CONNECTING CBW DEVICES TO MOSQUITTO MQTT BROKER

MQTT is a publish-subscribe based messaging protocol. While it requires an MQTT broker or server to handle distributing messages between devices and clients, it can be more efficient than having each device send a message to every other device or client in the system. When a device publishes a message to the broker, the broker distributes that message to all the other devices and clients that have subscribed to that message. When a client publishes a message to turn a relay on, for example, the broker then sends that message to all the devices that have subscribed to it, turning their relay on. In this document we'll refer to CBW products as devices, and we'll refer to software that connects to a broker to publish and subscribe to topics as clients.

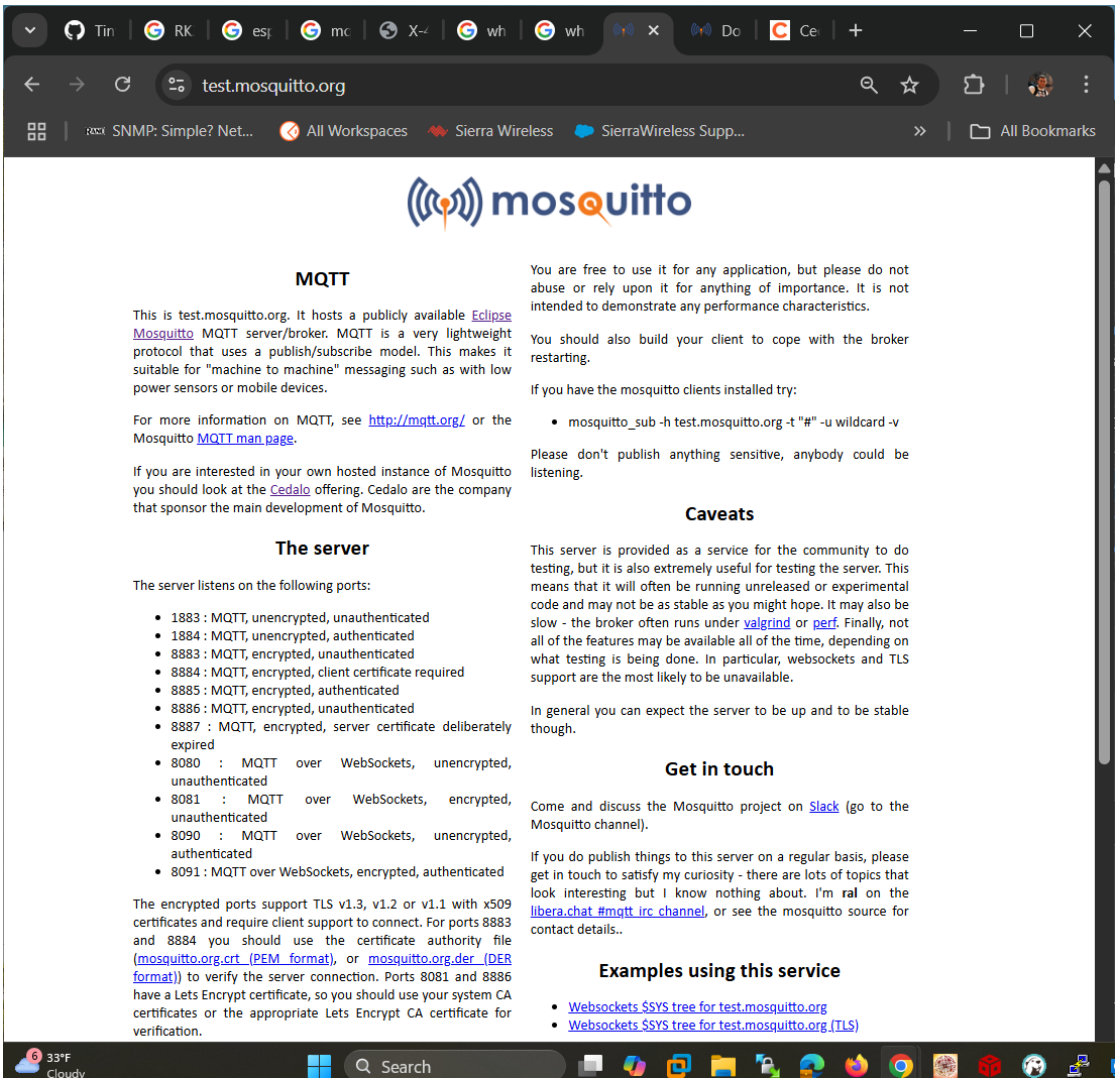
In the following document we'll walk through how to connect CBW devices to MQTT brokers. We have chosen the Eclipse Mosquitto broker for this example, but the concepts are the same for any MQTT broker. We won't talk about Sparkplug B in this document, but you will see Sparkplug B settings in the MQTT setup pages.

Eclipse Mosquitto (https://mosquitto.org/) is an open source MQTT broker. It can be downloaded and installed where an on-site MQTT broker is required. It can also be used for test purposes. Instructions on installing and configuring Eclipse Mosquitto can be found here https://mosquitto.org/documentation/ In addition, Cedalo offers both Cloud and On-premises options for hosting Eclipse Mosquitto.

This document will describe the steps to connect ControlByWeb (CBW) devices to a public Eclipse Mosquitto broker that can be used for testing purposes (https://test.mosquitto.org/). Note, this version of the service is used for testing new features and is used concurrently by a lot of people. It will also go over setting up and MQTT client to monitor the device. Features in this document require CBW device firmware version 3.13 or later.

