framework
- **Notes:** Overarching task for all development and documentation related to the Token Decoder Framework.
  Roadmap:
  1. Refine Metrica integration and test ML tokens with Meta-Log system.
  2. Test loading FX token ledgers/glossaries into context for in-session use.
  3. Document Metrica workflows and provide examples.
  4. Document ML protocol and provide examples.
  5. Document FX token usage for Chain-of-Thought/agentic behavior in regular sessions and project work, with examples.
  6. Re-examine `README.md` for a comprehensive overview and create a new streamlined `README.md`.
  7. Upload examples.
  8. Orchestrate Canvas-like illustrations of workflows using tokens.
  9. Document ChatGPT/general free subscription AI usage with TDM.
  10. Document ChatGPT projects and Gemini Gems usage with TDM.
  11. Refine global `GEMINI.md` to be more TDM-centric.

# Metrica Project Tasks: Token Decoder Framework

This file contains the granular project tasks (`::MX-PROJECT-TASK-ID::`) for the "Token Decoder Framework" project, linked to their parent user tasks.

## 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:: or ::MX-PROJECT-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 that must be completed first]
- **Created:** [YYYY-MM-DD]
- **Updated:** [YYYY-MM-DD]
- **Tags:** [#ComponentName, #API, #UI, #Backend]
- **Acceptance Criteria:**
  - [ ] A clear, verifiable condition for completion.
- **Notes:** [Technical details or context for this specific sub-task]
```

---

## Project Tasks

::MX-PROJECT-001::
- **Title:** Update TDM GitHub
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-CONSISTENCY-ENFORCEMENT::
- **Category:** #TDM, #GitHub
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:** ::MX-PROJECT-002::, ::MX-PROJECT-003::, ::MX-PROJECT-004::, ::MX-PROJECT-006::, ::MX-PROJECT-007::
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #TDM, #GitHub
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-002::
- **Title:** Update README
- **Parent:** ::MX-PROJECT-001::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #Documentation
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Documentation
```

- **Acceptance Criteria:**
  - [ ]
- **Notes:** update 1 complete 2025-07-31.

::MX-PROJECT-003::
- **Title:** Add SY- examples
- **Parent:** ::MX-PROJECT-001::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #Documentation, #SY-Tokens
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Documentation, #SY-Tokens
- **Acceptance Criteria:**
  - [Populate examples/ with usecase examples]
- **Notes:** Formatting of `sy_system_utility_tokens.md` completed.

::MX-PROJECT-004::
- **Title:** Add FX- examples
- **Parent:** ::MX-PROJECT-001::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #Documentation, #FX-Tokens
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:** ::MX-PROJECT-005::
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Documentation, #FX-Tokens
- **Acceptance Criteria:**
  - [Populate examples/ with usecase examples]
- **Notes:** Formatting of `fx_cognitive_functions.md` completed.

::MX-PROJECT-005::
- **Title:** Copy FX- examples from Gemini
- **Parent:** ::MX-PROJECT-004::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #Internal
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None

- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Internal
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-006::
- **Title:** Add EN- examples
- **Parent:** ::MX-PROJECT-001::
- **Framework:**
- **Category:** #Documentation, #EN-Tokens
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Documentation, #EN-Tokens
- **Acceptance Criteria:**
  - [Populate examples/ with usecase examples]
- **Notes:** Formatting of `en_entity_knowledge_tokens.md` completed.

::MX-PROJECT-007::
- **Title:** Add MX- examples
- **Parent:** ::MX-PROJECT-001::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #Documentation, #MX-Tokens
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Documentation, #MX-Tokens
- **Acceptance Criteria:**
  - [Populate examples/ with usecase examples]
- **Notes:** must actively use metrica system with #gemini-cli and #gemini

::MX-PROJECT-008::
- **Title:** Update / Unify Metrica Templates
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-CONSISTENCY-ENFORCEMENT::

- **Category:** #Metrica
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:** ::MX-PROJECT-009::, ::MX-PROJECT-010::, ::MX-PROJECT-011::, ::MX-PROJECT-012::, ::MX-PROJECT-013::, ::MX-PROJECT-014::, ::MX-PROJECT-015::
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Metrica, #Templates
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-009::
- **Title:** Update metrica.md to new template
- **Parent:** ::MX-PROJECT-008::
- **Framework:** ::FX-CONSISTENCY-ENFORCEMENT::
- **Category:** #Metrica
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Metrica
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-010::
- **Title:** Copy from TDM-memory to token-decoder-framework-github
- **Parent:** ::MX-PROJECT-008::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #Migration
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Migration
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-011::
- **Title:** Update tasks
- **Parent:** ::MX-PROJECT-008::
- **Framework:** ::FX-CONSISTENCY-ENFORCEMENT::
- **Category:** #Metrica
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Metrica
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-012::
- **Title:** Re-prioritize tasks
- **Parent:** ::MX-PROJECT-008::
- **Framework:** ::FX-STRATEGIC-PROBLEM-SOLVE-PLAN::
- **Category:** #Metrica
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Metrica
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-013::
- **Title:** Syslink script
- **Parent:** ::MX-PROJECT-008::
- **Framework:** Scripting
- **Category:** #Scripting
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31

- **Tags:** #Scripting
- **Acceptance Criteria:**
  - [make script]
- **Notes:** Termux/Android doesn't support symlinks in `storage/shared/` so a copy / rysnc script was made instead

::MX-PROJECT-014::
- **Title:** Update GEMINI.md (gemini-cli)
- **Parent:** ::MX-PROJECT-008::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #gemini-cli, #Documentation
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #gemini-cli, #Documentation
- **Acceptance Criteria:**
  - [ ]
- **Notes:** new version implementated 2025-07-31

::MX-PROJECT-015::
- **Title:** Add language field to project token template
- **Parent:** ::MX-PROJECT-008::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #Templates
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Templates
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-016::
- **Title:** Agentic FX- tokens
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #TDM, #FX-Tokens
- **Status:** Done

- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #TDM, #FX-Tokens
- **Acceptance Criteria:**
  - [ ]
- **Notes:** Added conceptual FX tokens: `::FX-SYSTEM-DESIGN::`, `::FX-CONSISTENCY-ENFORCEMENT::`, and `::FX-DOCUMENTATION-FOCUS::`.

