

## # Contest Log Analyzer - Programmer's Guide

**\*\*Version: 0.37.0-Beta\*\***

**\*\*Date: 2025-08-18\*\***

---

### ### --- Revision History ---

## [0.37.0-Beta] - 2025-08-18

### Added

# - Added a new "Regression Testing" section to describe the

# run\_regression\_test.py script and its methodology.

# - Added the `enable\_adif\_export` key to the JSON Quick Reference table.

### Changed

# - Updated the document's version to align with other documentation.

## [0.36.7-Beta] - 2025-08-15

### Changed

# - Updated the CLI arguments list to be complete.

# - Updated the JSON Quick Reference table to include all supported keys.

## [0.35.23-Beta] - 2025-08-15

### Changed

# - Updated the "Available Reports" list and the `--report` argument

# description to be consistent with the current codebase.

## [0.32.15-Beta] - 2025-08-12

### Added

# - Added documentation for the new "Custom Parser Module" plug-in pattern.

### Changed

# - Replaced nested markdown code fences with ``` placeholder.

## [1.0.1-Beta] - 2025-08-11

### Changed

# - Updated CLI arguments, contest-specific module descriptions, and the

# report interface to be fully consistent with the current codebase.

## [1.0.0-Beta] - 2025-08-10

### Added

# - Initial release of the Programmer's Guide.

# ---

## ## Introduction

This document provides a technical guide for developers (both human and AI) looking to extend

\* **\*\*Data-Driven:\*\*** The behavior of the analysis engine is primarily controlled by data, not c

\* **\*\*Extensible:\*\*** The application is designed with a "plugin" architecture. New reports and c

\* **\*\*Convention over Configuration:\*\*** This extensibility relies on convention. The dynamic dis

---

## ## Core Components

### ### Command-Line Interface (`main\_cli.py`)

This script is the main entry point for running the analyzer.

\* **\*\*Argument Parsing:\*\*** It uses Python's `argparse` to handle command-line arguments. Key arg

\* `log\_files`: A list of one or more log files to process.

\* `--report`: Specifies which reports to run. This can be a single `report\_id`, a comma-s

\* `--verbose`: Enables `INFO`-level debug logging.

\* `--include-dupes`: An optional flag to include duplicate QSOs in report calculations.

\* `--mult-name`: An optional argument to specify which multiplier to use for multiplier-s

\* `--metric`: An optional argument for difference plots, specifying whether to compare `q

\* `--debug-data`: An optional flag to save the source data for visual reports to a text f

\* **\*\*Report Discovery:\*\*** The script dynamically discovers all available reports by inspecting

### ### Logging System (`Utils/logger\_config.py`)

The project uses Python's built-in `logging` framework for console output.

\* **\*\*`logging.info()`:** Used for verbose, step-by-step diagnostic messages. These are only di

\* **\*\*`logging.warning()`:** Used for non-critical issues the user should be aware of (e.g., ig

\* **\*\*`logging.error()`:** Used for critical, run-terminating failures (e.g., a file not found

### ### Regression Testing (`run\_regression\_test.py`)

The project includes an automated regression test script to ensure that new changes do not br

```
* **Workflow**: The script follows a three-step process:
1.  **Archive**: It archives the last known-good set of reports by renaming the existing
2.  **Execute**: It runs a series of pre-defined test cases from a `regressiontest.bat` f
3.  **Compare**: It performs a `diff` comparison between the newly generated text reports
* **Methodology**: This approach focuses on **data integrity**. Instead of comparing images o
---
## How to Add a New Report

### The Report Interface
All reports must be created as `.py` files in the `contest_tools/reports/` directory. For the

1.  The file must contain a class named **`Report`**.
2.  This class must inherit from the **`ContestReport`** base class.
3.  The class must define the following required attributes:

| Attribute | Type | Description |
| --- | --- | --- |
| `report_id` | `str` | A unique, machine-friendly identifier (e.g., `score_report`). Used in
| `report_name` | `str` | A human-friendly name for the report (e.g., "Score Summary"). |
| `report_type` | `str` | The category of the report. Currently `text`, `plot`, `chart`, or `
| `supports_single` | `bool` | `True` if the report can be run on a single log. |
| `supports_multi` | `bool` | `True` if the report can be run on multiple logs (non-comparati
| `supports_pairwise` | `bool` | `True` if the report compares exactly two logs. |

4.  The class must implement a `generate(self, output_path: str, **kwargs) -> str` method. Th

### Dynamic Discovery
As long as a report file is in the `contest_tools/reports` directory and its class is named `

### Helper Functions and Factoring (`_report_utils.py`)
The `contest_tools/reports/_report_utils.py` module contains common helper functions. The phi

* **Keep it Self-Contained**: If a piece of logic is highly specific to a single report and u
* **Factor it Out**: If a function or component (like a chart style or data preparation step)

### Boilerplate Example
Here is a minimal "Hello World" report.

...

# contest_tools/reports/text_hello_world.py
from _report_interface import ContestReport

class Report(ContestReport):
    report_id = "hello_world"
    report_name = "Hello World Report"
    report_type = "text"
    supports_single = True

    def generate(self, output_path: str, **kwargs) -> str:
        log = self.logs[0]
        callsign = log.get_metadata().get('MyCall', 'N/A')
        report_content = f"Hello, {callsign}!"
        # In a real report, you would save this content to a file.
        print(report_content)

        return f"Report '{self.report_name}' generated successfully."
...

---
## How to Add a New Contest

Adding a new contest can range from simple (creating a new `.json` file) to complex (extendin
### JSON Quick Reference
The primary way to add a contest is by creating a new `.json` file in the `contest_tools/cont

| Key | Description | Example Value |
| --- | --- | --- |
```

