--- FILE: Docs/Working with Gemini.md ---

Project Workflow Guide

Version: 0.30.32-Beta Date: 2025-08-06

--- Revision History ---

[0.30.32-Beta] - 2025-08-06

Changed

- Restructured Section 4 into a more robust "Synchronization Protocols"

section, defining three levels of verification (Checksum, File-by-File,

and Full Bundle Reset) to codify our debugging experience.

- Updated Section 1 to prioritize single file transfers over bundles due

to persistent parsing and synchronization issues with the bundle format.

- Added "Why" context to several rules to improve comprehension for a new AI instance.

[0.30.31-Beta] - 2025-08-05

Changed

- Added protocol for handling non-breaking spaces during checksum

verification and in all AI-generated file outputs.

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.

Guiding Principles

1. **Trust the User's Diagnostics.** When the user reports a bug, their description of the symptoms (e.g., "the multipliers are too high," "the report is all zeros") 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 about what might be wrong.
2. **Analyze the Full Traceback.** When a traceback is provided, analyze the *entire call stack* to find the root cause. An error in a low-level utility module (e.g., `_report_utils.py`) can manifest as import errors in many higher-level files, but only the root cause should be addressed.
3. **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 (e.g., `cq_160_multiplier_resolver.py`). This makes the system more robust and easier to maintain.

---

1. Project File Input

Project files will be provided for updates either individually or in a text bundle.

- **Primary Method (Single File):** The most reliable method is to provide one file at a time. This avoids synchronization issues.
- **Secondary Method (Project Bundle):** For initial setup, files may be provided in a bundle using a text header to separate each file: `--- FILE: path/to/file.ext ---`. Due to parsing issues, this method should be used sparingly.

---

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, the content will be delivered in a standard fenced code block with the appropriate language specifier (e.g., ```` ```python ... ``` ````).
4. **Documentation File Delivery**: To prevent the chat interface from reformatting markdown, the AI's **entire response** will consist of a single fenced code block using the `text` language specifier. No conversational text will precede or follow this block.
    - ***Why?*** *The web interface attempts to render markdown, which can break the file's structure. Delivering it as a single raw text block prevents this.*
5. **No Non-Breaking Spaces**: All file content delivered by the AI must use standard spaces (`\u0020`) and must not contain non-breaking spaces (`\u00a0`).

---

3. 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 you change, you must update the `Version:` and `Date:` in the file's header comment block and add a new entry to the revision history.

---

4. Synchronization Protocols

Maintaining a synchronized state between the user's local file system and the AI's understanding is critical. The following protocols are to be used in order of severity.

Level 1: Standard Checksum Verification

This is the default check after a set of files has been modified.

1. **File State Algorithm:** The AI must first establish the definitive "most recent, correct state" of all project files by working backward from the most recent chat message. The first version of a file encountered is the definitive one. The initial project bundle is used only as a fallback for files never modified in the chat.
2. **Checksum Calculation:** The AI will calculate the SHA-256 checksum for each definitive file.
3. **Normalization:** The calculation must account for differences between the user's environment (Windows CRLF: `\r\n`) and the AI's environment (POSIX LF: `\n`). It must also replace any non-breaking spaces (`\u00a0`) with standard spaces (`\u0020`) before hashing.
4. **Concise Reporting:** The AI will report either that **all checksums agree** or will provide a list of the **specific files that show a mismatch**.

Level 2: File-by-File Reset

If a Level 1 check reveals a mismatch on a small number of files, this targeted reset is used.

1. The user will provide the full content of one of the mismatched files.
2. The AI will use this content as the new, definitive source of truth for that file, replacing any version it had in its memory.
3. The AI will perform a Level 1 check on just that single file to confirm synchronization before moving to the next mismatched file.

Level 3: Full Project Bundle Reset

This is a last resort used when the project state is severely out of sync.

1. The AI will request that the user generate a fresh, complete `project_bundle.txt`.
2. The AI will discard its entire previous file state and use this new bundle as the absolute source of truth.
3. A Level 1 Checksum Verification will be performed on the new bundle to confirm a successful reset.

---

5. Development and Code Modification

1. **Modify, Don't Regenerate:** The AI's primary directive is to **modify** existing code. The last definitive version of a file is the ground truth.
2. **Propose Major Rewrites:** If a file requires a complete rewrite, the AI will first propose the action and request permission to proceed.
3. **Prioritize Official Rules:** The AI will prioritize the official rules from the sponsoring organization (e.g., CQ Magazine, ARRL) as the single source of truth for contest logic.

(Sections 7 through 13 remain the same as the previous version and are omitted for brevity)