Modulus MQTT

Protocol Description & Implementation

Rev A2

www.iclinks.com

530.888.1800

Contents

Overview

MQTT (**M**essage **Q**ueuing **T**elemetry **T**ransport) is becoming popular as a protocol for SCADA (Supervisory Control and Data Acquisition) and IoT (Internet of Things) projects.

MQTT operates quite a bit differently from a traditional SCADA protocol such as Modbus. With a traditional protocol, there is a connection made between the two devices that are going to exchange data; usually a "Master" and a "Slave". The Master initiates a transaction to read or write a block of data and the addressed Slave responds.

In an MQTT system, there are three types of players instead of just the two:

- **Publishers** that generate the data that is required elsewhere in the system.
- **Subscribers** that are the consumers of the generated data.
- A **Broker** that receives the data from Publishers and distributes it to the Subscribers that want that data.

There is no direct link between the Publishers and the Subscribers. Everything goes through the Broker. A device can be both a Subscriber and a Publisher depending on its role in the system.

When a new Subscriber comes on-line, it logs into the Broker and subscribes to the "topics" (data) that it needs. The Subscriber and Broker maintain this connection. A topic is a tag name for the data from a Publisher. The topic is frequently a single piece of data, although it can also be a block of data. More on that later.

Publishers publish updated data for one or more topics by sending the data to the Broker. The Broker then looks to see what Subscribers have "signed up" for that topic, and forwards the data to them.

The one thing that stands out right away is that Subscribers cannot request data in the traditional sense. They must send a piece of data (bit?) that causes a Publisher to send an update for that topic to the broker who then distributes that data to all of the Subscribers signed up for that topic.

The big advantages of this system are:

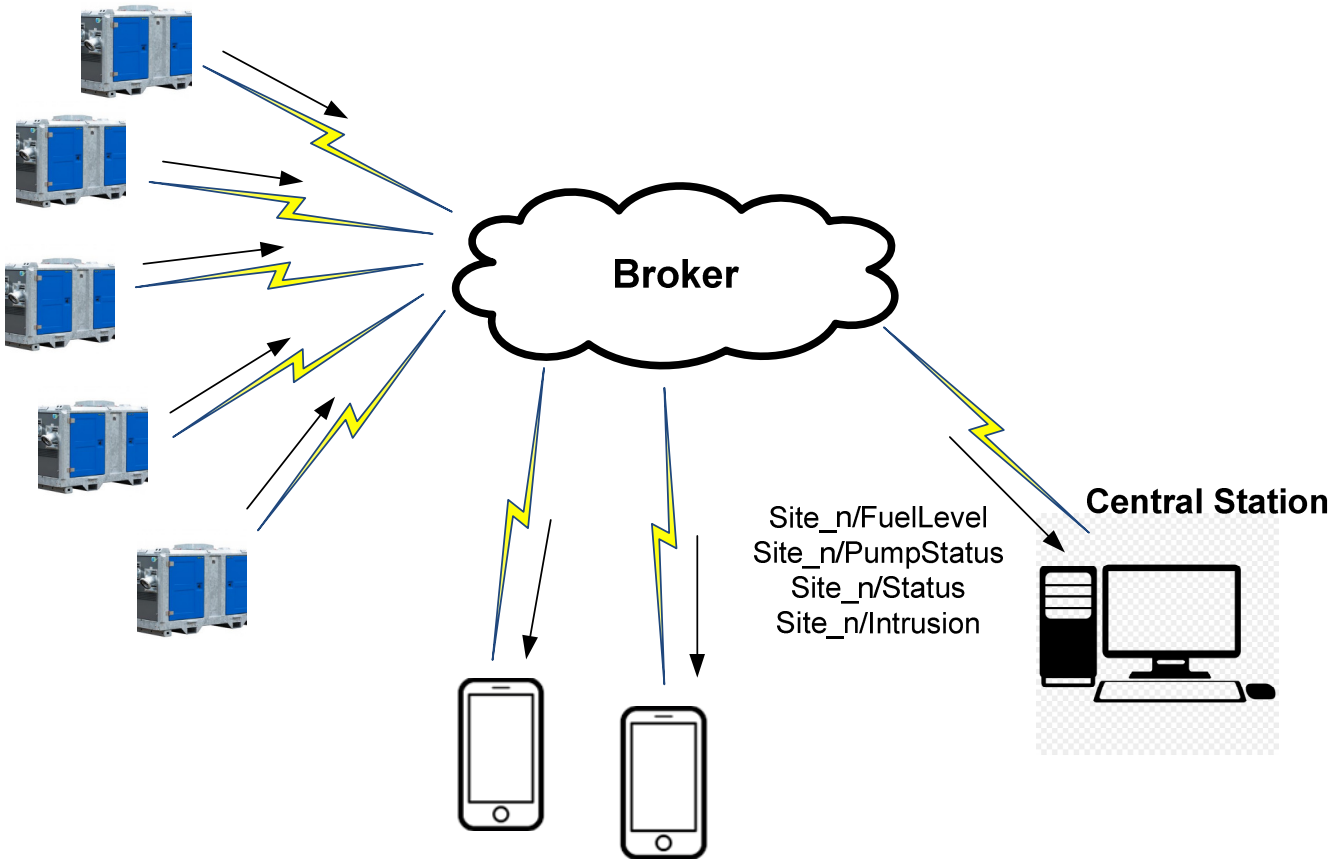
- **ANONIMITY** - None of the Publishers and Subscribers need to know information about any of the other devices in the system (such as IP addresses). They just need to know the (topic) name of the data. Devices in the field can be deployed without individual site specific configuration (Modulus controllers can use their serial numbers to automatically identity them in a system without manual configuration).
- **SECURITY** - Incoming sockets are not needed by MQTT. Modulus controllers allow the Ethernet and Cellular IP ports to be "turned off" to incoming data traffic from the Internet that can be used for attacks such as Denial of Service.
- **STATIC IPs NOT NEEDED** - The only entity that normally has a static IP address is the broker. All of the other nodes (Publishers and Subscribers - the remote sites) don't need a static IP.
- **WORKS WITH EXISTING FIREWALLS** - Corporate IT people like MQTT because they don't have to open up holes in their security firewalls to allow a SCADA device to come in via the Internet. All of the Publishers and Subscribers are clients going the other way logging into the Broker.

