ESP32 Irrigation Controller - Complete API Documentation

API Documentation

November 6, 2025

Executive Summary

This document provides comprehensive documentation for the ESP32 Irrigation Controller’s communication interfaces, including both MQTT and REST API protocols. The system supports real-time monitoring, zone control, scheduling, and configuration management through both interfaces.

Key Features: - Dual interface support (MQTT + REST API) - Real-time zone control and monitoring - Automated scheduling with RTC synchronization - Home automation integration (Home Assistant, Node-RED) - Network-based configuration management

---

1. MQTT Interface Documentation

1.1 Overview

The MQTT interface provides publish/subscribe messaging for real-time monitoring and control of the irrigation system. It supports automatic device discovery, periodic status updates, and bidirectional communication.

Connection Details

Parameter	Default Value	Description
Protocol	MQTT v3.1.1	Standard MQTT protocol
Port	1883	Default MQTT port (configurable)
Client ID	`esp32_irrigation_[MAC]`	Unique identifier per device
Keep Alive	60 seconds	Connection keep-alive interval
QoS	0 (At most once)	Message delivery quality
Retain	Configurable	Message retention on broker

Topic Hierarchy

All topics follow the pattern: `{prefix}/{device_id}/{category}/{subcategory}`

Default prefix: `irrigation/`
Default device ID: `esp32_irrigation`

1.2 Published Topics (Device → Broker)

1.2.1 Device Configuration

Topic: `irrigation/esp32_irrigation/config/device`
Frequency: Every minute on the minute (RTC synchronized)
Retain: Yes (configurable)

Payload Structure:

```
{
  "device_id": "esp32_irrigation",
  "client_id": "esp32_irrigation_AABBCCDDEEFF",
  "ip_address": "192.168.1.100",
  "mac_address": "AA:BB:CC:DD:EE:FF",
  "wifi_ssid": "MyNetwork",
  "wifi_rssi": -45,
  "heap_free": 180000,
  "timestamp": "2025-11-06T14:30:00+09:30",
  "uptime": 3600000,
  "firmware_version": "1.0.0",
  "mqtt_broker": "172.17.254.10",
  "mqtt_port": 1883,
  "topic_prefix": "irrigation/",
  "timezone": 9.5,
  "max_zones": 8
}
```

Field Descriptions: - `device_id`: Unique device identifier - `ip_address`: Current IP address on network - `wifi_rssi`: Signal strength in dBm (closer to 0 is better) - `heap_free`: Available RAM in bytes - `timestamp`: ISO 8601 format with timezone offset - `uptime`: Milliseconds since boot

---

1.2.2 Device Status

Topic: `irrigation/esp32_irrigation/status/device`
Frequency: Every minute on the minute
Retain: Yes (configurable)

Payload Structure:

```
{
  "zones": [
    {
      "zone": 1,
      "status": "active",
```

```
      "time_remaining": 900,
      "start_time": "2025-11-06T14:00:00"
    }
  ],
  "rain_delay": 0,
  "schedule_enabled": true,
  "active_program": 0,
  "timestamp": "2025-11-06T14:30:00+09:30"
}
```

Zone Status Values: - `idle`: Zone not running - `active`: Zone currently watering - `scheduled`: Zone scheduled to start - `paused`: Zone paused by rain delay

---

1.2.3 Schedule Status

Topic: `irrigation/esp32_irrigation/status/schedules`
Frequency: On change + every minute
Retain: Yes

Payload Structure:

```
{
  "schedules": [
    {
      "id": 1,
      "zone": 1,
      "days": [1, 2, 3, 4, 5],
      "start_hour": 6,
      "start_minute": 30,
      "duration": 15,
      "enabled": true,
      "next_run": "2025-11-07T06:30:00+09:30"
    }
  ],
  "total_schedules": 8,
  "active_schedules": 5
}
```

Day Encoding: - 1 = Monday, 2 = Tuesday, … 7 = Sunday

1.2.4 Zone Status (Individual)

Topic: `irrigation/esp32_irrigation/status/zone/{zone_number}`
Frequency: On zone state change
Retain: No

Payload Structure:

```
{
  "zone": 1,
  "status": "active",
  "time_remaining": 900,
  "timestamp": 1699270200000
}
```

1.2.5 Configuration Status

Topic: `irrigation/esp32_irrigation/status/config`
Frequency: On configuration change
Retain: Yes

Payload Structure:

```
{
  "timezone_offset": 9.5,
  "max_enabled_zones": 8,
  "mqtt_enabled": true,
  "mqtt_broker": "172.17.254.10",
  "mqtt_port": 1883,
  "mqtt_topic_prefix": "irrigation/",
  "mqtt_retain": true,
  "mqtt_keep_alive": 60,
  "scheduling_enabled": true
}
```