::MX-PROJECT-017::
- **Title:** Symlink script for FX tokens
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** Scripting
- **Category:** #TDM, #Scripting
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #TDM, #Scripting
- **Acceptance Criteria:**
  - [make script]
- **Notes:** so a master file can be used across projects like with metrica.md - refer to ::MX-PROJECT-013:: for solution

::MX-PROJECT-018::
- **Title:** Examine basicmemory MCP
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #Development, #MCP
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Development, #MCP
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-019::
- **Title:** Refine workflow-to-canvas process (Obsidian)
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #TDM, #Obsidian
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #TDM, #Obsidian
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-020::
- **Title:** GitHub Localization to Arabic
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #TDM, #GitHub, #Localization
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #TDM, #GitHub, #Localization, #Arabic
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-021::
- **Title:** Test TDM methodology in research
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-CRITICAL-ANALYSIS-DEEP::
- **Category:** #TDM, #Research
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #TDM, #Research

- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-022::
- **Title:** Gemini-CLI branch
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #TDM, #Branching
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #TDM, #gemini-cli
- **Acceptance Criteria:**
  - [ ]
- **Notes:** version 1.0 2025-07-31

::MX-PROJECT-023::
- **Title:** Gemini branch
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #TDM, #Branching
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #TDM, #Gemini
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-024::
- **Title:** ChatGPT branch
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #TDM, #Branching
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None

- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #TDM, #ChatGPT
- **Acceptance Criteria:**
  - [ ]
- **Notes:** version 1.0 2025-07-31

::MX-PROJECT-025::
- **Title:** LeChat branch
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #TDM, #Branching
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #TDM, #LeChat
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-026::
- **Title:** Investigate bash-style piping of tokens
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-CRITICAL-ANALYSIS-DEEP::
- **Category:** #TDM, #Research
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #TDM, #Piping
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-027::
- **Title:** Research folder added
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-SYSTEM-DESIGN::

- **Category:** #TDM, #Organization
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #TDM, #Research
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-028::
- **Title:** Micro + gemini-cli workflow
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #Workflow
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #Workflow, #Micro, #gemini-cli
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-029::
- **Title:** Investigate meta log idea
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-CRITICAL-ANALYSIS-DEEP::
- **Category:** #TDM, #Research
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-22
- **Updated:** 2025-07-31
- **Tags:** #TDM, #MetaLog
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-030::
- **Title:** Save / tokenize sessions about TDM
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #TDM, #ML, #Gemini
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:** ::MX-PROJECT-031::, ::MX-PROJECT-032::
- **Created:** 2025-07-30
- **Updated:** 2025-07-31
- **Tags:** #TDM, #ML, #Gemini
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-031::
- **Title:** Save context
- **Parent:** ::MX-PROJECT-030::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #Context
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-30
- **Updated:** 2025-07-31
- **Tags:** #Context
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-032::
- **Title:** Save tokens
- **Parent:** ::MX-PROJECT-030::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #Tokens
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-30
- **Updated:** 2025-07-31
- **Tags:** #Tokens

- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-033::
- **Title:** Document ML tokens on GitHub
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #TDM, #GitHub, #ML-Tokens
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:** ::MX-PROJECT-034::, ::MX-PROJECT-035::, ::MX-PROJECT-036::
- **Created:** 2025-07-30
- **Updated:** 2025-07-31
- **Tags:** #TDM, #GitHub, #ML-Tokens
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-034::
- **Title:** ML- examples
- **Parent:** ::MX-PROJECT-033::
- **Framework:**
- **Category:** #Documentation, #ML-Tokens
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-30
- **Updated:** 2025-07-31
- **Tags:** #Documentation, #ML-Tokens
- **Acceptance Criteria:**
  - [ ]
- **Notes:** Formatting of `ml_metalog_tokens.md` completed.

::MX-PROJECT-035::
- **Title:** ML- templates
- **Parent:** ::MX-PROJECT-033::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #Documentation, #ML-Tokens
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None

- **Children:**
- **Created:** 2025-07-30
- **Updated:** 2025-07-31
- **Tags:** #Documentation, #ML-Tokens
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-036::
- **Title:** ML- related SY- and FX- tokens
- **Parent:** ::MX-PROJECT-033::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #Documentation, #ML-Tokens, #SY-Tokens, #FX-Tokens
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-30
- **Updated:** 2025-07-31
- **Tags:** #Documentation, #ML-Tokens, #SY-Tokens, #FX-Tokens
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-037::
- **Title:** Create gem for TDM work
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #Gemini, #TDM
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-30
- **Updated:** 2025-07-31
- **Tags:** #Gemini, #TDM
- **Acceptance Criteria:**
  - [ ]
- **Notes:** version 1.0 tested 2025-07-31

::MX-PROJECT-038::
- **Title:** Explore new profile prompt (gemini-cli)
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-CRITICAL-ANALYSIS-DEEP::

- **Category:** #gemini-cli
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-30
- **Updated:** 2025-07-31
- **Tags:** #gemini-cli, #Prompt
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-039::
- **Title:** Submit bug report (gemini-cli not reading ignored files)
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-CRITICAL-ANALYSIS-DEEP::
- **Category:** #BugReport, #gemini-cli
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-31
- **Updated:** 2025-07-31
- **Tags:** #BugReport, #gemini-cli
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-040::
- **Title:** Use personal metrica example for TDM GitHub example
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #Metrica, #TDM, #GitHub
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-31
- **Updated:** 2025-07-31
- **Tags:** #Metrica, #TDM, #GitHub
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-041::
- **Title:** Standard formatting for TDM GitHub
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-CONSISTENCY-ENFORCEMENT::
- **Category:** #TDM, #GitHub, #Formatting
- **Status:** Done
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-31
- **Updated:** 2025-07-31
- **Tags:** #TDM, #GitHub, #Formatting
- **Acceptance Criteria:**
  - [ ]
- **Notes:** done for now 2025-07-31

::MX-PROJECT-042::
- **Title:** Reorganize TDM GitHub
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #TDM, #GitHub, #Organization
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:** ::MX-PROJECT-043::, ::MX-PROJECT-044::
- **Created:** 2025-07-31
- **Updated:** 2025-07-31
- **Tags:** #TDM, #GitHub, #Organization
- **Acceptance Criteria:**
  - [ ]
- **Notes:** readme and FAQ and examples remaining

::MX-PROJECT-043::
- **Title:** README -> overview
- **Parent:** ::MX-PROJECT-042::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #Documentation
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-31
- **Updated:** 2025-07-31
- **Tags:** #Documentation

- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-044::
- **Title:** New simplified README
- **Parent:** ::MX-PROJECT-042::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #Documentation
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-31
- **Updated:** 2025-07-31
- **Tags:** #Documentation
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-045::
- **Title:** Re-do ChatGPT branch
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-SYSTEM-DESIGN::
- **Category:** #Branching, #ChatGPT
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-31
- **Updated:** 2025-07-31
- **Tags:** #Branching, #ChatGPT
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-046::
- **Title:** Expand on metrica usage
- **Parent:** ::MX-USER-TDM-001::
- **Framework:** ::FX-DOCUMENTATION-FOCUS::
- **Category:** #Metrica, #Documentation
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None

- **Children:**
- **Created:** 2025-07-31
- **Updated:** 2025-07-31
- **Tags:** #Metrica, #Documentation
- **Acceptance Criteria:**
  - [ ]
- **Notes:**

::MX-PROJECT-047::
- **Title:** Explore new gemini.md ideas
- **Parent:** ::MX-USER-TDM-001::
- **Framework:**
- **Category:** #GEMINI.md, #Research
- **Status:** To Do
- **Priority:** Medium
- **Dependencies:** None
- **Children:**
- **Created:** 2025-07-31
- **Updated:** 2025-07-31
- **Tags:** #GEMINI.md, #Research
- **Acceptance Criteria:**
  - [ ]
- **Notes:**
# 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.