Another big advantage of MQTT is that it is quickly being widely adopted and there is an amazing amount of free and low cost resources like brokers "in the cloud" and apps that can be used to implement monitoring and SCADA systems.

A disadvantage of MQTT is its "openness". It's important to understand this if your data is mission critical. MQTT can be used with and without SSL/TLS encryption, and most MQTT brokers support the optional use of usernames and passwords to help secure the data. Unfortunately, if these features are not used, anyone that knows a portion or all of a topic name can snoop on the topic data or potentially insert "fake" data into a system for that topic. ICLs Modulus controllers support SSL/TLS encryption and broker username/password logins.

System Example - Monitoring/Alarming

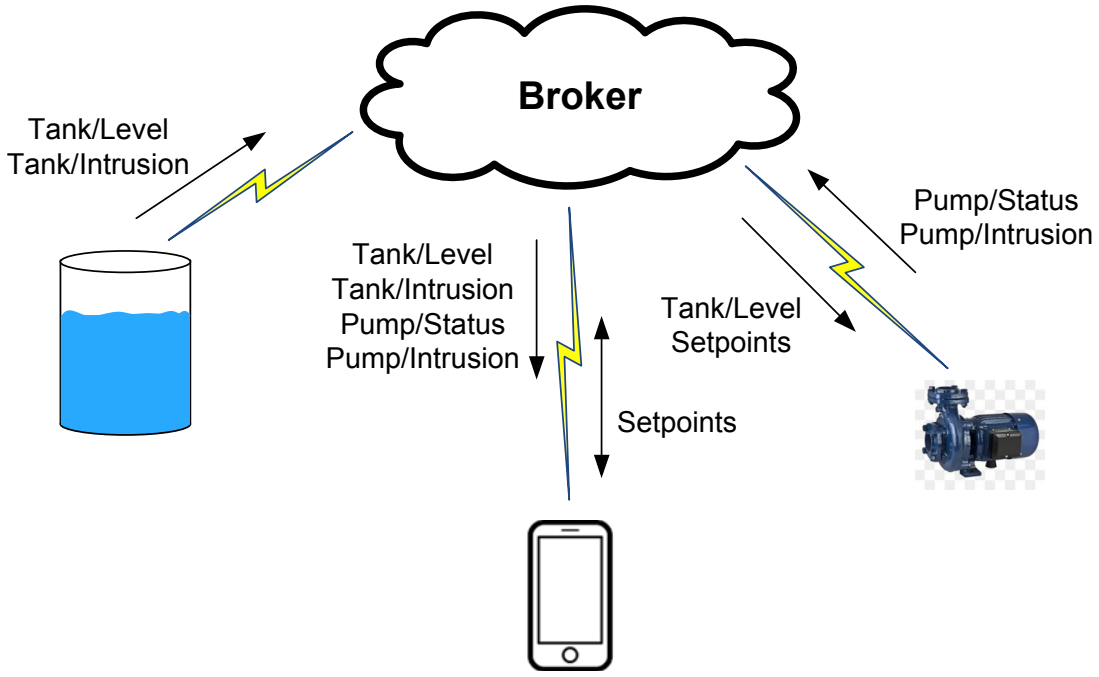
Systems with a large number of sites, but only a few I/O points per site are a good type of application for MQTT. For example, a company that leases pump skids. A small controller on each skid monitors various conditions such as fuel level, pumping status and runtimes, faults and intrusion. It might also provide latitude and longitude coordinates for the site location. With MQTT, the controllers at the sites can be deployed without any local configuration. They simply collect the sensor information and publish it to the broker.



Each site will publish its sensor information using the controller serial number as a unique prefix for each topics tag name. A Central Station can access this data by subscribing to the topics with all of the unit serial numbers. Likewise, users can access this same data and receive alarms/notifications by subscribing to the data on their cell phones/mobile devices. Unlike a conventional system, with MQTT, essentially any number of devices can subscribe to the data for which they have credentials to log into the broker account, and know the topic names <(serial number>/<signal name> for the data that they want to monitor and alarm from. No reconfiguration is needed as new users are brought on. The individual sites don't know (or need to know) who is subscribed to the data.

System Example - SCADA Control

A small simple SCADA system that controls a well pump to maintain the water level in a tank might look like this:



In the above system, the tank is a Publisher for the level data as well as an intrusion contact. The pump and the cell phone are Subscribers to this data. When the Broker receives data for the level "topic", it forwards that data to both the pump and the phone. Only the phone is subscribed to the intrusion data topic, so the broker only forwards that data to the phone.

The pump is both a Publisher and Subscriber. It receives level data and setpoints as a subscriber to make control decisions, and publishes pump status and setpoint feedback data.

Likewise, the phone is both a subscriber (for level, status, intrusion, and setpoint data) and a publisher (for setpoint changes).