1.2.6 Home Assistant Discovery

Topic: `homeassistant/switch/esp32_irrigation/config`
Frequency: On connection
Retain: Yes

Payload Structure:

```
{
  "name": "ESP32 Irrigation Controller",
  "unique_id": "esp32_irrigation",
  "device_class": "irrigation",
  "state_topic": "irrigation/esp32_irrigation/status/device",
  "command_topic": "irrigation/esp32_irrigation/command/status"
}
```

1.3 Subscribed Topics (Broker → Device)

1.3.1 Configuration Commands

Topic Pattern: `irrigation/esp32_irrigation/config/{setting}/set`

Supported Settings:

Setting	Payload Type	Example	Description
`timezone`	Float	`9.5`	Timezone offset in hours
`mqtt_broker`	String	`172.17.254.10`	MQTT broker address
`mqtt_port`	Integer	`1883`	MQTT broker port
`mqtt_username`	String	`user`	Authentication username
`mqtt_topic_prefix`	String	`home/irrigation/`	Topic prefix
`max_enabled_zones`	Integer	`8`	Maximum zones (1-16)

Example:

```
# Set timezone to Adelaide (UTC+9:30)
mosquitto_pub -h 172.17.254.10 -t
        "irrigation/esp32_irrigation/config/timezone/set" -m "9.5"


# Change max zones to 12
mosquitto_pub -h 172.17.254.10 -t
        "irrigation/esp32_irrigation/config/max_enabled_zones/set" -m
        "12"
```

Response: Device publishes updated configuration to `irrigation/esp32_irrigation/status/config`

1.3.2 Device Commands

Topic Pattern: `irrigation/esp32_irrigation/command/{command}`

Available Commands:

Command	Payload	Description	Example
`restart`	Any	Restart ESP32	`"restart"`
`status`	Any	Request status update	`"status"`
`rain_delay`	Integer	Set rain delay in minutes	`120`
`clear_rain`	Any	Clear rain delay	`"clear"`
`enable_schedule`	Boolean	Enable/disable scheduling	`true` or `1`

Examples:

```
# Restart the device
mosquitto_pub -h 172.17.254.10 -t
        "irrigation/esp32_irrigation/command/restart" -m "restart"

# Set 2-hour rain delay
mosquitto_pub -h 172.17.254.10 -t
        "irrigation/esp32_irrigation/command/rain_delay" -m "120"

# Request immediate status update
mosquitto_pub -h 172.17.254.10 -t
        "irrigation/esp32_irrigation/command/status" -m "status"

# Enable scheduling
mosquitto_pub -h 172.17.254.10 -t
        "irrigation/esp32_irrigation/command/enable_schedule" -m "true"
```

1.3.3 Schedule Management

Topic: `irrigation/esp32_irrigation/schedule/set`
Payload: JSON schedule object

Structure:

```
{
  "schedules": [
    {
      "zone": 1,
      "days": [1, 2, 3, 4, 5],
      "start_hour": 6,
      "start_minute": 30,
      "duration": 15,
      "enabled": true
    }
  ]
}
```

Response Topic: `irrigation/esp32_irrigation/schedule/result`
Response Payload: `"success"` or `"error"`

Example:

```
mosquitto_pub -h 172.17.254.10 \
  -t "irrigation/esp32_irrigation/schedule/set" \
  -m '{
    "schedules": [{
      "zone": 2,
```

```
      "days": [1,3,5],
      "start_hour": 18,
      "start_minute": 0,
      "duration": 20,
      "enabled": true
    }]
  }'
```

1.3.4 AI Schedule Management

Topic: `irrigation/esp32_irrigation/schedule/ai/set`
Payload: JSON schedule array

Structure:

```
{
  "schedules": [
    {
      "zone": 1,
      "start_time": "06:30",
      "duration": 15,
      "days": "MTWTFSS"
    }
  ]
}
```

Days Format: `M=Monday, T=Tuesday, W=Wednesday, T=Thursday, F=Friday, S=Saturday, S=Sunday`

Response Topic: `irrigation/esp32_irrigation/schedule/ai/result`
Response Payload: `"success"` or `"error"`

1.4 MQTT Integration Examples

1.4.1 Python with Paho MQTT

```
import paho.mqtt.client as mqtt
import json

BROKER = "172.17.254.10"
PORT = 1883
TOPIC_PREFIX = "irrigation/esp32_irrigation/"

def on_connect(client, userdata, flags, rc):
```

```
    print(f"Connected with result code {rc}")
    # Subscribe to all status topics
    client.subscribe(TOPIC_PREFIX + "status/#")
    client.subscribe(TOPIC_PREFIX + "config/#")

def on_message(client, userdata, msg):
    print(f"Topic: {msg.topic}")
    try:
        payload = json.loads(msg.payload)
        print(f"Payload: {json.dumps(payload, indent=2)}")
    except:
        print(f"Payload: {msg.payload.decode()}")

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect(BROKER, PORT, 60)

# Set rain delay
client.publish(TOPIC_PREFIX + "command/rain_delay", "120")

# Enable scheduling
client.publish(TOPIC_PREFIX + "command/enable_schedule", "true")

client.loop_forever()
```

1.4.2 Node-RED Flow

```
[
  {
    "id": "mqtt_in",
    "type": "mqtt in",
    "topic": "irrigation/esp32_irrigation/status/device",
    "broker": "172.17.254.10",
    "qos": "0"
  },
  {
    "id": "parse_json",
    "type": "json",
    "property": "payload"
  },
  {
```

```
    "id": "store_db",
    "type": "sqlite",
    "db": "/root/.node-red/data/irrigation.db",
    "sql": "INSERT INTO device_status (ip_address, wifi_rssi, heap_free,
        uptime, timestamp) VALUES ($ip_address, $wifi_rssi, $heap_free,
        $uptime, $timestamp)"
  }
]
```

1.4.3 Home Assistant Configuration

```yaml
# configuration.yaml

mqtt:
  sensor:
    - name: "Irrigation Controller Status"
      state_topic: "irrigation/esp32_irrigation/status/device"
      value_template: "{{ value_json.zones | length }}"
      json_attributes_topic: "irrigation/esp32_irrigation/config/device"
      json_attributes_template: "{{ value_json | tojson }}"

    - name: "Irrigation WiFi Signal"
      state_topic: "irrigation/esp32_irrigation/config/device"
      value_template: "{{ value_json.wifi_rssi }}"
      unit_of_measurement: "dBm"
      device_class: "signal_strength"

  switch:
    - name: "Irrigation Scheduling"
      command_topic:
        "irrigation/esp32_irrigation/command/enable_schedule"
      state_topic: "irrigation/esp32_irrigation/status/config"
      value_template: "{{ value_json.scheduling_enabled }}"
      payload_on: "true"
      payload_off: "false"

automation:
  - alias: "Rain Delay on Weather Alert"
    trigger:
      - platform: state
        entity_id: weather.home
        to: 'rainy'
    action:
      - service: mqtt.publish
```

```
      data:
        topic: "irrigation/esp32_irrigation/command/rain_delay"
        payload: "120"
```

---

2. REST API Documentation

2.1 Overview

The REST API provides HTTP-based access to all irrigation controller functions. It supports both GET and POST methods, with JSON and form-encoded data.

Base URL

`http://[ESP32_IP_ADDRESS]/api/`

Example: `http://192.168.1.100/api/`

Response Formats

- **Success Responses:** Plain text or JSON
- **Error Responses:** Plain text error messages
- **HTTP Status Codes:** Standard HTTP codes (200, 400, 404, 500)

2.2 Zone Control Endpoints

2.2.1 START ZONE

Start a specific irrigation zone for a specified duration.

Endpoint: `GET /api/start-zone`

Parameters:

Parameter	Type	Required	Range	Description
`zone`	Integer	Yes	1-16	Zone number
`time`	Integer	Yes	1-240	Duration in minutes

Example Request:

```
curl "http://192.168.1.100/api/start-zone?zone=3&time=15"
```

Response:

```
Zone 3 started for 15 minutes
```

Python Example:

```python
import requests

response = requests.get(
    "http://192.168.1.100/api/start-zone",
    params={"zone": 3, "time": 15}
)
print(response.text)
```

JavaScript Example:

```javascript
fetch('http://192.168.1.100/api/start-zone?zone=3&time=15')
  .then(response => response.text())
  .then(data => console.log(data));
```

2.2.2 STOP ZONE

Stop a specific irrigation zone immediately.

Endpoint: `GET /api/stop-zone`

Parameters:

Parameter	Type	Required	Range	Description
zone	Integer	Yes	1-16	Zone number

Example Request:

```
curl "http://192.168.1.100/api/stop-zone?zone=3"
```

Response:

```
Zone 3 stopped
```

2.3 Program Control

2.3.1 RUN PROGRAM

Execute a pre-configured irrigation program.

