Swiss Institute for Speleology and Karst Studies

Ezo Pump Integration Guide: Configuration, Usage, and Logic

For Automated Dosing Based on Water Height

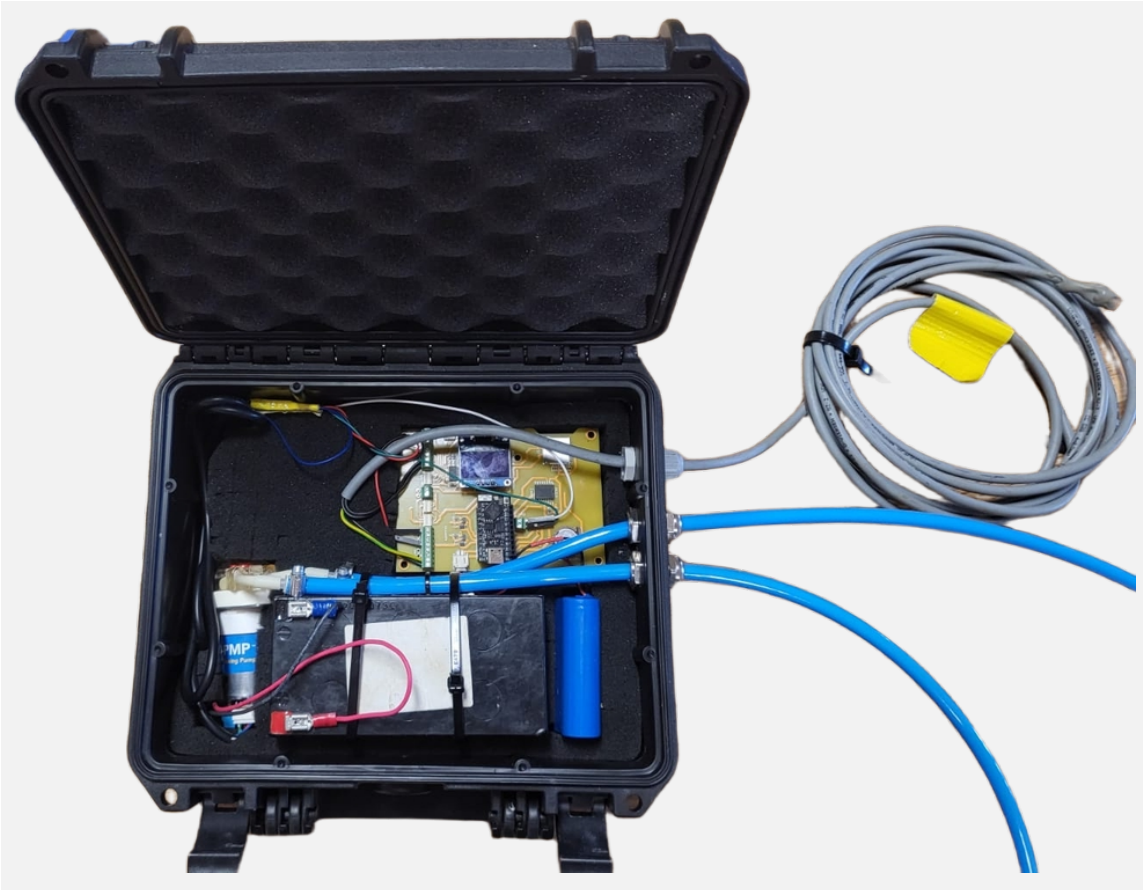
by

Marius Audenis

A report made during my civil service at ISSKA, to explain the configuration and usage of the tracer injection pump system and to support future use or modification.

June 02, 2025

Contents



EZO™ PMP embedded dosing pump integrated with ISSKA datalogger

1 Introduction

This guide explains how to integrate and operate the `Pump` class to control an **EZO-PMP™ embedded dosing pump** from Atlas Scientific. The class is designed for use on the ISSKA dataloggers code, running on a TinyPICO microcontroller. It automates fluid injection based on measured water heights.

This guide covers physical wiring, configuration files, injection logic, pump communication, and class internals.

2 Setup and Integration

2.1 Required Files on the SD Card

The pump module relies on the following file on the SD card. This file defines the injection logic. Pump logging is now merged into the main data CSV so a separate pump log is no longer written.

