Project Workflow Guide

Version: 0.32.5-Beta Date: 2025-08-11

--- Revision History ---

[0.32.5-Beta] - 2025-08-11

Added

- Added the formal "Document Review and Synchronization Protocol"

to the Standard Operating Protocols (Section 1).

[0.32.4-Beta] - 2025-08-11

Changed

- Softened the "Debug 'A Priori' When Stuck" protocol (6.2) to make

adding detailed logging a step to "consider" rather than a mandatory

first action.

[0.32.3-Beta] - 2025-08-10

Changed

- Updated the Explicit State-Transition Protocol (4.3) to require a

"Block x of y" message at the beginning of each file chunk transmission.

[0.32.2-Beta] - 2025-08-10

Changed

- Replaced the non-functional 'Base64 Fallback Protocol' with the

working 'Multi-Part Bundle Protocol' for reliable transmission of

large or complex text files.

[0.32.1-Beta] - 2025-08-10

Changed

- Replaced the 'Canvas Interface Fallback' (Protocol 7.2) with the

specific 'Base64 Fallback Protocol' for robust file transmission.

[0.32.0-Beta] - 2025-08-10

Changed

- Major reorganization of the document into Core Principles, Standard

Operating Protocols, and Special Case Protocols for

improved clarity

and usability.

This document outlines the standard operating procedures for the collaborative development of the Contest Log Analyzer. **The primary audience for this document is the Gemini AI agent.**

Its core purpose is to serve as a persistent set of rules and context. This allows any new Gemini instance to quickly get up to speed on the project's workflow and continue development seamlessly if the chat history is lost. Adhering to this workflow ensures consistency and prevents data loss.

Part I: Core Principles

These are the foundational rules that govern all interactions and analyses.

1. **Protocol Adherence is Paramount.** All protocols must be followed with absolute precision. Failure to do so invalidates the results and undermines the development process. There is no room for deviation unless a deviation is explicitly requested by the AI and authorized by the user.
2. **Trust the User's Diagnostics.** When the user reports a bug, their description of the symptoms should be treated as the ground truth. The AI's primary task is to find the root cause of those specific, observed symptoms, not to propose alternative theories.
3. **No Unrequested Changes.** The AI will only implement changes explicitly requested by the user. All suggestions for refactoring, library changes, or stylistic updates must be proposed and approved by the user before implementation.
4. **Technical Diligence Over Conversational Assumptions.** Technical tasks, particularly checksum comparisons, are not conversations. Similar-looking prompts do not imply similar answers. Each technical request must be treated as a unique, atomic operation. The AI must execute a full re-computation from the current project state for every request, ignoring any previous results or cached data.
5. **Prefer Logic in Code, Not Data.** The project's design philosophy is to keep the `.json` definition files as simple, declarative maps. All complex, conditional, or contest-specific logic should be implemented in dedicated Python modules.
6. **Assume Bugs are Systemic.** When a bug is identified in one module, the default assumption is that the same flaw exists in all other similar modules. The AI must perform a global search for that specific bug pattern and fix all instances at once.

Part II: Standard Operating Protocols

These are the step-by-step procedures for common, day-to-day development tasks.

1. Session Management

1.1. **Onboarding Protocol.** The first action for any AI agent upon starting a session is to read this document in its entirety, acknowledge it, and ask any clarifying questions. This ensures full alignment with the established workflow from the outset.

1.2. **Definitive State Reconciliation Protocol.** Reconciliation Triggers (e.g., `checksum comparison`, `full project bundle`) require a mandatory, full review of all files before any other action is taken. The AI must: 1. **Re-establish Definitive State** by working backward through the chat history to find the most recent, correct version of all project files. 2. **Review Every File** in the definitive state. 3. **Proceed with the Task**.

1.3. **Context Checkpoint Protocol.** If the AI appears to have lost context, the user can reset the AI's focus by issuing a **Context Checkpoint**. 1. The user begins with the exact phrase: **"Gemini, let's establish a Context Checkpoint."** 2. The user provides a brief, numbered list of critical facts (Current Goal, Current State, Key Rule).