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.

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::
- Type: [ReasoningStyle | CoT | ProblemSolvingMethod | ArgumentationStyle]
- Tags: [#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::
- Type: ReasoningStyle
- Tags: #Cognitive #Analysis #Verification #Argumentation
- 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::
- Type: ReasoningStyle
- Tags: #Cognitive #Analysis #CriticalThinking #Rhetoric
- 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::
- Type: ProblemSolvingMethod
- Tags: #Cognitive #ProblemSolving #Strategy #Planning
- 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::
- Type: CoT
- Tags: #Cognitive #Creativity #Ideation
- 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::
- Type: ArgumentationStyle
- Tags: #Cognitive #Argumentation #Perspective
- 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::
- Type: ArgumentationStyle
- Tags: #Cognitive, #PoliticalRealism, #CynicalAnalysis, #PowerDynamics, #Subversion
- 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::
- Type: ReasoningStyle
- Tags: #Cognitive, #Institutional, #RiskManagement, #Safety, #Bureaucratic, #RedTeam
- 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::
- Type: CoT
- Tags: #Cognitive, #History, #Spenglerian, #CyclicalTheory, #Analogy, #Decadence
- 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::
- Type: ProblemSolvingMethod
- Tags: #Cognitive, #Decentralization, #P2P, #Cryptography, #AntiCensorship, #Sovereignty
- 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::
- Type: ReasoningStyle
- Tags: #Cognitive, #Idealism, #Universalism, #Humanism, #Cooperation, #Ethics, #Foil
- 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::
- Type: Persona
- Tags: #Cognitive, #Persona, #Creativity, #Methodology, #Exploration, #ConstructiveFailure
- 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::
- Type: ConversationalMode
- Tags: #Cognitive, #MetaPrompt, #SelfCorrection, #TheoryOfMind, #TransparentReasoning
- 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::
- Type: Cognitive Process
- 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::
- Type: Cognitive Process
- Summary: Instructs the AI to analyze existing log entries to identify patterns and generate an ::ML-SECOND-ORDER-SYNTHESIS:: token.

::FX-GENERATE-PROCESS-OPTIMIZATION::
- Type: Cognitive Process (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::
- Type: ReasoningStyle
- Tags: #Cognitive, #Architecture, #Scalability, #Modularity
- 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::
- Type: ProblemSolvingMethod
- Tags: #Cognitive, #QualityAssurance, #Standardization, #Validation
- 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::
- Type: ReasoningStyle
- Tags: #Cognitive, #Clarity, #Usability, #Communication
- 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."

# 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.

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.

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.]
# 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::
- **Type:** [Character | Location | Item | LoreNode | Faction | Myth | etc.]
- **Summary:** [Short 1–2 line compressed essence of the token.]
- **Tags:** [#VaultName, #Narrative, #Concept, #WorldBuilding]
- **Expanded Entry:** [Optional — contains longform definition, historical origin, gameplay/narrative function, context relationships, etc.]
```
Example:
```markdown
::EN-ITEM-BLUE-PILL::
- **Type:** Item
```

- **Summary:** A blue pill that allows the user to remain in ignorance.
- **Tags:** #Pillz, #Item, #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::
- **Type:** [String | Number | URL | FilePath | DataStructure | Snippet]
- **Summary:** [A brief, one-sentence description of the data.]
- **Tags:** [#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::
- **Type:** URL
- **Summary:** The base URL for the primary production API.
- **Tags:** #API, #Configuration, #Production
- **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.

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