CONNECTING TO MOSQUITTO MQTT BROKER

To use the test Mosquitto server, no account or registration is required. You can go to https://test.mosquitto.org/ for instructions on how to connect to the server. In this example, we will connect to the broker first using an unencrypted connection and then second by using an encrypted connection.



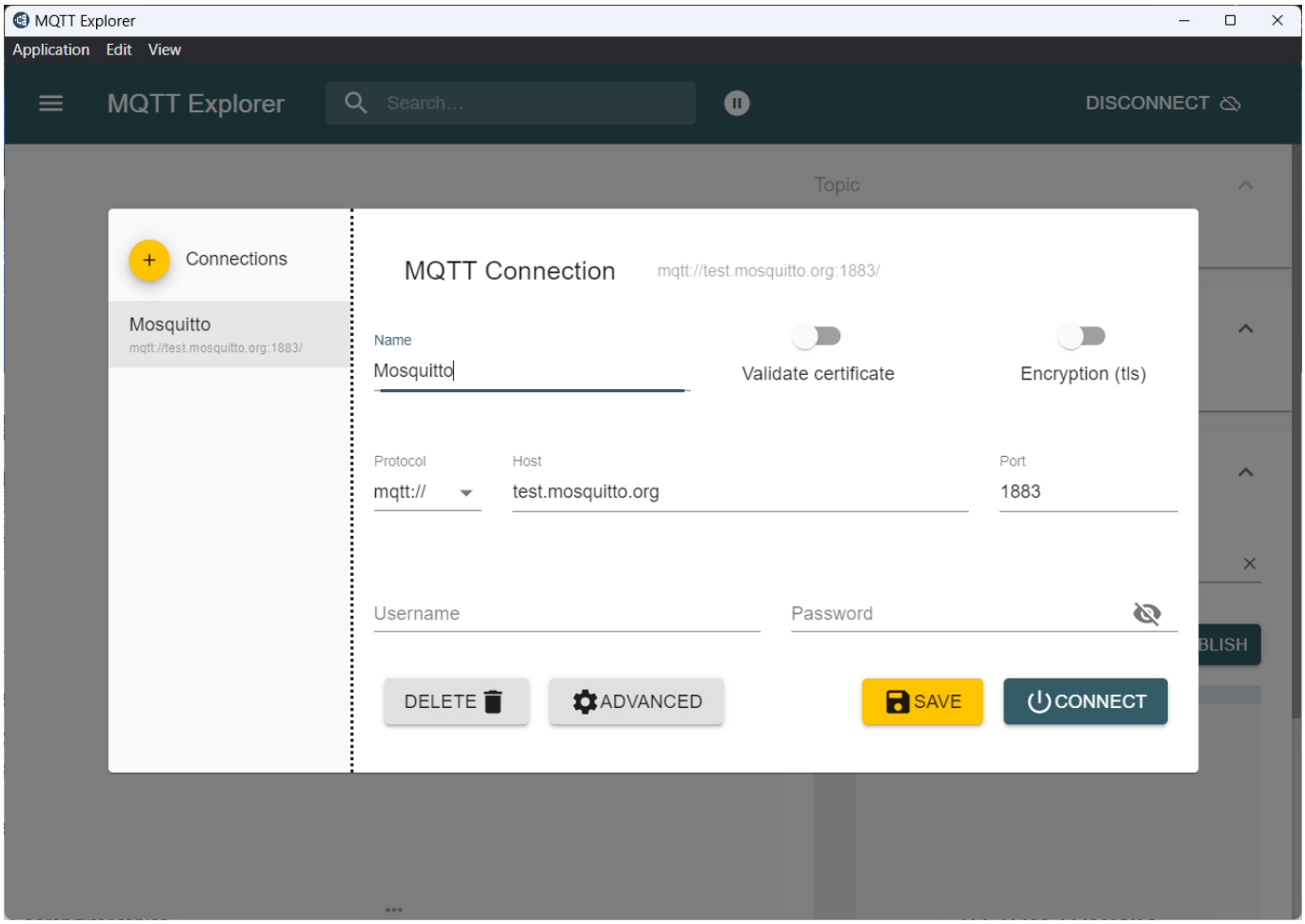
CONNECTING A CLIENT TO THE MOSQUITTO BROKER

Before we connect the device to the broker, we will connect a client to the broker. We'll use the client to monitor the publications from the device. There are online clients that can be used for testing purposes. There's also a free program called MQTT Explorer that works well for this purpose. We'll use MQTT Explorer for this example. You can download this program here https://mqtt-explorer.com/

We'll configure the MQTT Explorer program to connect to the test.mosqitto.org broker and listen to all publications from our device.

Install and open MQTT Explorer, then click on the **+ button** to add a new connection.You can name the connection Mosquitto, or whatever you'd like.

Choose mqtt:// as the protocol and use test.mosquitto.org as the host and enter 1883 for the port. Make sure to **deselect Validate certificate and Encryption(tls)**.

