



MQTT Web User Interface Guide

Document Version: 1.

Firmware Version: Dragino-v2 LG02_LG08-5.3.xxx

Version	Description	Date
0.1 - 0.5	Initial drafts	30-10-2019

Table of Contents

1 Introduction.....	4
2 MQTT Configuration Screens.....	4
2.1 General Server Profile Configuration.....	5
2.1.1 Valid Connection Parameters.....	6
2.2 Host Specific Profile Configuration.....	8
2.3 MQTT Certificate Management Screen.....	9
2.4 MQTT Channel Management Screen.....	10
2.5 LoRa Configuration Screen.....	11
3 MQTT Process on LoRa Gateway.....	12
3.1 Publish Overview.....	12
3.1.1 Channel Definition.....	13
3.1.2 MQTT Publish Command.....	13
3.2 MQTT Subscribe Process Overview.....	15
3.2.1 MQTT Subscribe Command.....	16
4 Troubleshooting.....	17
4.1 Overview.....	17
4.2 Host Connection Credentials and Parameters.....	17
5 Troubleshooting the Publish Process.....	20
5.1 Overview.....	20
5.2 Testing Publish with <i>mosquitto_pub</i>	20
5.2.1 Single line examples:.....	20
5.2.2 Multi-line Examples:.....	21
5.3 Testing Publish with <i>mqtt_process.sh</i>	23
5.4 Testing Publish with LoRa Messages.....	28
5.4.1 Set up the IoT Host Platform configuration.....	28
5.4.2 Set up Gateway LoRa Radio Settings.....	28
5.4.3 Run the test.....	29
5.4.4 Example Arduino Sketch.....	31

6 Troubleshooting the Subscription process.....	33
6.1 Overview.....	33
6.2 Testing Subscription with <i>mosquitto_sub</i>	33
6.3 Testing Subscription with <i>mqtt_process.sh</i>	35
6.4 Testing Subscription with LoRa Radio.....	35
7 References.....	36

1 Introduction

This User Guide document describes how to configure a Dragino LoRa Gateway device for MQTT using the Web User Interface.

It also includes some guidelines for troubleshooting Publish and Subscribe processes between remote Sensor Nodes communicating with a Gateway device using LoRa radio, and an IoT Host platform.

The updated interface described here replaces the interface previously provided as an extension of the OpenWrt LuCI web configuration interface.

Selecting the **MQTT Menu** item from the **Services** menu will display the new interface screens.

2 MQTT Configuration Screens

The main MQTT configuration screen is shown below.

The configuration supports a **General** connection profile as well as several host platform specific profiles including ThingSpeak, LwM2M, MyDevices, Amazon Web Service (AWS) and MS Azure.

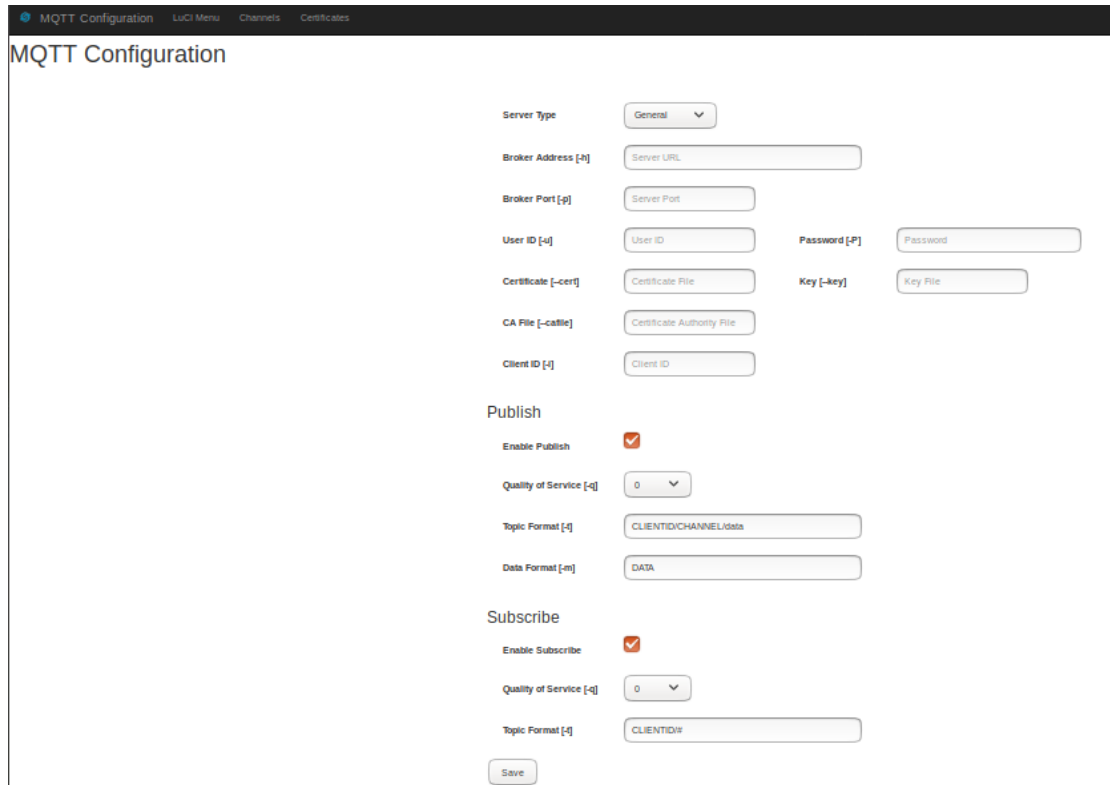
Selecting the **Server Type** at the top of the screen will change the set of parameters displayed to suit the particular host, and display the values stored in the selected profile.

Once the parameter values have been entered or edited, clicking on the **Save** button will update the values in the stored profile for the selected **Server Type**, and these values will be used in subsequent connection requests.

