Remote Sensor Control Using Secure MQTT Publish and Subscribe

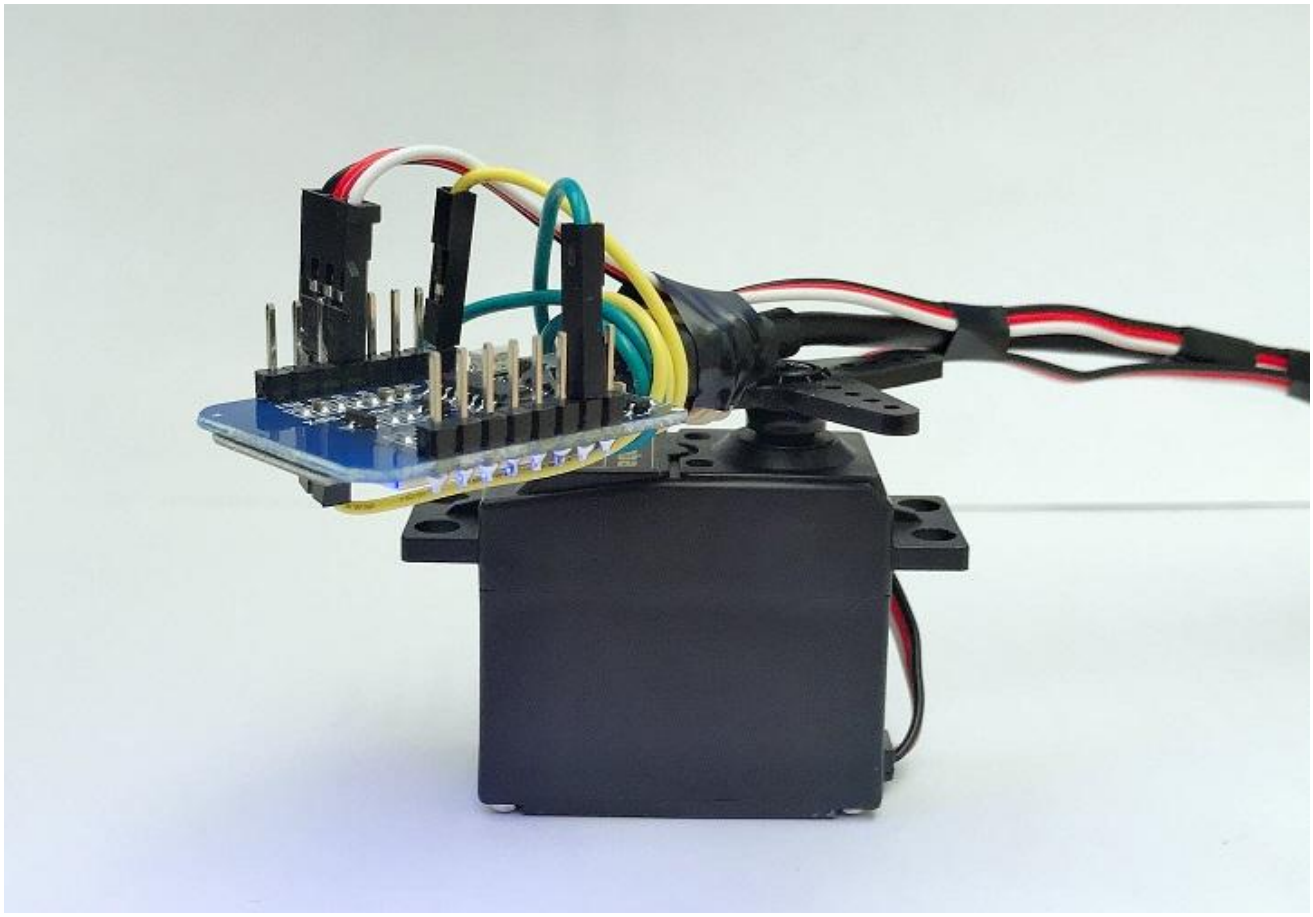
Copy Command

This example shows how to use the MQTT publish and subscribe architecture in ThingSpeak. The MQTT protocol is a low-overhead device messaging system. Using MQTT, subscribe to field 1 of a control channel. When you update the control channel, the posted value is sent to your device. The servo rotates to the specified angle. The device measures the network strength and updates the storage channel. Comments in the code indicate how to adapt this example for a nonsecure connection.