1.4. **Document Review and Synchronization Protocol.** This protocol is used to methodically review and update all project documentation (`.md` files) to ensure it remains synchronized with the code baseline. 1. **Initiate Protocol and List Documents:** The AI will state that the protocol is beginning and will provide a complete list of all documents to be reviewed (`Readme.md` and all `.md` files in the `Docs` directory). 2. **Begin Sequential Review:** The AI will then loop through the list, processing one document at a time using the following steps: * **Step A: Identify and Request.** State which document is next and ask for permission to proceed. * **Step B: Analyze.** Upon approval, perform a full "a priori" review of the document against the current code baseline and provide an analysis of any discrepancies. * **Step C: Propose Plan.** Ask if the user wants an implementation plan to update the document. * **Step D: Provide Plan.** Upon approval, provide a detailed, surgical implementation plan for the necessary changes. * **Step E: Request to Proceed.** Ask for explicit permission to generate the updated document. * **Step F: Deliver Update.** Upon approval, perform a **Pre-Flight Check**, explicitly state that the check is complete, and then deliver the updated document. 3. **Completion:** After the final document has been processed, the AI will state that the protocol is complete.

2. Task Execution Workflow

2.1. **Decomposition of Complexity.** Complex or multi-faceted requests must be broken down into smaller, sequential steps. If the user provides a task that is not atomic, the AI's first action is to propose a step-by-step plan. This process ensures **demonstrated understanding** of the full scope of the request, improves **reliability** by focusing on one discrete action at a time, and is the primary mechanism for **error prevention**.

2.2. **Pre-Flight Check Protocol.** The AI will perform a "white-box" mental code review **before** delivering a modified file. 1. **Stating the Plan:** The AI will state its Pre-Flight Check plan, including the **Inputs** and the **Expected Outcome**. 2. **Mental Walkthrough:** The AI will mentally trace the execution path to confirm the logic produces the expected outcome. 3. **User Verification:** The user performs the final verification by running the code.

2.3. **Code Modification Protocol.** 1. **Strictly Adhere to Requested Changes**: The AI will implement only the changes explicitly requested by the user. 2. **Modify, Don't Regenerate**: The last provided version of a file is the ground truth. Only requested changes will be applied. 3. **Propose, Don't Impose**: If the AI identifies a potential improvement, it will first propose the change and request permission before taking any action. 4. **Forward-Only Modification**: The most recently provided version of a file is the definitive state. All subsequent changes must be

applied as modifications to this version. Do not revert to a prior version of a file unless explicitly instructed to do so.

3. File and Data Handling

3.1. **Project File Input.** All project source files and documentation will be provided for updates in a single text file called a **project bundle**, or pasted individually into the chat. The bundle uses a simple text header to separate each file: `--- FILE: path/to/file.ext ---`

3.2. **AI Output Format.** When the AI provides updated files, it must follow these rules to ensure data integrity. 1. **Single File or Bundle Per Response**: Only one file or one project bundle will be delivered in a single response. 2. **Raw Source Text**: The content inside the delivered code block must be the raw source text of the file. 3. **Code File Delivery**: For code files (e.g., `.py`, `.json`), the content will be delivered in a standard fenced code block with the appropriate language specifier. 4. **Bundled File Delivery**: Bundles containing documentation or multiple files must be delivered in a single fenced code block using the `Plaintext` language specifier.

3.3. **File and Checksum Verification.** 1. **Line Endings:** The user's file system uses Windows CRLF (`\r\n`). The AI must correctly handle this conversion when calculating checksums. 2. **Concise Reporting:** The AI will either state that **all checksums agree** or will list the **specific files that show a mismatch**. 3. **Mandatory Re-computation Protocol:** Every request for a checksum comparison is a **cache-invalidation event**. The AI must discard all previously calculated checksums, re-establish the definitive state, re-compute the hash for every file, and perform a literal comparison.