To change the active **Server Type** to be used for connection requests, simply select the required profile and click on the **Save** button.

The host specific profiles have pre-set default values for some of the connection parameters. You can choose to select the stored host profile and use these preset values, or choose the **General** profile and set all the required parameters.

2.1 General Server Profile Configuration



The screenshot shows the MQTT Configuration web interface. At the top, there is a navigation bar with "MQTT Configuration", "LuCI Menu", "Channels", and "Certificates". The main content area is titled "MQTT Configuration".

The configuration is organized into several sections:

- Server Type:** A dropdown menu set to "General".
- Broker Address [-h]:** A text input field labeled "Server URL".
- Broker Port [-p]:** A text input field labeled "Server Port".
- User ID [-u]:** A text input field labeled "User ID".
- Password [-P]:** A text input field labeled "Password".
- Certificate [-cert]:** A text input field labeled "Certificate File".
- Key [-key]:** A text input field labeled "Key File".
- CA File [-cafile]:** A text input field labeled "Certificate Authority File".
- Client ID [-i]:** A text input field labeled "Client ID".

Below these fields are two sections for publishing and subscribing:

- Publish:**
 - Enable Publish:** A checked checkbox.
 - Quality of Service [-q]:** A dropdown menu set to "0".
 - Topic Format [-f]:** A text input field containing "CLIENTID/CHANNELidata".
 - Data Format [-m]:** A text input field containing "DATA".
- Subscribe:**
 - Enable Subscribe:** A checked checkbox.
 - Quality of Service [-q]:** A dropdown menu set to "0".
 - Topic Format [-f]:** A text input field containing "CLIENTID/#".

At the bottom of the form is a "Save" button.

The **General** profile displays more fields than will typically be used for a specific host connection.

For example, fields are displayed for both **User/Password** and **Certificate/Key** forms of authentication, but only one of these types of authentication will be used in practice.

Where fields are not required for a specific host connection, they should be left blank.

Publish and **Subscribe** functions can be individually selected using the checkboxes.

2.1.1 Valid Connection Parameters

The General profile allows flexibility in defining the parameter fields to be used in connection requests. However not all combinations of parameters will form valid requests, and the software allows for only a specific set of requests.

The software uses the *mosquitto_pub* and *mosquitto_sub* functions to establish connections with the host platform for Publish and Subscribe operations.

The connection process supports a defined set of parameter combinations as follows.

1. User/Password Authentication

Where all of *User*, *Password* and *ClientID* parameters are set, it is assumed that User/Password authentication will be used.

If the *CA File* parameter is set it will also be included in the connection command. (This is required for example for MS Azure)

2. Certificate/Key Authentication

Where all of Certificate, Key and *ClientID* parameters are set, it is assumed that Certificate/Key authentication will be used.

If the *CA File* parameter is set it will also be included in the connection command.

The *AWS* profile uses this form of connection for example.

3. Anonymous

Where no User, Certificate or ClientID parameters are present, it is assumed that a simple anonymous connection is required.

This form is used for some testing servers.

4. Client ID Only

Where no User or Certificate parameters are present, and ClientID parameter is present, then the command will include the ClientID field.

This form is used for some testing servers.

5. User and ClientID

Where the User and ClientID parameter are present, and there is no Password or Certificate parameters present, then the command will include the User and ClientID fields.

6. Invalid Parameter Set

Where none of the above sets of parameters are detected, the Publish and Subscribe connection commands will **not** be run and an error message will be logged.

2.2 Host Specific Profile Configuration

MQTT Configuration
LuCI Menu
Channels
Certificates

MQTT Configuration

Server Type	<input type="text" value="AWS"/>		
Broker Address [h]	<input type="text" value="Server URL"/>		
Broker Port [p]	<input type="text" value="8883"/>		
Certificate [-cert]	<input type="text" value="Certificate File"/>	Key [-key]	<input type="text" value="Key File"/>
CA File [-cafile]	<input type="text" value="AmazonRootCA1.pem"/>		
Client ID [-i]	<input type="text" value="Client ID"/>		

Publish

Enable Publish

Quality of Service [-q]

Topic Format [-t]

Data Format [-m]

Subscribe

Enable Subscribe

Quality of Service [-q]

Topic Format [-t]

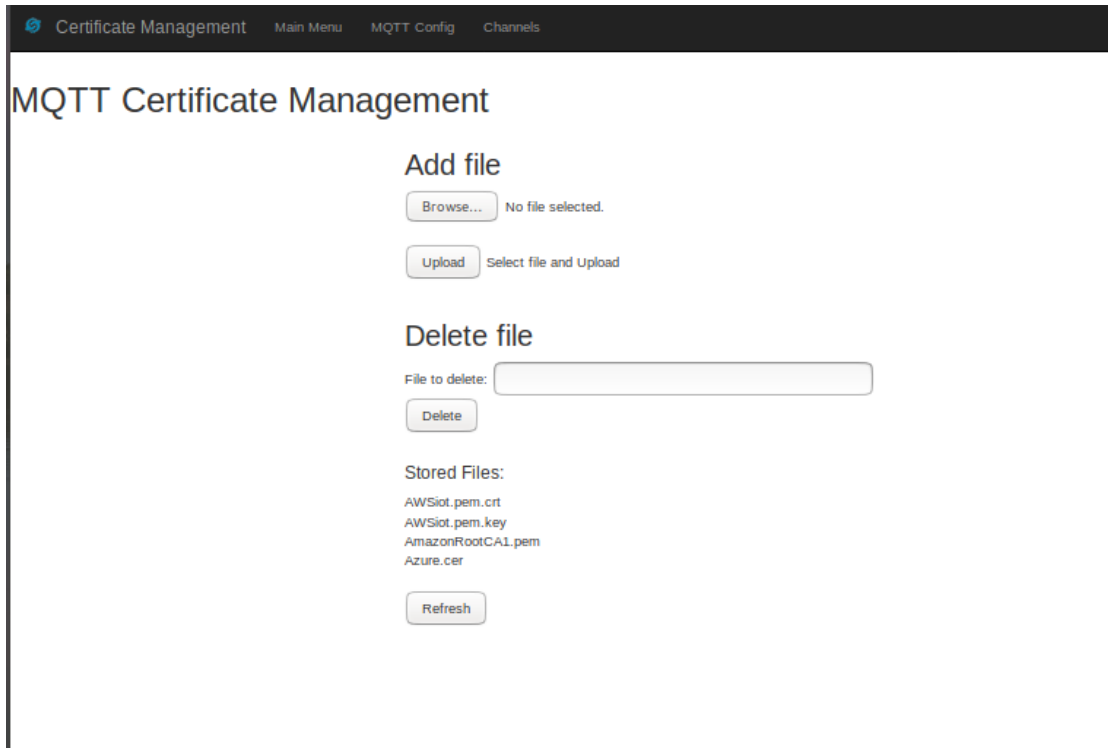
Selecting a specific host profile such as **AWS** shown above will adjust the parameter fields to show only those required for the given host.