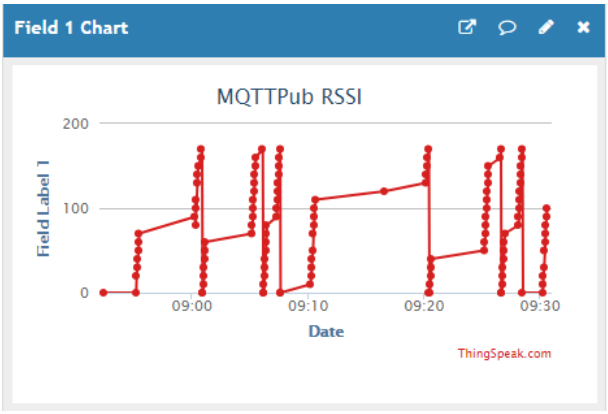
Supported Hardware

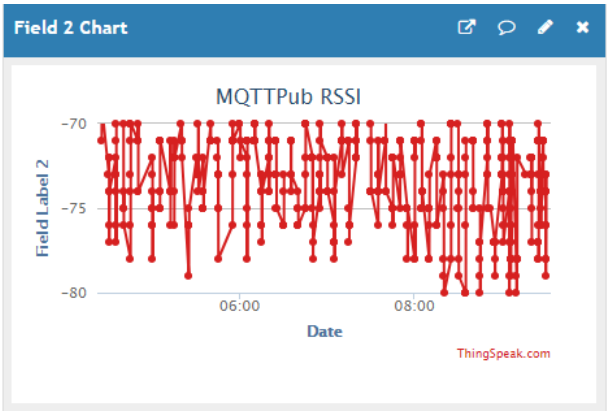
- ESP8266, NodeMCU, WeMOS
- Arduino MKR1000
- Arduino Uno, Mega, Due, or Leonardo with wireless network connection
- Particle Photon (with slight code and schematic adjustments)

The example is designed to work with only a few extra components, namely a single servo motor. You use the on board Wi-Fi antenna to make Wi-Fi strength measurements.