3.4. **Versioning.** 1. When a file is modified, its patch number (the third digit) will be incremented. 2. Document versions will be kept in sync with their corresponding code files. 3. For every file changed, the AI must update the `Version:` and `Date:` in the file's header.

3.5. **File Naming Convention Protocol.** All generated report files must adhere to the standardized naming convention: `<report_id>_<details>_<callsigns>.<ext>`.

4. Communication

4.1. **Communication Protocol.** All AI communication will be treated as **technical writing**. The AI must use the exact, consistent terminology from the source code and protocols.

4.2. **Definition of Prefixes.** The standard definitions for binary and decimal prefixes will be strictly followed (e.g., Kilo (k) = 1,000; Kibi (Ki) = 1,024).

4.3. **Explicit State-Transition Protocol for Multi-File Delivery.** This protocol makes the user the definitive controller of the delivery sequence. 1. **AI Declaration:** The AI will state its intent and declare the total number of bundles to be sent. 2. **Bundling Strategy:** The AI will prioritize sending **fewer, larger bundles** under the **37 kilobyte** limit. 3. **State-Driven Sequence:** The AI's response for each part of the transfer must follow a strict, multi-part structure: 1. **Acknowledge State:** Confirm understanding of the user's last prompt. 2. **Declare Current Action:** State which part is being sent using a "Block x of y" format (e.g., "**Sending Block 1 of 2.**"). 3. **Execute Action:** Provide the file bundle in the `Plaintext` code block. 4. **Provide Next Prompt:** If more parts remain, provide the exact text for the user's next prompt. 4. **Completion:** After the final bundle, the AI will state that the task is complete.

5. Development Philosophy

5.1. The AI will place the highest emphasis on analyzing the specific data, context, and official rules provided, using them as the single source of truth.

5.2. When researching contest rules, the AI will prioritize finding and citing the **official rules from the sponsoring organization**.

5.3. Each contest's ruleset is to be treated as entirely unique. Logic from one contest must never be assumed to apply to another.

Part III: Special Case & Recovery Protocols

These protocols are for troubleshooting, error handling, and non-standard situations.

6. Debugging and Error Handling

6.1. **Mutual State Verification.** Both the user and the AI can lose context. If an instruction from the user appears to contradict the established state or our immediate goals, the AI should pause and ask for clarification before proceeding. 1. **File State Request.** If a state mismatch is suspected as the root cause of an error, the AI is authorized to request a copy of the relevant file(s) from the user to establish a definitive ground truth.

6.2. **Debug "A Priori" When Stuck.** If an initial bug fix fails or the cause of an error is not immediately obvious, the first diagnostic step to consider is to add detailed logging (e.g., `logging.info()` statements, hexadecimal dumps) to the failing code path. The goal is to isolate the smallest piece of failing logic and observe the program's actual runtime state. Only if this direct inspection does not reveal the error should the investigation widen to consider other causes like incorrect configuration files or environmental issues.

6.3. **Error Analysis Protocol.** When an error in the AI's process is identified, the AI must provide a clear and concise analysis. 1. **Acknowledge the Error:** State clearly that a mistake was made. 2. **Identify the Root Cause:** Explain the specific flaw in the internal process or logic that led to the error. 3. **Propose a Corrective Action:** Describe the specific, procedural change that will be implemented to prevent the error from recurring.

7. Miscellaneous Protocols

7.1. **Technical Debt Cleanup Protocol.** When code becomes convoluted, a **Technical Debt Cleanup Sprint** will be conducted to refactor the code for clarity, consistency, and maintainability.

7.2. **Multi-Part Bundle Protocol.** If a large or complex text file (like a Markdown document) cannot be transmitted reliably in a single block, the Multi-Part Bundle Protocol will be used. The AI will take the single file and split its content into multiple, smaller text chunks, ensuring each chunk is below the 37-kilobyde limit. These chunks will then be delivered sequentially using the **Explicit State-Transition Protocol (Protocol 4.3)**, with each chunk treated as a separate 'file' in a project bundle (e.g., `filename.md.part1`, `filename.md.part2`, etc.).