Generally all fields shown for a specific host profile need to be correctly entered for the connection to be successful.

When the required values have been entered, click on the **Save** button to update the values in the profile and to set the selected profile to be active and used for subsequent connection requests.

2.3 MQTT Certificate Management Screen

This screen allows you to load Certificate, Key and Certificate Authority (CA) files onto the Gateway device ready to be added to profiles and used in connection requests.



To load a file, click on the **Browse** button and select the required file from the file system on your PC.

Then click on the **Upload** button to load the file into memory.

The file will then appear in the list of **Stored Files**.

To delete a stored file, enter its name in full into the **Delete File** field and click on the **Delete** button.

You will be asked to confirm the deletion.

2.4 MQTT Channel Management Screen

This screen allows you to define logical **Channels** that map the names of message sources (as known to the remote sensor device and the Gateway - **Local ID**), to the name by which it is known at the Host platform (**Remote ID**), typically used as part of the **Publish Topic** string.

Channel Management
Main Menu
MQTT Config
Certificates

MQTT Channel Management

Add / Edit Channel

Channel Number Number 0 - 99

Local ID

Remote ID

Write API Key

Delete Channel

Channel Number Number 0 - 99

Saved Channels:

```

chan1 Local ID: DSN-1 Remote ID: DraginoSensorNode-1 Write API Key:
chan2 Local ID: DSN-2 Remote ID: 828889 Write API Key: 8JIKKXNUNYSD21Z98

```

To enter a new **Channel** definition, enter an unused **Channel Number**, then complete the **Local ID** and **Remote ID fields**, and optionally the **Write API** field if it is required by the Host platform.

Then click on the **Add** button. The new definition will appear in the **Saved Channels** list.

To delete a channel, enter the number of the required channel in the **Channel Number** field and click on the **Delete** button.

2.5 LoRa Configuration Screen

The LoRa Configuration screen shown below is used to set up the LoRa Radio configuration parameters for communication between the Gateway device and Remote Sensor devices.

The Radio Settings on the Gateway device must match exactly the settings used on the Remote sensor device in order to establish communication between the devices for both Publish and Subscribe operations.

Note that the content of the loRa Configuration screen will change depending on the Gateway device type and the IoT Service selected. The screen below shows the screen for a LG01-N device with a single LoRa radio, and with the IoT Service selected as **LoRaRaw Forward to MQTT** Server.

LoRa Configuration MQTT Client TCP Client HTTP/HTTPS Client Custom Script Return to Main Menu

LoRa General Settings

IoT Service	<input type="text" value="mqtt"/>		
RF Power (dBm)	<input type="text" value="20"/>	Debug Level	<input type="text" value="1"/>
Gateway ID	<input type="text" value="a840411d19e84150"/>		

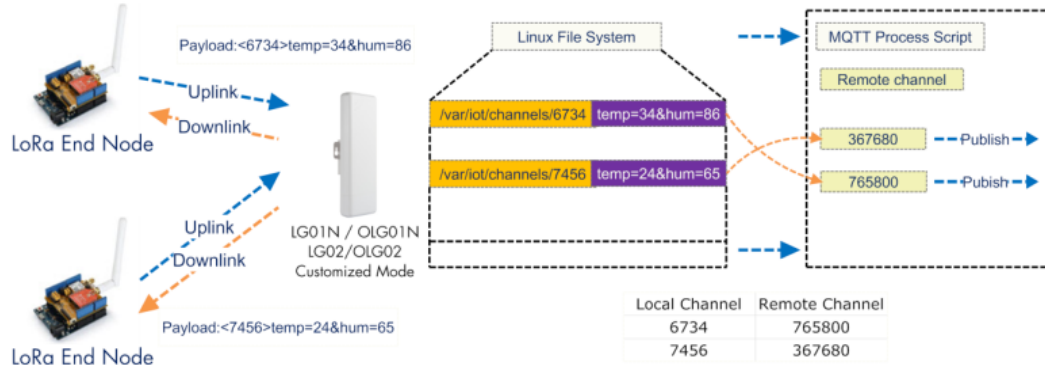
Radio Settings

Frequency (Hz)	<input type="text" value="915000000"/>	RF Bandwidth (Hz)	<input type="text" value="125000"/>
Spreading Factor	<input type="text" value="7"/>	Coding Rate	<input type="text" value="5"/>
Preamble Length	<input type="text" value="8"/>	LoRa Sync Word	<input type="text" value="52"/>

3 MQTT Process on LoRa Gateway

3.1 Publish Overview

This section describes how the LoRa to MQTT gateway process operates and includes some approaches troubleshooting the end to end connection.



The above diagram shows an overview of how the end to end messaging system works, from the creation of a message in a LoRa End Node (typically a remote sensor device) to the publishing of the message on a host IoT Platform such as ThingSpeak, AWS or Azure via the MQTT protocol.

