**Send Data to Cayenne Dashboard Using Node-Red**

Setup:  
Cayenne:  
Log in to your dashboard and click Add new…>Device/Widget  
Click the Bring Your Thing button - keep this window open for later  
That’s it for now!

Raspberry Pi server:  
Install a local MQTT server using apt-get:

$sudo apt-get install mosquitto

Set up Node-RED to start on boot using the command below:

$sudo systemctl enable nodered.service

Now you can either start Node-RED by rebooting or using “sudo service nodered start”

The Node-RED web interface can be accessed by going to your Pi’s IP address and appending:1880 to the end (ex. [http://192.168.0.10:1880 16](http://192.168.0.10:1880/))

For now, we will create a flow to receive an MQTT command from Cayenne and another to send a value to Cayenne using MQTT. The very basic flow below can be imported into Node-RED by clicking Settings (three lines in the top right)>Import>Clipboard then click where you want to place it on the flow tab.

After placing the flow edit the Set Global Variables node with your MQTT Username/Password listed on your Cayenne dashboard for the device we added earlier.

Edit the Cayenne Input node by double-clicking it. Change the topic to fit your settings, the channel should be 1. Below is the breakdown of what the topic means:

v1/username/things/clientID/cmd/channel

v1 - This is the API version which for now will always be v1

username - Change this to the MQTT username listed on your Cayenne dashboard for the device we added earlier

things - This can stay things - not exactly sure what this is

clientID - Change this to the Client ID listed on your Cayenne dashboard for the device we added earlier

cmd - This is the type of message. cmd is a command message from the Cayenne server

channel - This is where the data ends up on the dashboard. Each widget must have a unique channel

In the end your Topic string should look like this:

v1/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX/things/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX/cmd/X

After the topic is set click on the pencil next to the server. On this page paste in your Client ID in the Client ID field then click the Security tab and paste in your username and password from the Cayenne dashboard.

Next, edit the Cayenne Output Channel 1 node by double-clicking it. Change the topic to fit your settings. You can use the same topic from the input node, but this time we want to change the message type to digital to tell the Cayenne server we received the command, so it updates the button on the dashboard. The channel should be 1. Your Topic string should look like this:

v1/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX/things/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX/data/X

Finally, edit the Cayenne Output Channel 2 node by double-clicking it. Make the same changes as the Cayenne Output Channel 1 node but notice the second to the last parameter is data instead of digital this time. Sending the type data tells the server we are sending it a value to update a widget. The channel should be set to 2.

v1/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX/things/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX/data/X

After the 2 MQTT nodes are configured you can click Deploy to save the changes. At this point, the MQTT settings page should go away and take you to a blank dashboard. You can test that communication to the server is working by clicking the blue button to the left of the Send Temp Value of the 20c inject node. You should see a temporary widget appear on the blank dashboard. From here if you want to make it permanent just click the + in the upper right corner.

To test that values are being received from the server, click Add new…>Device/Widget>Custom Widgets>Button. Enter the following settings:

