ChatGPT Prompts:

# Prompt:

This is my code: #include <Adafruit\_MPU6050.h> #include <Adafruit\_Sensor.h> #include <Wire.h> #include <WiFi.h> #include <PubSubClient.h> WiFiClient espClient; PubSubClient mqttClient(espClient); Adafruit\_MPU6050 mpu; int led\_red = 26; int led\_green = 14; int led\_yellow = 27; int brightness = 0; // how bright the LED is int fadeAmount = 5; // how many points to fade the LED by #define IC2\_SDA 33 #define I2C\_SCL 32 const char\* ssid = "MaraudersMap"; const char\* password = "Page394%"; //bei public broker = "broker" teiweise const char\* mqttBroker = "192.168.0.89"; const int mqttPort = 1883; const char\* mpuTopic = "mpu/K05"; const char\* tempTopic= "temp/K05"; const char\* finishedTopic = "finished/K05"; void onMqttMessageReceived(char\* topic, byte\* payload, unsigned int length) { // Handle MQTT message received // Convert payload to a string String message; for (unsigned int i = 0; i < length; i++) { message += (char)payload[i]; } // Display the received message in the console Serial.print("Received message on topic: "); Serial.print(topic); Serial.print(", payload: "); Serial.println(message); } void connectToWifi() { WiFi.begin(ssid, password); Serial.print("Connecting to Wi-Fi..."); while (WiFi.status() != WL\_CONNECTED) { delay(100); } Serial.println("connected!"); Serial.print("IP address: "); Serial.println(WiFi.localIP()); } void setupMqtt() { mqttClient.setServer(mqttBroker, mqttPort); mqttClient.setCallback(onMqttMessageReceived); Serial.print("Connecting to MQTT broker..."); while (!mqttClient.connected()) { if (mqttClient.connect("ESP32Client")) { Serial.println("connected!"); mqttClient.subscribe(mpuTopic); mqttClient.subscribe(tempTopic); mqttClient.subscribe(finishedTopic); } else { Serial.print("failed, retrying in 5 seconds..."); delay(5000); } } } void setup() { pinMode(led\_green, OUTPUT); pinMode(led\_red, OUTPUT); pinMode(led\_yellow, OUTPUT); //sets the pins for the MPU6050 chip Wire.begin(33,32); Serial.begin(115200); while (!Serial) delay(10); Serial.println("Adafruit MPU6050 test!"); if (!mpu.begin()) { Serial.println("Failed to find MPU6050 chip"); while (1) { delay(10); } } Serial.println("MPU6050 Found!"); mpu.setAccelerometerRange(MPU6050\_RANGE\_8\_G); Serial.print("Accelerometer range set to: +-8G"); mpu.setGyroRange(MPU6050\_RANGE\_500\_DEG); Serial.print("Gyro range set to: +- 1000 deg/s"); mpu.setFilterBandwidth(MPU6050\_BAND\_21\_HZ); Serial.print("Filter bandwidth set to: 44hz"); Serial.println(""); delay(3000); // Connect to Wi-Fi connectToWifi(); // Setup MQTT setupMqtt(); } void blink(){ digitalWrite(led\_green, HIGH); digitalWrite(led\_red, HIGH); digitalWrite(led\_yellow, HIGH); delay(30); digitalWrite(led\_green, LOW); digitalWrite(led\_red, LOW); digitalWrite(led\_yellow, LOW); } void loop() { // Get new sensor events with the readings sensors\_event\_t a, g, temp; mpu.getEvent(&a, &g, &temp); // Publish the g.gyro.x value String gyroXValue = String(g.gyro.x); mqttClient.publish(mpuTopic, gyroXValue.c\_str()); blink(); /\* Print out the values \*/ Serial.print("Acceleration X: "); Serial.print(a.acceleration.x); Serial.print(", Y: "); Serial.print(a.acceleration.y); Serial.print(", Z: "); Serial.print(a.acceleration.z); Serial.println(" m/s^2"); Serial.print("Rotation X: "); Serial.print(g.gyro.x); Serial.print(", Y: "); Serial.print(g.gyro.y); Serial.print(", Z: "); Serial.print(g.gyro.z); Serial.println(" rad/s"); Serial.print("Temperature: "); Serial.print(temp.temperature); Serial.println(" degC"); Serial.println(""); delay(100); // Process MQTT messages mqttClient.loop(); } . Please modify the code with a precise timer, so that it will publish the tempTopic at precisely every second.