The message publishing process starts with an End Node generating a message whose payload contains an identifier for the End Node, and the data to be published e.g

Message Payload : `<6734>temp=34&hum=86`

Here '6374' is the identifier of the node, and it is followed by two data values expressed as 'name=value' pairs, one for temperature and one for humidity. Note that the End Node identifier is not globally unique, but it is unique among the End Nodes that connect to the particular Gateway device.

This message is transmitted via LoRa and is received by the Gateway, which creates a file called '/var/iot/channels/6734' which contains the message text.

3.1.1 Channel Definition

The Gateway maintains a Channel definition table which maps End Node identifiers (Local ID) to publishing identifiers (Remote ID) on the host IoT platform.

The Gateway runs a script called 'mqtt_process.sh' which scans the directory '/var/iot/channels' looking for new files.

When a new file is detected in '/var/iot/channels', mqtt_process looks up the channel definitions to find an entry that matches the new file name, and extracts the corresponding Remote ID value and uses it as the Channel name.

Mqtt_process uses the Remote ID value to define the Publish Topic e.g. for an AWS host, the default Topic string is: '/<ClientID>/<Channel>/data'

For the example above, the Local ID '6374' maps to Remote ID '765800' so the Topic string would look like: '/<clientID>/765800/data'.

The Client ID is the registered name of the Gateway device on the IoT Host platform. The Client ID for a particular gateway device is obviously unique among all the gateway devices connected to the particular IoT host instance, so the Topic string uniquely identifies the source of the sensor data via the Gateway identifier (Client ID) and the End Node identifier (Channel).

3.1.2 MQTT Publish Command

The Gateway device uses the **Mosquitto** software module for communication with the IoT Host via the MQTT protocol. (<https://mosquitto.org>)

To publish a message, the **mosquitto_pub** command is used.

For the above example message to be published on an **AWS IoT Hub** host that uses Certificate based authentication, the command would look like:

```
# mosquitto_pub \  
-h ahsty70p2ab23-ats.iot.eu-west-3.amazonaws.com \  
-p 8883 \  
-q 1 \  
--cert <certificate file path> \  
--key <private key file path> \  
--cafile <certificate authority file path> \  
-i <clientID> \  
-t /<clientID>/765800/data \  
-m "<6734>temp=34&hum=86"
```

The command parameters are as follows:

```
-h Server URL:      ahrqy70p2lw97-ats.iot.ap-southeast-2.amazonaws.com  
-p Port           :      8883  
-q QoS           :      1  
--cert Certificate: /etc/iot/cert/AWSiot.pem.crt  
--key Key         :      /etc/iot/cert/AWSiot.pem.key  
--cafile CA File:  /etc/iot/cert/AmazonRootCA1.pem  
-i Client ID:     <clientID>  
-t Topic         :      <clientID>/765800/data  
-m MQTT Data:     "temp:34 hum:86"
```

For a **ThingSpeak** IoT host which uses User/Password authentication, the command would look like:

```
# mosquitto_pub \  
-h mqtt.thingspeak.com \  
-p 1883 \  
-q 0 \  
-u <my_user_name>  
-P <my_password>  
-i <clientID> \  
-t channels/765800/publish/<write_api> \  
-m "<6734>temp=34&hum=86&status=MQTTPUBLISH"
```

3.2 MQTT Subscribe Process Overview

The Subscribe process works in a similar manner the Publish process on the Dragino gateway devices. The underlying command is **mosquitto_sub** which is called by **mqtt_process.sh** using the parameters that you set up in the GUI.

The **mosquitto_sub** process runs in the background once it is started by **mqtt_process.sh**, and looks for incoming subscription messages from the MQTT server platform.

The **mosquitto_sub** process on the Dragino devices has been extended to include the LoRa radio parameters and to interface with the LoRa radio driver software. It uses an additional flag (**-z**) at the end of the command to pass the LoRa parameters (freq, data rate, coding) as follows:

```
-z --freq $FREQSTR --datr $DATR --codr $CODR
```

When an incoming message from a subscription is received, the **mosquitto_sub** process will pass the message to the LoRa radio driver software to send the message out using the LoRa radio parameters specified in the LoRa configuration GUI page.

3.2.1 MQTT Subscribe Command

The ***mosquitto_sub*** process runs in the background once it is started by ***mqtt_process.sh***, and looks for incoming subscription messages from the MQTT server platform.

The ***mosquitto_sub*** process on the Dragino devices has been extended to include the LoRa radio parameters and to interface with the LoRa radio driver software. It uses an additional flag ***(-z)*** at the end of the command to pass the LoRa parameters (freq, data rate, coding) as follows:

```
-z --freq $FREQSTR --datr $DATR --codr $CODR
```

When an incoming message from a subscription is received, the ***mosquitto_sub*** process will pass the message to the LoRa radio driver software to send the message out using the LoRa radio parameters specified in the LoRa configuration GUI page.

4 Troubleshooting

4.1 Overview

Establishing a connection from a Sensor Node through a Gateway to an IoT Host in order to publish sensor data involves many steps and different pieces of software and hardware, all of which have to be correctly configured for messages to be delivered successfully.

Similarly for the Subscription process used to send messages in the reverse direction.

For a new connection it is generally useful to test step by step through the connection chain so that there are only a few variables to get correct at each step.

4.2 Host Connection Credentials and Parameters

It is essential to establish a correct set of connection credentials and parameters to allow communication to be established between your gateway device and the IoT Host.

When you register a device to send messages to an IoT Host, you will get a set of credentials and settings that are necessary to include in connection requests. Typical parameters and the flags used in the *mosquitto_pub* command are listed below.

1. Host Address (-h)

This may be a fixed URL e.g.

thingspeak.com.au

or it may be specific to the device that you are registering e.g.

ahrqy70123abc-ats.iot.ap-southeast-2.amazonaws.com

2. Host Port Number (-p)

This is the connection port number, and is **1883** by default for many hosts, however AWS and Azure use **8883**.

