SOURCE CODE - Real-Time Smart Agriculture System on µT-Kernel 3.0

Entry ID : 34686

1. Two Arduino/ESP32 sketches (one for the **ESP32 WROOM controller** with sensors & relays, one for the **ESP32-CAM** that captures & uploads images), both using MQTT to the same cloud broker + HTTP upload for camera images; and
2. a ready-to-adapt **Renesas EK-RA8D1** application (using Renesas Flexible Software Package / coreMQTT style) that subscribes to sensor topics, displays/uses the data, and publishes control commands (pump on/off, capture) back to the ESP devices through the broker — so the EK-RA8D1 can orchestrate automated agriculture via cloud commands.

I'll give full code for each device, topics and JSON formats, wiring & config notes, and build / flash instructions for the Renesas board (references to Renesas docs included).

---

1) Topics / JSON convention

Use this set of MQTT topics and payload conventions so all devices interoperate:

- farm1/sensors — device → broker (JSON):

{"device":"esp32-controller","soil_pct":42.1,"temp_c":28.34,"humidity":56.2,"pump":"ON"}

- farm1/pump — device → broker (string): "ON" or "OFF" (published by controller)
- farm1/commands — controller/camera/RA manager subscribe for commands (string):
  - "pump_on" / "pump_off" — start/stop pump
  - "capture" or "1" — make camera capture & upload
- farm1/camera/capture — RA or any client may publish "1" (request)
- farm1/camera/status — camera → broker (string) uploaded, upload_failed, capture_failed
- farm1/alerts — controller → broker string messages for alerts (e.g. pump_auto_stopped_max_runtime)

Use an MQTT broker reachable from both your LAN/cloud (Mosquitto, HiveMQ Cloud, AWS IoT, etc.). For local testing, Mosquitto on a LAN machine is easiest.

---

2) ESP32 WROOM controller (sensors + relay + MQTT)

This is your **controller.ino** (improved & split into one sketch). It reads soil ADC, SHT31, controls 4 relays, subscribes to farm1/commands, publishes sensors JSON, and receives camera capture requests (it forwards capture requests to the camera topic).

Notes/changes from your snippets

- Clean separation of raw ADC read vs percent mapping.

