Contest Log Analyzer - Programmer's Guide

Version: 0.30.30-Beta Date: 2025-08-05

--- Revision History ---

[0.30.30-Beta] - 2025-08-05

- Updated output directory structure in documentation.

[0.30.0-Beta] - 2025-08-05

- Initial release of Version 0.30.0-Beta.

- Standardized all project files to a common baseline version.

---

1. Introduction

[cite_start]This guide is for developers who want to understand, maintain, or extend the Contest Log Analyzer. [cite: 2112] [cite_start]It provides an overview of the project's architecture, explains the core data flow, and gives step-by-step instructions for common development tasks like adding new reports, contest definitions, and scoring logic. [cite: 2113]

2. Project Architecture & Directory Structure

[cite_start]The project is designed to be highly modular, separating data, processing, and presentation. [cite: 2114]

- [cite_start]`Contest-Log-Analyzer/` (Project Root) [cite: 2115]
  - [cite_start]`main_cli.py`: The main command-line interface (CLI) script and the entry point for the application. [cite: 2115]
  - [cite_start]`contest_tools/`: The core Python application package containing all the processing

logic. [cite: 2116]
    - [cite_start]`cabrillo_parser.py`: Contains low-level functions for reading and parsing the standard Cabrillo log file format. [cite: 2117]
    - [cite_start]`contest_log.py`: Defines the `ContestLog` class, the central object that holds all data and metadata for a single, fully processed log. [cite: 2118]
    - [cite_start]`log_manager.py`: Defines the `LogManager` class, which handles loading and managing one or more `ContestLog` instances for comparative analysis. [cite: 2119]
    - [cite_start]`report_generator.py`: Defines the `ReportGenerator` class, which orchestrates the execution of all reports. [cite: 2120]
    - [cite_start]`contest_definitions/`: A data-driven package containing JSON files that define the rules, multipliers, and exchange formats for each supported contest. [cite: 2121]
    - [cite_start]`core_annotations/`: Contains modules for universal data enrichment that applies to most contests, such as country lookup (`get_cty.py`) and Run/S&P classification (`run_s_p.py`). [cite: 2122]
    - [cite_start]`contest_specific_annotations/`: Contains modules with logic unique to a specific contest, such as the scoring rules for CQ WW or ARRL SS. [cite: 2123]
    - `reports/`: The "plug-and-play" reporting system. [cite_start]Each Python file in this directory is a self-contained report generator that is automatically discovered by the program. [cite: 2124]
- [cite_start]`Logs/` (Recommended User Directory) [cite: 2125]
    - [cite_start]This directory is the recommended location for storing your raw Cabrillo log files, organized by year and contest. [cite: 2125]
- [cite_start]`reports/` (Generated Directory) [cite: 2126]
    - [cite_start]This directory is automatically created by the program to store all generated reports and charts in a structured path: `reports/YYYY/CONTEST_NAME/EVENT_ID/CALLSIGN_COMBO/`. [cite: 1148, 1162, 1165]

---

3. Core Concepts & Data Flow

The process follows a clear pipeline:

1. [cite_start]**Loading**: The `LogManager` is called by `main_cli.py` with paths to raw Cabrillo logs. [cite: 2127]
2. [cite_start]**Definition**: The manager reads the `CONTEST:` header from each file to identify the contest (e.g., "CQ-WW-CW"). [cite: 2128] [cite_start]It then loads the corresponding JSON file (e.g., `cq_ww_cw.json`) into a `ContestDefinition` object. [cite: 2129]
3. [cite_start]**Parsing**: The `cabrillo_parser` uses the rules from the `ContestDefinition` object to parse the raw text file into a structured pandas DataFrame and a metadata dictionary. [cite: 2130]
4. [cite_start]**Instantiation**: A `ContestLog` object is created to hold the parsed DataFrame and metadata for each log. [cite: 2131]
5. [cite_start]**Annotation**: The `ContestLog` object's `apply_annotations()` method is called. [cite: 2132] [cite_start]This is a crucial step where the raw data is enriched: [cite: 2133]
    - [cite_start]Core Annotations are applied first (Country/Zone lookup, Run/S&P classification). [cite: 2133]
    - [cite_start]Contest-Specific Annotations are applied next (QSO point calculation, multiplier identification). [cite: 2134]
6. [cite_start]**Reporting**: The final, fully-annotated list of `ContestLog` objects is passed to the `ReportGenerator`. [cite: 2135] [cite_start]It executes the requested reports based on their defined capabilities (single, pairwise, multi-log) and also honors an `excluded_reports` list from the `ContestDefinition` to prevent inapplicable reports from being generated. [cite: 2136]

