**📘 Data Logger System**

**Setup Instructions for Raspberry Pi**

✅ **Prerequisites**

* Raspberry Pi is fully set up with a working Linux OS.
* Internet access (for package installation).
* logger.py and data\_logger.service files
* USB serial device (e.g., ESP32-C3) is connected and working.

### ✅ ****Recommended OS: Raspberry Pi OS Lite (Legacy, 32-bit)****

* Based on Debian **Bullseye**
* No desktop environment (CLI only)
* It is extremely lightweight, has good community support, is compatible with Python and USB serial libraries for the project, is easy to set up scripts to run on boot.

**🔧 1. Prepare the File System**

1. Open a terminal on your Raspberry Pi.
2. Create the main project folder:

mkdir -p /home/Username/data\_logger/LOGS

1. Place your logger.py script inside /home/Username/data\_logger/
2. Make sure the script is executable:

chmod +x /home/Username/data\_logger/logger.py

**📦 2. Install Python and Dependencies**

Make sure Python 3 and required libraries are installed:

sudo apt update

sudo apt install -y python3 python3-pip

pip3 install pyserial

**🧪 3. Test the Script Manually**

Run the script once manually to confirm it works:

cd /home/Username/data\_logger

python3 logger.py

🔍 Check that:

* A log file is created in /home/Username/data\_logger/LOGS/
* No errors are printed
* Serial data is being saved

Press Ctrl+C to stop the script.

**⚙️ 4. Install and Enable the Systemd Service**

1. Move the data\_logger.service file to the systemd directory:

/etc/systemd/system/data\_logger.service

1. Reload systemd daemon to recognize the new service:

sudo systemctl daemon-reexec

sudo systemctl daemon-reload

1. Enable the service so it starts on boot:

sudo systemctl enable data\_logger.service

1. Start the service immediately (for testing):

sudo systemctl start data\_logger.service

**🧩 5. Verify the Service Is Running**

Check if the service is active:

systemctl status data\_logger.service

If active, you should see "Active: active (running)".

If there's an issue, inspect the logs:

journalctl -u data\_logger.service -e

**🔁 6. Reboot to Confirm Autostart**

Reboot the Pi:

sudo reboot

After boot, check again:

systemctl status data\_logger.service

Check if new logs are being generated in:

ls /home/Username/data\_logger/LOGS/

**🧠** Program Logic: logger.py

**🔍 Goal**

The script monitors for an ESP32 device connected via USB. When the device is detected, it logs incoming serial data into a timestamped text file. If the device disconnects, the session is closed gracefully. The script runs in an infinite loop to monitor reconnects and log each session.

**📁 Configuration and Setup**

**Constants:**

* VENDOR\_ID and MANUFACTURER\_NAME: Used to uniquely identify the ESP32.
* LOG\_DIR: Where log files are saved.
* ISSUE\_LOG\_FILE: Where internal script errors are logged.
* RETRY\_DELAY, SESSION\_GRACE\_PERIOD, GRACE\_CHECK\_INTERVAL: Control timing and reconnection behavior.
* BAUD\_RATE: Serial communication speed.
* POST\_DISCONNECT\_FLUSH\_DELAY: A delay after disconnect to allow final data flushing (though currently ineffective).

**🧩 Core Components**

**1. log\_issue(message)**

Appends a timestamped error or warning message to data\_logger\_issues\_log.txt.

**2. find\_esp32\_port()**

Scans all available serial ports using pyserial.

* Filters ports by:
  + Matching port.vid with VENDOR\_ID
  + MANUFACTURER\_NAME in port.manufacturer
* Returns the port.device name (e.g., /dev/ttyUSB0) if found, else None.

**3. start\_logging\_session(port\_name)**

When a compatible ESP32 is detected:

* Generates a filename based on the current date and time.
* Opens the serial port and the log file.
* Continuously:
  + Reads all available bytes from the serial port (ser.in\_waiting)
  + Decodes and writes the data to the log file
  + Flushes the file buffer
* If the device path disappears (device unplugged):
  + Waits briefly to capture remaining data
  + Then exits the session loop

Exceptions during logging are caught and written to the issue log.

**4. main()**

The infinite loop that manages session control and reconnection logic:

* Ensures the log directory exists.
* Loops:
  + Calls find\_esp32\_port()
  + If the device is found:
    - If a session was recently active and still within the grace period:
      * Treat it as the same session and continue
    - Else:
      * Start a new logging session
  + If no device is detected:
    - If a session was active:
      * Start a grace period (wait for possible reconnect)
    - If the grace period expires:
      * Mark the session as inactive

The loop checks for device changes every GRACE\_CHECK\_INTERVAL seconds.

**🧭 High-Level Flow**

1. Wait for ESP32 device to appear.
2. When detected → start a logging session.
3. Log serial data to a timestamped text file.
4. If device disconnects:
   * Allow a grace period for reconnects.
   * If reconnects quickly → continue logging.
   * If not → end session and wait again.