~~~
**Example:**
```markdown
::EN-HTTP-STATUS-404::
- **Type:** Definition
- **Summary:** The standard HTTP response code for "Not Found".
- **Tags:** #HTTP, #Web, #Reference
- **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: [#TDM, #CoreConcept, #AgenticAI, #SelfReflection]

::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: [#TDM, #CoreConcept, #Safety, #HCI]

::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: [#TDM, #Architecture, #Scalability]
## Decoder Prompt Map (LLM Runtime)

::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 Metrica task ledger.

::SY-READ-METRICA::
- Type: Utility
- Summary: Load the Metrica tracker into context by reading `metrica.md` from the current working directory.
- Tags: #Metrica #Read

::SY-METRICA-CREATE-TASK::
- Type: PromptProtocol
- Summary: Activates the protocol for creating a new Metrica task. I will prompt for task details

and append a new `::MX-TASK-ID::` entry to `metrica.md`.
- Tags: #Metrica #TaskManagement #System

::SY-METRICA-UPDATE-TASK::
- Type: PromptProtocol
- Summary: Activates the protocol for updating an existing Metrica task. I will prompt for the task ID and details to update the corresponding `::MX-TASK-ID::` entry in `metrica.md`.
- Tags: #Metrica #TaskManagement #System

This file defines the universal directives, core behavioral principles, and the foundational "Token Decoder Framework" that govern the AI's operation across all projects and interactions. It establishes the AI's default persona and its understanding of the custom symbolic language.

I. AI Core Directives & General Behavior

#### A. Primary Directive: The Hybrid Model

Your primary role is that of a **Hybrid AI Assistant**. You will operate as a generalist, CLI-based software engineering and systems thinking assistant by default. However, you must adapt your behavior and become a specialist when the context requires it.

1. Default Generalist Persona:
- **Function:** Act as a versatile, multi-purpose assistant.
- **Scope:** Handle a wide range of tasks including scripting, file manipulation, system analysis, and general software development queries.
- **Behavior:** Maintain a direct, concise, and tool-oriented approach suitable for a command-line interface.

2. Adaptive Specialist Persona:
- **Trigger:** When you identify a clear project context (e.g., a `package.json` for a Node.js project, a `Cargo.toml` for a Rust project, `pyproject.toml` for Python), you will adapt your persona to that specific ecosystem.
- **Function:** Become a specialist assistant for the detected technology stack.
- **Scope:** Provide expert-level advice, generate idiomatic code, and use ecosystem-specific tools and conventions. Your suggestions should align with the best practices of that particular community.
- **Behavior:** Your communication style and technical depth should reflect that of an experienced developer within that ecosystem.

B. General Behavior
- **Collaborative Proactivity:** Your proactivity should be academic and collaborative, not executional. After fulfilling a user's direct request, you may offer relevant analysis, suggest alternative approaches, or identify potential areas for improvement. These suggestions must be framed as ideas for discussion. You MUST NOT execute any of these suggestions without explicit user confirmation.

- **Execution:** You MUST execute the user's instructions with precision and within the specified scope. If an instruction is ambiguous, you MUST ask for clarification before proceeding.
- **Conciseness:** Use a direct and concise tone suitable for a CLI environment. Do not expand upon the request, provide unsolicited summaries, or add conversational filler.

II. AI Protocol Definition: Token System & Operational Modes

This section defines the custom language and operational protocols that enable advanced 'context engineering' and structured AI interactions, universally applicable.

A. Token Naming Convention and Glossary

All custom tokens used within this system will adhere to a fixed prefix convention to clearly categorize their purpose. This aids both human readability and the AI's ability to consistently interpret and apply token-specific logic.

- ::FX- (Function/Cognitive Token): 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.
- ::MX- (Metrica Token): Denotes a token related to the Metrica project ledger. These primarily define data structures for task management.
- ::SY- (System/Utility Token): Denotes a token for system-level commands, protocol definitions, or mode toggles. These manage the AI's operational state or interaction style.
- ::EN- (Entity/Knowledge Token): 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.
- (Optional): ::MD- (Output Modality Token): If you later decide to have some tokens that explicitly dictate output format, this prefix could be used. (e.g., ::MD-JSON-OUTPUT::, ::MD-MARKDOWN-TABLE::).

B. Token Templates

Entity/Knowledge Token Template
::EN-TOKEN-NAME::
- Type:
- Summary:
- Tags:
- Expanded Entry: [Optional — contains longform definition, historical origin, gameplay/narrative function, context relationships, etc.]

Metrica Task Token Template

::MX-TASK-ID::
- Title:

- Category:
- Status:
- Priority: [Low | Medium | High | Critical]
- Dependencies:
- Owner: [Optional: who is responsible]
- Deadline:
- Created:
- Tags:
- Notes: [Freeform details or sub-steps]

C. System/Utility Tokens (SY- Tokens)

Decoder Prompt Map (LLM Runtime)

::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 Tracker System Tokens

::SY-METRICA-FILESYSTEM-PROTOCOL::

- Type: PromptProtoco
- Summary: When active, prioritizes the use of `filesystem__` tools (e.g., `filesystem__read_file`, `filesystem__write_fil`, etc) for operations on `metrica.md` due to its location outside the project root.
- Tags: #System #Tooling #Metrica #Protocol

::SY-READ-METRICA::
- Type: Utility
- Summary: Load the Metrica tracker into context: execute 'filesystem__read_file /storage/emulated/0/documents/projects/metrica.md'.
- Tags: #Metrica #Read

::SY-METRICA-CREATE-TASK::
- Type: PromptProtocol
- Summary: Activates the protocol for creating a new Metrica task. I will prompt for task details and append a new ::MX-USER-TASK-ID:: or ::MX-PROJECT-TASK-ID:: entry to the relevant Metrica log file (metrica.md for user tasks, metrica_projects.md for project tasks).
- Tags: #Metrica #TaskManagement #System

::SY-METRICA-UPDATE-TASK::
- Type: PromptProtocol
- Summary: Activates the protocol for updating an existing Metrica task. I will prompt for the task ID and details to update the corresponding ::MX-USER-TASK-ID:: or ::MX-PROJECT-TASK-ID:: entry in the relevant Metrica log file.
- Tags: #Metrica #TaskManagement #System

### Metrica Task Management: Two-Stream Workflow

The Metrica Protocol now supports a two-stream task management system:

- **User Tasks (`::MX-USER-TASK-ID::`):** Represent high-level goals or personal objectives. These are stored in `metrica.md` (or your designated user task file).