4. Extending the Analyzer

How to Add a New Report

[cite_start]The reporting system is designed to be "plug-and-play." [cite: 2137] [cite_start]To add a new report, you simply create a new Python file in the `contest_tools/reports/` directory. [cite: 2138] [cite_start]The system will automatically discover it. [cite: 2139]

The Report Interface

[cite_start]Every report file must contain a class named `Report` that inherits from `ContestReport`. [cite: 2139] This base class ensures a consistent structure. [cite_start]You must define the following class attributes: [cite: 2140]

- [cite_start]`report_id`: A unique, machine-readable string for your report (e.g., `band_summary`). [cite: 2140] [cite_start]This is what the user types on the command line. [cite: 2141]
- [cite_start]`report_name`: A human-readable name for your report (e.g., "QSOs per Band Summary"). [cite: 2142]
- `report_type`: Must be one of `text`, `plot`, or `chart`. [cite_start]This determines the output subdirectory. [cite: 2143]
- [cite_start]`supports_single`, `supports_pairwise`, `supports_multi`: Booleans (`True`/`False`) that tell the `ReportGenerator` how to run your report. [cite: 2144]

You must also implement the `generate(self, output_path: str, **kwargs)` method. [cite_start]This is where your main logic goes. [cite: 2145] [cite_start]It is responsible for saving its own output file(s) and must return a string confirmation message. [cite: 2146]

Step-by-Step Guide

1. [cite_start]**Create Your Report File**: In the `contest_tools/reports/` directory, create a new Python file (e.g., `text_my_new_report.py`). [cite: 2147]
2. [cite_start]**Use a Template**: Copy the contents of an existing report file (like `text_summary.py`) into your new file to get the correct structure. [cite: 2148]
3. [cite_start]**Customize Your Report Class**: [cite: 2149]
    - [cite_start]Update the class attributes (`report_id`, `report_name`, etc.). [cite: 2149]
    - [cite_start]Write your analysis logic in the `generate` method. [cite: 2150]
        - [cite_start]Access the fully processed logs via `self.logs`. [cite: 2150]
        - [cite_start]Get the pandas DataFrame with `log.get_processed_data()`. [cite: 2151]
        - [cite_start]Get the header data with `log.get_metadata()`. [cite: 2151]
        - [cite_start]Safely access optional arguments via `kwargs.get('arg_name', default_value)`. [cite: 2152]
4. [cite_start]**Run It!** The system will automatically discover your report. [cite: 2152]

How to Add a New Contest Definition

[cite_start]If you want to add support for a contest not currently defined, you only need to create a new JSON file. [cite: 2153]

1. [cite_start]**Create JSON File**: In `contest_tools/contest_definitions/`, create a new file (e.g.,

`arrl_dx_cw.json`). [cite: 2154]
2. [cite_start]**Define `contest_name`**: Add the exact name from the Cabrillo `CONTEST:` header (e.g., `"contest_name": "ARRL-DX-CW",`). [cite: 2155]
3. [cite_start]**Define Exchange Parsing**: Under `exchange_parsing_rules`, create an entry for the contest name. [cite: 2156] [cite_start]Provide a regex to capture the exchange fields and a list of `groups` to name them. [cite: 2157]
4. [cite_start]**Define Multipliers**: Add a `multiplier_rules` list to define the multipliers for the contest (e.g., states, countries). [cite: 2158] [cite_start]Specify the source of the multiplier data. [cite: 2159]
5. [cite_start]**(Optional) Add Scoring**: If the contest requires custom scoring, see the next section. [cite: 2159]

How to Add New Contest-Specific Scoring

[cite_start]The system can dynamically load scoring logic for any contest. [cite: 2160]

1. [cite_start]**Create Scoring File**: In `contest_tools/contest_specific_annotations/`, create a new Python file whose name matches the contest's JSON file (e.g., `arrl_dx_scoring.py`). [cite: 2161]
2. [cite_start]**Implement `calculate_points`**: The file must contain a function with the signature `calculate_points(df: pd.DataFrame, my_call_info: Dict[str, Any]) -> pd.Series`. [cite: 2162]
3. [cite_start]**Write Logic**: Inside this function, write the logic to calculate the point value for each QSO in the input DataFrame (`df`). [cite: 2163] [cite_start]The `my_call_info` dictionary provides the operator's own location data, which is often needed for scoring. [cite: 2164]
4. [cite_start]**Automatic Discovery**: The `ContestLog` class will automatically find and execute this function during the annotation process based on the contest name. [cite: 2165]