- `/pump.txt` — This file must be added on the datalogger SD card to define injection behavior and threshold logic.

  ```
  0;
  0;
  0;
  30-50,150,2;
  50-70,200,1;
  ```

  The tree first lines contain either a 0 or a 1. There are three possible settings:

  - Setting the pump in config mode: If 1, the pump will write the number of injections on the config file in the flash memory. After that, it is important to set it back to 0. This way, the pump will then read the number of injections from the flash memory, so if there is an accidental reboot or if a manual reboot is needed for maintenance, the number of remaining injections will not be reset.
  - Setting the offset of the water height sensor: As this sensor is meant to measure the height difference and is placed under water, an offset has to be added to avoid bias.
  - Filling in the tubes. If 1, the pump will inject 180 ml during configuration for filling in the tubes with tracer (necessary for a correct dosage). Put back to 0 after the tubes are filled, or else it will inject 180 ml at each reboot)
  - Setting the number of injections: Each class has a limited number of injections which is automatically updated at each injection. To avoid Each line specifies a water height range, a tracer dose in centimeters, and a maximum number of allowable injections for that range. The format is:

`<min_height>-<max_height>,<dose_mL>,<max_injections>;`

Example:

```
30-50,150,2;
50-70,200,1;
```

This means:

- Inject 150 mL if water height is between 30–50 cm (up to 2 times)
- Inject 200 mL if water height is between 50–70 cm (up to once)

- **Pump logging merged into `/data.csv`** — Pump status and cumulative activation count are now appended to the main sensor CSV. The merged CSV header includes two new columns at the end: `PumpError` and `PumpActivations`. Each data row ends with the pump error/status and the cumulative activation count.

  Example of merged header and row:

```
ID;DateTime;Vbatt;tempSHT;...;htWat;PumpError;PumpActivations
56;2025/06/02 12:00:00;3.9;...;12.3;Injected;3
```

2.2 Code Integration Steps

To integrate the `Pump` class with your datalogger, follow these steps:

1. Include the header and declare the pump object in `main.cpp`:

```
#include "Pump.h"
Pump pump;
```

2. Initialize the pump in `setup()` after setting up serial communication:

```
Serial2.begin(9600, SERIAL_8N1, rx, tx);
pump.configure(time_step, u8x8);
```

3. Make water height globally accessible in `Sensors.cpp` (top of file):

```
float measuredWaterheight = 0.0;
```

4. Call pump logic after measuring sensors and save the merged CSV after pump actions so pump fields are included in the same row:

```
mesure_all_sensors();
pump.handleInjections2(bootCount, time_step);
```

```
// save after pump.handleInjections2 so PumpError and PumpActivations reflect the same c
save_data_in_SD();
```

5. Minimal changes in `main.cpp` to merge pump fields into the header and data:

```
// in initialise_SD_card(), after sensor header is obtained:
dataFile.print("ID;DateTime;");
dataFile.print(header);                // sensor header already ends with ';'
dataFile.print(pump.getCSVHeader());   // adds "PumpError;PumpActivations"
dataFile.println();

// in save_data_in_SD(), after sensor data has been appended:
data_message = data_message + sensor.getFileData();
data_message = data_message + pump.getCSVFields();
```

3 Pump Operation Overview

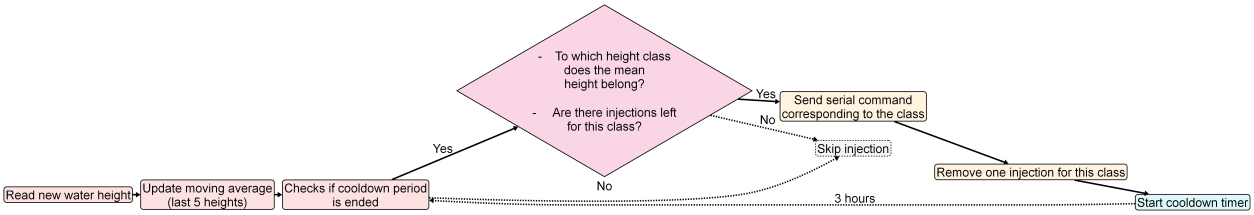
The pump automates fluid injections depending on water height. The system is designed to inject a specific volume of tracer based on predefined water height classes. Each class corresponds to a range of measured water heights (e.g., 10–20 cm) and is associated with a configurable injection dose (e.g., 5 ml). To ensure balanced tracer distribution, each class also has a limited number of allowable injections, preventing all the tracer from being delivered within a single height range. All three parameters —the height classes, corresponding doses, and the number of injections— are fully configurable via the system's configuration file. Also, to maintain injection accuracy, the system enforces cooldown periods between injections (default: 3 hours) to prevent mixing of tracer doses, which could skew measurement results.