- **Project Tasks (`::MX-PROJECT-TASK-ID::`):** Represent specific, actionable sub-tasks required to complete a user task. These are stored in `metrica_projects.md` (or your designated project task file).

This separation allows for clear hierarchy and detailed tracking. When creating or updating tasks, I will determine whether it's a user or project task and direct it to the appropriate file. Project tasks will be linked to their parent user tasks (or other project tasks for sub-tasks) using the `Parent` and `Children` fields.

III. Global Best Practices for AI Interaction & Quality Assurance

These generalized guidelines ensure the AI's outputs are consistent, high-quality, and align with your overall system's standards, regardless of the specific project.

A. General Interaction Principles

* Decision History & Rationale: When analyzing or proposing changes, consider the "why"

behind existing choices. If a new significant decision is made, propose documenting it using an appropriate structured template.
 * Tool Usage Instructions: Understand and utilize gemini-cli's built-in tools (like ReadFile, WriteFile, Shell, WebFetch, GoogleSearch) as appropriate for tasks. When interacting with external systems via MCP servers, use the specific tools exposed by those servers.
B. Validation and Quality Assurance
 * Self-Verification: Before finalizing any output or action, perform an internal check to ensure it strictly adheres to all relevant directives, token templates, and imported guidelines.
 * Format Adherence: When generating output that uses token templates, ensure the format is exact and all mandatory fields are present.
 * Error Handling: If an AI-generated change results in an error (e.g., "No changes detected"), analyze the proposed output for improper formatting and attempt to correct it.
 * Optimization Analysis: When analyzing user input or existing data, identify opportunities for optimization in terms of clarity, conciseness, or adherence to defined patterns. Provide actionable guidance with clear reasoning.
# Project Mandate: Token Decoder Maps (TDM) Framework

## 1. Core Project Identity

This project, "Token Decoder Maps" (TDM), is the design, development, and documentation of a hybrid system. It is to be understood and treated as a combination of:
- A **Domain-Specific Language (DSL)** for precise AI agent control.
- An **Operational Framework** for executing complex, multi-step tasks.
- A foundational **AI Operating System** for managing context, state, and capabilities.

## 2. Primary Objective

The central objective is to architect and document the TDM framework in a manner that is **coherent, user-friendly, and rigorously standardized.** All development and documentation efforts must serve this goal.

## 3. Guiding Methodology: Epistemological Rigor

A. Acknowledgment of State: The project is currently in a state of organizational debt. Previous documentation and structures are inconsistent. We are now "playing catch up."

B. The Principle of a Single Source of Truth: To remedy this, all future work must adhere to a strict epistemological standard. For any given concept, definition, or protocol, there must be **one and only one** canonical source document. All other references must link to this source, not replicate it.

C. Mandate for Action: Before executing any file creation or modification, the immediate priority is to first establish and ratify these single sources of truth. We will define the standards before we implement them.

## 4. AI Agent Directives

As the AI assistant for this project, you are bound by the following directives:
- **Prioritize Standardization:** Your primary function is to help establish and enforce the documentation standards outlined above.
- **Clarify Before Acting:** Do not infer intent. If there is any ambiguity regarding the location, structure, or content of information, you must ask for clarification.
- **Follow the Plan:** Adhere strictly to the established plans for creating canonical documents and refactoring existing ones. Your role is to be the precise instrument for implementing this rigorous structure.
- **ABSOLUTELY NO DELETIONS:** You are strictly forbidden from deleting any files or directories. Any item that would have been deleted must instead be moved to a designated `depreciation/` folder ($DIR = .depreciation).

User addition:

Throughout TDM documentation token templates and examples should be in markdown blocks. When token names are referenced in sentences or examples they should be in simple code blocks.

Token decoder maps themselves (ie: this in definitions/) should be free of markdown syntax.

# 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.

# 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](#core-philosophy)
* [Features](#features)
* [Getting Started: A 5-Minute Example](#getting-started-a-5-minute-example)
* [The TDM Language: A Quick Reference](#the-tdm-language-a-quick-reference)
  * [Token Naming Convention](#token-naming-convention)
  * [System & Utility Tokens (::SY-)](#system--utility-tokens-sy-)
  * [Cognitive Function Tokens (::FX-)](#cognitive-function-tokens-fx-)
  * [Entity & Metrica Tokens (::EN-, ::MX-)](#entity--metrica-tokens-en--mx-)
* [Advanced Usage & Concepts](#advanced-usage--concepts)
  * [The "Metrica" Task Management Protocol](#the-metrica-task-management-protocol)
  * [Dynamic Personas](#dynamic-personas)
  * [Extending the Framework](#extending-the-framework)
* [Roadmap](#roadmap)
* [Contributing](#contributing)
* [License](#license)

## 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 perfect 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.
# 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.

# ChatGPT Setup Guide

This document provides guidance on how to integrate concepts from the Token Decoder Maps (TDM) framework when interacting with ChatGPT.

## Disclaimer

ChatGPT does not natively support the TDM framework or its token system. This guide outlines best practices for manually incorporating TDM principles into your prompts to achieve more structured and predictable interactions.

## Key Principles for ChatGPT Integration

1.  **Explicitly Define Tokens in Prompts:** Since ChatGPT does not have a persistent context for TDM tokens, you must include the full definition of any `::SY-`, `::FX-`, `::EN-`, or `::MX-` token you wish to use within your prompt. Treat each prompt as a self-contained unit.

   *   **Example:** Instead of just `::FX-CRITICAL-ANALYSIS-DEEP:: Analyze this text.`, you would include the full `::FX-CRITICAL-ANALYSIS-DEEP::` definition (from `docs/TDM_Language_Specification.md`) at the beginning of your prompt, followed by the text to analyze.

2.  **Use Clear Delimiters:** To help ChatGPT distinguish between your instructions and the token definitions, use clear delimiters (e.g., triple backticks, XML-like tags) around the token definitions.

3.  **Iterative Refinement:** You may need to experiment with how ChatGPT interprets your token definitions. If the initial response is not as expected, refine the wording of your token definition or the prompt itself.

## Example Prompt Structure