# Answer:

To modify the code to publish the tempTopic at precisely every second, you can use a timer-based approach. Here's the modified code with a precise timer:

#include <Adafruit\_MPU6050.h>

#include <Adafruit\_Sensor.h>

#include <Wire.h>

#include <WiFi.h>

#include <PubSubClient.h>

WiFiClient espClient;

PubSubClient mqttClient(espClient);

Adafruit\_MPU6050 mpu;

int led\_red = 26;

int led\_green = 14;

int led\_yellow = 27;

int brightness = 0; // how bright the LED is

int fadeAmount = 5; // how many points to fade the LED by

#define IC2\_SDA 33

#define I2C\_SCL 32

const char\* ssid = "MaraudersMap";

const char\* password = "Page394%";

//bei public broker = "broker" teiweise

const char\* mqttBroker = "192.168.0.89";

const int mqttPort = 1883;

const char\* mpuTopic = "mpu/K05";

const char\* tempTopic= "temp/K05";

const char\* finishedTopic = "finished/K05";

unsigned long previousMillis = 0;

const unsigned long interval = 1000; // Interval in milliseconds

void onMqttMessageReceived(char\* topic, byte\* payload, unsigned int length) {

// Handle MQTT message received

// Convert payload to a string

String message;

for (unsigned int i = 0; i < length; i++) {

message += (char)payload[i];

}

// Display the received message in the console

Serial.print("Received message on topic: ");

Serial.print(topic);

Serial.print(", payload: ");

Serial.println(message);

}

void connectToWifi() {

WiFi.begin(ssid, password);

Serial.print("Connecting to Wi-Fi...");

while (WiFi.status() != WL\_CONNECTED) {

delay(100);

}

Serial.println("connected!");

Serial.print("IP address: ");

Serial.println(WiFi.localIP());

}

void setupMqtt() {

mqttClient.setServer(mqttBroker, mqttPort);

mqttClient.setCallback(onMqttMessageReceived);

Serial.print("Connecting to MQTT broker...");

while (!mqttClient.connected()) {

if (mqttClient.connect("ESP32Client")) {

Serial.println("connected!");

mqttClient.subscribe(mpuTopic);

mqttClient.subscribe(tempTopic);

mqttClient.subscribe(finishedTopic);

} else {

Serial.print("failed, retrying in 5 seconds...");

delay(5000);

}

}

}

void setup() {

pinMode(led\_green, OUTPUT);

pinMode(led\_red, OUTPUT);

pinMode(led\_yellow, OUTPUT);

// Sets the pins for the MPU6050 chip

Wire.begin(33,32);

Serial.begin(115200);

while (!Serial)

delay(10);

Serial.println("Adafruit MPU6050 test!");

if (!mpu.begin()) {

Serial.println("Failed to find MPU6050 chip");

while (1) {

delay(10);

}

}

Serial.println("MPU6050 Found!");

mpu.setAccelerometerRange(MPU6050\_RANGE\_8\_G);

Serial.print("Accelerometer range set to: +-8G");

mpu.setGyroRange(MPU6050\_RANGE\_500\_DEG);

Serial.print("Gyro range set to: +- 1000 deg/s");

mpu.setFilterBandwidth(MPU6050\_BAND\_21\_HZ);

Serial.print("Filter bandwidth set to: 44hz");

Serial.println("");

delay(3000);

// Connect to Wi-Fi

connectToWifi();

// Setup MQTT

setupMqtt();

}

void blink(){

digitalWrite(led\_green, HIGH);

digitalWrite(led\_red, HIGH);

digitalWrite(led\_yellow, HIGH);

delay(30);

digitalWrite(led\_green, LOW);

digitalWrite(led\_red, LOW);

digitalWrite(led\_yellow, LOW);