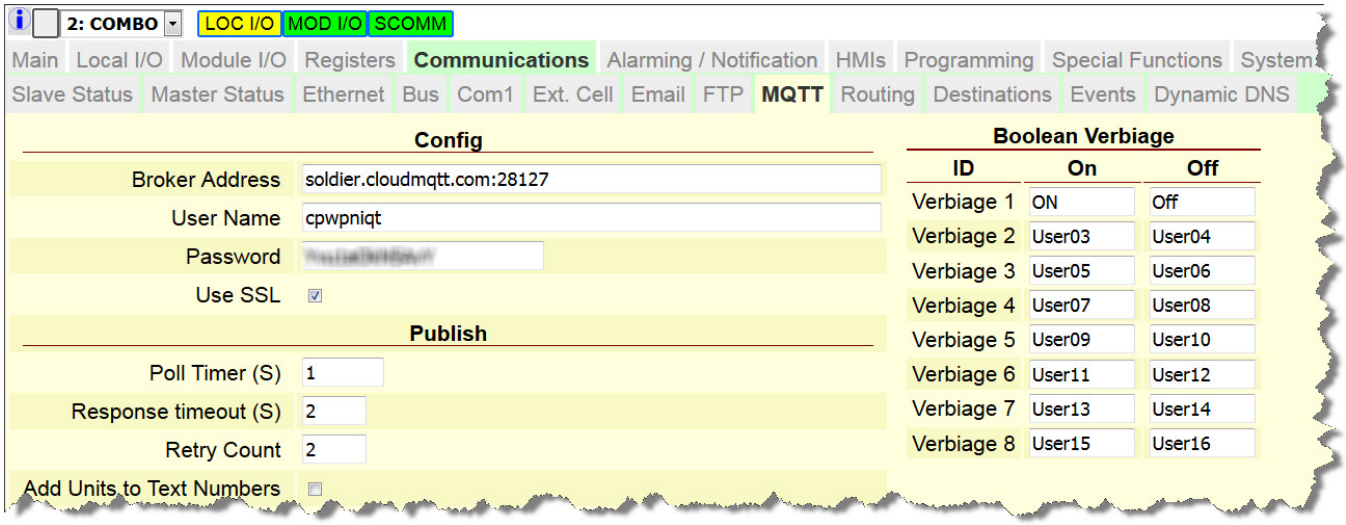
None of the three sites in this system know any details about the others. They simply subscribe to the data that they need, and produce the data needed by the rest of the system. The broker in the cloud manages the routing of the produced data to the subscribers that want it.

MQTT is unusual in that it does not specify the type and format of the data (also known as the payload) that is carried. Most MQTT mobile device apps utilize single pieces of ASCII data for each topic. So for example, the tank (above) would typically publish two separate messages with a human readable ASCII tank level, and an ASCII 0 or 1 for the intrusion switch. Modulus controllers also support exchange of blocks of binary data, more like a conventional SCADA system. This is more appropriate for when larger amounts of data must be handled.

Modulus MQTT Configuration

Broker

Communications configuration items specific to using the MQTT protocol are configured under the MQTT tab. These are primarily settings related to accessing the MQTT broker. Actual MQTT data transactions are configured under the Destinations and Events tabs.



Config

Broker Address

Enter the static IP address or URL name of the MQTT broker that you are using. If you are using a name, be sure to also configure the IP address of the name server ("DNS Server IP") under the Ethernet tab.

Username and Password

These fields are only needed if the broker is set up to use them. They only control access to the broker itself.

Use SSL

Check this box to enable SSL/TLS encryption of the MQTT packets. Be sure that the broker that you are using supports SSL encryption if you select this option (not all brokers do).

Use Cellular Data

Check this box to use MQTT through a built-in cellular modem (instead of the Ethernet port). This option is only available in Modulus Communications modules with a built-in cellular modem.

Publish

Poll Timer (S)

MQTT events will cause data to be published to the Broker at a periodic (polling) rate, and/or when triggered by logic in a control program. This value sets the rate at which events configured for polling are published.

Response Timeout (S) and Retry Count

If the Response Timeout value is non-zero, the controller looks for an acknowledgement message from the broker for each message that it publishes. This value sets the maximum time that the controller waits for that acknowledgement before retrying. A non-zero value also enables using a QoS (Quality of Service) level of 1. The number of retry attempts before a communications failure is annunciated is set in the Retry Count value. Note that this only checks for communications to the Broker. It does not include the entire path to the Subscribers.

Add units to Text numbers

Checking this box will cause the Controller to append the units text in the N register or I/O points configuration to each numeric value published as text.

Boolean Verbiage

ID

An identifying name for a Boolean verbiage pair.

On and Off

Eight different pairs of on and off text can be configured for publishing Boolean textual data. This text is used to represent the states of the data in textual MQTT messages. Typical verbiage pairs are On/Off, Running/Stopped, high/OK, low/OK, etc. If a verbiage pair is not selected, "0" and "1" will be used.

Topics - Destinations and Events

MQTT messages have "topics" (data tags) that consist of two or more parts. The first part is the "Destination". Typically, this would represent the site location such as "Tank 1", "Well 6", etc. The second part is the Event name. In many cases, this may be the name of a single piece of data such as "Tank Level". Modulus controllers do support transferring data as blocks.

The final topic name is the:

Destination Name / **Event Name**

Within each name, special text can be included in square brackets:

[#serial] inserts the serial number of the controller.

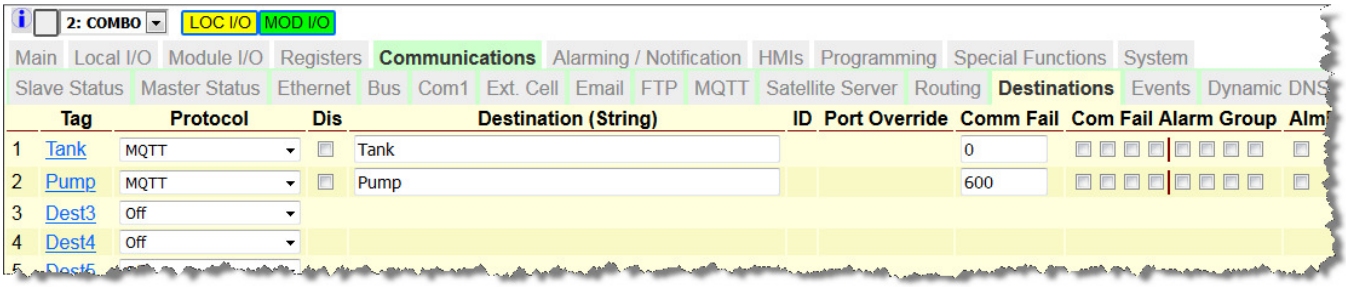
[#name] inserts the Unit Name as configured under the **HMI** | **General** tab.

