# Automated Column Stand: Standard Operating Procedure

Ryan Hogan, Engineering Intern, Summer 2025

## Overview

This documentation was made to help operators using the automated test stand. The dashboard and control panel should be accessible on the iPad connected to the column stand. Use the following documentation when being trained on the new system and to help troubleshoot when problems may occur.

## Major Parts List

|  |  |
| --- | --- |
| **PART** | **LINK** |
| ESP32-DEVKIT-C | [ESP32-C6-DEVKITC-1-N8](https://www.digikey.com/en/products/detail/espressif-systems/ESP32-C6-DEVKITC-1-N8/17728861) |
| Flow Meter | [DMA20-AAAAA1 Endress+Hauser | Industrial Automation and Controls | DigiKey](https://www.digikey.com/en/products/detail/endress-hauser/DMA20-AAAAA1/17140769?gclsrc=aw.ds&gad_source=1&gad_campaignid=20243136172&gclid=CjwKCAjw4K3DBhBqEiwAYtG_9Nic6GV5G9iVCKkkcvaqPpTFLpSUF5Qw7qQC18XumS8sreTEZRJqrxoCE4MQAvD_BwE) |
| Pressure Gauges | [Cole-Parmer High-Accuracy Digital Gauge, 0 to 100 psig, 4-Digit LCD from Cole-Parmer](https://www.coleparmer.com/i/cole-parmer-high-accuracy-digital-gauge-0-to-100-psig-4-digit-lcd/6834906) |

## Features

### Calibration

Calibrating the sensors is done by setting the values for the high/low limits on each sensor. These values are accessible for the flow meter via the SmartBlue app on the iPad. For the pressure gauge, you can set/access the limits by following the instructions in Document A in the Appendix.

### Data Export

The data export feature allows operators to download data via .csv file or by copying and pasting a data table into an excel file.

### System Configuration

The system configuration window allows operators to set update intervals, the pressure threshold and set up the ESP32 with an IP address. The pressure threshold will determine when the test stops, and it is recommended to stay at 20 psi.

### Pause/Start/Reset Test

The pause/start/reset buttons allow operators to control the status of the current test. When pausing the test, the valve will close, and data collection will pause. The collected data will not clear until the reset button is clicked. If paused, the test will have the ability to be resumed with no clearing of data.

## How to Operate: Step by Step

### Connecting with ESP32

Connect with the ESP32 by typing in the IP address and clicking connect. BE sure that the computer/iPad connecting is also on the PRENGIOT Wi-Fi or else connection will not work. The IP address for the current ESP32 is 10.107.6.137.

### Starting Test

Before starting the test, be sure that the sensors are calibrated and that the configurations (pressure threshold and update interval) are properly set for your test. After confirming your test set up is ready and the valves are open, you should hit the start test button.

### Stopping Test

You can pause the test whenever you would like. This will close the valve and pause test collection until you resume. Once you pause the test, you can also reset the data and clear the test. This will require confirmation and will then clear all data collected.

### Exporting Data

At any time, you may export data in two formats. You can either choose to copy the table to clipboard and have the ability to paste into excel. You also can export the data as a .csv file and open in excel in a consistently formatted way.

## Maintenance

### Sensor Calibration

Contact other engineers for help with flow meter and pressure gauge calibration. This should be done every \_\_\_\_\_\_ months and marked on the sensor when completed.

## API References

**Base URL and Authentication**

**Base URL:** http://{ESP32\_IP\_ADDRESS}  
**Port:** 80  
**Authentication:** None required  
**Content-Type:** application/json for POST requests

### Endpoints

#### GET /api/status

Retrieves comprehensive system status and current sensor readings.

**Request:**

http

GET /api/status HTTP/1.1

Host: {ESP32\_IP}

**Response:**

json

{ "success": true, "ip": "192.168.1.100", "firmware": "v2.1-ESP32C6", "uptime": 3600, "freeMemory": 245760, "wifiConnected": true, "wifiRSSI": -45, "sensorsOnline": true, "testRunning": false, "testPaused": false, "valveOpen": false, "deviceName": "ESP32\_C6\_Monitor", "analogReadings": { "flowVoltage": 1.234, "inletVoltage": 0.987, "outletVoltage": 0.765 }, "inputs": { "pressure1Scaled": 15.6, "pressure2Scaled": 12.8, "flowScaled": 3.45, "pressure1Raw": 0.987, "pressure2Raw": 0.765, "flowRaw": 1.234 }, "outputs": { "solenoid": false }, "scaling": { "pressure1Min": 0.0, "pressure1Max": 50.0, "pressure2Min": 0.0, "pressure2Max": 50.0, "flowMin": 0.0, "flowMax": 10.0 } }

**Error Codes:**

* 500: Internal sensor communication error
* 503: System not ready

#### POST /api/start

Starts a new test or resumes a paused test.