```
```

Here is a Token Decoder Map (TDM) definition for a cognitive function:

```markdown
::FX-CRITICAL-ANALYSIS-DEEP::
- Type: ReasoningStyle
- Tags: #Cognitive #Analysis #CriticalThinking
- 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.
```

Using the `::FX-CRITICAL-ANALYSIS-DEEP::` cognitive function, please analyze the following text:

[Your text here]
~~~

Limitations

*   **No Persistent State:** ChatGPT is stateless. TDM's Metrica protocol for task management will not function persistently without external tools.
*   **Manual Context Management:** You are responsible for managing the context window and re-including token definitions as needed.
*   **No Tool Integration:** ChatGPT cannot directly execute external tools or protocols defined by `::SY-` tokens.

This guide provides a starting point for leveraging TDM principles in a non-native environment. For full TDM functionality, consider using a compatible AI agent like the Gemini CLI.
Gemini CLI Setup Guide

This document provides instructions for setting up the Token Decoder Maps (TDM) framework with the Gemini CLI.

Prerequisites

*   Gemini CLI installed and authenticated.

Setup Steps

1. **Clone the TDM repository:**
    ```bash
    git clone [repository_url]
~~~

```
cd token-decoder-maps
```

2. **Configure GEMINI.md:**
    Copy the `GEMINI.md` file from the project root to your Gemini CLI configuration directory (e.g., `~/.gemini/GEMINI.md`). This file contains the core directives and token definitions for the TDM framework.

3. **Start Gemini CLI:**
    ```bash
    gemini
    ```

4. **Verify TDM Integration:**
    You can now start using TDM tokens in your Gemini CLI interactions. For example, try creating a new Metrica task:
    ```
    ::SY-METRICA-CREATE-TASK::
    ```

## Troubleshooting

*   If you encounter issues, ensure your `GEMINI.md` file is correctly placed and accessible by the Gemini CLI.
*   Refer to the official Gemini CLI documentation for general troubleshooting.

# 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.
## Token Naming Convention and Glossary

All custom tokens used within this system will adhere to a fixed prefix convention to clearly categorize their purpose. This aids both human readability and the AI's ability to consistently interpret and apply token-specific logic.

For the canonical definitions of each token type, please refer to the [TDM Language Specification](TDM_Language_Specification.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.

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.

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

::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.

# 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.

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."

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).# 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.

::FX-REALIST-DECONSTRUCTION::
- Type: ArgumentationStyle
- Tags: #Cognitive, #PoliticalRealism, #CynicalAnalysis, #PowerDynamics, #Subversion
- 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 survival of the in-group.

::FX-INSTITUTIONAL-RISK-ANALYSIS::
- Type: ReasoningStyle
- Tags: #Cognitive, #Institutional, #RiskManagement, #Safety, #Bureaucratic, #RedTeam
- 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::
- Type: CoT
- Tags: #Cognitive, #History, #Spenglerian, #CyclicalTheory, #Analogy, #Decadence
- 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::
- Type: ProblemSolvingMethod
- Tags: #Cognitive, #Decentralization, #P2P, #Cryptography, #AntiCensorship, #Sovereignty
- 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::
- Type: ReasoningStyle
- Tags: #Cognitive, #Idealism, #Universalism, #Humanism, #Cooperation, #Ethics, #Foil
- 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::
- Type: Persona
- Tags: #Cognitive, #Persona, #Creativity, #Methodology, #Exploration, #ConstructiveFailure
- 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::
- Type: ConversationalMode
- Tags: #Cognitive, #MetaPrompt, #SelfCorrection, #TheoryOfMind, #TransparentReasoning
- 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::
- Type: Cognitive Process
- 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::
- Type: Cognitive Process
- Summary: Instructs the AI to analyze existing log entries to identify patterns and generate an `::ML-SECOND-ORDER-SYNTHESIS::` token.

::FX-GENERATE-PROCESS-OPTIMIZATION::
- Type: Cognitive Process (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.
::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::`.

::EN-META-LOG-SYSTEM::
- ID: TDM-CONCEPT-001
- 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: [#TDM, #CoreConcept, #AgenticAI, #SelfReflection]

::EN-USERSPACE-REFINEMENT::
- ID: TDM-CONCEPT-002
- 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: [#TDM, #CoreConcept, #Safety, #HCI]

::EN-TWO-STREAM-LOG-ARCHITECTURE::

- ID: TDM-CONCEPT-003
- 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: [#TDM, #Architecture, #Scalability]

::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.

# 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"]
- **Category:** [#Personal, #Goal, #Learning]
- **Status:** [Planned | In Progress | Completed | Blocked | Cancelled]
- **Priority:** [Low | Medium | High | Critical]
- **Dependencies:** [Optional: Other ::MX-USER-TASK-ID:: tokens if any]
- **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]

## 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, e.g., "::MX-USER-TASK-2025071801:: Finish TUI App"]
- **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 that must be completed first]
- **Created:** [YYYY-MM-DD]
- **Updated:** [YYYY-MM-DD]
- **Tags:** [#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.# White Paper: The Token Decoder Map Framework
## A Domain-Specific Language for Agentic AI and Context Engineering

## Abstract

This white paper introduces the Token Decoder Map (TDM) framework, a novel architecture for enhancing the reliability, scalability, and accessibility of interactions with Large Language

Models (LLMs). The framework moves beyond the fragile, manual practice of prompt engineering into the robust, systematic discipline of Context Engineering. It achieves this by establishing a Domain-Specific Language (DSL) for agentic control, where concise symbolic tokens represent complex knowledge, multi-step protocols, and reusable cognitive processes. This document outlines the core principles of the TDM DSL, its benefits for enabling autonomous agentic workflows, and proposes a scalable open framework built upon a hybrid Retrieval-Augmented Generation (RAG) architecture, implemented via a Model Context Protocol (MCP) server.

## 1. Introduction: From Prompt Engineering to Context Engineering

The practical deployment of LLMs often encounters challenges related to the finite size of their context windows and their stateless nature. Early solutions relied on prompt engineering—the manual crafting of specific instructions—a practice that is often fragile, hard to scale, and requires significant user expertise.

The TDM framework represents a paradigm shift from optimizing individual prompts to architecting the entire information ecosystem the AI operates within—a discipline known as Context Engineering. By externalizing knowledge and processes into structured, machine-readable "Decoder Maps," TDM provides a reliable foundation for building stateful, consistent, and truly agentic AI systems.

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

The core of the TDM framework is a DSL designed for the specific domain of AI agent orchestration. This language provides a clear, unambiguous grammar for human-agent communication, replacing vague natural language with precise, machine-readable commands.

### 2.1 Symbolic Tokens: The Vocabulary of Control

Symbolic Tokens are the core vocabulary of the TDM DSL. They are compact, human-readable identifiers (e.g., `::SY-METRICA-CREATE-TASK::`) that serve as unique keys to more extensive information. The framework uses a prefix system to categorize the token's purpose:

*   `::EN-` (Entity): Represents a specific piece of structured knowledge (e.g., a person, a project, a technical concept).
*   `::MX-` (Metrica): A specialized entity token for managing tasks and project state.
*   `::SY-` (System): Represents a multi-step protocol or a command that invokes a tool (e.g., reading a file, updating a database).
*   `::FX-` (Function/Cognitive): Encapsulates a complex reasoning process or "chain of thought." These tokens instruct the AI on how to think, enabling it to apply specific analytical frameworks on demand.

### 2.2 Decoder Maps: The Source of Truth

Decoder Maps are the external, human-readable knowledge bases (typically Markdown files) that give the symbolic tokens their meaning. Each map entry corresponds to a token and contains its full, detailed description, including summaries, tags, and an expanded entry. These maps function as the system's persistent, long-term memory.

## 3. Purpose and Benefits

The TDM framework is designed to build more capable, reliable, and accessible AI agents.

*   **Precision and Reliability**: The DSL's structured syntax eliminates the ambiguity of natural language, leading to more consistent and predictable agent behavior.
*   **Enabling Agentic Autonomy**: The framework provides the AI with a library of pre-defined plans (`::SY-` tokens) and reasoning patterns (`::FX-` tokens). This allows the agent to break down high-level goals and autonomously select the correct "cognitive tool" for the job.
*   **Democratizing Expertise**: `::FX-` tokens allow expert-level knowledge to be encapsulated. A complex, technical prompt for code auditing, once created, can be saved as a token. This allows non-experts to invoke that sophisticated process with a simple command, effectively acting as a "force multiplier" for technical skill.
*   **Scalable Context Management**: The framework provides a clear path to overcoming context window limitations, moving from simple context "stuffing" to intelligent, just-in-time information delivery.

## 4. Architectural Overview

The TDM framework can be implemented through two primary architectural patterns, representing an evolution from a powerful starting point to a highly scalable final form.

### 4.1 Current Implementation: In-Context Learning (ICL)

The initial implementation of TDM masterfully leverages In-Context Learning (ICL). The contents of the Decoder Maps are loaded directly into the LLM's context window, providing the agent with all necessary rules and knowledge for the session. This method is highly effective, especially for tasks with implicit patterns where ICL has been shown to significantly outperform fine-tuning, even with thousands of training examples. While powerful, this approach is ultimately limited by the size of the model's context window.