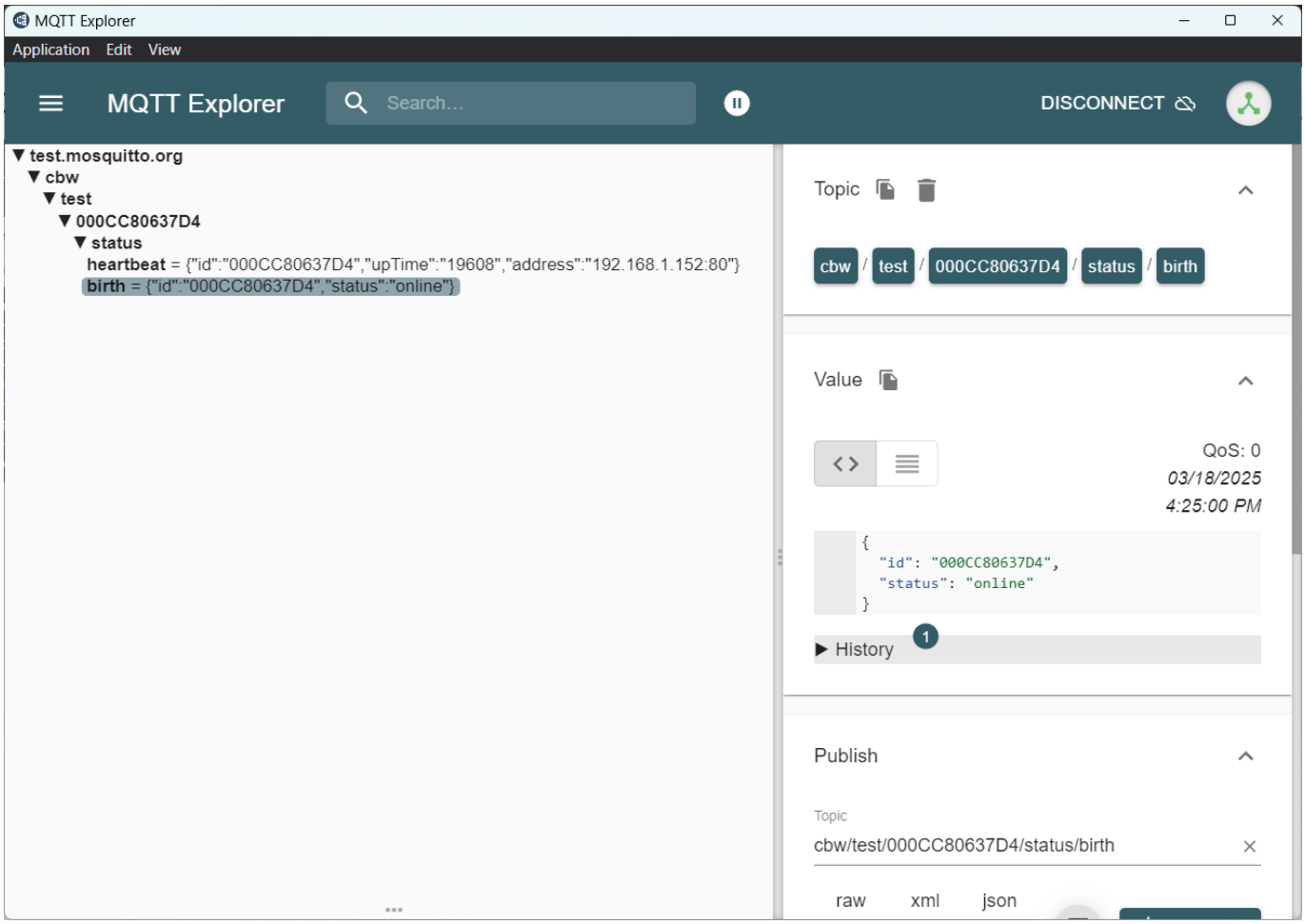


Next, click on the **advanced** button to subscribe to our test device.

Delete any other subscriptions by clicking on the **trash can** button next to them, then enter cbw/test/000CC80637D4/# qos 0 and click **+ Add**.

Replace the MAC address in the example with the MAC address of your device all uppercase. Then click **Back**.

Now you can click **Save** for connection and click **Connect** to connect to the Mosquitto broker. As we connect our device to the Mosquitto broker, the message from the device will appear in MQTT Explorer. We'll also send messages to the device from MQTT Explorer.



ADDING A DEVICE TO MOSQUITTO

Devices in Mosquitto, as with other brokers, require a unique id. Using the CBW device serial number (such as 000CC80637D4), which is the Ethernet MAC address, can be this unique identifier. Some hosted MQTT brokers require adding the device and identifying its unique identifier before the device connects. In this example, the test Mosquitto broker is open and allows connections from any device. What we will do is look at the ports that the test server listens to. Each port is used for a different type of connection to the server. In the first example we'll connect to port 1883. This port allows connection from unencrypted and unauthenticated devices. In the second example we'll connect using port 8883 which is an encrypted connection.