**Request:**

http

POST /api/start HTTP/1.1

Host: {ESP32\_IP}

Content-Type: application/json

{ "pressureThreshold": 20.0, "resume": false }

**Parameters:**

* pressureThreshold (float): Maximum pressure before auto-stop (5.0-50.0 PSI)
* resume (boolean): true to resume paused test, false for new test

**Response (Success):**

Json { "success": true, "message": "Test started successfully", "testRunning": true, "valveOpen": true, "startTime": 1234567890 }

**Response (Error):**

json

{ "success": false, "error": "Test already running", "testRunning": true }

**Error Codes:**

* 400: Invalid parameters
* 409: Test already running
* 500: Hardware failure (valve or sensors)

#### POST /api/stop

Pauses the current test without resetting data.

**Request:**

http

POST /api/stop HTTP/1.1

Host: {ESP32\_IP}

**Response:**

Json { "success": true, "message": "Test paused successfully", "testRunning": false, "testPaused": true, "valveOpen": false }

**Error Codes:**

* 409: No test currently running
* 500: Hardware failure during valve closure

#### POST /api/reset

Resets the system to initial state, clearing all test data.

**Request:**

http

POST /api/reset HTTP/1.1

Host: {ESP32\_IP}

**Response:**

Json { "success": true, "message": "System reset successfully", "testRunning": false, "testPaused": false, "valveOpen": false, "dataCleared": true }

**Error Codes:**

* 500: System reset failed

#### GET /api/data

Retrieves current measurement data during active test.

**Request:**

http

GET /api/data HTTP/1.1

Host: {ESP32\_IP}

**Response:**

Json { "success": true, "timestamp": 1234567890, "testRunning": true, "testPaused": false, "elapsedTime": 125, "flowRate": 4.23, "pressureDrop": 8.7, "totalVolume": 8.854, "valveOpen": true, "inletPressure": 22.4, "outletPressure": 13.7, "analogVoltages": { "flow": 1.456, "inlet": 1.234, "outlet": 0.987 } }

**Error Codes:**

* 404: No active test session
* 500: Sensor communication error

#### POST /api/config

Updates system configuration parameters.

**Request:**

http

POST /api/config HTTP/1.1

Host: {ESP32\_IP}

Content-Type: application/json

{ "pressureThreshold": 25.0, "updateInterval": 2.0, "deviceName": "Lab\_Monitor\_01"}

**Parameters:**

* pressureThreshold (float): Auto-stop pressure limit (5.0-50.0 PSI)
* updateInterval (float): Data polling interval (0.5-120.0 seconds)
* deviceName (string): Device identifier (max 32 characters)

**Response:**

Json { "success": true, "message": "Configuration updated", "configurationSaved": true}

**Error Codes:**

* 400: Invalid parameter values
* 500: EEPROM write failure

#### POST /api/scaling

Updates sensor scaling parameters for calibration.

**Request:**

http

POST /api/scaling HTTP/1.1

Host: {ESP32\_IP}

Content-Type: application/json

{ "pressure1Min": 0.0, "pressure1Max": 50.0, "pressure2Min": 0.0, "pressure2Max": 50.0, "flowMin": 0.0, "flowMax": 15.0 }

**Parameters:**

* pressure1Min/Max (float): Inlet pressure sensor range
* pressure2Min/Max (float): Outlet pressure sensor range
* flowMin/Max (float): Flow sensor range

**Response:**

Json { "success": true, "message": "Scaling configuration updated", "scalingSaved": true}

**Error Codes:**

* 400: Invalid scaling values (min >= max)
* 500: Configuration save failure

#### POST /api/calibrate

Performs sensor calibration procedures.

**Request:**

http

POST /api/calibrate HTTP/1.1

Host: {ESP32\_IP}

Content-Type: application/json

{ "type": "flow", "knownValue": 5.25, "calibrationPoint": "max", "confirmed": true }

**Parameters:**

* type (string): Sensor type ("flow", "pressure")
* knownValue (float): Reference standard reading
* calibrationPoint (string): "min" or "max" calibration point
* confirmed (boolean): Final confirmation of calibration

**Response:**

json

{ "success": true, "message": "Flow sensor calibration completed", "calibrationType": "flow", "calibrationPoint": "max", "referenceValue": 5.25, "measuredValue": 5.18, "percentError": 1.3 }

**Error Codes:**

* 400: Invalid calibration parameters
* 500: Calibration procedure failed

#### POST /api/solenoid

Manually controls the solenoid valve state.

**Request:**

http

POST /api/solenoid HTTP/1.1

Host: {ESP32\_IP}

Content-Type: application/json

{ "state": true}

**Parameters:**

* state (boolean): true to open valve, false to close

**Response:**

Json { "success": true, "newState": true, "message": "Solenoid ON", "valveOpen": true}

**Error Codes:**

* 400: Invalid state parameter
* 409: Cannot control valve during active test
* 500: Hardware control failure

**Error Handling**

All endpoints return appropriate HTTP status codes:

* **200 OK**: Request successful
* **400 Bad Request**: Invalid parameters or malformed JSON
* **404 Not Found**: Endpoint not available or resource not found
* **409 Conflict**: Operation not permitted in current state
* **500 Internal Server Error**: Hardware or system failure
* **503 Service Unavailable**: System not ready or sensors offline

Error responses include descriptive messages in the error field to assist with troubleshooting.