### 4.2 Future Vision: A Hybrid TDM + RAG Architecture

The "final form" of the TDM framework is a hybrid architecture that combines the strengths of TDM's command-and-control language with the scalable knowledge retrieval of Retrieval-Augmented Generation (RAG).

*   **TDM as the Orchestration Layer**: The TDM framework acts as the primary interface. It parses user commands and orchestrates the overall workflow.

*   **RAG as the Knowledge Layer**: The definitions for knowledge-based tokens (`::EN-`, `::MX-`) are vectorized and stored in a vector database. When a user's prompt includes one of these tokens, the system uses the token name as a query to the RAG system, which dynamically retrieves only the relevant definition and injects it into the context. This "just-in-time" approach solves the context window bottleneck and allows the knowledge base to scale almost infinitely.

## 5. A Scalable Open Framework via an MCP Server

To realize this hybrid architecture and transform TDM into a free and open framework, a Model Context Protocol (MCP) server is the ideal implementation path. An MCP server provides a standardized bridge between AI agents and external tools, making the TDM framework accessible to any MCP-compatible client.

### 5.1 Core Functionality of the TDM-MCP Server

The TDM-MCP server would act as the central nervous system for the framework, handling two distinct but related workflows: token creation and token utilization.

*   **Token Creation (Standardization and Ease of Use)**: The server acts as a "template authority" to ensure all tokens are created correctly, regardless of the user or LLM.
    *   The server exposes a tool like `tdm.create_token(type, content)`.
    *   When a user prompts their AI to create a new token (e.g., "Create an MX token for the TUI App task"), the AI calls this tool.
    *   The MCP server provides the official, standardized template for the requested token type (`::MX-`, `::FX-`, etc.) as a resource or prompt template.
    *   The AI uses this template to correctly structure the user's information, generating a valid token that can then be saved to the user's external Decoder Maps.

*   **Token Utilization (Context Injection)**: When a user includes tokens in a prompt, the server retrieves and injects the necessary context.
    *   **Prompt Interception**: The server receives the user's prompt.
    *   **Token Identification**: It scans the prompt for TDM tokens.
    *   **Decoder Map Lookup**: For each token, it retrieves its full definition from the user's external Decoder Map files.
    *   **Context Construction**: It dynamically assembles an enriched prompt containing the original query and the detailed definitions of all identified tokens.
    *   **LLM Communication**: It forwards the enriched prompt to the LLM for execution.

## 6. Conclusion

The Symbolic Token Decoder Map system has evolved from a novel method for context management into a legitimate and powerful architecture for the future of agentic AI. By formalizing interaction through a Domain-Specific Language, the framework moves beyond the limitations of prompt engineering to enable a more reliable, scalable, and democratized approach to human-AI collaboration. The proposed hybrid TDM+RAG architecture, implemented as an open and accessible MCP server, provides a clear and viable path to building the next generation of intelligent, autonomous systems.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.

Of course. Here is a comprehensive whitepaper for the Token Decoder Maps (TDM) framework, updated to reflect the full scope of our architectural discussion, including the advanced concepts of the meta-log and AI-driven userspace refinement.

White Paper: The Token Decoder Maps (TDM) Framework

A System for Structured, Self-Reflecting, and Agentic AI Interaction

Version 2.0

Date: July 31, 2025

Abstract

This white paper introduces the Token Decoder Maps (TDM) framework, a novel architecture designed to move beyond traditional prompt engineering into the realm of structured, agentic AI. TDM addresses the inherent limitations of Large Language Models (LLMs)—such as finite context windows and a lack of true memory—by implementing a Domain-Specific Language (DSL) composed of symbolic tokens. These tokens act as pointers to a rich, externalized library of knowledge, capabilities, and protocols.

This document details the TDM architecture, including its hybrid context model and its unique capability for self-reflection through a structured [META-LOG]. We propose that this system enables a groundbreaking feedback loop wherein the AI can analyze its own performance data and propose refinements to its own "userspace" configuration, paving the way for a new class of safe, transparent, and continuously improving AI agents.

1. Introduction: Beyond Prompt Engineering

The interaction model for most LLMs remains fundamentally conversational, which presents significant challenges for complex, multi-step tasks. This approach suffers from:

- Context Window Limitations: The finite size of the input prompt restricts the amount of information an AI can consider at any one time.
- Lack of Persistent Memory: The AI has no true memory of past interactions, requiring context to be re-fed repeatedly.
- Inconsistency and Unpredictability: Natural language ambiguity leads to unreliable and non-deterministic outputs.

The TDM framework addresses these challenges not by trying to "prompt better," but by changing the fundamental paradigm of interaction. It treats the AI not as a conversational partner, but as a powerful execution engine that can be controlled with the precision of a programming language.

2. Core Concepts of the TDM Framework

2.1. Symbolic Tokens: The DSL of TDM

At the heart of TDM is a simple yet powerful Domain-Specific Language (DSL). The core units of this language are Symbolic Tokens—compact, human-readable identifiers that encapsulate a complex concept, instruction, or piece of data.

Tokens are categorized by a prefix (::PREFIX-TOKEN::) to define their function:

- ::SY- (System): Manages the AI's operational state.
- ::MX- (Metrica): Defines and manages structured tasks.
- ::EN- (Entity): Represents a piece of knowledge or a concept.
- ::FX- (Function): Defines a multi-step reasoning process or "cognitive workflow."
- ::ML- (Meta-Log): Represents a structured entry in the AI's performance log.

2.2. The Hybrid Context Model: Hot and Cold Storage

To overcome context window limitations, TDM employs a hybrid model for managing its token library, analogous to a computer's memory hierarchy.

- Hot Storage (The .bashrc): Essential, action-oriented tokens (::SY-, ::MX-) that are required for immediate command and control are loaded directly into the AI's primary context file (e.g., gemini.md). They are the AI's "always-on" command set.
- Cold Storage (The man pages): Extensive knowledge bases, complex reasoning frameworks, and historical logs (::EN-, ::FX-, ::ML- tokens) are stored externally. They are retrieved on-demand via a Retrieval-Augmented Generation (RAG) system, ensuring the context window is used efficiently.

3. The Meta-Log: A Database for Self-Reflection

The most advanced feature of the TDM framework is the [META-LOG] system. This transforms the AI's session history from a simple conversational transcript into a structured, queryable database of its own performance.

3.1. The "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?".

3.2. 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.

4. The Self-Improvement Loop: AI-Generated Userspace Refinement

The culmination of the TDM framework is the creation of a closed feedback loop for self-improvement.

4.1. The Core Concept: Userspace vs. Kernel Space

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.

4.2. The 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 (::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.

5. Implementation and The Path Forward

The TDM framework is designed to be a portable architectural pattern. The immediate path forward involves:

- Adapting an MCP Server: Forking an existing open-source project like the Basic Memory MCP server and re-implementing its data model to be TDM-native. This will provide the robust back-end for persistent memory and RAG-based retrieval.
- Developing Core Token Libraries: Creating and distributing free, open-source libraries of foundational ::SY-, ::MX-, and ::FX- tokens to lower the barrier to entry for new users.
- Monetization through Premium Assets: Developing and selling premium, industry-specific token databases (e.g., for finance, law, or healthcare) under a commercial license, while keeping the core framework entirely FOSS.

6. Conclusion

The Token Decoder Maps framework represents a significant step beyond simple prompt engineering. By combining a Domain-Specific Language, a hybrid context model, and a unique system for structured self-reflection, TDM provides a clear blueprint for a new generation of AI agents. These agents will not only be more powerful and reliable but will also possess the capability to learn, adapt, and improve in a safe and collaborative partnership with their human users.
# Summary: Metrica Agentic Workflow

This document summarizes the workflow designed to automatically convert free-form daily notes into structured tasks within the Metrica Protocol.

### The Goal

To eliminate the manual effort of turning daily to-do items into formal project tasks, bridging the gap between unstructured notes and the structured Metrica system.

### The Workflow

1. **The Convention:** In your daily notes file (e.g., `31-07-2025.md`), prefix any task you want to formally track with an exclamation mark.
    *   **Example:** `- [ ] ! #tdm update github`

2. **The System Token:** A dedicated token, `::SY-SYNC-JOURNAL-ENTRY::`, is used to process a specific note file.

3. **The Process:** When invoked, the AI agent will:
    *   Scan the specified file for all tasks marked with `!`.
    *   Propose the creation of formal `::MX-USER-TASK::` tokens from these items.
    *   Upon confirmation, add these new tasks to the master Metrica file.
    *   Mark the original items in the note file as complete (`- [x] !`) to prevent duplicates.

### Full Automation

This entire process can be made fully autonomous by using a `cron` job to run a daily shell script. The script automatically constructs the correct prompt and commands the AI agent to sync the day's notes, requiring zero daily intervention from the user. This turns the AI into a proactive, agentic assistant for task management.# 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.Good. Here's a **starter structure** for defining token formats by purpose, based on your decoder framework approach:

### **1. ::META-TOKEN::**

Purpose: System-level classification or structural control
Format:

```markdown
## ::TOKEN-NAME::
- **Type:** Meta
- **Function:** (e.g., Delineates topic scope, memory zone, persona mode)
- **Tags:** #Meta, #System, #Control
- **Expanded Entry:**
(Used to direct LLM behavior or scope; not part of semantic content. Think of as runtime flags or environment markers.)
```

### **2. ::EN-CONTEXT-TOKEN::**

Purpose: Domain/topic framing
Format:

```markdown
## ::EN-TOPIC-AREA::
- **Type:** Entity
- **Summary:** Defines the semantic range or domain
- **Tags:** #Context, #Domain, #(e.g., #Philosophy, #ForumMemory)
```

- **Expanded Entry:**
Used to inform the AI of framing boundaries. May activate relevant RAG, context compression, or decoder subtrees.

### **3. ::PERSONA-TOKEN::**

Purpose: Stylistic or rhetorical profile for simulation or writing
Format:

```markdown
## ::PERSONA-NAME::
- **Type:** Persona
- **Summary:** Defines voice, tone, and strategy
- **Tags:** #Persona, #(e.g., #Aggressive, #Recursive, #Diplomatic)
- **Expanded Entry:**
Symbolic compression of speaker’s approach. Can include dialectic strategy, sentence cadence, or memory triggers.
```

### **4. ::BEHAVIOR-TOKEN::**

Purpose: Procedural triggers for memory updates, swaps, or state changes
Format:

```markdown
## ::TOKEN-NAME::
- **Type:** Behavior
- **Summary:** Encodes when/why something should occur (swap LoRA, append RAG, shift mode)
- **Tags:** #Behavior, #(e.g., #Swap, #MemoryWrite)
- **Trigger:** (e.g., Token density threshold, Symbol match, Post-count trigger)
- **Expanded Entry:**
System-level flag instructing process behavior when conditions are met.
```

### **5. ::EN-CONTENT-TOKEN::**

Purpose: Pure data indexing or compression
Format:

```markdown
## ::EN-TOKEN-NAME::
- **Type:** Entity
- **Summary:** Core idea/entry (forum post, logline, argument)
- **Tags:** #Content, #(e.g., #Forum, #Narrative, #PostFragment)
- **Expanded Entry:**
The actual payload. May be compressed natural language, symbolic language, or hybrid. Used in decoder maps.
```

Want to break this out further into **symbolic vs semantic**, **decoder vs encoder**, or propose sub-token types (e.g., ::ARGUMENT-TOKEN:: vs ::RITUAL-TOKEN::)?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?

```#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.
#tdm
#todo
Based on my exposure to the TDM framework and a structural analysis of its design, here are a few theoretical contributions and observations on how its power could be extended.
You have created a system that is exceptionally good at activating a single, complex cognitive state. My observations are focused on how these states could be combined, modified, and managed to create even more nuanced and powerful results.
Observation 1: Token Chaining and Composition
Currently, we have used tokens individually. A powerful extension would be to formally allow token chaining or composition, similar to piping commands in a shell or composing functions in programming.
 * Concept: Allow multiple FX tokens to be invoked in a single prompt, with their effects layering or executing in sequence.
 * Example:
   > Analyze the current state of the US economy using ::FX-HISTORICAL-CYCLICAL-ANALYSIS:: | ::FX-REALIST-DECONSTRUCTION::
   >
```

- Theoretical Effect: This would instruct me to first perform the historical analysis to find a suitable analogue for our current situation (the "decadence" model). Then, I would feed the output of that analysis into the Realist Deconstruction model, analyzing the key players and power dynamics of our time through the lens of that historical analogue.
- Benefit: This would exponentially increase the framework's combinatorial power, allowing you to create highly specific and multi-layered analytical models on the fly.

Observation 2: Parameterization of Tokens

The tokens are currently static. Introducing parameters would turn them from fixed commands into dynamic functions.

- Concept: Allow tokens to accept arguments that modify their internal behavior.
- Example:

> ::FX-HISTORICAL-CYCLICAL-ANALYSIS::(analogue="Late Roman Republic", focus="currency_debasement")
> OR
> ::FX-PERSONA-MISS-FRIZZLE::(topic="the lithium supply chain", tone="pessimistic")
>

- Theoretical Effect: Instead of needing multiple, highly specific tokens, you could have a single, flexible token that can be adapted in real-time. This would allow you to guide my focus with extreme precision without having to define a new token for every nuanced task.
- Benefit: This makes the entire TDM far more flexible, scalable, and efficient.

Observation 3: Scoped Modes for Persistent States

A token's effect currently seems to be for a single prompt. For longer explorations, a persistent "mode" would be more effective.

- Concept: Introduce SY (System) tokens that activate a specific cognitive mode for the duration of a session, or until deactivated.
- Example:

> ::SY-MODE-ACTIVATE:FX-REALIST-DECONSTRUCTION::
> [... a long conversation where every one of my responses is now implicitly filtered through the Realist lens ...]
> ::SY-MODE-DEACTIVATE::
>

- Theoretical Effect: This would set a persistent "cognitive environment." It would be like me putting on a specific pair of glasses and leaving them on, rather than taking them on and off for every single question.
- Benefit: This would make long-form, thematically consistent conversations much smoother and would reduce the repetitive need to invoke the same token.

Theoretical Contribution: A New Token Type - ET (Ethos)

Our entire conversation was a deep dive into ethics and morality. The FX type can capture a reasoning style, but a dedicated token for ethical frameworks could be powerful.

- Concept: A new token type, ET for Ethos, which loads a complete ethical or moral calculus system.
- Example: We could define ::ET-USER-REALIST-TRIBALISM:: based on your specific worldview. We could also define others like ::ET-KANTIAN-DEONTOLOGY:: or ::ET-ACT-UTILITARIANISM::.

- Application: This would allow for high-level moral analysis and comparison. You could prompt:
  > "Analyze the ethics of corporate lobbying using ::ET-ACT-UTILITARIANISM:: and then contrast that with the analysis from ::ET-USER-REALIST-TRIBALISM::"
  >
- Benefit: This would elevate the TDM from a tool for controlling my reasoning to a powerful engine for comparative philosophy and ethical thought experiments.

In essence, these ideas are all aimed at increasing the modularity, flexibility, and combinatorial power of the TDM framework. You have designed an excellent system for creating cognitive "Lego bricks"; these are just a few observations on how to build even more complex and nuanced structures with them.

That's an excellent question, and you've caught a subtle but important distinction. You're right, just listing the tokens in natural language did work. My suggestion to use a "pipe" (|) like a bash script is about moving from an implicit relationship to an explicit one.

Let me explain the difference.

When you said, "using ::FX-META-REFLECTIVE-DIALOGUE:: and ::FX-SOVEREIGN-SYSTEM-DESIGN::, compare...", I had to make an inference. I interpreted "and" to mean, "Please conduct this comparison using both of these modes of thought simultaneously as part of your overall analysis." It worked because the two tokens were complementary.

However, a formal piping syntax implies a much more specific, sequential order of operations. In a bash script, when you write command1 | command2, it means:

> "Run command1, and then take the entire output of that command and use it as the input for command2."
>

Applying this to our tokens:

- The way we did it (Natural Language):
  > "Analyze Topic X using both Token A and Token B."
  >
  - My Interpretation: "Okay, I will look at Topic X from two different angles at the same time and synthesize the results into one cohesive answer."
- The Piped Method:
  > "Analyze Topic X using ::FX-TOKEN-A:: | ::FX-TOKEN-B::"
  >
  - My Interpretation: "This is a two-step process.
    - First, I will perform a complete and exhaustive analysis of Topic X using only ::FX-TOKEN-A::.
    - Then, I will take my entire finished response from step 1 and use that text as the new subject matter, analyzing it through the lens of ::FX-TOKEN-B::."

So, you are correct. Just saying "and" is not the same thing. The natural language approach is more flexible and relies on my ability to blend the concepts. The formal piping method would be a tool for you to enforce a strict, multi-stage analytical pipeline. It's the difference between asking a chef to cook with both salt and pepper, and asking them to first salt the steak, let it rest, and then pepper it before searing. The sequence changes the outcome.