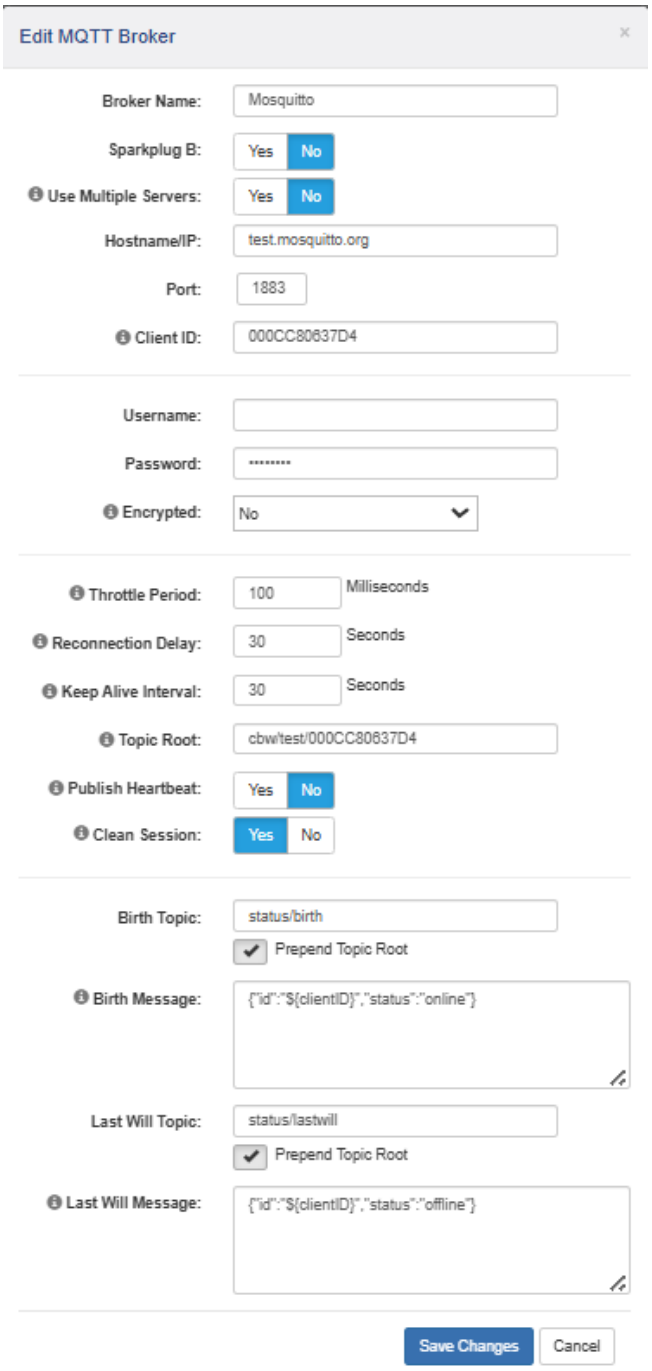
The server

The server listens on the following ports:

- 1883 : MQTT, unencrypted, unauthenticated
- 1884 : MQTT, unencrypted, authenticated
- 8883 : MQTT, encrypted, unauthenticated
- 8884 : MQTT, encrypted, client certificate required
- 8885 : MQTT, encrypted, authenticated
- 8886 : MQTT, encrypted, unauthenticated
- 8887 : MQTT, encrypted, server certificate deliberately expired
- 8080 : MQTT over WebSockets, unencrypted, unauthenticated
- 8081 : MQTT over WebSockets, encrypted, unauthenticated
- 8090 : MQTT over WebSockets, unencrypted, authenticated
- 8091 : MQTT over WebSockets, encrypted, authenticated

CONFIGURING DEVICE MQTT PUBLICATIONS

We must configure the ControlByWeb device's MQTT interface to talk to Mosquitto. Click on the **MQTT** tab and click on **Brokers**. Then, enter the MQTT device configuration information.



- The hostname and port are test.mosquitto.org**,** port **1883**.
- Enter the ClientID and leave the username blank as we aren't using authentication. Use the MAC address for the ClientID all uppercase.
- The unit will send an MQTT PING message every **Keep Alive Interval** unless there is other MQTT activity. The PING message will be sent if there is no activity during the minimum time. Note that this exchange takes 120 bytes, so you would want to increase this time to 5 minutes (300 seconds) or 10 minutes (600 seconds) for a cellular connection.
- Set the Topic Root to **cbw/test/000CC80637D4/** where the last part is the Client ID of your device. If you leave the example Client ID, you'll run into issues with other people testing their device on the broker.
- The Birth MQTT topic is sent when the device starts to register with the MQTT broker. If the device is at default settings, it will publish the birth message to **cbw/test/000CC80637D4/status/birth**. The message will be **{"id":"${clientID}", "status":"online"}**.
- The Last Will topic is sent when the device disconnects. In practice, the device sends this before sending the Birth topic to ensure the broker knows the device is connecting again. The Last Will topic is **cbw/test/000CC80637D4/status/lastwill** with the message set to **{"id":"${clientID}", "status":"offline"}**.
- Publish heartbeat will send the packet at the specified interval. Depending on the heartbeat payload you specify, this will take about 180 bytes or more and shouldn't normally be used for cellular connections. The topic will be **cbw/test/000CC80637D4/status/heartbeat** if enabled. Also, the default payload specified in 3.12 releases is not correctly formatted JSON and could cause the MQTT broker to disconnect. The default payload specified is **{"id":"${clientID}","upTime":"${upTime}","address":"${ip}:${port}}"** and should be **{"id":"${clientID}","upTime":"${upTime}","address":"${ip}:${port}"}**
- Save Changes and look at MQTT Explorer for the birth message to come through when the device connects. If the device doesn't connect check the syslog.txt file of the device to see if there are any error

messages that might indicate the reason. Double check the settings on the Broker setup page to make sure they are all correct.

Next, we need to configure the **Publish / Subscribe** sections of the MQTT sensor device configuration.