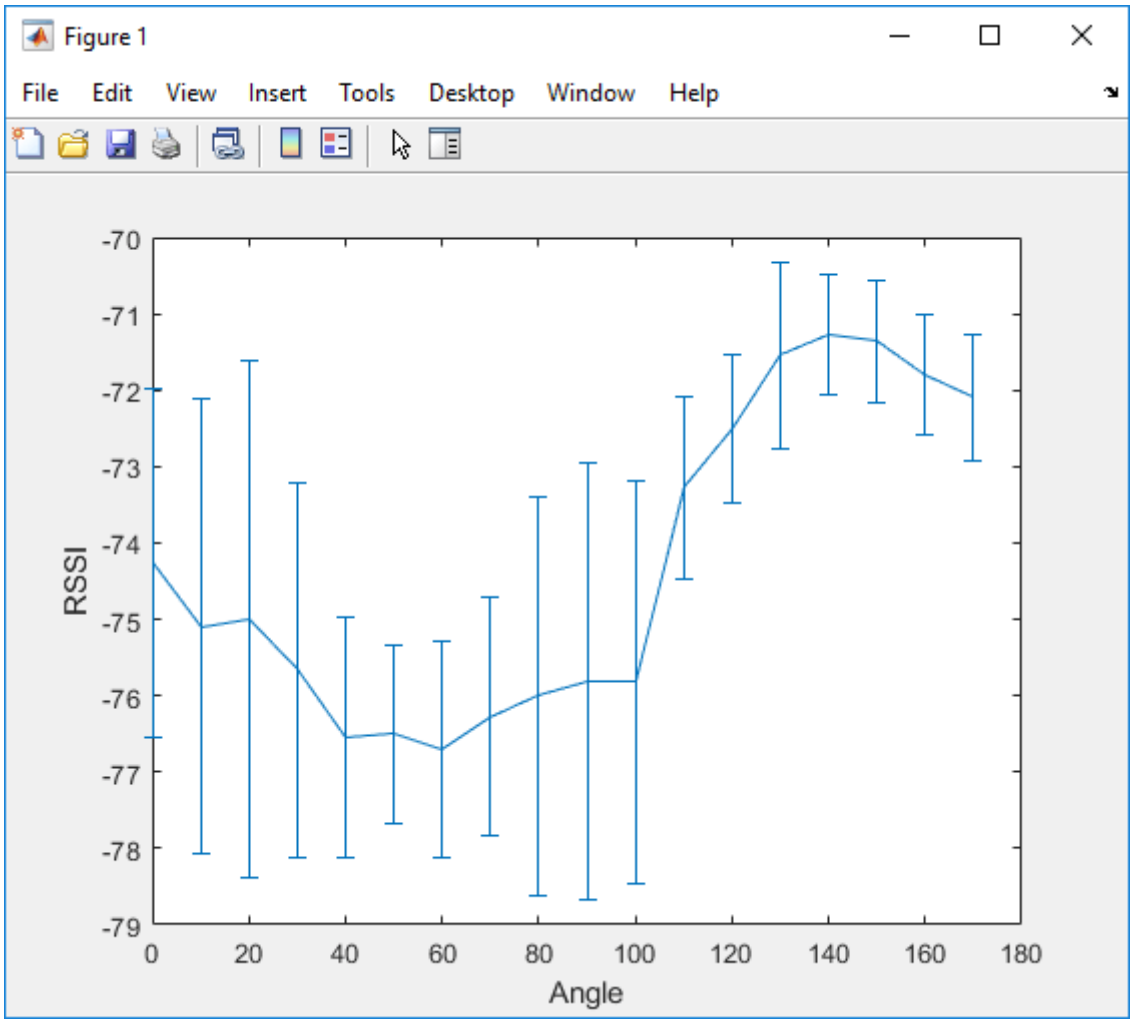


These images show sample channel output from the storage channel. Field 1 stores the angle of the servo motor that is set from the control channel, and field 2 shows the measured Wi-Fi strength value.





Data analysis shows that the orientation of the WeMOS hardware has a directional effect on the measured signal strength.

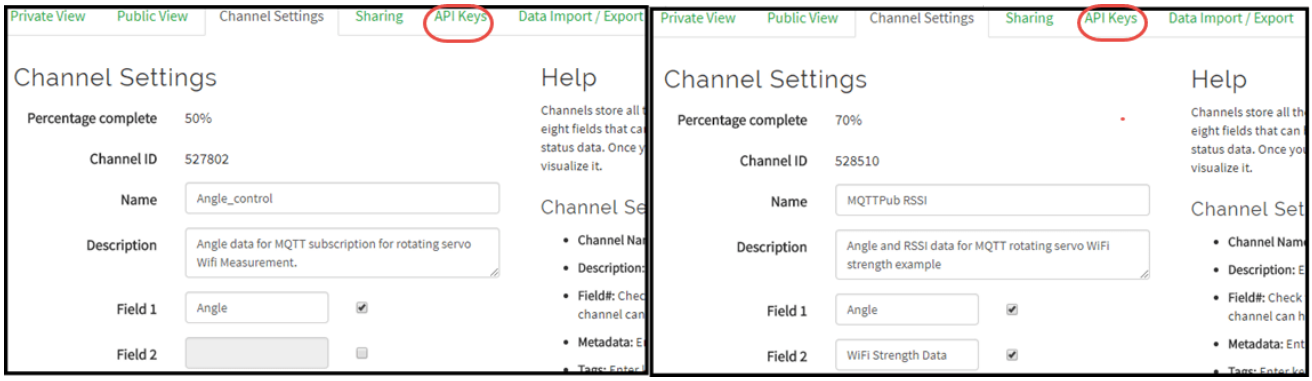


Prerequisites

1) Create a ThingSpeak Channel for the subscribe control as shown in Collect Data in a New Channel. The subscribe channel holds the angle for the servo motor. When the servo motor angle is updated, the subscribed device receives the angle as an MQTT message. The device sets the servo angle and measures the new wireless network strength at that angle.