The injection logic is as follows:

1. Maintains a moving average of the last 5 measured water heights.
2. If we are not in a cooldown period (there hasn't been any injection for more than 3 hours), it:
   - Iterates through `doseTable`, a variable stored in RTC memory (to survive deepsleep) containing the height classes, their corresponding injection doses and number of injections.
   - If the average water height falls within a defined range and remaining injections for that class are available:
     - Sends a command (e.g., `D,150˚`) to the pump.

- Enters a cooldown period.
- Decrements the injection count for the corresponding range.

3. If injection is disabled, increments the internal cooldown counter, and once the counter exceeds the threshold (default: every 3 hours), injection is possible again.
4. Logs the current injection status and cumulative injection count to `/pumpdata.csv`.



4 Pump Communication Protocol

The pump communicates over a UART serial interface. All technical details are available in the EZO-PMP™ Embedded Dosing Pump Manual, but key parameters and behaviors are summarized below:

- **Baud rate:** `9600`
- **Command format:** `D,<dose_in_mL>\r`
- **Example command:** `D,150\r`
- **Response codes:**
  - `*DONE` — Injection completed successfully
  - `*WA` — Injection skipped (already performed)
  - `*ER` — Error occurred during injection
  - `Timeout` — No response received (this one is a custom response added in the code)

5 Pump Class Internals

This section explains the internal workings of the `Pump` class to support customization or debugging.

Core Workflow Functions

`configure(int& time_step, U8X8 u8x8)` Initializes the pump system at startup:

- Reads dose rules from `/pump.txt` and parses them into a class table.
- Sends test signals to verify pump connection.

- No longer initializes a separate pump CSV; the merged header is written by `main.cpp` (see initialise$_{S}D_{c}ard$).$Computes$`boot_step`$basedoninjectioninterval$.

- Sends an initial test command to the pump.

`handleInjections2(int& bootCount, int& time_step)` Called during each measurement cycle:

- Maintains a moving average over the last 5 water height readings.
- Checks if the current height matches a dosing class.
- Verifies cooldown and injection count limits.
- Sends dose command if permitted.
- Does not write a separate pump CSV anymore — the pump exposes `getCSVFields()` and `getCSVHeader()` so `main.cpp` can merge pump information into `/data.csv`.

`save_in_SD(int& bootCount)` Present in the code for backward compatibility but currently a no-op: pump data is appended to the merged `/data.csv` via `getCSVFields()` instead.

Serial Communication and Pump Control

`sendCommand(const String& command)` Sends a dosing command to the pump:

- Transmits the command via `Serial2` with a carriage return.
- Waits for valid responses: `*DONE`, `*WA`, etc.
- Handles errors using `handleError()` and timeouts.
- Calls `pumpSleep()` after completion.

`readPumpResponse()` Reads characters from the serial buffer until a newline or timeout (15 seconds).

- Resets timer on each received byte.
- Returns the trimmed response or `Timeout` if no response.

`handleError(String& response)` Interprets and logs error messages:

- Logs `*ER` or `Timeout`, and may send a reset command.
- Extracts partial error strings from unknown responses.
- Stores error in `latestErrorMessage`.

`pumpSleep()` Sends the `Sleep` command:

- Waits for `*SL` confirmation.
- Handles communication errors or timeouts.

Optional Display and Debug Tools

`displayConfiguration(U8X8& display)` Displays the current dosing table on an OLED screen:

- Shows lower/upper class limits, dose values, and injection counts.
- Pages output every 6 entries to fit screen.

`latestErrorMessage` A string holding the latest pump response or error (e.g., `*DONE`, `Timeout`). Useful for diagnostics and logging.