Lora Scanner API Documentation

The Lora Scanner is a Flask web application integrated with SocketIO for real-time communication, primarily focused on managing and processing serial data from various frequencies. The application's core functionality revolves around connecting to, managing, and disconnecting from serial ports, corresponding to three distinct frequency bands: 433 MHz, 868 MHz, and 915 MHz. This is achieved through routes that allow users to attach or delete serial connections for each frequency, with the backend handling the setup and teardown of these connections. Additionally, the application includes functionality to transmit data over these serial connections, facilitated by POST routes for each frequency.

The app features a real-time data processing aspect, where it reads from serial ports in separate threads, extracts and processes the data using regular expressions, and stores this data in a structured format. The processed data includes signal strength and decoded values from raw serial data, which are then appended to a survey data structure for further use. SocketIO is employed to emit this serial data to the frontend, enabling real-time data streaming and display. The application also includes routes for transmitting data via the serial ports, and a route to check the status of each serial port.

The Flask application provides several web pages for user interaction, rendered through specific routes. These include the main dashboard, analysis, survey, and tracking pages, each serving different aspects of the application, like displaying initial data from serial buffers and the global dataframe. The app also features a WebSocket event for handling connections, ensuring initial serial data is emitted to connected clients. Overall, the `app.py` file presents a comprehensive web application designed for real-time serial data communication and processing, with a user-friendly interface for monitoring and interacting with connected devices across different frequencies.

Each route description includes the HTTP method used (GET or POST), a brief description of what the route does, the expected parameters (if any), and the format of the response. This documentation is intended to help users understand how to interact with the application's API.

---

1. Homepage

- **Route:** `/`
- **Method:** `GET`
- **Description:** Renders the homepage.
- **Parameters:** None
- **Returns:** HTML content of the `index.html` page.

2. Analysis Page

- **Route:** `/analysis`
- **Method:** `GET`
- **Description:** Renders the analysis page with initial data from serial buffers.
- **Parameters:** None
- **Returns:** HTML content of the `analysis.html` page with initial serial buffer data.

3. Survey Page

- **Route:** `/survey`
- **Method:** `GET`
- **Description:** Renders the survey page.
- **Parameters:** None
- **Returns:** HTML content of the `survey.html` page with data from `global_dataframe`.

4. Tracking Page

- **Route:** `/tracking`
- **Method:** `GET`
- **Description:** Renders the tracking page with initial data.
- **Parameters:** None
- **Returns:** HTML content of the `tracking.html` page with initial serial buffer data.

5. Attach Serial Port for 433 MHz

- **Route:** `/attach_serial_433`
- **Method:** `GET`
- **Description:** Attaches a serial port for 433 MHz frequency based on user input.
- **Parameters:** `user_input` (string): Serial port to connect.
- **Returns:** JSON object with the result of the operation.

6. Delete Serial Port for 433 MHz

- **Route:** `/delete_serial_433`
- **Method:** `GET`
- **Description:** Disconnects the serial port for 433 MHz.
- **Parameters:** None
- **Returns:** JSON object indicating port disconnection.

7. Attach Serial Port for 868 MHz

- **Route:** `/attach_serial_868`
- **Method:** `GET`
- **Description:** Attaches a serial port for 868 MHz frequency based on user input.
- **Parameters:** `user_input` (string): Serial port to connect.
- **Returns:** JSON object with the result of the operation.

8. Delete Serial Port for 868 MHz

- **Route:** `/delete_serial_868`
- **Method:** `GET`
- **Description:** Disconnects the serial port for 868 MHz.
- **Parameters:** None
- **Returns:** JSON object indicating port disconnection.

9. Attach Serial Port for 915 MHz

- **Route:** `/attach_serial_915`
- **Method:** `GET`
- **Description:** Attaches a serial port for 915 MHz frequency based on user input.
- **Parameters:** `user_input` (string): Serial port to connect.
- **Returns:** JSON object with the result of the operation.

10. Delete Serial Port for 915 MHz

- **Route:** `/delete_serial_915`
- **Method:** `GET`
- **Description:** Disconnects the serial port for 915 MHz.
- **Parameters:** None
- **Returns:** JSON object indicating port disconnection.

11. Transmit Data at 433 MHz

- **Route:** `/transmit433`
- **Method:** `POST`
- **Description:** Transmits data via the 433 MHz serial port.
- **Parameters:** JSON payload with `user_input` (string).
- **Returns:** JSON object with the status of the transmission.

12. Transmit Data at 868 MHz

- **Route:** `/transmit868`
- **Method:** `POST`
- **Description:** Transmits data via the 868 MHz serial port.
- **Parameters:** JSON payload with `user_input` (string).
- **Returns:** JSON object with the status of the transmission.

13. Transmit Data at 915 MHz

- **Route:** `/transmit915`
- **Method:** `POST`
- **Description:** Transmits data via the 915 MHz serial port.
- **Parameters:** JSON payload with `user_input` (string).
- **Returns:** JSON object with the status of the transmission.

14. Check Serial Port Status

- **Route:** `/checkSer`
- **Method:** `GET`
- **Description:** Checks the status of a specified serial port.
- **Parameters:** `port` (string): Serial port to check.
- **Returns:** JSON object with the port status.

15. Get Table Data

- **Route:** `/get_table_data`
- **Method:** `GET`
- **Description:** Retrieves data for display in a table format.
- **Parameters:** None
- **Returns:** JSON object with survey data.

16. WebSocket Connect

- **Socket Event:** `connect`
- **Description:** Handles WebSocket connection and emits initial serial data.
- **Parameters:** None
- **Returns:** Emits initial serial data for each port.