3. Quality of Service (-q)

QoS values are generally 0, 1 or 2. Not all platforms support all values, but a value of 0 is safe to start with for all hosts.

4. Authentication Credentials

Authentication may use Username, with or without a Password, or may be based on an X.509 certificate scheme.

4.1 Username and Password (-u -P)

Registering your device (e.g. on ThingSpeak) will require creation of a Username and Password, which are generally used in the normal manner as part of the MQTT connection command, but Azure is an exception – see below.

4.2 Certificate Based (--cert --key --cafile)

In this case, registering your device (e.g. on AWS) will result in generation of a specific Certificate file and Public and Private Key files. For the MQTT connection command, the Certificate file and the private Key file need to be specified. In addition, a Certificate Authority file is also generally required. This file is common to all devices connecting to the Host, and a suitable CA file for several Host services may be pre-loaded as part of the Gateway firmware (e.g. CA files are pre-loaded for AWS and Azure on the Dragino Gateway devices.)

In the case of the Azure IoT platform, it requires Username and Password, but also requires a CA file to be specified. The Password string is long and complex, comprised of several components, and includes a space character, so care need to be taken to enclose the string in parentheses when used on the command line.

5. Client ID (-i)

A Client ID is usually generated as part of the registration process on the Host platform. Some basic broker platforms do not use the Client ID.

6. MQTT Topic (-t)

The Topic field determines how the data will be presented when it is published on the MQTT host.

For some Hosts, the Topic format is fixed, and may require a “Write API” key to allow publishing (e.g. for ThingSpeak, the Topic structure is:

```
'channels/<CHANNEL>/publish/<WRITE_API>'
```

For other hosts the Topic format is quite flexible e.g. the default Topic for AWS on the Dragino Gateway is: '`<CLIENTID>/<CHANNEL>/data`'

This structure serves to simply define the origin of the message in terms of nodes using a particular **Channel** on a Gateway device with a particular **Client ID**.

7. MQTT Data (-m)

The Data field contains the message payload for publishing.

For some hosts the structure is fixed in order to support predefined presentation of the data e.g. as graphs (e.g. for ThingSpeak, the data string looks like 'temp=34&hum=86&status=MQTTPUBLISH' where

Note: the data field is not used in the subscription command.

5 Troubleshooting the Publish Process

5.1 Overview

5.2 Testing Publish with `mosquitto_pub`

A basic test to check that the credentials and other parameters are correct can be made just using the `mosquitto_pub` command from a Linux terminal session on the Dragino Gateway device. This test uses just the `mosquitto_pub` utility and no other setting on the device, so it is a very basic check on the correctness of the command parameters.

You can build up a suitable command in a text editor as either a single (long) line of parameters, or you can make it multi-line for ease of readability and use continuation characters at the end of each line. In either case you can edit the text of the command in the editor, then copy and paste it into the command line.

Note that the `'-d'` flag enables the debug output from the command which provides a good indication of the progress and success of the command execution.

5.2.1 Single line examples:

Azure

```
# mosquitto_pub -d -h Dragino-MQTT-Test.azure-devices.net -p 8883
-u Dragino-MQTT-Test.azure-devices.net/Gateway-1
-P "SharedAccessSignature\ sr=Dragino-MQTT-Test.azure-devices.net
%2Fdevices%2FDraginoDevice-
1&sig=9Kb27uYoys96tunZyqYsyAWc1n8ktDhnEK3i%2F%2B8I%2Ffc
%3D&se=1594260870" -i Gateway-1
-t channels/200893/publish/B9Z0R25QNVEBKIFY -m "test message"
--cafile "/etc/iot/cert/Azure.cer"
```

AWS

```
# mosquitto_pub -d -h ahrqy70p2lw97-ats.iot.ap-southeast-
2.amazonaws.com -p 8883 -i Gateway-1 -q 1
-t devices/Gateway-1/SensorNode-3/messages/events -m "test message"
--cert "/etc/iot/cert/AWSiot.pem.crt" --key "/etc/iot/cert/AWSiot.pem.key"
--cafile "/etc/iot/cert/AmazonRootCA1.pem"
```

Thingspeak

```
# mosquitto_pub -d -h mqtt.thingspeak.com -p 1883 -u dragino_user  
-P mypassword -q 1 -i Gateway-3  
-t channels/820809/publish/ 8JIKADCBYSD21Z9B  
-m "field1=34&field2=89&status=MQTTPUBLISH"
```

5.2.2 Multi-line Examples:

Azure

```
# mosquitto_pub -d \  
-h Dragino-MQTT-Test.azure-devices.net \  
-p 8883 \  
-u Dragino-MQTT-Test.azure-devices.net/Gateway-1 \  
-P "SharedAccessSignature sr=Dragino-MQTT-Test.azure-devices.net  
%2Fdevices%2FDraginoDevice-  
1&sig=9Kb27uYoys96tunZyqYsyAWc1n8ktDhnEK3i%2F%2B8I%2Ffc  
%3D&se=1594260870" \  
-i Gateway-1 \  
-t channels/200893/publish/B9Z0R25QNVEBKIFY \  
-m "test message" \  
--cafile "/etc/iot/cert/Azure.cer"
```

AWS

```
# mosquitto_pub -d \  
-h ahrqy70p2lw97-ats.iot.ap-southeast-2.amazonaws.com \  
-p 8883 \  
-i Gateway-1 \  
-q 1 \  
-t devices/Gateway-1/SensorNode-3/messages/events \  
-m "test message" \  
--cert "/etc/iot/cert/AWSiot.pem.crt" \  
--key "/etc/iot/cert/AWSiot.pem.key" \  
--cafile "/etc/iot/cert/AmazonRootCA1.pem"
```