2) Create another ThingSpeak channel for the published data. The publish channel records the set angle and the signal strength data.

3) In the **Channel Settings** view, enable fields 1 and 2 for the publish channel. To distinguish between the fields, give each field a descriptive name.



4) Note the read and write API keys on the **API Keys** tab in the **Channel Settings** view (circled in the image).

5) Create an MQTT device by clicking **Devices** > **MQTT** at the top of the page, then **Add a new device**. When you set up the device, authorize both channels for publish and subscribe. For details, see Create a ThingSpeak MQTT Device.

6) While adding the new device, click **Download Credentials** > **Arduino (mqtt_secrets.h)**. Keep this downloaded secrets file for access in the Code section below.

Required Hardware

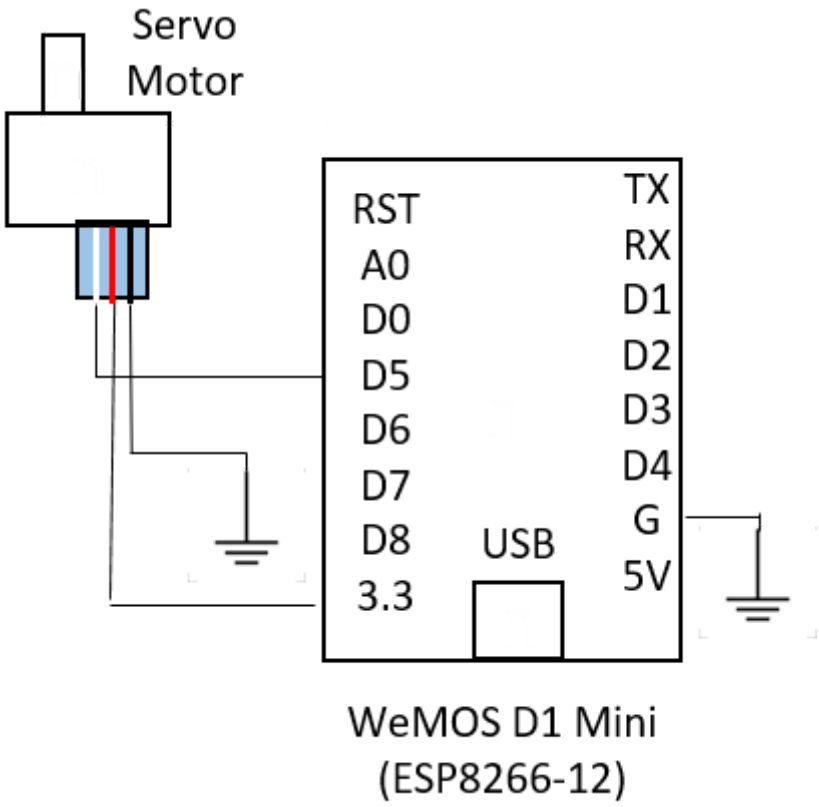
- WeMOS D1 Mini, or one of the following devices with changes to the libraries used: NodeMCU, ESP8266-01, ESP8266-04, ESP8266-12, ESP8266-12E, Arduino® MKR1000, or other Arduino with Ethernet or wireless network connection
- Servo motor (for example, Futaba S3003)
- Jumper wires (at least 3)
- USB cable

Schematic and Connections

1) Connect D5 on the WeMOS D1 Mini the signal line of the servo.