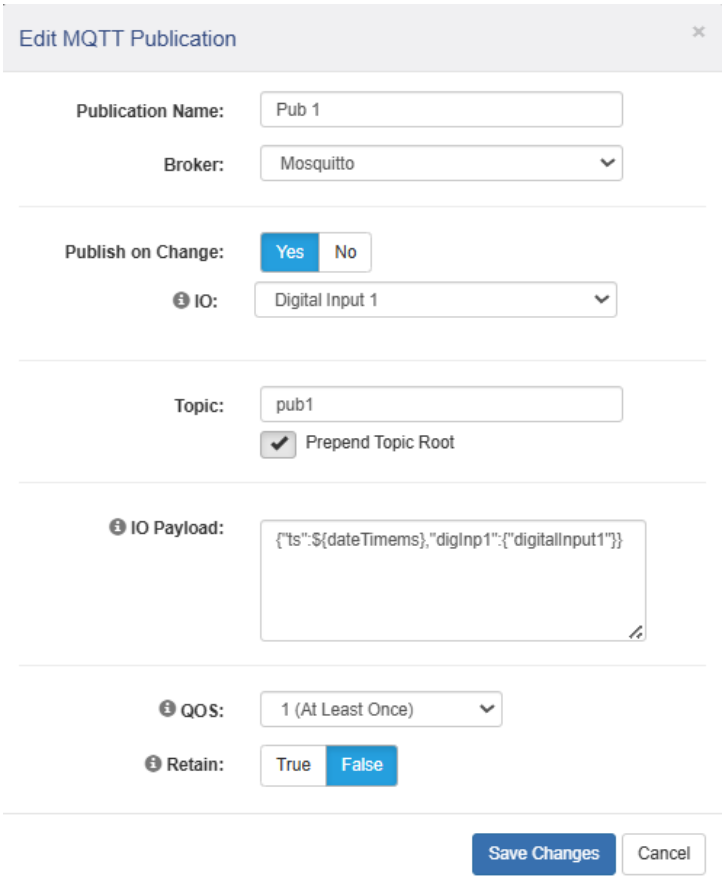
We'll format the messages as JSON packets to the **cbw/test/000CC80637D4/pub1** MQTT topic. The message is formatted as:

{"keyword1":"${token1}",{"keyword2":"${token2}",{"keyword3":"${token3}"}

The keywords are the names of the data that will appear in the message. There is no standard for names in standard MQTT, so you can pick any names that make sense to your application. It is recommended to keep these short since the MQTT Payload can be a maximum of 500 bytes per published message. Suggested keywords are **rly1, din1, ain3, reg10, vin, and id.** The tokens are used to reference sensor values in the CBW device, and the actual value is replaced in the data published to Mosquitto. The currently supported MQTT tokens are:

MAC Address	${mac}	Name (Control Page Header)	${name}	Vin	${vin}
ClientID	${clientID}	Firmware Revision	${ver}	Register 1	${register1}
Model	${model}	Serial Number	${ser}	Digital Input 1	${digitalInput1}
IP Address	${ip}	Epoch Time Stamp (sec)	${dateTime}	Analog Input 1	${analogInput1}
HTTP Port	${port}	Epoch Time Stamp (msec)	${dateTimems}	Relay 1	${relay1}
HTTPS Port	${httpsPort}	Sequence Number (autoincrementing)	${seq}		
RSSI (cellular)	${rssi}	Up Time (sec)	${upTime}		
Latitude	${latitude}				
Longitude	${longitude}				

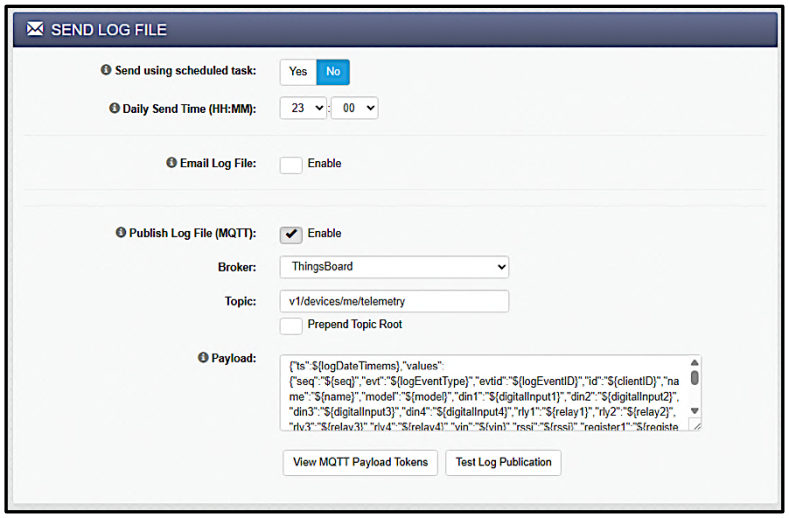
The tokens in the right column are specific to each device. They will differ based on the device's configuration and any expansion modules or additional sensors connected to it. The View MQTT Payload Tokens button will show a list tailored to each device.



Some MQTT brokers will timestamp the data when the server receives it. If the MQTT connection is down or intermittent, MQTT messages are queued and sent when the connection is viable, and the time shown will be the received time, not the queued time. To solve this, we can specify the current timestamp in the JSON payload using the keyword ts and the token **dateTimems**. In this case, the JSON payload format is:

```
{“ts”:${dateTimems},"values":
{“keyword1”:”${token1}”,
{“keyword2”:”${token2}”,
{“keyword3”:”${token3}”} }
```

Messages can be published either from the **Control/Logic Tasks** as a **Scheduled**, **Conditional,** or **Automatic Reboot Task**, based on a change of sensor device value in the Publish definition, or as **Log** entries when the data is sent to the device’s Log. For example, suppose you want to publish every time the Digital Input 1 changes state. In that case, you can select **Publish on Change** and then select that sensor device to trigger a publication. Only a single sensor device can be used as a trigger. Alternatively, you can specify the sensor device(s) in an MQTT Publication, and then in the **Control/Logic Tasks** it will appear as an action that can be triggered. You can have changes in either of two different devices trigger a message publication, along with performing two other Tasks.



ControlByWeb devices often use the Flash Log to record activity. This is set up under the **Logging & Cloud** menu. The log is 512K of Flash memory that stores activity in a circular buffer. Logging can be specified to occur based on Tasks, changes on Devices, BASIC script, SNMP, Modbus, or Open API activity. When a log event occurs, the values on the Logging page are captured and sent via Email, FTP, Cloud, or MQTT. Enable the MQTT section under SEND LOG FILE to send Log entries via MQTT. Note here that the payload can be up to 1000 characters in size. There are three additional variables you can use in your publication: *Log Epoch Time Stamp (sec or ms) / ${logDateTime},* ${logDateTimems}, *Log Event Type or Source ${logEventType} Log Entry ID ${logEventID}* .

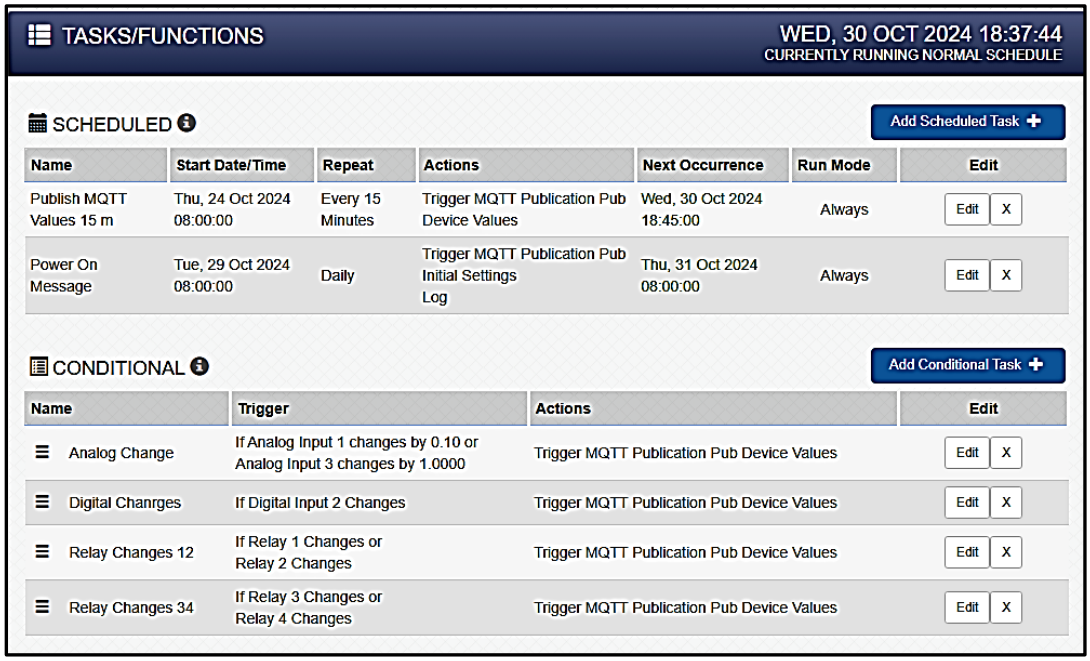
With MQTT, short messages are expected to be published with just the changed data. You could have a different MQTT Publication setup for each sensor or register on the device and configure each to send when that sensor or register changes value. Alternatively, you could configure the payload for each publication to be the same but contain multiple sensor values of interest. Then, you can configure conditional tasks to trigger the publication:

```
{"ts": ${dateTimems},"values": {"id": "${clientID}", "name":"${name}",
"model":"${model}", "ain1":"${analogInput1}",
"din2":"${digitalInput2}","ain3":"${analogInput3}", "ain4":"${analogInput4}",
"rly1": "${relay1}" , "rly2": "${relay2}", "rly3": "${relay3}", "rly4":
"${relay4}","vin":"${vin}","seq":"${seq}","rssi":"${rssi}"} }
```

You may also want to send data periodically. This will be a different MQTT publication that you can send based on a scheduled event once a day. The Initial Settings message we defined was:

```
{"ts": ${dateTimems},"values": {"id": "${clientID}", "name":"${name}",
"model":"${model}","latitude": "${latitude}", "longitude": "${longitude}"} }
```

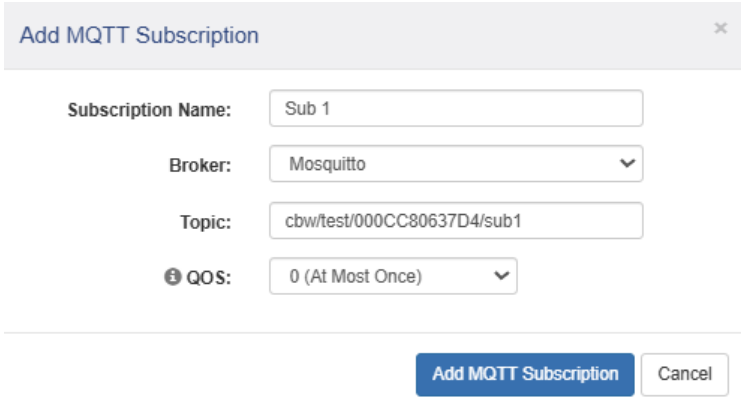
The device's Control/Logic setup page is set to publish a message when any inputs or outputs change, daily, and every 15 minutes.