| `contest\_name` | The official name from the Cabrillo `CONTEST:` tag. | `"CQ-WW-CW"` |

| `dupe\_check\_scope` | Determines if dupes are checked `per\_band` or across `all\_bands`. | `all\_bands` |

| `exchange\_parsing\_rules` | An object containing regex patterns to parse the exchange portion of the log. | `{ "exchange": "RST", "exchange\_field": "RcvdExchangeField" }` |

| `multiplier\_rules` | A list of objects defining the contest's multipliers. | `[ { "name": "MyMults", "source\_column": "RcvdExchangeField", "value\_column": "Mult1", "totaling\_method": "once\_per\_log" } ]` |

| `score\_formula` | Scoring method. Can be `qsos\_times\_mults` or `points\_times\_mults`. | `qsos\_times\_mults` |

| `multiplier\_report\_scope` | Determines if mult reports run `per\_band` or `per\_mode`. | `per\_mode` |

| `excluded\_reports` | A list of `report\_id` strings to disable for this contest. | `[ "point\_report" ]` |

| `operating\_time\_rules` | Defines on-time limits for the `score\_report`. | `{ "single\_op\_max": 10, "multi\_op\_max": 30 }` |

| `mults\_from\_zero\_point\_qsos` | `True` if multipliers count from 0-point QSOs. | `true` |

| `enable\_adif\_export` | `True` if the log should be exported to an N1MM-compatible ADIF file. | `true` |

| `valid\_bands` | A list of bands valid for the contest. | `[ "160M", "80M", "40M" ]` |

| `contest\_period` | Defines the official start/end of the contest. | `{ "start\_day": "Saturday", "start\_time": "00:00", "end\_day": "Sunday", "end\_time": "23:59" }` |

| `custom\_parser\_module` | \*Optional.\* Specifies a module to run for complex, asymmetric parsing. | `my\_contest\_parser` |

| `custom\_multiplier\_resolver` | \*Optional.\* Specifies a module to run for complex multiplier resolution. | `my\_contest\_resolver` |

| `contest\_specific\_event\_id\_resolver` | \*Optional.\* Specifies a module to create a unique event ID. | `my\_contest\_event\_resolver` |

| `scoring\_module` | \*Implied.\* The system looks for a `[contest\_name]\_scoring.py` file with a `calculate\_score` function.

### ### Basic Guide: Creating a New Contest Definition

1. Create a new `.json` file in `contest\_tools/contest\_definitions/`.
2. Define the `contest\_name` to match the Cabrillo logs.
3. Define the `exchange\_parsing\_rules`. If the exchange can have multiple valid formats, you can use `exchange` and `exchange\_field` to specify the format.
4. Define the `multiplier\_rules`. For simple multipliers, you can use `source\_column` to specify the column to parse for multipliers.

### ### Boilerplate Example

```
...
{
  "_filename": "contest_tools/contest_definitions/my_contest.json",
  "_version": "1.0.0-Beta",
  "_date": "2025-08-10",
  "contest_name": "MY-CONTEST-CW",
  "dupe_check_scope": "per_band",
  "exchange_parsing_rules": {
    "MY-CONTEST-CW": {
      "regex": "((\\d{3})\\s+(\\w+))",
      "groups": [ "RST", "RcvdExchangeField" ]
    }
  },
  "multiplier_rules": [
    {
      "name": "MyMults",
      "source_column": "RcvdExchangeField",
      "value_column": "Mult1",
      "totaling_method": "once_per_log"
    }
  ]
}
...
```

### ### Advanced Guide: Extending Core Logic

If a contest requires logic that cannot be defined in JSON, you can extend the Python code. Create the following files in the `contest\_tools` directory:

- \* \*\*Custom Parser Module:\*\* Create a file (e.g., `my\_contest\_parser.py`) containing a `parse\_log` function.
- \* \*\*Custom Multiplier Resolver:\*\* Create a file (e.g., `my\_contest\_resolver.py`) containing a `resolve\_multipliers` function.
- \* \*\*Event ID Resolver:\*\* Create a file (e.g., `my\_contest\_event\_resolver.py`) with a `resolve\_event\_id` function.
- \* \*\*Scoring Module:\*\* Create a file named `my\_contest\_cw\_scoring.py` containing a `calculate\_score` function.
- \* \*\*Multiplier Calculation Module:\*\* Create a file (e.g., `my\_contest\_mult\_calc.py`) with a `calculate\_mult` function.