}

void loop() {

unsigned long currentMillis = millis();

// Get new sensor events with the readings

sensors\_event\_t a, g, temp;

mpu.getEvent(&a, &g, &temp);

// Publish the g.gyro.x value

String gyroXValue = String(g.gyro.x);

mqttClient.publish(mpuTopic, gyroXValue.c\_str());

blink();

/\* Print out the values \*/

Serial.print("Acceleration X: ");

Serial.print(a.acceleration.x);

Serial.print(", Y: ");

Serial.print(a.acceleration.y);

Serial.print(", Z: ");

Serial.print(a.acceleration.z);

Serial.println(" m/s^2");

Serial.print("Rotation X: ");

Serial.print(g.gyro.x);

Serial.print(", Y: ");

Serial.print(g.gyro.y);

Serial.print(", Z: ");

Serial.print(g.gyro.z);

Serial.println(" rad/s");

// Check if it's time to publish the temperature

if (currentMillis - previousMillis >= interval) {

previousMillis = currentMillis;

// Publish the temp.temperature value

String tempValue = String(temp.temperature);

mqttClient.publish(tempTopic, tempValue.c\_str());

}

Serial.print("Temperature: ");

Serial.print(temp.temperature);

Serial.println(" degC");

Serial.println("");

delay(100);

// Process MQTT messages

mqttClient.loop();

}.

In this modified code, a timer-based approach is used to determine when to publish the tempTopic. The variable previousMillis keeps track of the previous time the temperature was published, and currentMillis holds the current time. By comparing the difference between currentMillis and previousMillis with the interval (which is set to 1000 milliseconds or 1 second), you can determine if it's time to publish the temperature again. If the condition currentMillis - previousMillis >= interval is true, the temperature is published, and previousMillis is updated with the current time.

Note: Make sure to include the required libraries and set the necessary configurations for Wi-Fi and MQTT according to your specific setup.

# Prompt:

thanks but that's not precise enough unfortunately. I need something like a global timer or something

# Answer:

If you need a more precise timing mechanism for publishing the tempTopic at exactly every second, you can utilize the ESP32's hardware timer. Here's the modified code that uses a hardware timer for precise timing:

#include <Adafruit\_MPU6050.h>

#include <Adafruit\_Sensor.h>

#include <Wire.h>

#include <WiFi.h>

#include <PubSubClient.h>

WiFiClient espClient;

PubSubClient mqttClient(espClient);

Adafruit\_MPU6050 mpu;

int led\_red = 26;

int led\_green = 14;

int led\_yellow = 27;

int brightness = 0; // how bright the LED is

int fadeAmount = 5; // how many points to fade the LED by

#define IC2\_SDA 33

#define I2C\_SCL 32

const char \*ssid = "MaraudersMap";

const char \*password = "Page394%";

const char \*mqttBroker = "192.168.0.89";

const int mqttPort = 1883;

const char \*mpuTopic = "mpu/K05";

const char \*tempTopic = "temp/K05";

const char \*finishedTopic = "finished/K05";

hw\_timer\_t \*timer = NULL;

portMUX\_TYPE timerMux = portMUX\_INITIALIZER\_UNLOCKED;

volatile unsigned long previousMillis = 0;

const unsigned long interval = 1000; // Interval in milliseconds

void IRAM\_ATTR onTimer() {

portENTER\_CRITICAL\_ISR(&timerMux);

previousMillis += interval;

portEXIT\_CRITICAL\_ISR(&timerMux);

}

void onMqttMessageReceived(char \*topic, byte \*payload, unsigned int length) {

String message;

for (unsigned int i = 0; i < length; i++) {

message += (char)payload[i];

}

Serial.print("Received message on topic: ");

Serial.print(topic);

Serial.print(", payload: ");

Serial.println(message);

}

void connectToWifi() {

WiFi.begin(ssid, password);

Serial.print("Connecting to Wi-Fi...");

while (WiFi.status() != WL\_CONNECTED) {

delay(100);

}

Serial.println("connected!");

Serial.print("IP address: ");

Serial.println(WiFi.localIP());

