Air Conditioner Instruction Manual v2

Table of Contents

---

1. Introduction

This document describes the operation and maintenance of the MQTT-based Air Conditioner simulator. It is intended for technicians responsible for:

- Deploying and configuring the AC simulator to publish real-time telemetry to AWS IoT Core.
- Monitoring and troubleshooting the AC unit's performance.
- Injecting and resolving various faults or simulated errors.
- Managing device runtime and filter status.

> **Note:** This manual applies to a simulated air conditioner. However, the structure and procedures closely mimic those of a real-world IoT-enabled AC unit for training and testing purposes.

---

2. System Overview

The simulated AC system is an IoT device that:

- Connects securely to an AWS IoT endpoint via MQTT.
- Publishes telemetry data (e.g., temperatures, humidity, pressure).
- Receives commands to change operating parameters (e.g., setpoint temperature, operational mode).
- Reports device shadow updates to keep AWS IoT in sync with the AC unit's state.

KEY FEATURES

- **Indoor & Outdoor Temperature/Humidity** measurements.
- **Setpoint Temperature** control.
- **Compressor** on/off simulation based on setpoint requirements.
- **Fan Speed** adjustments (in RPM).
- **Refrigerant Pressure** monitoring.
- **Power Consumption** estimates in watts.
- **Injectable Faults** for testing troubleshooting procedures.
- **Device Shadow** updates for remote state synchronization.
- **Error Reporting:** Any detected errors (e.g., "E1", "E2", "E3") are published to the `aircon/errors` topic.

---

3. Hardware & Software Requirements

1. **AWS IoT Account**: An active AWS account configured to accept MQTT connections for IoT devices.
2. **Python Environment**: Python 3.7+ installed.
3. **AWS IoT SDK for Python**: Required libraries (specifically `AWSIoTPythonSDK`).
4. **Certificates and Keys**:
    - **Root CA (AmazonRootCA1.pem)**
    - **Private Key** (`<device_name>-private.pem.key`)
    - **Certificate** (`<device_name>-certificate.pem.crt`)

1. **Device Information File**: `device_info.json` containing:

```
json
Copy
{
"thingName": "your_device_name",
"endpoint": "your_aws_iot_endpoint",
"rootCAPath": "/path/to/AmazonRootCA1.pem"
}
```

1. **Internet Connectivity**: Required for publishing telemetry and subscribing to commands.

---

4. Installation & Setup

1. **Organize Certificates**:
   Place the **private key** and **certificate** files in the same directory as the `device_info.json` file for each device.
2. **Update `device_info.json`**:
   Ensure the following keys and values are correct:
    - `thingName`: The name registered in AWS IoT.

  - `endpoint`: The AWS IoT endpoint URL for your region.
  - `rootCAPath`: The full path to the Amazon Root CA file.

1. **Download the Root CA (If Not Present)**:
   If the root CA file is missing, the simulator will attempt to automatically download it from:

```
arduino
Copy
https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

1. **Run the Simulator**:

```
bash
Copy
python ac_simulator.py --devices-folder devices --device-name <DEVICE_NAME> --topic a
```

  - `--devices-folder`: Path to the folder containing the individual device folder(s).
  - `--device-name`: (Optional) The specific device folder name to run. If not specified, simulator runs for all found devices.
  - `--topic`: MQTT topic for telemetry publishing (default: `aircon/telemetry`).
  - `--interval`: Time in seconds between telemetry publishes (default: 10).

---

5. Operational Modes

The simulator supports three operational modes:

1. **Cool**:

  - Compressor cycles on/off to maintain the **setpoint temperature**.
  - Fan runs when the compressor is active to circulate cool air.

1. **Fan Only**:

  - Compressor remains off.
  - Fan spins at a consistent RPM to provide ventilation.

1. **Off**:

  - Compressor and fan remain off.
  - The indoor temperature gradually approaches the outdoor temperature.

> **Note:** Mode transitions can be triggered via a device shadow update (see Section 12) or by publishing a command message to the command topic.

---

6. Telemetry & Data Points

Every interval seconds, the AC unit generates and publishes the following telemetry data:

	Field	Description	Example
1	`device_name`	Name of the AC unit (thing name).	`"ACUnit1"`
2	`indoor_temperature_c`	Current indoor temperature (°C).	24
3	`outdoor_temperature_c`	Current outdoor temperature (°C).	30
4	`setpoint_temperature_c`	Desired target indoor temperature (°C).	23
5	`mode`	Current operational mode. (`cool`, `fan_only`, or `off`)	`"cool"`
6	`indoor_humidity_percent`	Current indoor humidity (%).	45
7	`outdoor_humidity_percent`	Current outdoor humidity (%).	70
8	`power_consumption_watts`	Real-time power usage in watts.	2100.54
9	`compressor_status`	Status of compressor (`On` or `Off`).	`"On"`
10	`fan_speed_rpm`	Fan speed in rotations per minute (RPM).	1200.57
11	`refrigerant_pressure_psi`	Current refrigerant pressure in PSI.	198.75
12	`error_code`	Current error/fault code.	`"None"` or `"E1"`
13	`filter_status`	Current state of the air filter (`Clean`, `Needs Cleaning`, or `Replace`).	`"Clean"`
14	`runtime_hours`	Total operational hours since last reset.	10.5

---

7. Command & Control

Commands can be published to the command topic:

```
bash
Copy
aircon/commands/<device_name>
```

Each command is a JSON payload with an `action` field (and optional parameters). For example:

	Action	JSON Payload Example	Description
1	`inject_fault`	`{"action": "inject_fault", "fault_type": "high_temperature"}`	Injects a fault into the system. Supported `fault_type` values: `high_temperature`, `low_pressure`, `compressor_failure`.
2	`clear_fault`	`{"action": "clear_fault"}`	Clears any active faults and resets the error code to `"None"`.
3	`update_filter_status`	`{"action": "update_filter_status", "filter_status": "Needs Cleaning"}`	Updates the filter status to one of: `Clean`, `Needs Cleaning`, or `Replace`.
4	`reset_runtime`	`{"action": "reset_runtime"}`	Resets the runtime counter (`runtime_hours`) to `0`.
5	`disconnect`	`{"action": "disconnect"}`	Instructs the simulator to disconnect from MQTT and terminate.

---

8. Fault Codes & Troubleshooting

During normal operation, `error_code` remains `"None"`. The following fault types may be injected to simulate errors:

	Fault Type	Internal Effect	Error Code	Troubleshooting Steps
1	`high_temperature`	Indoor temperature increases drastically by 5 to 10 °C.	`E1`	1. Check compressor functionality. <br/>2. Verify refrigerant pressure. <br/>3. Inspect the filter and ensure proper airflow.
2	`low_pressure`	Refrigerant pressure drops toward the lower threshold (50–60 PSI).	`E2`	1. Check for refrigerant leaks. <br/>2. Verify correct refrigerant charge. <br/>3. Inspect compressor for abnormal sounds or vibrations.
3	`compressor_failure`	Compressor remains off, power consumption drops to 0.	`E3`	1. Verify power supply to compressor. <br/>2. Check compressor windings/resistance. <br/>3. Test capacitor and relay.

CLEARING FAULTS

TO CLEAR ANY ACTIVE FAULT, PUBLISH:

```
json
Copy
{"action": "clear_fault"}
```

This resets `error_code` to `"None"` and returns the unit to normal operation.

> **Note:** Error codes are also published to the `aircon/errors` topic for easier monitoring.

---

9. Maintenance & Filter Management

1. **Filter Status Values:**

    - **Clean:** Filter is new or recently maintained.
    - **Needs Cleaning:** Filter has reached recommended runtime hours.
    - **Replace:** Filter is no longer effective and needs replacement.

1. **Auto-Update Mechanism:**
   When the simulator’s `runtime_hours` surpass 500 hours, the filter status automatically changes from `"Clean"` to `"Needs Cleaning"`. Technicians can manually override this by publishing:

```
json
Copy
{
"action": "update_filter_status",
"filter_status": "Replace"
}
```

1. **Runtime Reset:**
   After cleaning or replacing the filter, reset the operational hours by publishing:

```
json
Copy
{"action": "reset_runtime"}
```

10. Runtime & Behavior

Every telemetry interval (default: 10 seconds), the AC unit updates internal calculations:

- **Runtime Hours:** Increases based on elapsed time (`interval / 3600`).
- **Indoor Temperature:** Adjusts toward the setpoint when the compressor is on; otherwise drifts toward the outdoor temperature.
- **Outdoor Temperature:** Varies slightly by ±1 °C periodically.
- **Power Consumption:**
  - Ranges from **0 W** (off) to around **2000+ W** with compressor load.
  - **Fan Only** mode consumes around **200 W**.
  - Idle consumes **100 W**.
- **Refrigerant Pressure:** Fluctuates randomly within realistic bounds (50–300 PSI).

---

11. MQTT Communication Topics

1. **Telemetry Publish Topic:** `aircon/telemetry`

   - The simulator publishes AC data at regular intervals.

1. **Command Topic:** `aircon/commands/<device_name>`

   - The simulator subscribes to this topic to receive commands.

1. **Device Shadow:**

   - **Shadow Delta Topic:** `$aws/things/<device_name>/shadow/update/delta`
   - **Shadow Update Topic:** `$aws/things/<device_name>/shadow/update`

1. **Error Topic:** `aircon/errors`

   - When an error is detected (i.e., `error_code` is not `"None"`), an error message is published to this topic with details including the device name, error code, and timestamp.

---

12. Shadow Updates

The AC simulator integrates with the AWS IoT Device Shadow. When the device receives a delta message, it updates its internal state accordingly. Currently, the simulator listens for:

- **Setpoint Temperature** (`setpoint_temperature_c`)
- **Mode** (can be `cool`, `off`, or `fan_only`)

Example **delta message**:

```
json
Copy
{
"state": {
"setpoint_temperature_c": 22,
"mode": "fan_only"
}
}
```

Upon receiving this message:

- The simulator adjusts the setpoint to `22 °C`.
- Updates the mode to `"fan_only"`.
- Reports the new state back via the shadow **update** topic, visible in the AWS IoT console or consumable by other services.

---

13. FAQ & Best Practices

1. **How do I simulate a specific error without waiting?**
   Publish a JSON command with `"action": "inject_fault"` and specify the `fault_type`.
2. **Why does the indoor temperature not reach the setpoint immediately?**
   The simulator mimics natural drift and a realistic cooling rate, so temperature adjustments occur gradually.
3. **Can I run multiple units simultaneously?**
   Yes. Place multiple device folders (each containing its own `device_info.json` and certificates) under the `devices` directory. The simulator starts each device in a separate thread.
4. **What happens if I set the mode to `off` while a fault is injected?**
   The simulator respects the operational mode (i.e., the compressor remains off), but the injected fault continues until explicitly cleared.

---

14. Reference: Simulator Code Summary

The core simulator logic (`ACUnitSimulator`) handles:

- **Initialization:** Loads configuration, assigns initial random values for temperature, humidity, etc., and subscribes to command/shadow topics.
- **Command Processing:** (`on_command_received`) Handles injected faults, runtime resets, filter status updates, and disconnect instructions.
- **Shadow Delta Processing:** (`on_shadow_delta`) Updates setpoint temperature or operational mode based on received delta messages and reports the updated state.
- **Telemetry Generation:** (`generate_telemetry_data`)
  - Applies temperature, humidity, and pressure calculations.
  - Adjusts compressor and fan status.
  - Injects faults when instructed.

    - Publishes data to the telemetry topic.
    - Additionally, publishes error messages to the `aircon/errors` topic if an error is detected.
- **Execution Loop:** (`run`) Continuously publishes telemetry data at specified intervals until a disconnect command is received or the program is terminated.