Name - Enter a name. The button is a good choice

Device - Select your MQTT device

Channel - Enter a 1

Icon - Whatever your preference. Light is usually the one I select

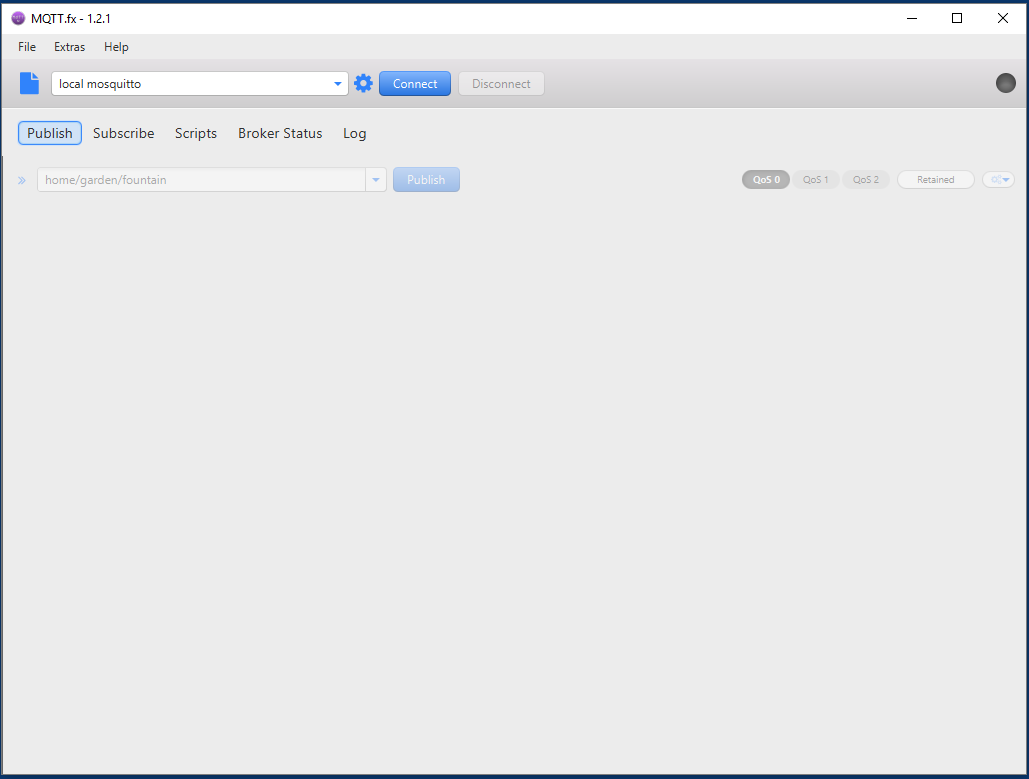
Click Add Widget then click on the button that shows up on the dashboard. Go back to Node-RED and click on the Debug tab in the top right. You should see two debug messages:

XXXXXXXXXXXXXXX,1 which means your message was received. 1 means the button was turned on and 0 means it was turned off. The first part of the message is a random sequence.  