- Configurable MQTT/client reconnect logic.
- Use String only minimally in the MQTT publish to reduce heap pressure.
- Camera capture request is forwarded to farm1/camera/capture to trigger camera device.

```
/* controller_combined.ino

  ESP32 WROOM controller:

  - Capacitive soil moisture (ADC GPIO34)

  - SHT31 (I2C SDA 21, SCL 22)

  - 4 relays (GPIO26,27,32,25)

  - MQTT publish/subscribe

  - Basic irrigation with hysteresis and max runtime

*/


#include <WiFi.h>

#include <PubSubClient.h>

#include <Wire.h>

#include <Adafruit_SHT31.h>


// ---------------- CONFIG ----------------

const char* ssid = "YOUR_WIFI_SSID";

const char* password = "YOUR_WIFI_PASSWORD";


const char* mqtt_server = "YOUR_MQTT_BROKER_IP_OR_HOST";

const uint16_t mqtt_port = 1883;

const char* mqtt_user = "mqtt_user"; // or NULL

const char* mqtt_pass = "mqtt_pass"; // or NULL


const char* baseTopic = "farm1";


// pins

const int PIN_SOIL_ADC = 34; // ADC1_CH6
```

```
const int PIN_RELAY1 = 26; // pump (active LOW assumed)
const int PIN_RELAY2 = 27;
const int PIN_RELAY3 = 32;
const int PIN_RELAY4 = 25;
const int I2C_SDA = 21;
const int I2C_SCL = 22;

// irrigation params (tune for your sensor)
const int SOIL_DRY_ADC_THRESHOLD = 2000; // raw: > = dry
const int SOIL_WET_ADC_THRESHOLD  = 1200; // raw: < = wet
const unsigned long MAX_PUMP_RUNTIME_MS = 2UL * 60UL * 1000UL; // 2 minutes
const unsigned long SENSOR_PUBLISH_INTERVAL = 30UL * 1000UL; // 30s
// ----------------------------------------

WiFiClient espClient;
PubSubClient mqtt(espClient);
Adafruit_SHT31 sht31 = Adafruit_SHT31();

unsigned long lastPublish = 0;
unsigned long pumpStartTime = 0;
bool pumpOn = false;

void setPumpHardware(bool on) {
  // active LOW relay assumed - adapt if yours is active HIGH
  digitalWrite(PIN_RELAY1, on ? LOW : HIGH);
}

void setPump(bool on) {
  if (on && !pumpOn) {
    setPumpHardware(true);
    pumpStartTime = millis();
```

```
    pumpOn = true;
    mqtt.publish(String(baseTopic) + "/pump", "ON");
    Serial.println("Pump ON");
  } else if (!on && pumpOn) {
    setPumpHardware(false);
    pumpOn = false;
    mqtt.publish(String(baseTopic) + "/pump", "OFF");
    Serial.println("Pump OFF");
  }
}

void callback(char* topic, byte* payload, unsigned int length) {
  String t = String(topic);
  String msg;
  for (unsigned int i = 0; i < length; i++) msg += (char)payload[i];
  msg.trim();
  Serial.print("MQTT in: "); Serial.print(t); Serial.print(" => "); Serial.println(msg);

  if (t.equals(String(baseTopic) + "/commands")) {
    if (msg == "pump_on") setPump(true);
    else if (msg == "pump_off") setPump(false);
    else if (msg == "capture") {
      mqtt.publish(String(baseTopic) + "/camera/capture", "1");
    }
  } else if (t.equals(String(baseTopic) + "/camera/status")) {
    // camera replied; you may take action on it (log or update display)
    Serial.print("Camera status: "); Serial.println(msg);
  }
}

void connectWiFi() {
```

```
  Serial.printf("Connecting to WiFi %s\n", ssid);
  WiFi.begin(ssid, password);
  unsigned long started = millis();
  while (WiFi.status() != WL_CONNECTED) {
    delay(300);
    Serial.print(".");
    if (millis() - started > 20000UL) {
      Serial.println("\nWiFi connect timeout, restarting...");
      ESP.restart();
    }
  }
  Serial.println();
  Serial.print("Connected, IP: "); Serial.println(WiFi.localIP());
}

void connectMQTT() {
  mqtt.setServer(mqtt_server, mqtt_port);
  mqtt.setCallback(callback);
  while (!mqtt.connected()) {
    Serial.print("Connecting MQTT...");
    String clientId = "esp32_controller_";
    clientId += String((uint32_t)esp_random(), HEX);
    if (mqtt.connect(clientId.c_str(), mqtt_user, mqtt_pass)) {
      Serial.println("connected");
      mqtt.subscribe(String(baseTopic) + "/commands");
      mqtt.subscribe(String(baseTopic) + "/camera/status");
    } else {
      Serial.print("failed, rc=");
      Serial.print(mqtt.state());
      Serial.println(" try again in 3s");
      delay(3000);
```

```
    }
  }
}

void setupPins() {
  pinMode(PIN_RELAY1, OUTPUT);
  pinMode(PIN_RELAY2, OUTPUT);
  pinMode(PIN_RELAY3, OUTPUT);
  pinMode(PIN_RELAY4, OUTPUT);
  // default off (active LOW)
  digitalWrite(PIN_RELAY1, HIGH);
  digitalWrite(PIN_RELAY2, HIGH);
  digitalWrite(PIN_RELAY3, HIGH);
  digitalWrite(PIN_RELAY4, HIGH);
}

void setup() {
  Serial.begin(115200);
  delay(100);
  setupPins();

  analogReadResolution(12); // 0-4095
  analogSetPinAttenuation(PIN_SOIL_ADC, ADC_11db);

  Wire.begin(I2C_SDA, I2C_SCL);
  if (!sht31.begin(0x44)) {
    Serial.println("Couldn't find SHT31");
  } else {
    Serial.println("SHT31 found");
  }
```

```
  connectWiFi();
  connectMQTT();
  lastPublish = millis() - SENSOR_PUBLISH_INTERVAL;
}

int readSoilRaw() {
  return analogRead(PIN_SOIL_ADC);
}

float soilRawToPercent(int raw) {
  // calibrate these values to your sensor
  const int rawWet = 200;   // raw value for fully wet
  const int rawDry = 3800;  // raw for fully dry
  int clamped = raw;
  if (clamped < rawWet) clamped = rawWet;
  if (clamped > rawDry) clamped = rawDry;
  float pct = 100.0 * (1.0 - float(clamped - rawWet) / float(rawDry - rawWet));
  if (pct < 0) pct = 0;
  if (pct > 100) pct = 100;
  return pct;
}

void publishSensorData(float soilPct, float tempC, float humidity) {
  char payload[256];
  snprintf(payload, sizeof(payload),
         "{\"device\":\"esp32-
controller\",\"soil_pct\":%.1f,\"temp_c\":%.2f,\"humidity\":%.2f,\"pump\":\"%s\"}",
         soilPct, tempC, humidity, pumpOn ? "ON":"OFF");
  mqtt.publish(String(baseTopic) + "/sensors", payload);
  Serial.print("Published: "); Serial.println(payload);
}
```

```
void loop() {
 if (WiFi.status() != WL_CONNECTED) connectWiFi();
 if (!mqtt.connected()) connectMQTT();
 mqtt.loop();

 unsigned long now = millis();
 if (now - lastPublish >= SENSOR_PUBLISH_INTERVAL) {
  lastPublish = now;
  int raw = readSoilRaw();
  float soilPct = soilRawToPercent(raw);

  float temp = NAN, hum = NAN;
  // Adafruit_SHT31 check
  if (sht31.isConnected()) {
   temp = sht31.readTemperature();
   hum = sht31.readHumidity();
  }

  publishSensorData(soilPct, temp, hum);

  // irrigation: raw comparison
  if (!pumpOn && raw > SOIL_DRY_ADC_THRESHOLD) {
   setPump(true);
  }

  if (pumpOn) {
   if (raw < SOIL_WET_ADC_THRESHOLD) {
    setPump(false);
   } else if (millis() - pumpStartTime > MAX_PUMP_RUNTIME_MS) {
    setPump(false);
```

```
      mqtt.publish(String(baseTopic) + "/alerts", "pump_auto_stopped_max_runtime");
    }
  }
}

delay(50);
}
```

Wiring notes

- Soil sensor → ADC pin 34 (input only). Provide ground & Vcc per sensor datasheet.
- SHT31 SDA→21, SCL→22, Vcc→3.3V, GND→GND.
- Relays → GPIO26/27/32/25. If using mechanical relay module power Vcc (5V) and ensure common GND.

---

3) ESP32-CAM (AI-Thinker) — capture & upload to cloud

This sketch subscribes to farm1/camera/capture. When it receives "1" or "capture" it captures a photo and uploads to your HTTP endpoint (PHP/Flask/etc.). It publishes farm1/camera/status with outcome.

```
/* esp32_cam_uploader_combined.ino
 AI-Thinker ESP32-CAM. subscribe to MQTT farm1/camera/capture.
 On command: capture and multipart POST upload to uploadURL.
*/

#include "esp_camera.h"
#include <WiFi.h>
#include <PubSubClient.h>
#include <HTTPClient.h>

// ---------- CONFIG ----------
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";

const char* mqtt_server = "YOUR_MQTT_BROKER_IP_OR_HOST";
```

```
const uint16_t mqtt_port = 1883;
const char* mqtt_user = "mqtt_user";
const char* mqtt_pass = "mqtt_pass";

const char* baseTopic = "farm1";
const char* uploadURL = "http://YOUR_SERVER/upload_image.php"; // change to your endpoint
// -----------------------------

// AI-Thinker board default pins (common mapping)
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27
#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22

WiFiClient espClient;
PubSubClient mqtt(espClient);

void callback(char* topic, byte* payload, unsigned int length) {
  String t = String(topic);
```

```
  String msg;
  for (unsigned int i=0; i<length; i++) msg += (char)payload[i];
  msg.trim();
  Serial.printf("MQTT in: %s => %s\n", t.c_str(), msg.c_str());
  if (t == String(baseTopic) + "/camera/capture") {
    if (msg == "1" || msg == "capture") {
      captureAndUpload();
    }
  }
}

void connectWiFi() {
  WiFi.begin(ssid, password);
  Serial.print("Connecting WiFi");
  unsigned long started = millis();
  while (WiFi.status() != WL_CONNECTED) {
    delay(300);
    Serial.print(".");
    if (millis() - started > 20000UL) {
      Serial.println("\nWiFi connect timeout, restarting...");
      ESP.restart();
    }
  }
  Serial.println("\nConnected. IP: " + WiFi.localIP().toString());
}

void connectMQTT() {
  mqtt.setServer(mqtt_server, mqtt_port);
  mqtt.setCallback(callback);
  while (!mqtt.connected()) {
    Serial.print("Connecting MQTT...");
```

```
    String clientId = "esp32cam_";
    clientId += String((uint32_t)esp_random(), HEX);
    if (mqtt.connect(clientId.c_str(), mqtt_user, mqtt_pass)) {
      Serial.println("connected");
      mqtt.subscribe(String(baseTopic) + "/camera/capture");
    } else {
      Serial.print("failed rc=");
      Serial.print(mqtt.state());
      Serial.println(" try again in 3s");
      delay(3000);
    }
  }
}

void initCamera() {
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
```

```
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size = FRAMESIZE_VGA; // reduce bandwidth
  config.jpeg_quality = 12; // adjust
  config.fb_count = 1;

  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed 0x%x\n", err);
    while (true) { delay(1000); }
  }
}

// upload using HTTP multipart; server must accept multipart image field "image"
bool uploadImage(uint8_t *buf, size_t len) {
  HTTPClient http;
  String boundary = "----ESP32Boundary";
  http.begin(uploadURL);
  http.addHeader("Content-Type", "multipart/form-data; boundary=" + boundary);

  WiFiClient *client = http.getStreamPtr();
  client->print("--" + boundary + "\r\n");
  client->print("Content-Disposition: form-data; name=\"image\"; filename=\"img.jpg\"\r\n");
  client->print("Content-Type: image/jpeg\r\n\r\n");
  client->write(buf, len);
  client->print("\r\n--" + boundary + "--\r\n");

  int httpCode = http.POST(""); // server should accept
```

```
  if (httpCode > 0) {
    String resp = http.getString();
    Serial.printf("Upload httpCode=%d resp=%s\n", httpCode, resp.c_str());
    mqtt.publish(String(baseTopic) + "/camera/status", "uploaded");
    http.end();
    return true;
  } else {
    Serial.printf("Upload failed, error: %s\n", http.errorToString(httpCode).c_str());
    mqtt.publish(String(baseTopic) + "/camera/status", "upload_failed");
    http.end();
    return false;
  }
}

void captureAndUpload() {
  camera_fb_t * fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Camera capture failed");
    mqtt.publish(String(baseTopic) + "/camera/status", "capture_failed");
    return;
  }
  Serial.printf("Captured %u bytes\n", fb->len);
  bool ok = uploadImage(fb->buf, fb->len);
  esp_camera_fb_return(fb);
  if (ok) Serial.println("Image uploaded.");
}

void setup() {
  Serial.begin(115200);
  connectWiFi();
  connectMQTT();
```

```
  initCamera();
}


void loop() {
  if (WiFi.status() != WL_CONNECTED) connectWiFi();
  if (!mqtt.connected()) connectMQTT();
  mqtt.loop();
  delay(10);
}
```

Server-side: upload_image.php (or Flask endpoint) must accept POST multipart/form-data file field named image. Save to disk or S3 and return 200.

---

4) Renesas EK-RA8D1 application (MQTT client & cloud bridge)

Goal: EK-RA8D1 runs a networked application that subscribes to farm1/sensors, farm1/camera/status and can publish commands to farm1/commands or farm1/camera/capture. The board typically runs code using Renesas Flexible Software Package (FSP) and uses the coreMQTT / AWS MQTT integration provided by Renesas. Use e2 studio to build and flash.

I'll provide a clear sample C source (single-file sketch-like) showing using FSP networking + coreMQTT APIs. **You will need to import this into a Renesas FSP project** and enable the appropriate middleware (IPv4/v4 Wi-Fi or Ethernet, TLS if required, coreMQTT) in the FSP configuration.

Note: Renesas provides example application projects for MQTT in the FSP; refer to the EK-RA8D1 product pages and RA FSP MQTT docs while integrating. Renesas Electronics+1

4A) High-level steps (prepare FSP project)

1. Install **e2 studio** and **Renesas Flexible Software Package (FSP)**.
2. Create a new RA FSP project for **EK-RA8D1** (select board EK-RA8D1 in new project wizard).
3. In the FSP configurator (RBA/GUI), enable:
    - Network stack (Ethernet or Wi-Fi interface available on your board).
    - coreMQTT or AWS MQTT integration module (example exists in FSP).
    - Sockets / TLS (mbedTLS or wolfSSL) if using MQTT over TLS.
    - A simple JSON parser (you can add jsmn or cJSON as a middleware).
4. Configure network interface to obtain IP via DHCP or set static IP.

5. Fill broker address, port, username/password in the MQTT config.
6. Add the C source below as an application file, adjust include paths and project settings, build and flash.

4B) Application code (conceptual FSP + coreMQTT)

This is a practical template. You will need to adapt to FSP callback names and project layout (e.g., R_SCI, sf_mqtt, or coreMQTT integration). The code focuses on subscription callback, JSON parsing (lightweight), and publishing commands.

Important: This code uses coreMQTT-style APIs and a simple string parser for the sensor JSON. Replace mqtt_connect(), mqtt_subscribe() and mqtt_publish() with your actual FSP wrapper functions (Renesas example uses aws_mqtt wrapper names). The code is intentionally complete and clear so you can map functions to your FSP project easily.

```
/* ek_ra8d1_mqtt_controller.c

  Example: EK-RA8D1 subscribes to farm1/sensors & camera status,

  displays/parses sensor JSON and publishes commands (pump_on/pump_off/capture).

  Adapt and integrate into an RA FSP project (e2 studio).

*/


#include <stdio.h>

#include <string.h>

#include <stdint.h>

#include <stdbool.h>

#include "rm_mqtt_api.h"   // replace with your FSP MQTT header or coreMQTT wrapper

#include "r_os_abstraction_api.h" // if using OS threads

// include your project's JSON parser (cJSON or jsmn)

#include "cJSON.h" // if you have it integrated


/* CONFIG - match your FSP config */

#define MQTT_BROKER_HOST "YOUR_MQTT_BROKER_IP_OR_HOST"

#define MQTT_BROKER_PORT 1883

#define MQTT_USERNAME    "mqtt_user"

#define MQTT_PASSWORD    "mqtt_pass"


static rm_mqtt_client_t g_mqtt_client; // replace with your project's mqtt client struct
```

```
/* topics */
const char *TOPIC_SENSORS = "farm1/sensors";
const char *TOPIC_COMMANDS = "farm1/commands";
const char *TOPIC_CAMERA_CAPTURE = "farm1/camera/capture";
const char *TOPIC_CAMERA_STATUS = "farm1/camera/status";
const char *TOPIC_PUMP = "farm1/pump";
const char *TOPIC_ALERTS = "farm1/alerts";

void mqtt_message_callback(const char *topic, const uint8_t *payload, size_t payload_len)
{
    // topic string, payload (not null terminated)
    char msgbuf[512];
    size_t n = payload_len < sizeof(msgbuf)-1 ? payload_len : sizeof(msgbuf)-1;
    memcpy(msgbuf, payload, n);
    msgbuf[n] = '\0';

    printf("MQTT RX topic=%s payload=%s\n", topic, msgbuf);

    if (strcmp(topic, TOPIC_SENSORS) == 0) {
        // parse JSON (using cJSON for clarity; integrate library into project)
        cJSON *root = cJSON_Parse(msgbuf);
        if (root) {
            cJSON *device = cJSON_GetObjectItem(root, "device");
            cJSON *soil = cJSON_GetObjectItem(root, "soil_pct");
            cJSON *temp = cJSON_GetObjectItem(root, "temp_c");
            cJSON *hum  = cJSON_GetObjectItem(root, "humidity");
            cJSON *pump = cJSON_GetObjectItem(root, "pump");

            if (soil && temp && hum) {
                printf("Device:%s Soil:%.1f Temp:%.2f Hum:%.2f Pump:%s\n",
```

```
                device ? device->valuestring : "unknown",
                soil->valuedouble, temp->valuedouble, hum->valuedouble,
                pump ? pump->valuestring : "N/A");

            // Example automation policy on RA: if soil < 30% request pump
            double soil_pct = soil->valuedouble;
            if (soil_pct < 30.0) {
                // publish pump_on command to cloud
                const char *cmd = "pump_on";
                rm_mqtt_publish(&g_mqtt_client, TOPIC_COMMANDS, (const uint8_t*)cmd, strlen(cmd),
1);
                printf("Published command: %s\n", cmd);
            } else if (soil_pct > 45.0) {
                const char *cmd = "pump_off";
                rm_mqtt_publish(&g_mqtt_client, TOPIC_COMMANDS, (const uint8_t*)cmd, strlen(cmd),
1);
                printf("Published command: %s\n", cmd);
            }
          }
          cJSON_Delete(root);
      }
  } else if (strcmp(topic, TOPIC_CAMERA_STATUS) == 0) {
      // camera status updates (uploaded, upload_failed etc.)
      printf("Camera status: %s\n", msgbuf);
  }
}

/* wrapper publish function (adapt to FSP's mqtt publish) */
void mqtt_publish_text(const char *topic, const char *text)
{
  rm_mqtt_publish(&g_mqtt_client, topic, (const uint8_t*)text, strlen(text), 1);
}
```

```
/* initialize and connect MQTT - adapt to your platform functions */
bool mqtt_connect_and_subscribe(void)
{
  rm_mqtt_config_t cfg = {0};
  // fill cfg from FSP settings (hostname, port, user/pass, callback)
  cfg.broker_host = MQTT_BROKER_HOST;
  cfg.broker_port = MQTT_BROKER_PORT;
  cfg.username = MQTT_USERNAME;
  cfg.password = MQTT_PASSWORD;
  cfg.message_callback = mqtt_message_callback; // your wrapper's callback pattern

  if (rm_mqtt_init(&g_mqtt_client, &cfg) != RM_MQTT_SUCCESS) {
    printf("MQTT init failed\n");
    return false;
  }

  if (rm_mqtt_connect(&g_mqtt_client) != RM_MQTT_SUCCESS) {
    printf("MQTT connect failed\n");
    return false;
  }

  // subscribe
  rm_mqtt_subscribe(&g_mqtt_client, TOPIC_SENSORS, 1);
  rm_mqtt_subscribe(&g_mqtt_client, TOPIC_CAMERA_STATUS, 1);

  printf("MQTT connected and subscribed\n");
  return true;
}

/* main app entry (adapt to RTOS or bare-metal main) */
```

```
void app_main(void)
{
  // 1) initialize board, network (DHCP), etc. Usually done in FSP generated code.
  // 2) ensure network is up before mqtt_connect_and_subscribe()

  // Wait for network ready (example; replace with your event/wait)
  // ... network bring-up code ...

  if (!mqtt_connect_and_subscribe()) {
    printf("MQTT startup failed\n");
    // retry or handle error
    return;
  }

  // example: periodically request camera capture (publish "1")
  while (1) {
    // e.g., hourly capture or on a button event — here we trigger every 10 minutes for demo
    const char *capture_cmd = "1";
    mqtt_publish_text(TOPIC_CAMERA_CAPTURE, capture_cmd);
    printf("Published camera capture request\n");

    // sleep for 600 seconds (10 minutes); in RTOS use vTaskDelay
    rm_os_delay_ms(600000);
  }
}
```

5) Build & flash instructions for EK-RA8D1 (summary)

1. Install e2 studio and FSP (download from Renesas site). Renesas Electronics
2. Create new project → choose board **EK-RA8D1** → pick a sample template or empty project.
3. In the FSP Configuration window, enable Networking (Ethernet/Wi-Fi) and the coreMQTT/AWS MQTT middleware. Configure broker host/port/credentials.

4. Add cJSON or similar to your project (import source).
5. Add the C file from **4B** to the project, adapt function names to FSP generated network/MQTT functions.
6. Build, connect the EK-RA8D1 via the debug USB port, and flash using e2 studio. Monitor serial output for MQTT messages.
7. Test: publish a sample sensors message from Mosquitto or from your ESP32 controller and watch RA logs. Then let the RA publish commands — the ESP32 controller will act on them.

---

6) End-to-end test plan

1. Start MQTT broker (Mosquitto) reachable by EK-RA8D1 and ESP devices.
2. Flash ESP32 controller and ESP32-CAM with the above sketches (set same MQTT broker + credentials).
3. Bring up EK-RA8D1 app. Check all devices register/subscribe.
4. Observe sensor publication to farm1/sensors. EK-RA8D1 should parse the JSON and publish pump_on / pump_off as per automation thresholds.
5. EK-RA8D1 or any client can publish 1 to farm1/camera/capture. ESP32-CAM captures and uploads image; camera publishes farm1/camera/status.
6. Tune thresholds, hysteresis and max runtime as required.

---

7) Security & production notes

- For production, use **MQTT over TLS** with client/server certs or a secure cloud broker (AWS IoT Core, Azure IoT Hub). Renesas FSP has TLS & coreMQTT examples; enable them.
- Use unique client IDs and handle offline cases (queueing messages locally, backoff strategies).
- Calibrate your soil sensor thresholds by measuring raw ADC on wet and dry soil samples and adjusting SOIL_DRY_ADC_THRESHOLD / SOIL_WET_ADC_THRESHOLD accordingly.
- If the camera upload endpoint is public, protect it (auth tokens / signed URLs / server-side rate limits).

---

8) References (useful docs)

- Renesas EK-RA8D1 evaluation kit page (board details & quick start).
- Renesas RA Flexible Software Package — AWS / coreMQTT documentation and examples (shows MQTT/TLS integrations and APIs used in FSP).
- EK-RA8D1 User Manual (hardware guide & pinout).