Endpoint: `GET /api/run-program`

Parameters:

Parameter	Type	Required	Range	Description
`program`	Integer	Yes	1-4	Program number

Example Request:

```
curl "http://192.168.1.100/api/run-program?program=1"
```

Response:

```
Program 1 started
```

2.4 Time and Clock Management

2.4.1 GET CURRENT TIME

Retrieve current RTC time.

Endpoint: `GET /api/time`

Parameters: None

Example Request:

```
curl "http://192.168.1.100/api/time"
```

Response:

```
2025-11-06 14:30:00
```

2.4.2 SET TIME

Manually set the RTC time.

Endpoint: `POST /api/set-time`

Content-Type: `application/x-www-form-urlencoded`

Parameters:

Parameter	Type	Required	Range	Description
`year`	Integer	Yes	2020-2099	Year
`month`	Integer	Yes	1-12	Month
`day`	Integer	Yes	1-31	Day
`hour`	Integer	Yes	0-23	Hour
`minute`	Integer	Yes	0-59	Minute
`second`	Integer	Yes	0-59	Second

Example Request:

```
curl -X POST "http://192.168.1.100/api/set-time" \
     -H "Content-Type: application/x-www-form-urlencoded" \
     -d "year=2025&month=11&day=6&hour=14&minute=30&second=0"
```

2.4.3 SYNC WITH NTP

Synchronize RTC with Network Time Protocol servers.

Endpoint: `GET /api/sync-ntp`

Parameters: None

Example Request:

```
curl "http://192.168.1.100/api/sync-ntp"
```

Response:

```
Time synchronized with NTP server
```

2.5 System Status and Information

2.5.1 GET SYSTEM STATUS

Retrieve comprehensive system status information in HTML format.

Endpoint: `GET /api/status`

Parameters: None

Example Request:

```
curl "http://192.168.1.100/api/status"
```

Response: HTML formatted system information

2.5.2 GET DEVICE STATUS

Get comprehensive device status in JSON format.

Endpoint: `GET /api/device/status`

Parameters: None

Example Request:

```
curl "http://192.168.1.100/api/device/status"
```

Response:


```
{
  "device_id": "esp32_irrigation",
  "ip_address": "192.168.1.100",
  "mac_address": "AA:BB:CC:DD:EE:FF",
  "wifi_ssid": "MyNetwork",
  "wifi_rssi": -45,
  "heap_free": 180000,
  "uptime": 3600000,
  "firmware_version": "1.0.0",
  "zones": [
    {
      "zone": 1,
      "status": "idle",
      "time_remaining": 0
    }
  ]
}
```

2.6 Configuration Management

2.6.1 GET CONFIGURATION

Retrieve current system configuration.

Endpoint: `GET /api/config`

Parameters: None

Example Request:

```
curl "http://192.168.1.100/api/config"
```

Response:


```
{
  "timezone_offset": 9.5,
  "max_enabled_zones": 8,
  "mqtt_enabled": true,
  "mqtt_broker": "172.17.254.10",
  "mqtt_port": 1883,
  "mqtt_topic_prefix": "irrigation/",
  "mqtt_retain": true,
```

```
  "mqtt_keep_alive": 60,
  "scheduling_enabled": true
}
```

2.6.2 SET CONFIGURATION

Update system configuration settings.

Endpoint: `POST /api/config`

Content-Type: `application/json`

Parameters (JSON body):

Parameter	Type	Description
`timezone_offset`	Float	Timezone offset in hours
`max_enabled_zones`	Integer	Maximum zones (1-16)
`mqtt_broker`	String	MQTT broker address
`mqtt_port`	Integer	MQTT broker port
`mqtt_enabled`	Boolean	Enable MQTT
`scheduling_enabled`	Boolean	Enable scheduling

Example Request:

```
curl -X POST "http://192.168.1.100/api/config" \
     -H "Content-Type: application/json" \
     -d '{
       "timezone_offset": 9.5,
       "max_enabled_zones": 8,
       "mqtt_broker": "172.17.254.10",
       "mqtt_port": 1883
     }'
```

2.6.3 GET MQTT CONFIGURATION

Retrieve MQTT-specific configuration.

Endpoint: `GET /api/mqtt/config`

Parameters: None

Example Request:

```
curl "http://192.168.1.100/api/mqtt/config"
```

Response:

```
{
  "mqtt_enabled": true,
  "mqtt_broker": "172.17.254.10",
  "mqtt_port": 1883,
  "mqtt_username": "user",
  "mqtt_topic_prefix": "irrigation/",
  "mqtt_retain": true,
  "mqtt_keep_alive": 60
}
```

2.6.4 SET MQTT CONFIGURATION

Update MQTT-specific configuration.

Endpoint: `POST /api/mqtt/config`

Content-Type: `application/json`

Example Request:

```
curl -X POST "http://192.168.1.100/api/mqtt/config" \
     -H "Content-Type: application/json" \
     -d '{
       "mqtt_broker": "172.17.254.20",
       "mqtt_port": 1883,
       "mqtt_username": "newuser",
       "mqtt_password": "newpass"
     }'
```

2.7 Schedule Management

2.7.1 GET ALL SCHEDULES

Retrieve all configured irrigation schedules.

Endpoint: `GET /api/schedules`

Parameters: None

Example Request:

```
curl "http://192.168.1.100/api/schedules"
```

Response:

```
{
  "schedules": [
    {
      "id": 1,
      "zone": 1,
      "days": [1, 2, 3, 4, 5],
      "start_hour": 6,
      "start_minute": 30,
      "duration": 15,
      "enabled": true
    }
  ]
}
```

2.7.2 CREATE SCHEDULE

Create a new irrigation schedule.

Endpoint: `POST /api/schedules`

Content-Type: `application/json`

Parameters:

Parameter	Type	Required	Description
`zone`	Integer	Yes	Zone number (1-16)
`days`	Array	Yes	Days of week (1-7)
`start_hour`	Integer	Yes	Start hour (0-23)
`start_minute`	Integer	Yes	Start minute (0-59)
`duration`	Integer	Yes	Duration in minutes
`enabled`	Boolean	Yes	Enable schedule

Example Request:

```
curl -X POST "http://192.168.1.100/api/schedules" \
     -H "Content-Type: application/json" \
     -d '{
       "zone": 1,
       "days": [1,2,3,4,5],
       "start_hour": 6,
       "start_minute": 30,
       "duration": 15,
       "enabled": true
     }'
```

2.7.3 GET ACTIVE ZONES

Get information about currently running zones.

Endpoint: `GET /api/schedules/active`

Parameters: None

Example Request:

```
curl "http://192.168.1.100/api/schedules/active"
```

Response:

```
{
  "active_zones": [
    {
      "zone": 3,
      "time_remaining": 840,
      "start_time": "2025-11-06T14:00:00+09:30"
    }
  ]
}
```

2.7.4 SET AI SCHEDULES

Set schedules generated by AI or automation systems.

Endpoint: `POST /api/schedules/ai`

Content-Type: `application/json`

Example Request:

```
curl -X POST "http://192.168.1.100/api/schedules/ai" \
     -H "Content-Type: application/json" \
     -d '{
       "schedules": [
         {
           "zone": 1,
           "start_time": "06:30",
           "duration": 15,
           "days": "MTWTFSS"
         }
       ]
     }'
```

2.7.5 CLEAR AI SCHEDULES

Remove all AI-generated schedules.

Endpoint: `DELETE /api/schedules/ai`

Parameters: None

Example Request:

```
curl -X DELETE "http://192.168.1.100/api/schedules/ai"
```

2.8 Device Commands

2.8.1 GET NEXT SCHEDULED EVENT

Get information about the next scheduled irrigation event.

Endpoint: `GET /api/device/next`

Parameters: None

Example Request:

```
curl "http://192.168.1.100/api/device/next"
```

Response:

```
{
  "next_event": {
    "zone": 2,
    "scheduled_time": "2025-11-07T06:30:00+09:30",
    "duration": 15
  }
}
```

2.8.2 SEND DEVICE COMMAND

Send control commands to the device.

Endpoint: `POST /api/device/command`

Content-Type: `application/json`

Available Commands:

Command	Parameter	Description
`restart`	None	Restart ESP32
`rain_delay`	`minutes` (Integer)	Set rain delay
`clear_rain`	None	Clear rain delay

Example Request:

```
# Restart device
curl -X POST "http://192.168.1.100/api/device/command" \
     -H "Content-Type: application/json" \
     -d '{"command": "restart"}'

# Set rain delay
curl -X POST "http://192.168.1.100/api/device/command" \
     -H "Content-Type: application/json" \
     -d '{"command": "rain_delay", "minutes": 120}'
```

---

3. Improvement Suggestions

3.1 MQTT Interface Improvements

3.1.1 High Priority

1. **Add Zone Control via MQTT**
   - **Current Gap:** Zone start/stop operations only available via REST API
   - **Recommendation:** Add command topics for zone control

   ```
   Topic: irrigation/esp32_irrigation/command/zone/start
   Payload: {"zone": 1, "duration": 15}

   Topic: irrigation/esp32_irrigation/command/zone/stop
   Payload: {"zone": 1}
   ```

2. **Implement MQTT Authentication**
   - **Current:** Password stored but not enforced in some paths
   - **Recommendation:** Mandatory authentication with username/password
   - Add TLS/SSL support for encrypted connections
3. **Add QoS Level Configuration**
   - **Current:** Fixed QoS 0 (fire and forget)
   - **Recommendation:** Configurable QoS levels (0, 1, 2) for critical messages
   - Use QoS 1 for commands, QoS 0 for status updates
4. **Implement Last Will and Testament (LWT)**
   - **Current:** No offline detection
   - **Recommendation:**

```
LWT Topic: irrigation/esp32_irrigation/status/availability
LWT Payload: "offline"
Online Payload: "online"
```

3.1.2 Medium Priority

5. **Add Message Timestamps**
   - Include UTC timestamp in all published messages
   - Enable time-series analysis and late message detection
6. **Implement Batch Status Updates**
   - Reduce MQTT traffic by batching non-critical updates
   - Configurable batch interval (1-60 seconds)
7. **Add Diagnostic Topics**
   - Publish error logs, warnings, and debug information

```
Topic: irrigation/esp32_irrigation/diagnostic/error
Topic: irrigation/esp32_irrigation/diagnostic/warning
```

8. **Implement Command Acknowledgments**
   - Publish ACK/NACK for all received commands

```
Topic: irrigation/esp32_irrigation/command/ack
Payload: {"command": "zone_start", "status": "success", "message":
"Zone 1 started"}
```

3.1.3 Low Priority

9. **Add Historical Data Topics**
   - Publish daily/weekly summaries
   - Water usage statistics per zone
10. **Implement Remote Logging**
    - Forward system logs via MQTT for centralized monitoring

---

3.2 REST API Improvements

3.2.1 High Priority

1. **Add Authentication and Authorization**
   - **Current:** No authentication
   - **Recommendation:**
     - API key authentication via header: `X-API-Key: your_key`
     - Optional username/password with JWT tokens
     - Role-based access control (read-only, operator, admin)
2. **Implement Rate Limiting**
   - **Current:** Unlimited requests
   - **Recommendation:**
     - Limit to 60 requests per minute per IP
     - Return HTTP 429 (Too Many Requests) when exceeded

        - Include `X-RateLimit-*` headers
3. **Add Request/Response Validation**
    - **Current:** Limited input validation
    - **Recommendation:**
        - JSON schema validation for POST/PUT requests
        - Detailed error messages with field-specific feedback
        - Return HTTP 422 (Unprocessable Entity) for validation errors
4. **Standardize JSON Response Format**
    - **Current:** Mix of plain text and JSON responses
    - **Recommendation:** Consistent JSON structure:

```json
{
  "success": true,
  "data": { ... },
  "message": "Operation completed successfully",
  "timestamp": "2025-11-06T14:30:00Z"
}
```

5. **Add HTTPS Support**
    - **Current:** HTTP only
    - **Recommendation:**
        - Self-signed certificate for local access
        - Let's Encrypt for internet-facing deployments
        - Redirect HTTP to HTTPS

3.2.2 Medium Priority

6. **Implement API Versioning**
    - **Current:** No versioning
    - **Recommendation:**
        - URL-based: `/api/v1/`, `/api/v2/`
        - Header-based: `Accept: application/vnd.irrigation.v1+json`
        - Maintain backward compatibility for at least one major version
7. **Add Pagination for List Endpoints**
    - **Current:** Returns all records
    - **Recommendation:**

```
GET /api/schedules?page=1&limit=10
Response: {
  "data": [...],
  "pagination": {
    "page": 1,
    "limit": 10,
    "total": 45,
    "pages": 5
  }
}
```

8. **Implement PATCH Method for Partial Updates**
    - **Current:** Only full object updates

- **Recommendation:** Support partial updates

```
PATCH /api/config
{"max_enabled_zones": 12}
```

9. **Add Filtering and Sorting**
   - Enable query parameters for list endpoints

```
GET /api/schedules?zone=1&enabled=true&sort=start_time
```

10. **Implement OPTIONS Method**
    - Return available methods and parameters for each endpoint
    - Support CORS for web applications

3.2.3 Low Priority

11. **Add Webhook Support**
    - Allow registration of external URLs for event notifications
    - POST to webhook URL on zone start/stop, errors, etc.
12. **Implement GraphQL Endpoint**
    - Alternative to REST for flexible data queries
    - Single endpoint with query language
13. **Add Bulk Operations**
    - Start/stop multiple zones in single request
    - Batch schedule creation
14. **Implement ETag/Cache Headers**
    - Support HTTP caching with ETag headers
    - Reduce bandwidth for frequently accessed resources

3.3 General System Improvements

3.3.1 Security

1. **Network Security**
   - Implement MAC address filtering
   - Add firewall rules configuration via API
   - Support VPN tunnel configuration
2. **Audit Logging**
   - Log all configuration changes
   - Track zone activations with user/source
   - Store logs locally and forward to syslog server
3. **Secure Credential Storage**
   - Encrypt stored passwords (MQTT, WiFi)
   - Use secure element (if hardware supports)

3.3.2 Reliability

4. **Watchdog Timer**
   - Implement hardware watchdog for crash recovery
   - Automatic restart on hang/freeze

5. **Configuration Backup**
   - Periodic backup to SD card or cloud
   - Export/import configuration via API
6. **Failsafe Modes**
   - Manual override switch detection
   - Fallback to default schedule on RTC failure
   - Continue basic operations if MQTT/WiFi unavailable

3.3.3 Monitoring

7. **Health Check Endpoint**

```
GET /api/health
Response: {
  "status": "healthy",
  "checks": {
    "rtc": "ok",
    "wifi": "ok",
    "mqtt": "connected",
    "memory": "ok"
  }
}
```

8. **Metrics Endpoint**

   - Prometheus-compatible metrics
   - Zone run counts, durations, error rates

```
GET /api/metrics
```

9. **Alert Configuration**

   - Define alert conditions (low memory, connection loss)
   - Send alerts via MQTT, email, or push notification

3.3.4 Usability

10. **Web Dashboard**
    - Built-in web interface served by ESP32
    - Real-time zone status visualization
    - Configuration UI
11. **Schedule Templates**
    - Pre-defined schedule templates (lawn, garden, etc.)
    - Import/export schedules
12. **Zone Grouping**
    - Create zone groups for simultaneous control
    - Master valve support
13. **Water Budget/Flow Tracking**
    - Integrate with flow meters
    - Track water consumption per zone
    - Budget-based scheduling adjustments

3.4 Protocol Comparison

Feature	MQTT	REST API	Recommendation
Zone Control	❌ Missing	✅ Available	Add to MQTT
Real-time Updates	✅ Pub/Sub	❌ Polling	Keep MQTT
Authentication	⚠️ Partial	❌ None	Implement both
Bidirectional	✅ Yes	⚠️ Request only	Keep MQTT
Browser Compatible	❌ No	✅ Yes	Keep REST
Bandwidth Efficiency	✅ High	⚠️ Medium	Prefer MQTT
Learning Curve	⚠️ Moderate	✅ Easy	Support both

Summary: Maintain both interfaces with feature parity. Use MQTT for real-time monitoring and automation, REST API for manual control and web interfaces.

3.5 Implementation Priority Matrix

Priority	MQTT Improvements	REST Improvements	Timeline
Critical	Add zone control commands	Add authentication	1-2 weeks
High	Implement LWT, QoS config	Add HTTPS, rate limiting	2-4 weeks
Medium	Add diagnostics, ACKs	Standardize responses, versioning	1-2 months
Low	Historical data topics	GraphQL, webhooks	3-6 months

---

4. Appendices

4.1 Complete Topic Reference

Published Topics

```
irrigation/esp32_irrigation/config/device
irrigation/esp32_irrigation/status/device
irrigation/esp32_irrigation/status/schedules
irrigation/esp32_irrigation/status/zone/{1-16}
irrigation/esp32_irrigation/status/config
homeassistant/switch/esp32_irrigation/config
```

Subscribed Topics

```
irrigation/esp32_irrigation/config/{setting}/set
irrigation/esp32_irrigation/command/{command}
irrigation/esp32_irrigation/schedule/set
irrigation/esp32_irrigation/schedule/ai/set
```

4.2 Complete REST Endpoint Reference

Zone Control

- `GET /api/start-zone?zone={n}&time={m}`
- `GET /api/stop-zone?zone={n}`

Program Control

- `GET /api/run-program?program={n}`

Time Management

- `GET /api/time`
- `POST /api/set-time`
- `GET /api/sync-ntp`

Status

- `GET /api/status`
- `GET /api/device/status`
- `GET /api/device/next`

Configuration

- `GET /api/config`
- `POST /api/config`
- `GET /api/mqtt/config`
- `POST /api/mqtt/config`

Schedules

- `GET /api/schedules`
- `POST /api/schedules`
- `GET /api/schedules/active`
- `POST /api/schedules/ai`
- `DELETE /api/schedules/ai`

Commands

- `POST /api/device/command`

4.3 Error Codes Reference

HTTP Status Codes

Code	Meaning	Example
200	Success	Request completed successfully
400	Bad Request	Invalid zone number
401	Unauthorized	Authentication required (future)
404	Not Found	Endpoint does not exist
422	Unprocessable Entity	Validation error (future)
429	Too Many Requests	Rate limit exceeded (future)
500	Internal Server Error	System error

MQTT Result Payloads

Payload	Meaning
`success`	Command executed successfully
`error`	Command failed
`invalid`	Invalid parameters
`timeout`	Operation timed out

4.4 Testing Checklist

MQTT Testing

- ☐ Connect to broker with credentials
- ☐ Subscribe to all status topics
- ☐ Verify periodic status updates (every minute)
- ☐ Send configuration change command
- ☐ Send device restart command
- ☐ Set rain delay via MQTT
- ☐ Update schedule via MQTT
- ☐ Verify LWT message on disconnect
- ☐ Test QoS levels
- ☐ Verify retained messages persist

REST API Testing

- [ ] Start zone via GET request
- [ ] Stop zone via GET request
- [ ] Run program via GET request
- [ ] Get current time
- [ ] Set time via POST
- [ ] Sync with NTP
- [ ] Get device status JSON
- [ ] Get configuration JSON
- [ ] Update configuration via POST
- [ ] Create new schedule via POST
- [ ] Get active zones
- [ ] Set AI schedules
- [ ] Clear AI schedules
- [ ] Get next scheduled event
- [ ] Send device commands

Integration Testing

- [ ] Control zone via REST, verify MQTT status update
- [ ] Configure via MQTT, verify via REST API
- [ ] Test Home Assistant integration
- [ ] Test Node-RED flows
- [ ] Verify timezone handling
- [ ] Test RTC synchronization
- [ ] Verify schedule execution
- [ ] Test rain delay functionality

4.5 Troubleshooting Guide

MQTT Connection Issues

Problem: Device not connecting to broker

Solutions: 1. Verify broker IP and port: `ping 172.17.254.10` 2. Check credentials if authentication enabled 3. Verify firewall allows port 1883 4. Check broker logs: `journalctl -u mosquitto -f` 5. Test broker: `mosquitto_pub -h 172.17.254.10 -t test -m "hello"`

Problem: Messages not received

Solutions: 1. Verify subscription: `mosquitto_sub -h 172.17.254.10 -t "irrigation/#" -v` 2. Check topic prefix configuration 3. Verify retain flag settings 4. Check buffer size (must be ≥ payload size)

REST API Issues

Problem: 404 Not Found

Solutions: 1. Verify correct IP address 2. Check endpoint path (case-sensitive) 3. Ensure `/api/` prefix included 4. Verify device is on network: `ping 192.168.1.100`

Problem: No response / timeout

Solutions: 1. Check ESP32 is powered and running 2. Verify WiFi connection 3. Check firewall rules 4. Try different network/device

System Issues

Problem: Schedules not running

Solutions: 1. Verify RTC time: `GET /api/time` 2. Check schedule enabled: `GET /api/config` 3. Verify timezone offset 4. Check rain delay status 5. Review schedule configuration

Problem: High memory usage

Solutions: 1. Reduce MQTT publish frequency 2. Decrease buffer sizes 3. Limit number of active schedules 4. Restart device to clear memory leaks

---

4.6 Support and Resources

Documentation

- GitHub Repository: https://github.com/Zanoroy/esp32-irrigation-controller
- API Reference: This document
- Source Code: See repository `/src` directory

Community Support

- GitHub Issues: Report bugs and request features
- Email Support: (To be configured)

Development Tools

- PlatformIO: ESP32 development environment
- MQTT Explorer: GUI MQTT client for testing
- Postman: REST API testing tool
- curl: Command-line HTTP client
- mosquitto_pub/sub: MQTT command-line tools

4.7 Glossary

Term	Definition
MQTT	Message Queuing Telemetry Transport - lightweight pub/sub protocol
REST	Representational State Transfer - HTTP-based API architecture
QoS	Quality of Service - MQTT message delivery guarantee level
LWT	Last Will and Testament - MQTT message sent on unexpected disconnect
RTC	Real-Time Clock - hardware clock for timekeeping
NTP	Network Time Protocol - internet time synchronization
JSON	JavaScript Object Notation - data interchange format
API	Application Programming Interface
ESP32	Microcontroller with WiFi and Bluetooth capabilities
Home Assistant	Open-source home automation platform
Node-RED	Flow-based automation tool
ISO 8601	International date/time format standard
JWT	JSON Web Token - authentication token format
CORS	Cross-Origin Resource Sharing - browser security mechanism
TLS/SSL	Transport Layer Security - encryption protocol

Document Version: 1.0
Last Updated: November 6, 2025
Firmware Version: 1.0.0
API Version: 1.0

End of Documentation