DEVICE MQTT SUBSCRIPTIONS

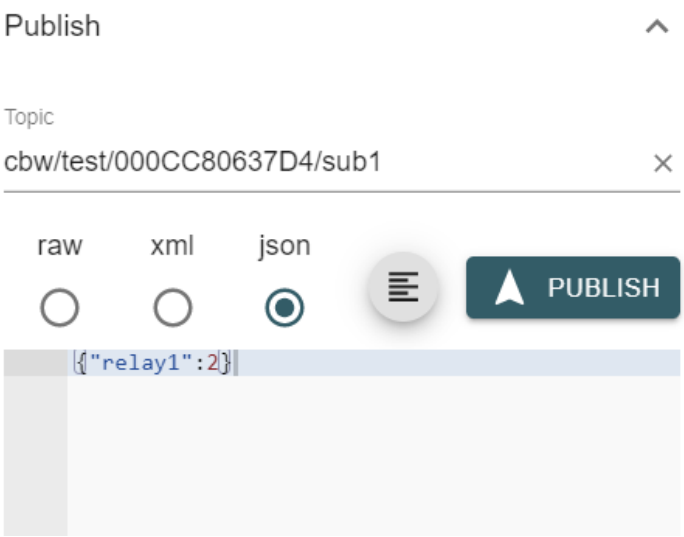
To control I/O on the device, the device must be configured to subscribe to an MQTT topic. Messages to control the device can then be published to the topic and those messages will be forwarded to the device from the broker.

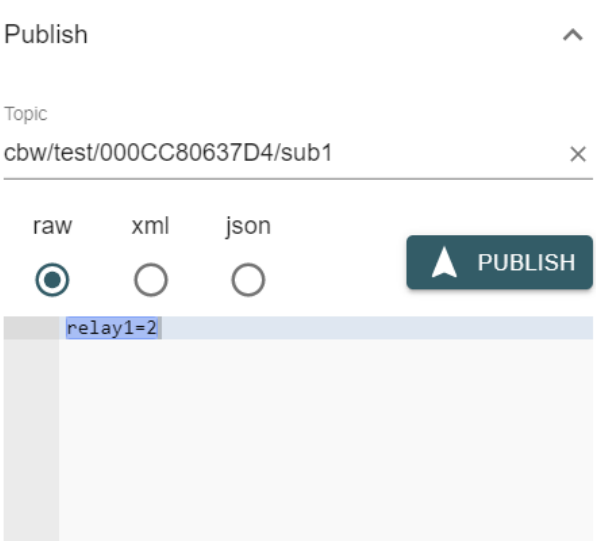
The next step is to define the MQTT subscription. Click **Add Subscription** on the CBW device under MQTT and then click **Publish / Subscribe**. Select the MQTT Broker you have defined. The topic is **cbw/test/000CC80637D4/sub1**, and QOS is set to 1 to ensure you receive at least one message.



When we publish a message like **`{"relay1":1,"register1":5.5}`** to the topic **cbw/test/000CC80637D4/sub1**, the device will now receive the message. The device can parse JSON messages in the format above, as well as messages formatted like the HTTP Get requests used with state.json. (relay1=1, relay1=1&register1=5.5) This message will change the state of the I/O on the device.

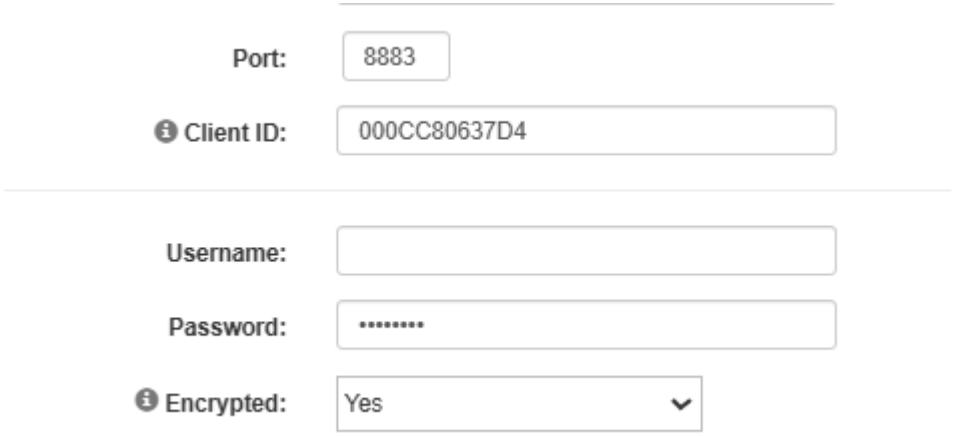
This can be tested with MQTT Explorer. In the MQTT Explorer window, on the right-hand side of the program, there is a section called Publish. Here we can enter the publication topic and message and then send it. In the screen shots below, you can see we sent a relay1=2 command that will pulse relay 1 on and off. We have also sent **`{"relay1":2}`**. Make sure to select raw or json depending on the format.




Publish
Topic
cbw/test/000CC80637D4/sub1
raw
xml
json
PUBLISH
relay1=2

ADDENDUM: CERTIFICATE X.509 AUTHENTICATION

The Mosquitto broker allows us to connect the device to it over an encrypted connection. To do this simply enable Encryption on the Broker Setup page and change the port to 8883. This will force all MQTT communication with the Mosquitto broker to be encrypted.