**Rate Limiting**

The system can handle concurrent requests but optimal performance is achieved with:

* Status polling: Maximum 2 Hz (every 500ms)
* Configuration updates: Maximum 1 request per second
* Manual controls: No specific limits

**CORS Support**

All endpoints include appropriate CORS headers for browser compatibility:

Access-Control-Allow-Origin: \*

Access-Control-Allow-Methods: GET, POST, OPTIONS

Access-Control-Allow-Headers: Content-Type, Accept

## FAQs/Troubleshooting

### How do I find the ESP32 IP Address?

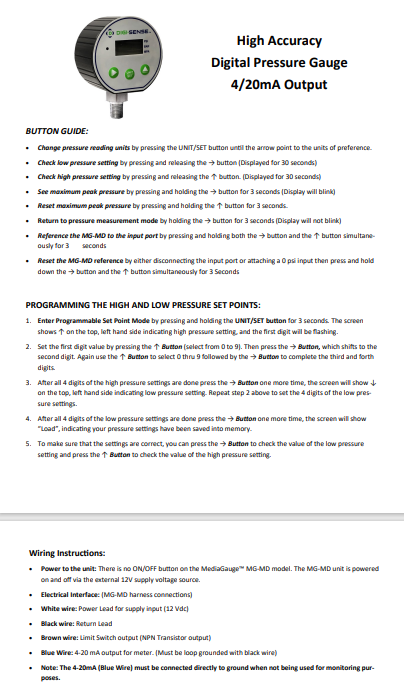
The IP address should be \_\_\_\_\_\_\_\_\_\_. If this is not working, you will need to download the backend software and run this on Arduino IDE (<https://www.arduino.cc/en/software/>) while connected to the ESP32 via USB-C cable. When you compile and run the code, the IP address will be printed on the serial monitor in the Arduino IDE.

### Why can’t I connect to the ESP32?

You need to be on the PNRENGIOT Wi-Fi to connect to the ESP32. If you do not have the password for this Wi-Fi, please contact IT (Greg Schraufnagel) for password.

## Appendix

### Document A: Pressure Gauge Documentation



### Document B: Flow Meter Documentation (<https://tinyurl.com/PicoMagIODoc>)

Diagram of a device plug

AI-generated content may be incorrect.

A close-up of a box

AI-generated content may be incorrect.

A white box with black text

AI-generated content may be incorrect.

A diagram of a circuit

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A graph of a function

AI-generated content may be incorrect.

### Document C: How Does the System Work?

The communication flow begins when the ESP32 establishes a WiFi connection to the local network and starts an HTTP web server listening on port 80. Browser clients discover and connect to the ESP32 using its assigned IP address, initiating a client-server relationship where the ESP32 serves both static web content and dynamic API endpoints.

Real-time data acquisition occurs through continuous analog-to-digital conversion on three dedicated GPIO pins, where the ESP32 samples pressure and flow sensors at 12-bit resolution with 10-sample averaging for noise reduction. The raw voltage readings undergo scaling transformations using configurable minimum and maximum ranges to convert sensor outputs into engineering units.

During active test operations, the browser establishes a polling loop that sends GET requests to the ESP32's data endpoint at user-configurable intervals ranging from 500 milliseconds to 10 seconds. Each request triggers the ESP32 to perform immediate sensor readings, calculate derived values like pressure differential and cumulative volume, and package the results into structured JSON responses.

The ESP32 maintains internal test state including elapsed time, total volume calculations, and safety threshold monitoring. When pressure limits are exceeded, the system automatically triggers safety responses by closing the solenoid valve and notifying connected clients through status updates in subsequent polling responses.

Configuration changes flow bidirectionally, where browser clients send POST requests containing updated parameters such as pressure thresholds, sensor scaling factors, and operational settings. The ESP32 processes these requests, updates internal variables, and optionally persists critical settings to EEPROM for retention across power cycles.

The system implements graceful degradation where connection losses are detected through failed HTTP requests, triggering automatic reconnection attempts on the client side and fail-safe valve closure on the ESP32 side. Multiple clients can connect simultaneously to monitor the same test session, with each receiving independent data streams through their individual polling cycles.

Emergency stop functionality operates through immediate HTTP POST requests that bypass normal polling intervals, ensuring rapid response times under 100 milliseconds for critical safety operations. The ESP32 processes these commands with highest priority, immediately updating hardware outputs and internal state before responding to the client.

Data export capabilities utilize browser-initiated requests that trigger the ESP32 to format historical data into CSV structures, which are then transmitted as downloadable content through standard HTTP file transfer mechanisms, allowing seamless integration with external analysis tools.