}

void setupMqtt() {

mqttClient.setServer(mqttBroker, mqttPort);

mqttClient.setCallback(onMqttMessageReceived);

Serial.print("Connecting to MQTT broker...");

while (!mqttClient.connected()) {

if (mqttClient.connect("ESP32Client")) {

Serial.println("connected!");

mqttClient.subscribe(mpuTopic);

mqttClient.subscribe(tempTopic);

mqttClient.subscribe(finishedTopic);

} else {

Serial.print("failed, retrying in 5 seconds...");

delay(5000);

}

}

}

void setup() {

pinMode(led\_green, OUTPUT);

pinMode(led\_red, OUTPUT);

pinMode(led\_yellow, OUTPUT);

Wire.begin(33, 32);

Serial.begin(115200);

while (!Serial)

delay(10);

Serial.println("Adafruit MPU6050 test!");

if (!mpu.begin()) {

Serial.println("Failed to find MPU6050 chip");

while (1) {

delay(10);

}

}

Serial.println("MPU6050 Found!");

mpu.setAccelerometerRange(MPU6050\_RANGE\_8\_G);

Serial.print("Accelerometer range set to: +-8G");

mpu.setGyroRange(MPU6050\_RANGE\_500\_DEG);

Serial.print("Gyro range set to: +- 1000 deg/s");

mpu.setFilterBandwidth(MPU6050\_BAND\_21\_HZ);

Serial.print("Filter bandwidth set to: 44hz");

Serial.println("");

delay(3000);

connectToWifi();

setupMqtt();

// Set up the hardware timer

timer = timerBegin(0, 80, true); // Timer 0, prescaler 80 (1MHz tick), count up

timerAttachInterrupt(timer, &onTimer, true); // Attach the timer ISR

timerAlarmWrite(timer, interval \* 1000, true); // Set the alarm to trigger every interval (in microseconds)

timerAlarmEnable(timer); // Enable the alarm

}

void blink() {

digitalWrite(led\_green, HIGH);

digitalWrite(led\_red, HIGH);

digitalWrite(led\_yellow, HIGH);

delay(30);

digitalWrite(led\_green, LOW);

digitalWrite(led\_red, LOW);

digitalWrite(led\_yellow, LOW);

}

void loop() {

unsigned long currentMillis;

unsigned long tempMillis;

portENTER\_CRITICAL(&timerMux);

currentMillis = previousMillis;

portEXIT\_CRITICAL(&timerMux);

sensors\_event\_t a, g, temp;

mpu.getEvent(&a, &g, &temp);

String gyroXValue = String(g.gyro.x);

mqttClient.publish(mpuTopic, gyroXValue.c\_str());

blink();

Serial.print("Acceleration X: ");

Serial.print(a.acceleration.x);

Serial.print(", Y: ");

Serial.print(a.acceleration.y);

Serial.print(", Z: ");

Serial.print(a.acceleration.z);

Serial.println(" m/s^2");

Serial.print("Rotation X: ");

Serial.print(g.gyro.x);

Serial.print(", Y: ");

Serial.print(g.gyro.y);

Serial.print(", Z: ");

Serial.print(g.gyro.z);

Serial.println(" rad/s");

if (currentMillis - tempMillis >= interval) {

tempMillis = currentMillis;

String tempValue = String(temp.temperature);

mqttClient.publish(tempTopic, tempValue.c\_str());

}

Serial.print("Temperature: ");

Serial.print(temp.temperature);

Serial.println(" degC");

Serial.println("");

delay(100);

mqttClient.loop();

}

In this modified code, a hardware timer is utilized to provide a more precise timing mechanism. The timer is set up in the setup() function using timerBegin() and timerAttachInterrupt(). The onTimer() function is called when the timer triggers, and it increments the previousMillis variable by the interval duration. The critical sections (portENTER\_CRITICAL and portEXIT\_CRITICAL) are used to ensure the variable is accessed atomically.

Inside the loop() function, the current value of previousMillis is stored in currentMillis, and the temperature publishing is triggered if the difference between currentMillis and tempMillis is greater than or equal to the interval. The tempMillis variable keeps track of the last time the temperature was published.

