Data Logger GUI - Developer Documentation

This document serves as an informal guide to the Python-based Data Logger GUI. It covers the code structure, important modules and methods, threading and serial communication, GUI components, plotting, data processing, and how everything ties together.

Overview

The Data Logger GUI is a desktop application that:

- **Manages serial communication:**
  Connects to a device over a serial port (using PySerial) to receive sensor data.
- **Logs sensor data to CSV:**
  Captures and stores incoming CSV-formatted data from the device for later analysis.
- **Real-time plotting:**
  Displays live sensor data using Matplotlib integrated into a Tkinter GUI.
- **Interactive controls:**
  Provides buttons and menus for starting/stopping data logging, refreshing ports, setting sampling rates, and sending commands (like configuration uploads and heater profile requests).
- **Data processing and storage:**
  Uses Pandas to store logged data in a DataFrame with thread locks for safety.

Key Dependencies

The code relies on several libraries:

- **Standard Library:**
  - `os`, `threading`, `time`, `csv`, `datetime`, `queue`, `io`
    For file operations, multithreading, timing, CSV parsing, and string handling.
- **Tkinter:**
  For building the GUI components including buttons, dialogs, and text widgets.
- **PySerial:**
  Handles serial port communication for connecting to and interacting with the external device.
- **Matplotlib:**
  Used for plotting sensor data in real-time. The TkAgg backend integrates plots into the Tkinter application.
- **Pandas:**
  Manages data storage and manipulation in a DataFrame format, making it easy to filter and process

sensor readings.

Code Structure

The entire application is structured around the main class `DataLoggerGUI`. Here's a breakdown of the core components:

1. **Initialization and Global Setup**

- **Imports:**
  The top of the file loads all necessary modules from the standard library and third-party packages.
- **Tkinter and Matplotlib Configuration:**
  - The `TkAgg` backend is set for Matplotlib to ensure it works with the Tkinter GUI.
  - A root window is created and passed to the main class.

2. **DataLoggerGUI Class**

This class is the heart of the application and manages the GUI, serial communication, data logging, and plotting.

Constructor (`__init__`)

- **Window Setup:**
  Configures the main window (title, size, minimum dimensions).
- **State Variables:**
  Initializes flags for logging, serial connection, heater profile response, and a data queue for thread communication.
- **Data Storage:**
  Creates an empty Pandas DataFrame with predefined CSV column headers to store logged data. Uses a thread lock (`self.data_lock`) to ensure safe access.
- **GUI Variables:**
  Sets up Tkinter variables (e.g., for Label Tag, Heater Profile display, selected parameter for plotting).
- **Component Initialization:**
  Calls `create_widgets()` to build the GUI and `create_plot()` for initializing the plot. Also schedules periodic processing of incoming data via `process_queue()`.

GUI Construction (`create_widgets`)

- **Layout Management:**
  Uses grid layout to arrange frames for:
  - Serial Connection Controls
  - Controller Panel (start/stop logging, send commands)
  - Plotting Area (sensor selection, live plot, file path display)

    - Data Transferred Display (scrollable text box for real-time data)
    - Status Bar (application status messages)

- **Interactive Elements:**
  Dropdowns, buttons, checkboxes, and entry fields are created for:

    - Selecting serial ports and refreshing the list.
    - Starting and stopping logging.
    - Sending commands (e.g., `GETHEAT`, `START_CONFIG_UPLOAD`).
    - Configuring the sampling rate and time window for the plot.

Serial Communication Methods

- **Connecting/Disconnecting:**

    - `connect_serial()`: Opens the serial port with a specified baud rate (115200) and starts a background thread (`read_serial_data`) to continuously read incoming data.
    - `disconnect_serial()`: Safely closes the serial port and stops the reading thread.

- **Command Handling:**

    - `send_command()`: Sends commands (e.g., "START", "STOP", "MS_") over the serial connection.
    - `send_new_config()`: Reads a configuration file (JSON), splits it line by line, and sends it over serial using a start-content-end protocol.

- **Retrieving Heater Profiles:**

    - `get_heater_profiles()`: Sends a `GETHEAT` command to request heater profile data and prepares to collect the response in a buffer.

Data Logging and Processing

- **Logging Management:**

    - `start_logging()`: Prompts the user for a file path, opens a CSV file for writing, writes the header row, and starts sending the "START" command.
    - `stop_logging()`: Stops logging by sending a "STOP" command, closing the file, and resetting UI elements.

- **Serial Data Reading:**

    - `read_serial_data()`: Runs in a separate thread; continuously reads from the serial port, decodes lines, and enqueues them for processing.
    - Handles heater profile responses and standard CSV data lines separately.

- **Data Parsing and Storage:**

    - `parse_and_store_data()`: Parses CSV-formatted lines into a structured format, updates the Pandas DataFrame, and refreshes the Label Tag and Heater Profile displays. Ensures only data within the selected time window is kept.

- **Queue Processing:**
  - `process_queue()`: Periodically checks the thread-safe queue for new data lines, processes them, updates the GUI display, and triggers plot updates.

Plotting and Data Visualization

- **Real-time Plot Updates:**
  - `update_plot()`: Redraws the plot with new sensor data. Uses the selected parameter (e.g., Temperature, Pressure) and the chosen sensors from the checkboxes.
  - Automatically adjusts the time window (controlled by a dropdown) and formats the x-axis using Matplotlib's date formatting.
- **Dynamic Controls:**
  - Users can change the selected parameter and time window, which triggers an immediate plot update.

Utility Methods

- **Status and Confirmation:**
  - `update_status()`: Updates the status bar at the bottom of the GUI.
  - `confirm_action()`: Displays confirmation dialogs for potentially disruptive actions (like stopping logging).
- **Heater Profile Formatting:**
  - `show_heater_profiles()` and `format_heater_profiles()`: When heater profile data is received, these methods display the formatted information in a pop-up window, making it easier to read.

Application Lifecycle

- **Window Closing:**
  - `on_closing()`: Ensures that logging is properly stopped and the application exits cleanly when the window is closed.

Main Function

- **Entry Point (`main()`):**
  Creates the main Tkinter window, initializes the `DataLoggerGUI` class, and starts the Tkinter event loop. The application is launched when running the script directly.

Final Notes

- **Threading & Synchronization:**
  The design carefully uses a separate thread for reading serial data and a thread-safe queue (`Queue`)

for passing data back to the main thread. A threading lock protects the Pandas DataFrame from concurrent access issues.

- **User Experience:**
  The GUI is designed to be user-friendly, with feedback provided through the status bar, pop-up dialogs, and real-time data displays. It handles error conditions gracefully, ensuring that issues like lost serial connections or file errors are promptly communicated to the user.

- **Extensibility:**
  With clear separation between the GUI, serial communication, and data processing, developers can easily extend the code to add more functionality (e.g., additional commands, advanced data analysis, or enhanced visualization).