2) Connect the ground wire of the servo to the ground on the WeMOS board.

3) Connect the servo power to 3.3 V. Using 5 V directly can overload the USB power limit in some cases.



Program Your Arduino

Use the Arduino IDE to program your device. You can download the latest Arduino IDE here.

1) Add the ESP8266 Board Package:

a. Under **File** > **Preferences**, enter `https://arduino.esp8266.com/stable/package_esp8266com_index.json` into **Additional Board Manager URLs**.

b. Select **Tools** > **Boards** > **Board Manager**. Enter `ESP8266` in the search bar and install the package.

2) Modify the allowed packet size.

a. Navigate to the folder with the pub sub library header file, generally `Documents\Arduino\libraries\PubSubClient\src`.

b. Edit `PubSubClient.h to` change the max packet size to 4096. When complete, the line should read:

```
#define MQTT_MAX_PACKET_SIZE 4096
```

3) Create the application:

a. Open a new window in the Arduino IDE, and save the file.

b. Add the code provided in the Code section.

c. Be sure to edit the wireless network information and channel IDs in the code.

4) Add libraries and secrets file to the sketch:

a. If not already present, add the following libraries to the library manager by selecting **Sketch** > **Include Library** > **Manage Libraries**. For each library search its name and select **Install**.

- `PubSubClient`
- `ESP8266Wifi`
- `servo`

b. Add the `mqtt_secrets.h` file.

Test Your Device

After you successfully upload your program, you can monitor the output using the serial monitor. Upload a value to your ThingSpeak control channel in the range of 0 to 175. You can copy the GET request format from the **API Keys** tab or modify this text with your write API key. Enter each URL directly into the address bar of your browser, changing YOUR WRITE API KEY to the write API key for your channel.

```
https://api.thingspeak.com/update?api_key=YOUR_WRITE_API_KEY&field1=ANGLE_VALUE
```

The device publishes the angle and the Wi-Fi signal strength to the storage channel whenever you post to the subscribe channel. Make sure that your angle value is in the range of 0 to 175.

Code

1) Include required libraries and define data fields:

```
#include <PubSubClient.h>
#include <WiFiClientSecure.h>                          // Needed only if using secure connec
#include <ESP8266WiFi.h>
#include <Servo.h>
#include "mqtt_secrets.h"
#define ANGLE_FIELD 0
#define DATA_FIELD 1                                   // Data field to post the signal stre
```

2) Define and initialize the variables. Be sure to edit the wireless network information, channel ID, and credentials. Find your channel ID on the top of the main page for your channel.

```
char ssid[] = "YOUR_SSID";                  // Change to your network SSID (name).
char pass[] = "YOUR_WIFI_PASSWORD";         // Change to your network password.
const char* server = "mqtt3.thingspeak.com";
char mqttUserName[] = SECRET_MQTT_USERNAME;  // Change to your MQTT device username.
char mqttPass[] = SECRET_MQTT_PASSWORD;      // Change to your MQTT device password.
char clientID[] = SECRET_MQTT_CLIENT_ID;     // Change to your MQTT device clientID.
long readChannelID = 85;
long writeChannelID = 86;

// Here's how to get ThingSpeak server fingerprint: https://www.a2hosting.com/kb/security/s
const char* thingspeak_server_fingerprint = "27 18 92 dd a4 26 c3 07 09 b9 7a e6 c5 21 b9 5

// WiFiClient client;                               // Initialize the Wi-Fi client librar
WiFiClientSecure client;                            // Uncomment for secure connection.
PubSubClient mqttClient( client );                  // Initialize the PuBSubClient librar
Servo myservo;  // Create servo object to control a servo .

int fieldsToPublish[8]={1,1,0,0,0,0,0,0};           // Change to allow multiple fields.
float dataToPublish[8];                             // Holds your field data.
int changeFlag=0;                                   // Let the main loop know there is ne
int servo_pos=0;                                    // Servo position
```

3) Define the function prototypes in this code.

```
//
// Prototypes
//

// Handle messages from MQTT subscription.
void mqttSubscriptionCallback(char* topic, byte* payload, unsigned int length);

// Generate a unique client ID and connect to MQTT broker.
void mqttConnect();

// Subscribe to a field or feed from a ThingSpeak channel.
int mqttSubscribe( long subChannelID,int field, int unSub);

// Publish messages to a channel feed.

// Connect to a given Wi-Fi SSID.
int connectWifi();

// Measure the Wi-Fi signal strength.
void updateRSSIValue();
```

4) Initialize pins for input and output, start the serial monitor, and initialize the MQTT client in the `setup` routine.

```
void setup() {
Serial.begin( 115200 );
Serial.println( "Start" );
int status = WL_IDLE_STATUS; // Set temporary Wi-Fi status.

    connectWifi();                                      // Connect to Wi-Fi network.
    // mqttClient.setServer( server, 1883 );            // Set the MQTT broker details, r
    mqttClient.setServer( server, 8883 );               // Set the MQTT broker details, s
    mqttClient.setCallback( mqttSubscriptionCallback ); // Set the MQTT message handler f
    myservo.attach(14);                                 // Attach the servo on GIO2 to th
    myservo.write(90);                                  // Start in the middle.
}
```

5) Each time the main loop executes, check to see if data from the MQTT subscription is available for processing. Then set the servo position to match the data. Make sure the wireless and MQTT clients are active and maintain a connection to the client server.

```
void loop() {

    if (WiFi.status() != WL_CONNECTED) {
        connectWifi();
    }

    if (!mqttClient.connected())
    {

       mqttConnect(); // Connect if MQTT client is not connected.

         if(mqttSubscribe( readChannelID,1,0 )==1 ){
                Serial.println( " Subscribed " );
            }
    }

    mqttClient.loop(); // Call the loop to maintain connection to the server.

    if ((servo_pos>175)||(servo_pos<0)){
    servo_pos=0;
    }

    if (changeFlag){

        changeFlag=0;
        myservo.write(servo_pos);
        dataToPublish[ANGLE_FIELD]=servo_pos;
        delay(1100);                      // Wait for ThingSpeak to publish.
        Serial.println( "Servo value " + String( servo_pos ) );
        mqttPublish( writeChannelID, dataToPublish, fieldsToPublish );
    }

    delay(1);
}
```

6) Use the `mqttSubscriptionCallback` function to handle incoming MQTT messages. The program runs smoother if the main loop performs the processing steps instead of the callback. In this function, use flags to cause changes in the main loop.

```
/**
 * Process messages received from subscribed channel via MQTT broker.
 *   topic - Subscription topic for message.
 *   payload - Field to subscribe to. Value 0 means subscribe to all fields.
 *   mesLength - Message length.
 */

void mqttSubscriptionCallback( char* topic, byte* payload, unsigned int mesLength ) {

    char p[mesLength + 1];
    memcpy( p, payload, mesLength );
    p[mesLength] = NULL;
    Serial.print( "Answer: " );
    Serial.println( String(p) );
    servo_pos=atoi( p );
    changeFlag=1;
}
```

7) Use the `MQTTConnect` function to set up and maintain a connection to the MQTT.

```
void mqttConnect()
{
    // Loop until connected.
    while ( !mqttClient.connected() )
    {
      Serial.println(String( mqttUserName)+ " , " + mqttPass + " , " + clientID);

        // Connect to the MQTT broker.
        Serial.print( "Attempting MQTT connection..." );
        if ( mqttClient.connect( clientID, mqttUserName, mqttPass ) )
        {
            Serial.println( "Connected with Client ID:  " + String( clientID ) + " User "+

        } else
        {
            Serial.print( "failed, rc = " );
            // See https://pubsubclient.knolleary.net/api.html#state for the failure code e
            Serial.print( mqttClient.state() );
            Serial.println( " Will try again in 5 seconds" );
            delay( 5000 );
        }
    }
}
```

8) Use `mqttSubscribe` to receive updates from the LED control field. In this example, you subscribe to a field, but you can also use this function to subscribe to the whole channel feed. Call the function with `field = 0` to subscribe to the whole feed.

```
/**
 * Subscribe to fields of a channel.
 *   subChannelID - Channel to subscribe to.
 *   field - Field to subscribe to. Value 0 means subscribe to all fields.
 *   readKey - Read API key for the subscribe channel.
 *   unSub - Set to 1 for unsubscribe.
 */

int mqttSubscribe( long subChannelID, int field, int unsubSub ){
    String myTopic;

    // There is no field zero, so if field 0 is sent to subscribe to, then subscribe to the
    if (field==0){
        myTopic="channels/"+String( subChannelID )+"/subscribe";
    }
    else{
        myTopic="channels/"+String( subChannelID )+"/subscribe/fields/field"+String( field
    }

    Serial.println( "Subscribing to " +myTopic );
    Serial.println( "State= " + String( mqttClient.state() ) );

    if ( unsubSub==1 ){
        return mqttClient.unsubscribe(myTopic.c_str());
    }
    return mqttClient.subscribe( myTopic.c_str() ,0 );
}
```

9) The `mqttUnsubscribe` function is not used in the code, but you can use it to end a subscription.

```
/**
 * Unsubscribe channel
 *   subChannelID - Channel to unsubscribe from.
 *   field - Field to unsubscribe subscribe from. The value 0 means subscribe to all fields
 *   readKey - Read API key for the subscribe channel.
 */

int mqttUnSubscribe(long subChannelID,int field,char* readKey){
    String myTopic;

    if (field==0){
         myTopic="channels/"+String( subChannelID )+"/subscribe";
    }
    else{
        myTopic="channels/"+String( subChannelID )+"/subscribe/fields/field"+String( field
    }
    return mqttClient.unsubscribe( myTopic.c_str() );
}
```

10) Use the `mqttPublish` function to send your angle and Wi-Fi RSSI data to a ThingSpeak channel.

```
/**
 * Publish to a channel
 *   pubChannelID - Channel to publish to.
 *   pubWriteAPIKey - Write API key for the channel to publish to.
 *   dataArray - Binary array indicating which fields to publish to, starting with field 1.
 *   fieldArray - Array of values to publish, starting with field 1.
 */

void mqttPublish(long pubChannelID, float dataArray[], int fieldArray[]) {
    int index=0;
    String dataString="";

    updateRSSIValue();  // Make sure the stored value is updated.

    //
    while (index<8){

        // Look at the field array to build the posting string to send to ThingSpeak.
        if (fieldArray[ index ]>0){

            dataString+="&field" + String( index+1 ) + "="+String( dataArray [ index ] );
        }
        index++;
    }

    Serial.println( dataString );

    // Create a topic string and publish data to ThingSpeak channel feed.
     String topicString ="channels/" + String( pubChannelID ) + "/publish";
    mqttClient.publish( topicString.c_str(), dataString.c_str() );
    Serial.println( "Published to channel " + String( pubChannelID ) );
}
```

11) Connect your device to a wireless network using the `connectWiFi` function.

```
int connectWifi()
{
    while ( WiFi.status() != WL_CONNECTED ) {
        WiFi.begin( ssid, pass );
        delay( 8500 );
        Serial.println( "Connecting to Wi-Fi" );
    }
    Serial.println( "Connected" );
    client.setFingerprint(thingspeak_server_fingerprint);  // Comment this line if using nc
}
```

12) Use the `updateRSSIValue` function to read the signal strength for the network that you are presently connected to.

```
void updateRSSIValue(){

   long rssi = WiFi.RSSI();
   Serial.print( "RSSI:" );
   Serial.println(rssi);
   dataToPublish[ DATA_FIELD ]=float( rssi );

}
```

See Also

`Write Data` | `MQTT Publish`

Related Topics

- Troubleshoot MQTT Publish
- REST API