# Prompt:

do you see anything, that could cause memory issues/leaks?

# Answer:

In the provided code, I don't see any obvious memory-related issues or leaks. However, it's always a good practice to review your code for potential memory concerns. Here are some general suggestions to help you avoid memory issues:

Avoid unnecessary dynamic memory allocation: Minimize the use of String objects and dynamic memory allocation (new, malloc, etc.) in your code. Instead, prefer using fixed-size character arrays (char[]) or statically allocated memory where possible.

Properly manage memory for received MQTT messages: In the onMqttMessageReceived function, ensure that you handle memory properly when concatenating the payload into a String object. Consider using fixed-size character arrays or buffer management techniques to avoid memory fragmentation or leaks.

Free any dynamically allocated memory: If you do use dynamic memory allocation, make sure to release it (delete, free) when it is no longer needed to prevent memory leaks.

Check for memory fragmentation: Monitor the free heap memory using ESP.getFreeHeap() periodically or when necessary. If you observe a continuous decrease in free heap memory over time, it might indicate memory fragmentation issues. In such cases, consider using static allocation or implementing memory optimization techniques.

Optimize code and data storage: Analyze your code for unnecessary variables, arrays, or data structures that consume excessive memory. Consider using more memory-efficient alternatives or optimizing your algorithms to reduce memory usage.

Review external libraries: Verify that the external libraries you're using are memory-efficient and don't have any known memory-related issues. Keep them up to date to benefit from bug fixes and improvements.

Test and monitor: Thoroughly test your code and monitor its memory usage over extended periods of operation. Use tools like the Arduino IDE's Serial Monitor and heap memory monitoring functions to detect any abnormal behavior or memory-related issues.

By following these suggestions and applying general best practices for memory management, you can minimize the risk of memory issues and leaks in your code.

# Prompt:

thanks, now apply all these guidelines to my code and modify it

# Answer:

#include <Adafruit\_MPU6050.h>

#include <Adafruit\_Sensor.h>

#include <Wire.h>

#include <WiFi.h>

#include <PubSubClient.h>

WiFiClient espClient;

PubSubClient mqttClient(espClient);

Adafruit\_MPU6050 mpu;

int led\_red = 26;

int led\_green = 14;

int led\_yellow = 27;

int brightness = 0; // how bright the LED is

int fadeAmount = 5; // how many points to fade the LED by

#define IC2\_SDA 33

#define I2C\_SCL 32

const char\* ssid = "MaraudersMap";

const char\* password = "Page394%";

const char\* mqttBroker = "192.168.0.89";

const int mqttPort = 1883;

const char\* mpuTopic = "mpu/K05";

const char\* tempTopic = "temp/K05";

const char\* finishedTopic = "finished/K05";

hw\_timer\_t\* timer = NULL;

portMUX\_TYPE timerMux = portMUX\_INITIALIZER\_UNLOCKED;

volatile unsigned long previousMillis = 0;

const unsigned long interval = 1000; // Interval in milliseconds

void IRAM\_ATTR onTimer() {

portENTER\_CRITICAL\_ISR(&timerMux);

previousMillis += interval;

portEXIT\_CRITICAL\_ISR(&timerMux);

}

void onMqttMessageReceived(char\* topic, byte\* payload, unsigned int length) {

// Handle MQTT message received

// Convert payload to a string

char message[length + 1];

memcpy(message, payload, length);

message[length] = '\0';

// Display the received message in the console

Serial.print("Received message on topic: ");

Serial.print(topic);

Serial.print(", payload: ");

Serial.println(message);

}

void connectToWifi() {

WiFi.begin(ssid, password);

Serial.print("Connecting to Wi-Fi...");

while (WiFi.status() != WL\_CONNECTED) {

delay(100);

}

Serial.println("connected!");

Serial.print("IP address: ");

Serial.println(WiFi.localIP());

}

void setupMqtt() {

mqttClient.setServer(mqttBroker, mqttPort);

mqttClient.setCallback(onMqttMessageReceived);