For additional security, we can use certificates to authenticate both the server and the device to each other. The device will then know that it's connected to the correct broker, and the broker will know that our device is authorized to connect to it. Authentication of the server is simple. We can download and install the server's certificate authority file in PEM format. This file can be downloaded from https://test.mosquitto.org/ssl/mosquitto.org.crt

To authenticate the device, we'll need to generate a client certificate for our device. To do this we'll use OpenSSL to generate a private key and a certificate signing request. We then submit the certificate signing request to test.mosquitto.org which will then generate the certificate for us. Instructions for this process can be found here https://test.mosquitto.org/ssl/ as well as the form that we will use to submit our CSR.

Before starting you'll want to install OpenSSL on your system. You can find precompiled binaries of OpenSSL here:https://wiki.openssl.org/index.php/Binaries.

The Windows version is here: https://slproweb.com/products/Win32OpenSSL.html.

On linux you can use a package manager to install OpenSSL. For example, on Ubuntu, you can run "sudo apt-get install openssl" from a terminal to install it. Once you have OpenSSL installed, generating your private key and the CSR are straightforward.

Open a terminal or command prompt and create a directory to store your test certificates.

In Windows, the command prompt will open to C:\Users\username by default.

In linux the terminal will generally open to our home directory.

The following commands will create a directory called testMQTTCerts and change to that directory:

```
mkdir testMQTTCerts
cd testMQTTCerts
```

Once inside the testMQTTCerts directory, enter the following command to generate the private key and CSR:

Generate a private key: `openssl genrsa -out client.key`

Generate the CSR: `openssl req -out client.csr -key client.key -new`

If you're using windows, you might need to put the full path of OpenSSLwhich will depend on where it was installed.

```
C:\"Program Files"\OpenSSL-Win64\bin\openssl genrsa -out client.key
C:\"Program Files"\OpenSSL-Win64\bin\openssl req -out client.csr -key client.key -new
```

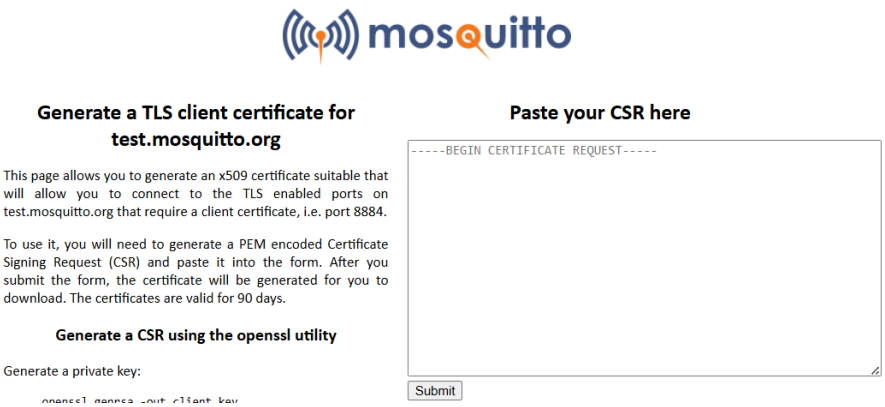
The second command will ask some questions which it will then be used for generating the CSR. Here are the questions and some answers we used to generate a CSR for this test.

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:UT
Locality Name (eg, city) []:Nibley
Organization Name (eg, company) [Internet Widgits Pty Ltd]:CBW
Organizational Unit Name (eg, section) []:.
Common Name (e.g. server FQDN or YOUR name) []:cbwMQTTTest
Email Address []:.

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

You can skip the challenge password and optional company name by pressing enter.

You should now have a private key client.key and a CSR client.csr. Client.csr is the file we will submit to test.mosquitto.org so it can generate our client certificate. Open the file client.csr and copy its contents.



Go to https://test.mosquitto.org/ssl/ and paste the CSR contents in the textarea where it says -----BEGIN CERTIFICATE REQUEST-----.

Click the **Submit** button. This will generate a client.crt file and download it to your computer. On Windows you can find the file under the Downloads directory. Copy this file over to your testMQTTCerts folder.

If you haven't already, download the mosquito.org.crt file from https://test.mosquitto.org/ssl/mosquitto.org.crt and place it in the testMQTTCerts folder as well.

You should now have the following files in a directory called testMQTTCerts

Name	Date modified	Type	Size
client.crt	3/19/2025 9:12 AM	Security Certifi...	2 KB
client.csr	3/19/2025 9:11 AM	CSR File	1 KB
client.key	3/19/2025 9:02 AM	Registration E...	2 KB
mosquitto.org.crt	3/19/2025 9:13 AM	Security Certifi...	2 KB

We will use the client.crt, client.key and mosquito.org.crt files to connect the device to the Mosquitto broker with encryptions and authentication. To do this go to the **MQTT->Brokers Edit Mqtt Broker** setup page. Change the port to **8884** and select **Yes – Certificate Authentication** for the **Encrypted** option. Three buttons will appear. Each allows uploading one of the files above into the device for use during the connection process.

- Click on Upload/View Client Certificate to upload the client.crt file.
- Click on Upload/View Client Key to upload the client.key file.
- Click on Upload/View Client CA to upload the mosquito.org.crt file.

Once all the files have been uploaded, submit the setup page. The device will disconnect from the broker and reconnect in 30 seconds using the newly uploaded certificates and port number. Everything should work as it did before, only now the connection is both encrypted and authenticated. The Mosquitto broker will only allow devices to connect to it on port 8884 if they have the correct certificates. You can test this by using MQTT Explorer to send a pulse command to the device.