[any text] inserts the text contained with the brackets verbatim. This allows for more complex topic names, including topics with "/" characters representing more than two topic sections.

Both the Destination Name and the Event Name may be up to 32 characters each. The total topic name may be up to 80 characters.

Destinations

Destinations represent sites, and are used as the first portion of topic names.



Destination
A name that typically describes the site. This is for descriptive purposes only.

Protocol
MQTT

Dis (disable)
Checking this box disables publishing and receipt of subscribed messages for a destination. This can be useful for system troubleshooting.

Destination (String)
This is the first part of the topic (Destination Name).

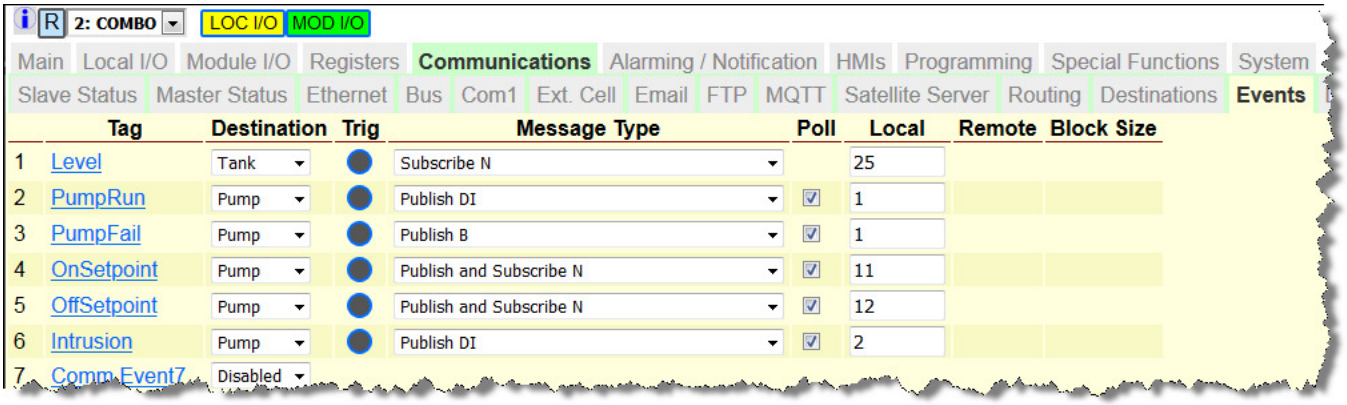
Comm Fail
Setting this to a non-zero value enables a timer in the controller that verifies that messages are being received from a particular destination that has been subscribed to. The time value is in seconds and can be set up to 65535 (18.2 hours). Any message received with a topic from that destination will restart the communications timer for that destination.

Comm Alarm Groups and Disable

If a communications failure is detected (see above), an alarm can be initiated to one or more alarm groups as selected by a set of 8 check boxes. Alarms for a particular destination can be disabled by checking the destination's **AlmDis** box.

Events

Events describe the actual data transfers (publishes and subscribes) and the source or destination of that data in the controller.



Event

A name that typically describes the type of data being transferred (i.e. a tank level, a flow, a pump status, setpoints, etc.)

Destination

The site name configured under the Destinations tab. The entire MQTT topic is the:

Destination Name / Event Name

So for example, the topic *Tank1/Level* might be the full topic name that would be used for a Subscriber to monitor the water level in the tank. Note that the slash is automatically added between the destination and event names in the topic by the controller.

Message Type

Message Type not only describes the type of data to be transferred, but the mode (publish or subscribe) and the representation (ASCII or binary). ASCII text selections transfer one piece of data at a a time, while binary selections can transfer data in blocks. The mode and type selections include:

- **Subscribe** and write to analog or discrete output(s), or local N (numeric), B (Boolean), or string (text) register(s).
- **Publish** analog or discrete input(s) or output(s), or local N (numeric), B (Boolean), or string (text) register(s)
- **Publish and Subscribe** local N (numeric), B (Boolean), or string (text) register(s). This combines the functionality of separate publish and subscribe actions in a single event.

For each of the selections, there is a "RETAINED" form. When a transfer is done as retained, the Broker holds onto the data. This alllows off-line devices to get the last value sent for a piece of data

when it comes on line. Non retained data is simply passed to the current set of subscribers as a publishing device creates it.

Textual Publish and Subscribe

The standard message types at the top of the Message Type list publish and subscribe as ASCII text which is what most mobile devices expect. All text functions publish or subscribe to a single piece of data per topic.

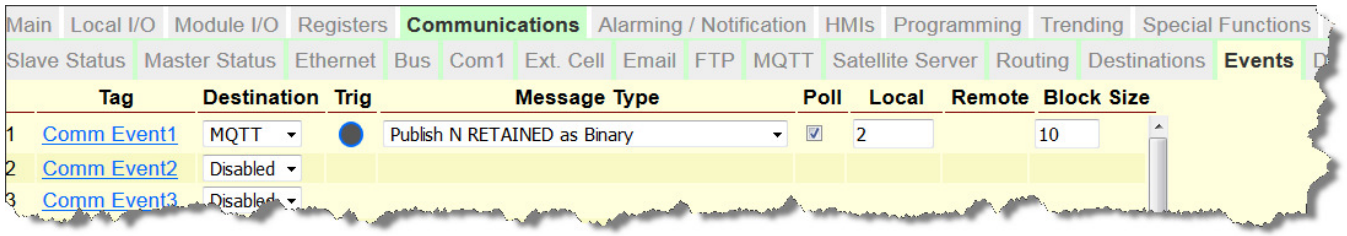
In standard textual data transfers, Booleans are represented as ASCII "0" and "1" for OFF and ON. An example of this is "PUBLISH B". If you would prefer to have text other than 0 or 1, such as "OFF" and "ON", select one of the "verbiage n" functions. Be sure to specify the text for the two states under the appropriate pair under the COMMUNICATIONS | MQTT tab.

For example, PUBLISH B (Verbiage 1). The (verbiage 1) causes the controller to use the text in the first pair of user configured textual elements. There are 8 different pairs available.

In some cases, it may be desirable to send an entire textual message for an event. For example, to publish and subscribe to alarm messages. In this case, use the Publish String_Buffer or Subscribe String_Buffer . Strings are created in the controller programming section.

Binary Publish and Subscribe

When data is being passed between controllers instead of human interface devices (i.e.phones), it is more efficient to transfer the data as block of binary data binary instead of individual textual elements. For this purpose, use Publish or Subscribe "as Binary" events. When these types of events are selected, an additional set of values specify the starting Local register or I/O address, and the number of elements (Block Size) to be transferred.



Note that amount of data exchanged per topic (message size) may be limited by the broker that you use and the "plan" that you select (paid plans generally support larger messages than the free ones). Verify this limit if you plan to transfer larger packets of data.

Poll

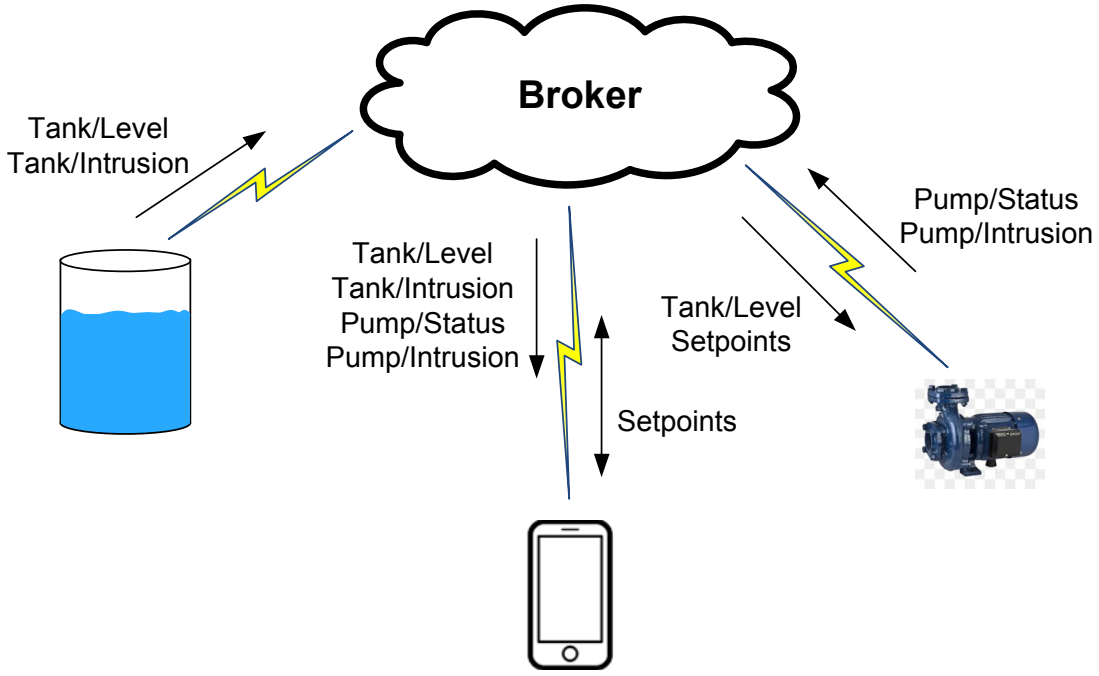
Check this box if the event should execute periodically at the polling rate defined under the MQTT tab. Uncheck it to strictly use triggers in the programming section. This only applies to publish events. It is ignored for subscribe events.

Local and Block Size

The Local field defines the starting registers in the module (local) for transfer of data for this event. The first piece of data to be transferred will be the specified I/O point or register. When multiple pieces of data are transferred, the additional points or registers will be the next successive ones. The Block Size is the number of values to be exchanged for binary transfers. When transfers are done as ASCII data, the block size is always one and not displayed.

Sample System Setup

To demonstrate how to set up an entire MQTT system, we'll use the sample system configuration from above. This system consists of a tank site that feeds level information to a pump site that controls the pump to maintain the water level in the tank. A smart phone is used to access the level, pump status and any alarms such as intrusion, pump failure, etc. The phone can also be used to adjust the setpoints that determine the water level to turn the pump on and off.



STEP 1 - Get a Broker account

The heart of any MQTT system is the broker. For initial testing, you can use an open public broker such as Hive MQ (www.hivemq.com/public-mqtt-broker). The problem is that anyone that knows the topic names on a public broker can access the data in your system. Ultimately for privacy and security, you will want to register for your own account. For very small configurations (typically around five connections or less), this can be done for no charge. For "real" production systems, you will want to get a paid account or create your own secure broker. A paid account with 100 connections or less will cost from $5 to $20 per month (i.e. www.cloudmqtt.com/plans.html).

Note: This is just an example, not an endorsement. A quick web search will reveal multiple available broker sites as well as broker source code if you want to do your own.

When you sign up for an account, you will receive a server address, username, password, and possibly a port# if the broker uses a non-standard port (which in this example, cloudmqtt does).

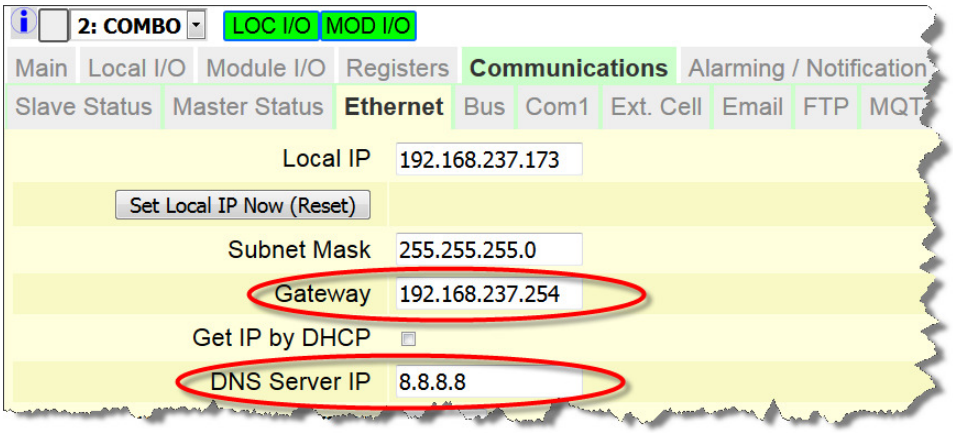
The server address will normally not be an IP address but a name that the controller will need to translate into an IP address. For this, you will need to configure the controller with the IP address of a DNS (name) server. There are multiple free DNS servers available. For example:

8.8.8.8 Google Public DNS

208.67.222.222 Open DNS (Cisco)

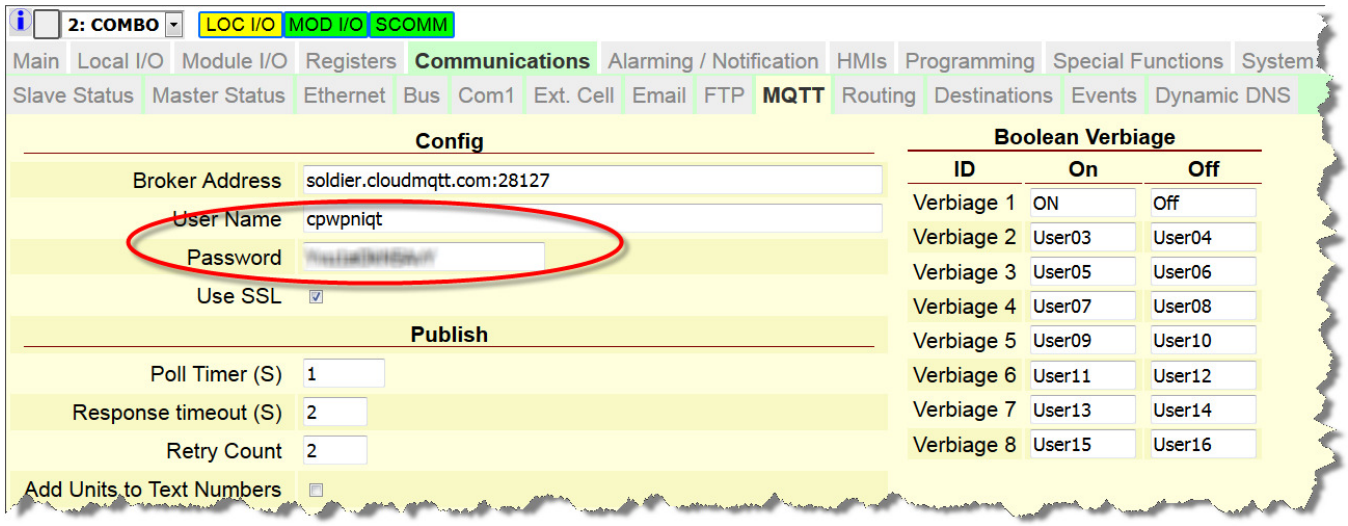
For every controller in the system (both Tank and Pump in this system) . . .

you will need to enter the Gateway address for your network and the address of a DNS server under the **COMMUNICATIONS** | **ETHERNET** tab:



Note that the **Gateway Address** will most likely be different from the one above (that one is for our network). If you are not sure, check with your IT people. If you are using an internal cellular modem, MQTT will not need this gateway address.

You will need to enter the broker information under the **Communications** | **MQTT** tab:



Note that the :28127 after the URL is only needed if the broker uses a non-standard port number. The standard port numbers for MQTT are 1883 for non-SSL and 8883 for using SSL security.

If your broker is set to use SSL security, check the next box (Use SSL) as shown.

Some Modulus Communications modules have a built-in cellular modem. If the connection to the Internet will be by internal cellular modem instead of the Ethernet port, check the Cellular Data box.

Under the Publish heading are the parameters that determine how fast polled published data is sent to the broker and if the controller should use retries if the message is not acknowledged within the specified "Response Time" period. The defaults of 1 second **Poll Time**, 3 second **Response Timeout** period and 2 **Retries** will work well for most systems.

Technical note: When using retries, the controller uses QoS (Quality of Service) level 1. With retries disabled (0), it uses QoS level 0.

In cellular systems, you may want to stretch out the polling time to reduce data charges, especially if triggered "on change" operation is also being used.

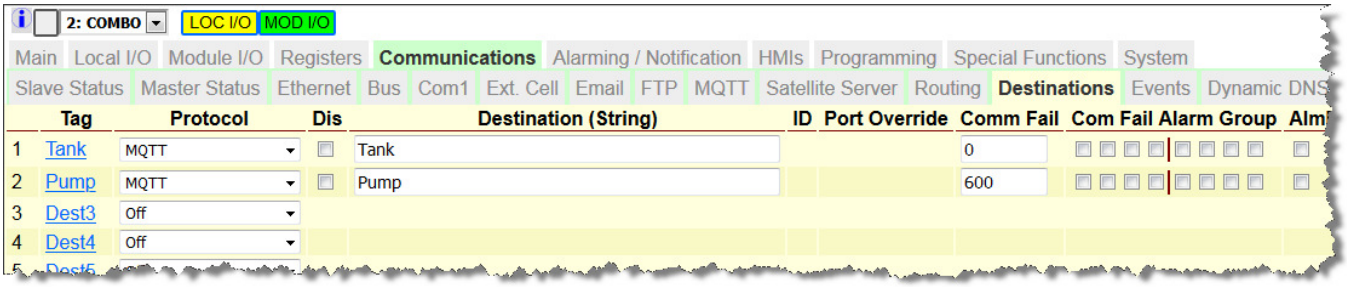
STEP 2 - Set Up MQTT Topics and Transactions

MQTT topics are the tag names that get associated with the data being exchanged. Topics have two parts; the Destination (site name) and the Event (specific data from that site). When the MQTT topic is sent, the controller automatically places a slash (/) between the two halves so it looks like this:

Destination/Event

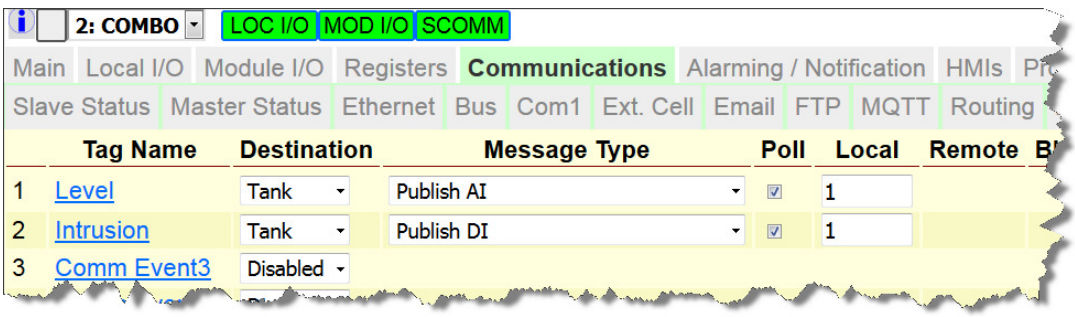
It is possible to support more than two pieces for a topic if needed, but in this example, we'll use the most common two-piece case.

The Destination name is set up under the **Communications** | **Destinations** tab:



For the first site that we are configuring, we will use **Tank** as the destination (in the example, we've also made the tag name "Tank", but the controller does not use the tag name. It's for documentaion only. Every message that we send using that destination name will go out as Tank/<event name> . Note that topic names **are** case sensitive! For the tank, we will not be receiving data (the tank is not a subscriber) so we don't need to set up the (slave) communications fail alarm parameters on the right. These are recommended for publishers.

The second part of the topics will be the Events under the **Communications** | **Events** tab:

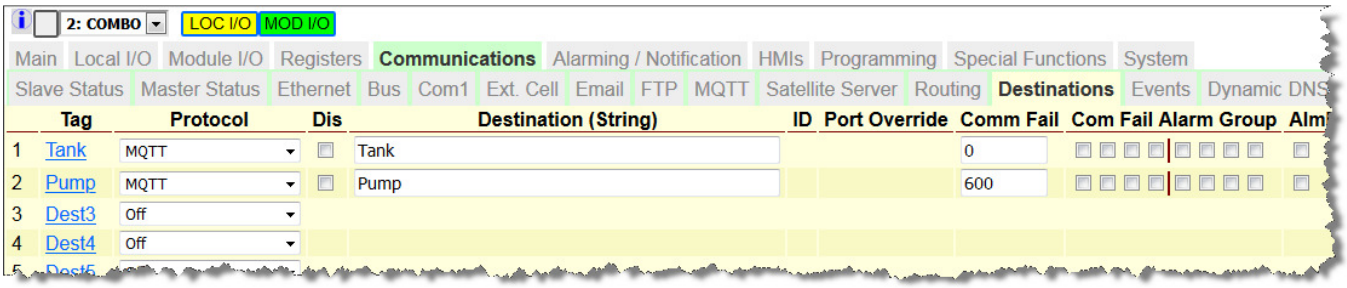


The above example will set up two publish topics from the Tank: **Tank/Level** and **Tank/Intrusion**.

For the second controller (Pump), you will need to repeat the Ethernet (Gateway and DNS Server Address) and MQTT broker setup that you used for the Tank controller.

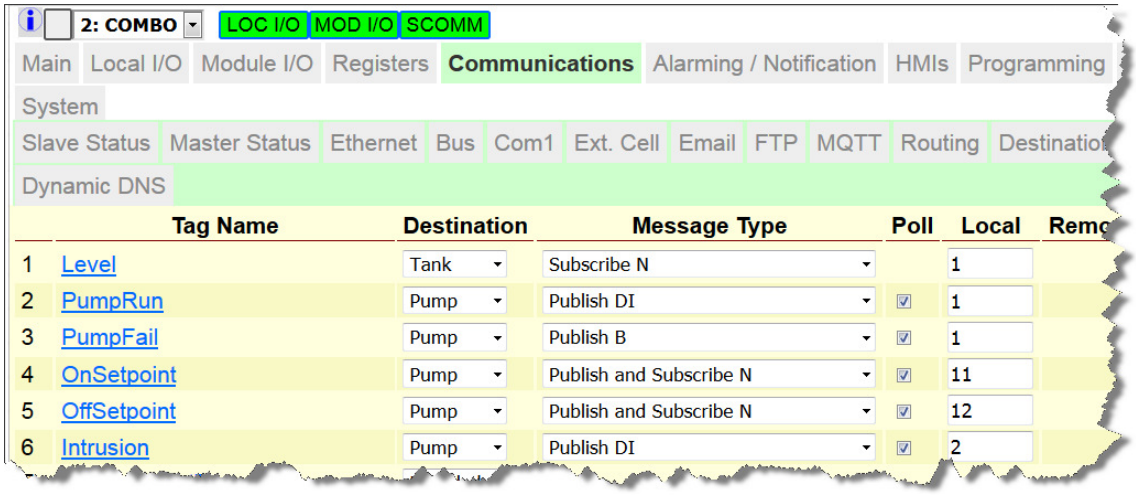
Under the Destination tab, you will need topics with two destinations:

- **Tank** so that the controller can subscribe to the level information
- **Pump** so that the controller can publish the Pump Status and setpoint information



At the pump site, we will want to alarm, shut off the pump, or use a different strategy if we stop getting level updates from the tank. In the configuration above, we've set up to alarm if we don't get a level update from the tank within 10 minutes. The details of setting up the actual broadcast of the alarm is a discussion out of the scope of this document. If needed, please contact technical support.

The second part of the topics will be the Events under the **Communications** | **Events** tab. These will be for the transactions with both the tank and the phone:



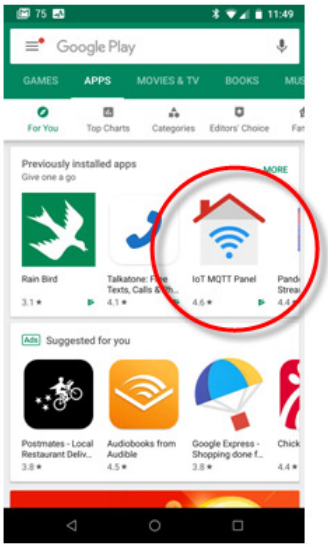
This will set up one topic that will be subscribed from the Tank (**Tank/Level)** and five topics that will be published from this site:

- **Pump/PumpRun**
- **Pump/PumpFail**
- **Pump/OnSetpoint** (publish and subscribe)
- **Pump/OffSetpoint** (publish and subscribe)

- **Pump/Intrusion**

STEP 3 - Set Up MQTT Phone or Mobile Device Application

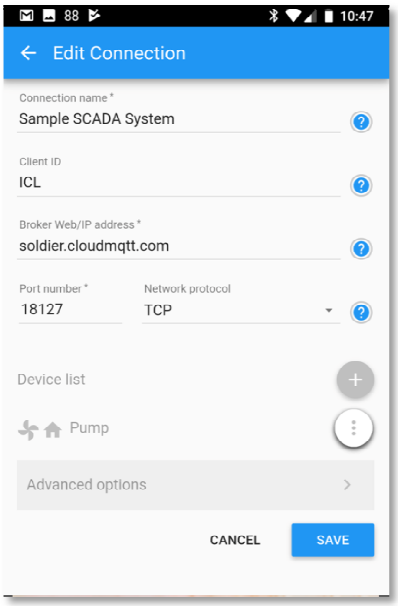
The final step is to set up a mobile device app so that users can access the published information such as tank level, pump status and alarms.



For this example, we're going to use a free application called **IoT MQTT Panel** written by Rahul Kundu and running on an Android phone. Get started by downloading and installing the app from the Google Play Store.

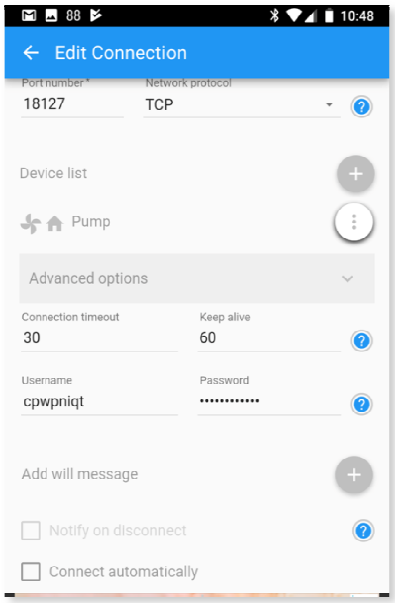
Start it.

For more information on this app and to interact with the developer, go to the developers web site at:
https://www.snrelectronicsblog.com/iot/iot-mqtt-panel-user-guide/



First you will need to configure the broker connection using the information that you received when you signed up for an account. We used an account that we signed up for from CloudMQTT for this example.

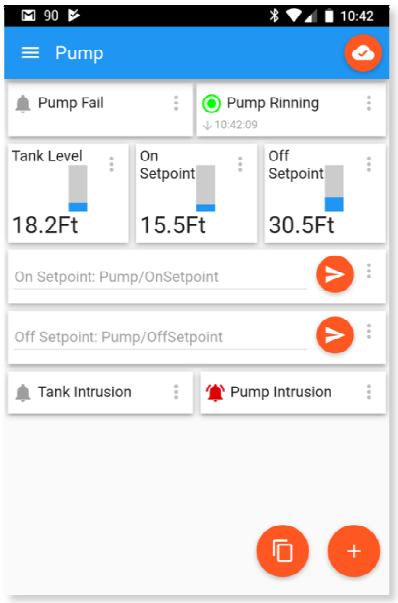
Under the "Advanced Options", you can set up the Username and Password if you are using that functionality with your broker (recommended).

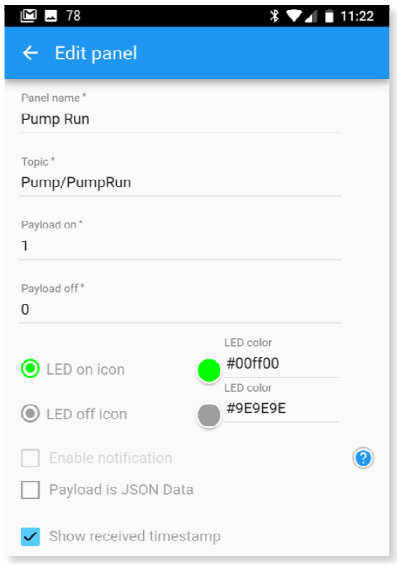


For this example, we're going to display the pump running status as well as the pump failure and intrusion alarms using the LED indicator object. Part of that configuration allows you to change the icon displayed as we have done in this example (circular indicator and alarm bells).

The completed panel is pictured on the right. Note that you are able to configure the different objects to take the full width, or fractions to stack them side-by-side.

For display of the tank level and pump on/off setpoints, we've used the vertical meter object. To change the setpoints, we've used the text object (we originally tried to use sliders, but they wouldn't allow for increments of less than 1 so we use the text object to get down to tenths of a foot).





The configuration screen for a Boolean "LED" object looks like this. You will need to configure each object individually with its own topic. Be sure that when you enter the topic name that the case exactly matches the Destination/Event naming in the controller.

To edit the icons, tap on the "LED on icon" and "LED off icon" labels.

The configuration screen for a vertical meter object looks like this. Again, you will need to configure each object individually with its own topic making sure that the spelling and case exactly matches the Destination/Event naming in the controller.

Note that you need to enter minimum and maximum values so that the app can scale to display. In this case, we selected 100 for the maximum (as if we had a 100ft. tank).