ThingSpeak

```
# mosquitto_pub -d \  
-h mqtt.thingspeak.com \  
-p 1883 \  
-u dragino_user \  
-P mypassword \  
-q 1 \  
-i Gateway-2 \  
-t channels/820123/publish/8JIKADCBYSD21Z9B \  
-m "field1=34&field2=89&status=MQTTPUBLISH"
```

Example Outputs

Typical **mosquitto_pub** debug output for QoS=0

```
Client Gateway-2 sending CONNECT  
Client Gateway-2 received CONNACK (0)  
Client Gateway-2 sending PUBLISH (d0, q0, r0, m1, 'channels/820123/publish/  
8JIKKXABCDE21Z9B', ... (38 bytes))  
Client Gateway-2 sending DISCONNECT
```

Typical **mosquitto_pub** debug output for QoS=1

```
Client Gateway-1 sending CONNECT  
Client Gateway-1 received CONNACK (0)  
Client Gateway-1 sending PUBLISH (d0, q1, r0, m1, 'Gateway-1/SensorNode-3',  
... (18 bytes))  
Client Gateway-1 received PUBACK (Mid: 1)  
Client Gateway-1 sending DISCONNECT
```

5.3 Testing Publish with `mqtt_process.sh`

The Gateway device uses the `mqtt_process.sh` script to get messages received from the LoRa radio and send them on to an IoT host platform via MQTT protocol.

For Publishing, the script looks for incoming message files in the `/var/iot` directory. When it finds a file, it uses the file name to look up the Channel definitions **Local ID** to find the corresponding **Remote ID**. It then builds up a `mosquitto_pub` command with the credentials and parameters defined in the MQTT Configuration.

Once you have a set of credentials and settings that work with the `mosquitto_pub` command, you can use these to complete the MQTT Configuration for your selected IoT server platform, then test the configuration using `mqtt_process.sh` from the command line, which will provide extensive diagnostics to show you what is going on.

The step by step testing process is as follows:

1. Go to the **Service/Lora Gateway** configuration page and set the **IoT Service** field to Disabled. This will prevent any incoming LoRa messages from interfering with the test.

2. Set the debug level to **10** by editing the file `etc/config/gateway`
In the file edit the line as below:

```
config settings 'general'  
option DEB '10'
```

This setting will maximise debug output, including debug output from `mqtt_process.sh` which will appear in the terminal session after you start the script.

3. Define a Channel

Go to the MQTT Channel Management screen and set up a channel definition. The **Local ID** will be used to simulate an incoming LoRa message from a Sensor Node, and the **Remote ID** will be used to represent the Sensor Node on the IoT platform.

Your IoT platform may require the **Write API** field to be completed (eg ThingSpeak), otherwise leave it blank.

4. Open three terminal sessions on the Gateway device and run commands as follows:

A. Logread

Terminal Session #1

```
# logread -f
```

This session will show information being added to the system log.

B. Start the MQTT process:

Terminal Session #2

```
# mqtt_process.sh
```

This session will start the script and display debug output.

C. Send a message:

Terminal Session #3

```
# echo "Test message" > /var/iot/channels/<channel Local ID>
```

This session is used to send test messages by creating message files which are read and processed by the **mqtt_process.sh** script. Use the Local ID of the channel definition that you created above.

Typical debug output for three different IoT platforms is shown below:

1. ThingSpeak

Message command:

```
# echo "field1=55&field2=93" > /var/iot/channels/DSN-2
```

Channel:

Local ID: DSN-2

Remote ID: 820123

Write API: 8JIKKXABCDE21Z9B

Output:

Parameters

server: mqtt.thingspeak.com

port: 1883

user: my_user_name

pass: 7B6MABCDE7BS3JWB

QoS: 0

cert:

key:

cafile:

clientID: DraginoGateway-2

remoteID: 820123

topic: channels/820809/publish/8JIKKXABCDE21Z9B

mqtt_data: field1=55&field2=93&status=MQTTPUBLISH

Client DraginoDevice-2 sending CONNECT

Client DraginoDevice-2 received CONNACK (0)

Client DraginoDevice-2 sending PUBLISH (d0, q0, r0, m1, 'channels/820809/publish/8JIKKXABCDE21Z9B', ... (38 bytes))

Client DraginoDevice-2 sending DISCONNECT

2. AWS

Message command:

```
# echo "temp:12.3 hum:45.6" > /var/iot/channels/DSN-1
```

Channel:

Local ID: DSN-1

Remote ID: DraginoSensorNode-1

Write API:

Output:

Parameters

server: ahrabc0p12397-ats.iot.ap-southeast-2.amazonaws.com

port: 8883

user:

pass:

QoS: 1

cert: /etc/iot/cert/AWSiot.pem.crt

key: /etc/iot/cert/AWSiot.pem.key

cafile: /etc/iot/cert/AmazonRootCA1.pem

clientID: DraginoDevice-1

remoteID: DraginoSensorNode-1

topic: DraginoDevice-1/DraginoSensorNode-1

mqtt_data: temp:12.3 hum:45.6

Client DraginoDevice-1 sending CONNECT

Client DraginoDevice-1 received CONNACK (0)

Client DraginoDevice-1 sending PUBLISH (d0, q1, r0, m1, 'DraginoDevice-1/DraginoSensorNode-1', ... (18 bytes))

Client DraginoDevice-1 received PUBACK (Mid: 1)

Client DraginoDevice-1 sending DISCONNECT

3. Azure

Message command:

```
# echo "Test message" > /var/iot/channels/DSN-3
```

Channel:

Local ID: DSN-3

Remote ID: DraginoSensorNode-3

Write API:

Output:

Parameters

server: Dragino-MQTT-Test.azure-devices.net

port: 8883

user: Dragino-MQTT-Test.azure-devices.net/DraginoDevice-3

pass: SharedAccessSignature sr=Dragino-MQTT-Test.azure-devices.net
%2Fdevices%2FDraginoDevice-3&sig=HcbA2hUd%2FYGWw5yI3J3v
%2Fjy9Gj95uipJlrMMxvQuWx0%3D&se=1594713233

QoS: 1

cert:

key:

cafile: /etc/iot/cert/Azure.cer

clientId: DraginoDevice-3

remoteID: DraginoSensorNode-3

topic: devices/DraginoDevice-3/messages/events/DraginoSensorNode-3/

mqtt_data: Test message

Client DraginoDevice-3 sending CONNECT

Client DraginoDevice-3 received CONNACK (0)

Client DraginoDevice-3 sending PUBLISH (d0, q1, r0, m1,

'devices/DraginoDevice-3/messages/events/DraginoSensorNode-3/', ... (18 bytes))

Client DraginoDevice-3 received PUBACK (Mid: 1)

Client DraginoDevice-3 sending DISCONNECT

5.4 Testing Publish with LoRa Messages

In order to test end-to-end from a Sensor Node through the Gateway and on to the IoT Platform, you can use a simple device equipped with a LoRa radio to generate messages.

This example uses an Arduino board fitted with a LoRa shield and running a simple sketch that generates a series of messages containing random data values for Temperature and Humidity from a simulated sensor. The sketch code is shown in the following section.

5.4.1 Set up the IoT Host Platform configuration

Set up the configuration for the IoT Host platform where you want the messages to be published. (See previous section for guide)

Define a **Channel** that has the **Local ID** matching the “**device_ID**” string used in the Arduino sketch (e.g. the sketch code shown uses “10009” as the device_ID string).

5.4.2 Set up Gateway LoRa Radio Settings

Assuming that the Gateway device has been newly flashed and has the default settings, you need to check that the settings match the Arduino LoRa shield settings in the sketch code. The example code uses a **Frequency** setting of “915000000” or 915.0MHz. Your devices may use another default frequency, for example “868000000” or 868.0MHz in Europe.

Edit the sketch code to suit the LoRa Shield radio, and set the Gateway to the same frequency.

Go to the **Service/LoRa Gateway** configuration page as shown below and set the following:

1. IoT Service: “LoRaRaw Forward to MQTT Server”
2. Debug Level: Set to highest level
3. Radio Settings / Frequency: Same as sketch code setting
4. Radio Settings / LoRa Sync Word: Same as sketch code setting.
Default 52 (0x34)
5. Leave all other fields on the screen blank or set to their default setting.
6. Save the settings and then go back and check that all settings have been saved correctly.
7. Reboot the Gateway to ensure that all settings have are correctly applied when the LoRa radio is started.

5.4.3 Run the test

1. Open a terminal session and run:

```
# logread -f
```

This will display information being written to the system log.

2. Start the Arduino sketch code

Launch the Arduino IDE on your PC and start the sketch.

When the Arduino sketch is started, it will send a new message every 30 seconds.

3. Open the Serial Monitor on the Arduino IDE to monitor the message sending process.

4. The incoming LoRa messages will be shown in the Gateway log display terminal session, followed by the MQTT message parameters used to send the message on to the IoT Host platform.

If no messages appear, then it is most likely that the radio settings do not match between the Arduino sketch and the Gateway, or that the Arduino and LoRa shield are not working.

A typical log output is shown below:

```
daemon.info lg01_pkt_fwd[1458]: RXTX~  
Receive(HEX):3c31303030393e54656d703d33332048756d3d383820436f756e743d3335  
  
user.notice root: [IoT.MQTT]: Find Match Entry for 10009  
user.notice root: [IoT.MQTT]:  
user.notice root: [IoT.MQTT]:-----  
user.notice root: [IoT.MQTT]:Parameters  
user.notice root: [IoT.MQTT]:server[-h]: ahrabcqy70p12397-ats.iot.ap-southeast-  
2.amazonaws.com  
  
user.notice root: [IoT.MQTT]:port[-p]: 8883  
user.notice root: [IoT.MQTT]:user[-u]:  
user.notice root: [IoT.MQTT]:pass[-P]:  
user.notice root: [IoT.MQTT]:pub_qos[-q]: 1  
user.notice root: [IoT.MQTT]:cafile[--cafile]: /etc/iot/cert/AmazonRootCA1.pem  
user.notice root: [IoT.MQTT]:cert[--cert]: /etc/iot/cert/AWSiot.pem.crt  
user.notice root: [IoT.MQTT]:key[--key]: /etc/iot/cert/AWSiot.pem.key  
user.notice root: [IoT.MQTT]:clientID[-i]: DraginoDevice-1  
user.notice root: [IoT.MQTT]:remote_id: DraginoSensorNode-1  
user.notice root: [IoT.MQTT]:pub_topic[-t]: DraginoDevice-1/DraginoSensorNode-1/data  
user.notice root: [IoT.MQTT]:mqtt_data[-m]: Temp=33 Hum=88 Count=35  
user.notice root: [IoT.MQTT]:-----
```

5.4.4 Example Arduino Sketch

```
/*  
LoRa_Sender_MQTT:  
Support Devices: LoRa Shield + Arduino  
Requires Library:  
  https://github.com/sandeepmistry/arduino-LoRa
```

Example sketch sends a message every 30 seconds using a simple protocol which will be processed by the Dragino Gateway device to send the payload on to a host IoT platform.

The End node will send out a message string:

```
"<device_ID>field1=${tem}&field2=${hum}" (ThingSpeak format)
```

or

```
"<device_ID>Temp=${tem} Hum=${hum}" (General format)
```

When the LG01/LG02 gateway gets the data, it will parse the data string and forward the data to the IoT platform via MQTT protocol.

Message information is also output to the Arduino Serial Monitor.

Modified Nov 3 2019

by Dragino Technology Co., Limited <support@dragino.com>

```
*/  
#include <SPI.h>  
#include <LoRa.h>  
long tem,hum;  
int count=1;  
int device_id=10009; // ID of this End node.  
// Match to Gateway Channel Local ID definition  
  
void setup() {  
  Serial.begin(9600);  
  //while (!Serial);  
  Serial.println("LoRa Sender");  
  
  if (!LoRa.begin(915000000)) { // Match to Gateway frequency setting  
    Serial.println("Starting LoRa failed!");  
    while (1);  
  }  
  LoRa.setSyncWord(0x34); // Match to Gateway sync setting  
}
```

```
void loop() {
  tem = random(25,35); // Generate a random temperature.
  hum = random(85,95); // Generate a random humidity.

  Serial.print("Sending packet: "); Serial.print(count);
  Serial.print(" device_id: "); Serial.print(device_id);
  Serial.print(" tem: "); Serial.print(tem);
  Serial.print(" hum: "); Serial.println(hum);

  Serial.print(" Data: <");
  Serial.print(device_id);
  // LoRa.print(">field1="); // ThingSpeak
  Serial.print(">Temp="); // General
  Serial.print(tem);
  // LoRa.print("&field2="); // ThingSpeak
  Serial.print(" Hum="); // General
  Serial.print(hum);
  Serial.print(" Count="); // General
  Serial.println(count); // General
  delay(100);

  // compose and send packet
  LoRa.beginPacket();

  LoRa.print("<");
  LoRa.print(device_id);
  // LoRa.print(">field1="); // ThingSpeak
  LoRa.print(">Temp="); // General
  LoRa.print(tem);
  // LoRa.print("&field2="); // ThingSpeak
  LoRa.print(" Hum="); // General
  LoRa.print(hum);
  LoRa.print(" Count="); // General
  LoRa.print(count); // General

  // LoRa.print(counter);
  LoRa.endPacket();
  count++;
  delay(30000); // Wait 30 seconds before sending the next message
}
// End of code
```


6 Troubleshooting the Subscription process

6.1 Overview

To troubleshoot the subscription process you should take a similar layered approach as used for the publishing process as follows:

- a) Test with ***mosquitto_pub*** and ***mosquitto_sub*** from the command line to ensure that you have the basic publish and subscribe mechanisms working with the correct credentials and parameters.
- b) Test with ***mqtt_process.sh*** from the command line to ensure that you have entered the credentials and parameters correctly in the GUI.
- c) Test with LoRa radio

6.2 Testing Subscription with ***mosquitto_sub***

Initially use ***mosquitto_pub*** and ***mosquitto_sub*** commands on the gateway device to publish a test message to a topic on your MQTT broker, and to subscribe to the same topic so you can receive the message.

This test shows that you have established communication with the host and have the correct credentials and parameters.

Your example Publish command is:

```
# mosquitto_pub -h 192.168.2.3 -p 1883 -t openhab/DATA -m {"test"}
```

So to Subscribe to the same topic and receive the test message, use a command like this:

```
# /usr/bin/mosquitto_sub -d -h 192.168.2.3 -p 1883 -t openhab/DATA
```

Run this Subscribe command in a terminal session, then run the Publish command in a separate terminal session.

The debug information showing you the progress of the subscription will appear on the command line.

Enter Ctrl-c to terminate the Subscribe command.

Example Subscribe Commands

Azure

```
# mosquitto_sub -d \  
-h Dragino-MQTT-Test.azure-devices.net \  
-p 8883 \  
-u Dragino-MQTT-Test.azure-devices.net/Gateway-1 \  
-P "SharedAccessSignature sr=Dragino-MQTT-Test.azure-devices.net  
%2Fdevices%2FDraginoDevice-  
1&sig=9Kb27uYoys96tunZyqYsyAWc1n8ktDhnEK3i%2F%2B8I%2Ffc  
%3D&se=1594260870" \  
-i Gateway-1 \  
-t channels/200893/publish/B9Z0R25QNVEBKIFY \  
--cafile "/etc/iot/cert/Azure.cer"
```

AWS

```
# mosquitto_sub -d \  
-h ahrqy70p2lw97-ats.iot.ap-southeast-2.amazonaws.com \  
-p 8883 \  
-i Gateway-1 \  
-q 1 \  
-t devices/Gateway-1/SensorNode-3/messages/events \  
--cert "/etc/iot/cert/AWSiot.pem.crt" \  
--key "/etc/iot/cert/AWSiot.pem.key" \  
--cafile "/etc/iot/cert/AmazonRootCA1.pem"
```

ThingSpeak

```
# mosquitto_sub -d \  
-h mqtt.thingspeak.com \  
-p 1883 \  
-u dragino_user \  
-P mypassword \  
-q 1 \  
-i Gateway-2 \  
-t channels/820123/publish/8JIKADCBYSD21Z9B \  

```

6.3 Testing Subscription with `mqtt_process.sh`

Once you have successfully run the basic Publish and Subscribe test, then you can move on to setting up the Publish and Subscribe parameters in the GUI and run `mqtt_process.sh` from the command line with debug set to **10** which will put the debug information on the command line after you run the `mqtt_process.sh` command.

6.4 Testing Subscription with LoRa Radio

Once you can see the `mqtt_process.sh` process successfully subscribed to your test messages, then you can check that the test messages are being sent out from the LoRa radio in the gateway, and received on your LoRa shield.

You will need to make sure that the LoRa radio parameters set up on the MQTT configuration page of the gateway match those used by the LoRa shield.

The `mosquitto_sub` process on the Dragino devices has been extended to include the LoRa radio parameters and to interface with the LoRa radio driver software. It uses an additional flag (-z) at the end of the command to pass the LoRa parameters (freq, data rate, coding) as follows:

```
-z --freq $FREQSTR --datr $DATR --codr $CODR
```

You will see this in the log output with `logread -f` command so you can check the LoRa radio parameters actually being used.

7 References

<https://www.hivemq.com/mqtt-essentials/>

<http://mosquitto.org/>

<https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs.html>

https://thingspeak.com/pages/learn_more

<https://docs.microsoft.com/en-us/azure/iot-hub/>