

Simulated Sensor Data Logger

Tools: C/C++, Python, File I/O, Virtual Serial Terminal (Tera Term, PuTTY, etc.)

Skills: Data simulation, file handling, serial communication basics, embedded software logic

Introduction

Sensor data logging is a key part of embedded systems and IoT development. In this project, students will simulate sensor behavior entirely in software, generating synthetic data for temperature, humidity, and light intensity. The goal is to structure, timestamp, and store or transmit this data as would be done in a real embedded system, without requiring any physical sensors or hardware components.

Objectives

- Understand how to emulate real-world sensors in software.
 - Build structured logs of sensor data using file I/O or virtual serial communication.
 - Implement periodic data collection and processing.
 - Document configuration and design decisions in an embedded-style project.
-

Key Features

1. Sensor Simulation

Simulated sensors generate fake but realistic data values using software logic. Instead of connecting real hardware, you'll use mathematical models or random number functions to mimic the behavior of physical sensors:

- **Temperature Sensor:** Generate values in a common ambient range (e.g., 20°C to 40°C). You can use a random function with slight fluctuations between each reading to simulate environmental changes.
- **Humidity Sensor:** Simulate relative humidity levels ranging from 30% to 90%. You may introduce larger or slower variations to mimic natural humidity trends.
- **Light Sensor:** Emulate ambient light levels using a scale from 0 (dark) to 1000 (bright sunlight), adding variation over time or simulating day/night cycles.

This gives a realistic feel of fluctuating sensor input, which is useful for testing embedded software.

2. Data Formatting & Logging

Each sensor reading should be recorded in a structured format, with the option to store it for later analysis. The format typically includes:

- A timestamp indicating when the reading was taken
- The sensor names and their corresponding values

- A consistent output format such as CSV (Comma-Separated Values), which is widely used in data logging and can be opened in Excel or plotted in analysis tools

Example log line (CSV format):

2025-05-28 10:45:23, Temperature: 26.4°C, Humidity: 57%, Light: 690

Logs can be written to:

- A text file for later retrieval or
 - A console window or virtual serial terminal for real-time observation
-

3. Serial Output (Optional)

In embedded systems, sensors often send data to external systems (like a PC or a cloud service) through serial communication. This project can optionally include:

- Sending the simulated data to a virtual serial port
- Viewing the output using Tera Term, PuTTY, or other terminal emulators

This helps understand how serial interfaces work and gives experience with stream-based output methods, even without actual embedded hardware.

Optional Tools:

- com0com (Windows) or tty0tty (Linux) to create virtual COM port pairs if needed
-

4. Periodic Sampling Mechanism

In real embedded applications, sensors are read at fixed time intervals. Your software should:

- Use a loop that runs continuously
- Wait for a fixed duration (e.g., 1 second) between iterations
- Read (simulate) sensor values and log them each cycle

This simulates how embedded systems regularly collect and process data, such as in IoT monitoring systems.

You may use:

- `sleep()` in Python or
 - `delay()` / `Sleep()` in C/C++
-

5. Configuration Options (Optional)

To make the program more flexible and user-friendly, allow configuration of key parameters without modifying the code:

- Sampling interval: e.g., choose between 1, 2, or 5 seconds

- Logging method: file logging vs. serial output
- Run duration: run the logger for a set number of samples or until the user stops it

Configurations can be taken:

- Through command-line arguments
- Via input prompts
- Using a config file (e.g., .ini, .json, or .txt)

This simulates real-world flexibility in embedded systems where runtime configuration is essential.

6. Error Handling & Edge Case Simulation (Optional)

Introduce simulated problems that reflect real-world issues like:

- Sensor disconnection (skip values or log error messages)
- Sensor malfunction (output extreme values or zero)
- Data outliers (simulate sudden spikes or drops)

This will help in designing systems that can detect and manage errors, a crucial skill in embedded programming.


Example:

2025-05-28 10:47:45, Temperature: ERROR, Humidity: 57%, Light: 690

Technical Components

Software Tools Only

- Programming Language: C/C++ or Python
- Text Editor/IDE: VS Code, Code::Blocks, or equivalent
- Terminal Emulator: Tera Term, PuTTY (for serial output simulation)
- Random Number Functions: To simulate variable sensor readings
- System Time Functions: For timestamps
- File I/O Libraries: To log data in readable format
- Sleep/Delay Functions: To maintain sampling rate

 **Note:** No external sensors or microcontroller hardware is required. All sensor values are simulated using software logic. Serial output is virtual and optional.

How to Perform This Task Using Only Software

1. Set Up the Environment

- Choose Python or C/C++ as the development language.
- Use an IDE or text editor of your choice.
- Ensure access to system libraries for time, random number generation, and file operations.

2. Simulate Sensor Data

- Use pseudo-random number generators to produce:
 - Temperature values (e.g., 20–40°C)
 - Humidity values (e.g., 30–90%)
 - Light values (e.g., 0–1000)
- Optionally, introduce logic to simulate realistic fluctuation or noise.

3. Create Logging Mechanism

- Format the simulated readings into lines with timestamps.
- Save each line to a .csv or .txt file using standard file I/O.

4. Implement Sampling Logic

- Use a loop with a time delay to generate new readings every second (or configurable interval).
- Continue for a specified number of samples or until a termination condition is met.





5. Optional: Add Serial Output

- If using Python, print output to the console and redirect to a virtual serial port.
- If using C/C++, redirect standard output to a virtual COM port (e.g., using com0com on Windows).
- Open a serial terminal (e.g., Tera Term or PuTTY) to view real-time data.

6. Test and Validate

- Verify output file content.
- Validate correct timing between entries.
- Confirm optional serial output displays as expected.

Deliverables

-  Complete source code in chosen language
-  Sample output log file (CSV or TXT)
-  Configuration summary (sampling interval, run mode)
-  README or report containing:
 - Description of sensors and data format

- How to build and run the simulation
 - Screenshots of console or terminal output
 - Optional enhancements or future ideas
-