Serial.print("Connecting to MQTT broker...");

while (!mqttClient.connected()) {

if (mqttClient.connect("ESP32Client")) {

Serial.println("connected!");

mqttClient.subscribe(mpuTopic);

mqttClient.subscribe(tempTopic);

mqttClient.subscribe(finishedTopic);

} else {

Serial.print("failed, retrying in 5 seconds...");

delay(5000);

}

}

}

void setup() {

pinMode(led\_green, OUTPUT);

pinMode(led\_red, OUTPUT);

pinMode(led\_yellow, OUTPUT);

Wire.begin(33, 32);

Serial.begin(115200);

while (!Serial)

delay(10);

Serial.println("Adafruit MPU6050 test!");

if (!mpu.begin()) {

Serial.println("Failed to find MPU6050 chip");

while (1) {

delay(10);

}

}

Serial.println("MPU6050 Found!");

mpu.setAccelerometerRange(MPU6050\_RANGE\_8\_G);

Serial.print("Accelerometer range set to: +-8G");

mpu.setGyroRange(MPU6050\_RANGE\_500\_DEG);

Serial.print("Gyro range set to: +- 1000 deg/s");

mpu.setFilterBandwidth(MPU6050\_BAND\_21\_HZ);

Serial.print("Filter bandwidth set to: 44hz");

Serial.println("");

delay(3000);

connectToWifi();

setupMqtt();

// Set up the hardware timer

timer = timerBegin(0, 80, true); // Timer 0, prescaler 80 (1MHz tick), count up

timerAttachInterrupt(timer, &onTimer, true); // Attach the timer ISR

timerAlarmWrite(timer, interval \* 1000, true); // Set the alarm to trigger every interval (in microseconds)

timerAlarmEnable(timer); // Enable the alarm

}

void blink() {

digitalWrite(led\_green, HIGH);

digitalWrite(led\_red, HIGH);

digitalWrite(led\_yellow, HIGH);

delay(30);

digitalWrite(led\_green, LOW);

digitalWrite(led\_red, LOW);

digitalWrite(led\_yellow, LOW);

}

void loop() {

static unsigned long previousTempMillis = 0;

static char tempValue[8]; // Buffer to store temperature value

portENTER\_CRITICAL(&timerMux);

unsigned long currentMillis = previousMillis;

portEXIT\_CRITICAL(&timerMux);

sensors\_event\_t a, g, temp;

mpu.getEvent(&a, &g, &temp);

// Publish gyro X value

char gyroXValue[8]; // Buffer to store gyro X value

snprintf(gyroXValue, sizeof(gyroXValue), "%f", g.gyro.x);

mqttClient.publish(mpuTopic, gyroXValue);

blink();

Serial.print("Acceleration X: ");

Serial.print(a.acceleration.x);

Serial.print(", Y: ");

Serial.print(a.acceleration.y);

Serial.print(", Z: ");

Serial.print(a.acceleration.z);

Serial.println(" m/s^2");

Serial.print("Rotation X: ");

Serial.print(g.gyro.x);

Serial.print(", Y: ");

Serial.print(g.gyro.y);

Serial.print(", Z: ");

Serial.print(g.gyro.z);

Serial.println(" rad/s");

if (currentMillis - previousTempMillis >= interval) {

previousTempMillis = currentMillis;

// Publish temperature value

snprintf(tempValue, sizeof(tempValue), "%f", temp.temperature);

mqttClient.publish(tempTopic, tempValue);

}

Serial.print("Temperature: ");

Serial.print(temp.temperature);

Serial.println(" degC");

Serial.println("");

delay(100);

mqttClient.loop();

}

In this modified version, I've made the following changes:

Replaced the usage of String objects with character arrays (char[]) where appropriate to avoid unnecessary dynamic memory allocation.

In the onMqttMessageReceived function, I've used a character array (char[]) to store the payload instead of a String object.

Introduced static character arrays (gyroXValue and tempValue) to store the sensor values instead of dynamically allocating memory each time.

By following these memory management guidelines, the code reduces dynamic memory allocation and uses statically allocated memory where possible, minimizing the risk of memory issues and leaks.

# Prompt: