```markdown
# 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`. | `"
| `exchange_parsing_rules` | An object containing regex patterns to parse the exchange portio
| `multiplier_rules` | A list of objects defining the contest's multipliers. | `[ { "name": "
| `score_formula` | Scoring method. Can be `qsos_times_mults` or `points_times_mults`. | `"po
| `multiplier_report_scope`| Determines if mult reports run `per_band` or `per_mode`. | `"per
| `excluded_reports`| A list of `report_id` strings to disable for this contest. | `[ "point_
| `operating_time_rules`| Defines on-time limits for the `score_report`. | `{ "single_op_max_
| `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
| `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": "Saturd
| `custom_parser_module` | *Optional.* Specifies a module to run for complex, asymmetric pars
| `custom_multiplier_resolver` | *Optional.* Specifies a module to run for complex multiplier
| `contest_specific_event_id_resolver` | *Optional.* Specifies a module to create a unique ev
| `scoring_module` | *Implied.* The system looks for a `[contest_name]_scoring.py` file with

### 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
4.  Define the `multiplier_rules`. For simple multipliers, you can use `"source_column"` to c

### 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. C
* **Custom Parser Module:** Create a file (e.g., `my_contest_parser.py`) containing a `parse_
* **Custom Multiplier Resolver:** Create a file (e.g., `my_contest_resolver.py`) containing a
* **Event ID Resolver:** Create a file (e.g., `my_contest_event_resolver.py`) with a `resolve_
* **Scoring Module:** Create a file named `my_contest_cw_scoring.py` containing a `calculate_
* **Multiplier Calculation Module:** Create a file (e.g., `my_contest_mult_calc.py`) with a f