1 - 1 means the button was turned on and 0 means it was turned off. This is the value sent back to the server to verify the command was successful and update the button on the dashboard.

**Publish data from MQTT.FX**

**Install MQTT.fx**

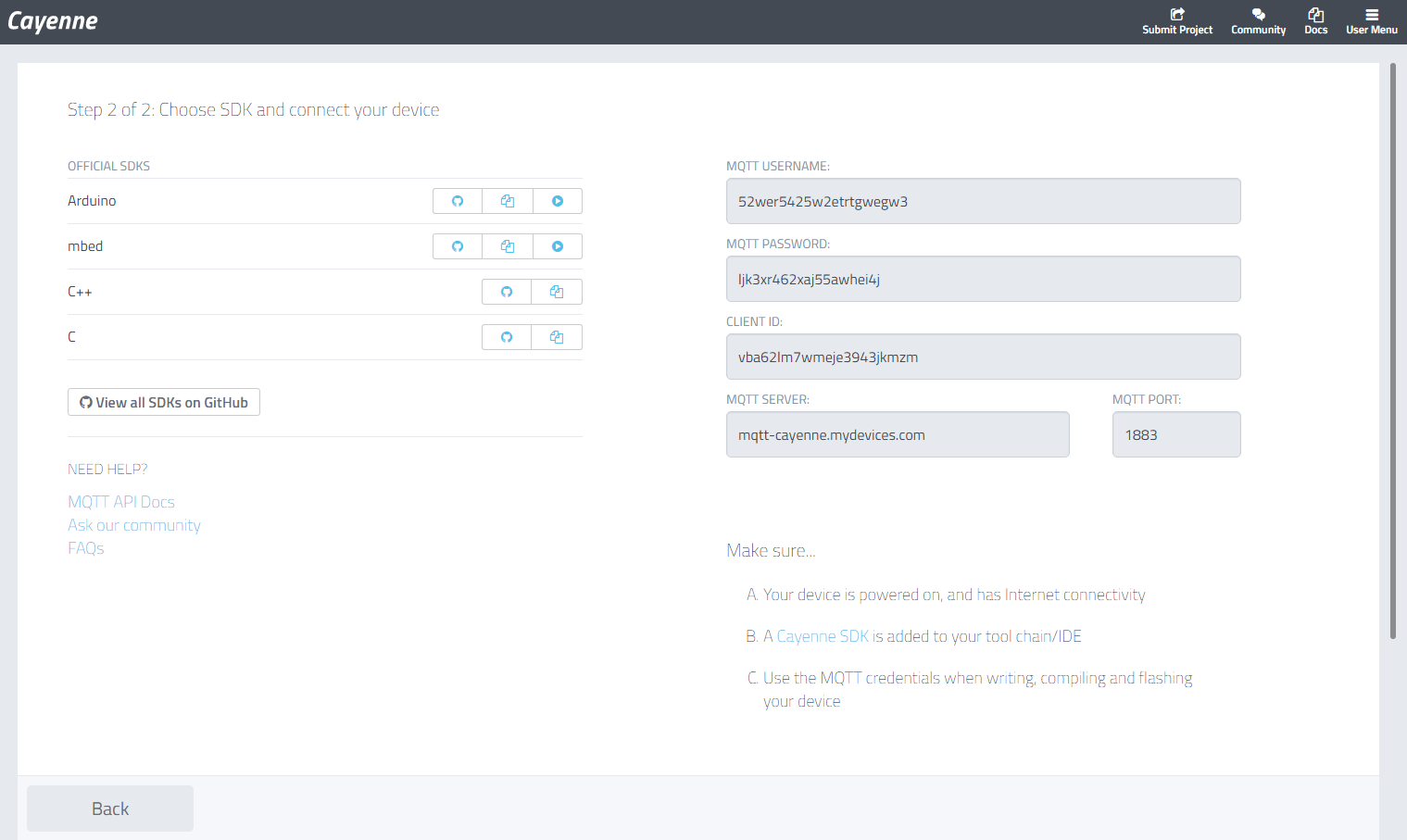
To begin using MQTT.fx, we must download and install it. Doing so is easy and straightforward. Simply visit the [MQTT.fx download page](http://mqttfx.jensd.de/index.php/download). Download and install the correct version for the Operating System that you are using. Once installed, launch the MQTT.fx client to get started.

[](https://camo.githubusercontent.com/5f6dbd2d8a7e6176c7a7558da3cb41e8bba72a6b/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4d5154542e66782d312d696e697469616c2d6c61756e63682e706e67)

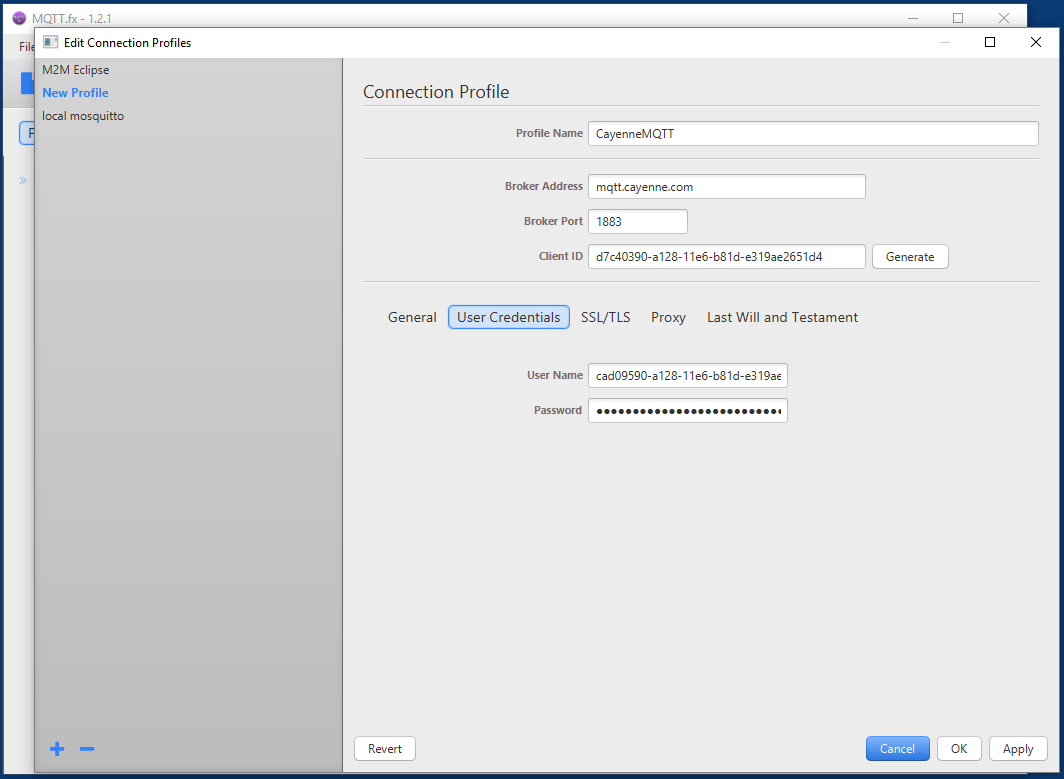
**Add Connection Profile**

To connect to the Cayenne Cloud, you will need to set up a **Connection Profile** in MQTT.fx. To do so, click on the **cogwheel** icon or select **Extras** > **Edit Connection Profiles** from the menu. The *Edit Connection Profiles* screen appears. From this screen, you can enter all the required information needed to complete a profile for connecting to Cayenne.

*TIP: All the required information we need can be found on the Cayenne dashboard’s 'Choose SDK and connect your device' screen. Refer to this screen and the information below to complete creating your connection profile to Cayenne.*

[](https://camo.githubusercontent.com/a21ebf4f2a7442779ad417ea6833f41236ae9336/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f537465702d322d42594f542d43686f6f73652d53444b2e706e67)

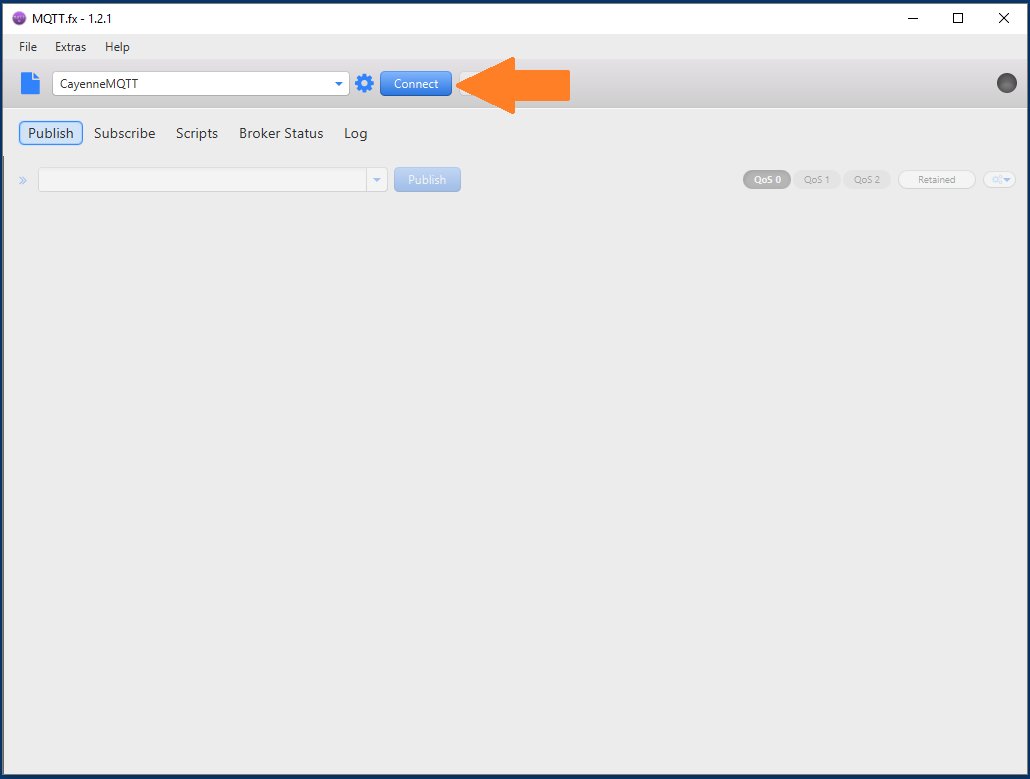
1. Give the profile a name, such as Cayenne MQTT, in the **Profile Name** field.
2. Copy & paste the **MQTT Server** URL from the dashboard into the **Broker Address** field.
3. Leave the **Broker Port** at its default of 1883.
4. Copy & paste the **CLIENT ID** field from the dashboard into the **Client ID** field.
5. On the *User Credentials* tab, copy & paste **MQTT Username** from the dashboard into the **Username** field.
6. Also on the *User Credentials* tab, copy & paste **MQTT Password** from the dashboard into the **Password** field.
7. The default values for fields in the other tabs are OK to leave as-is. Click the **OK** button to save our profile.

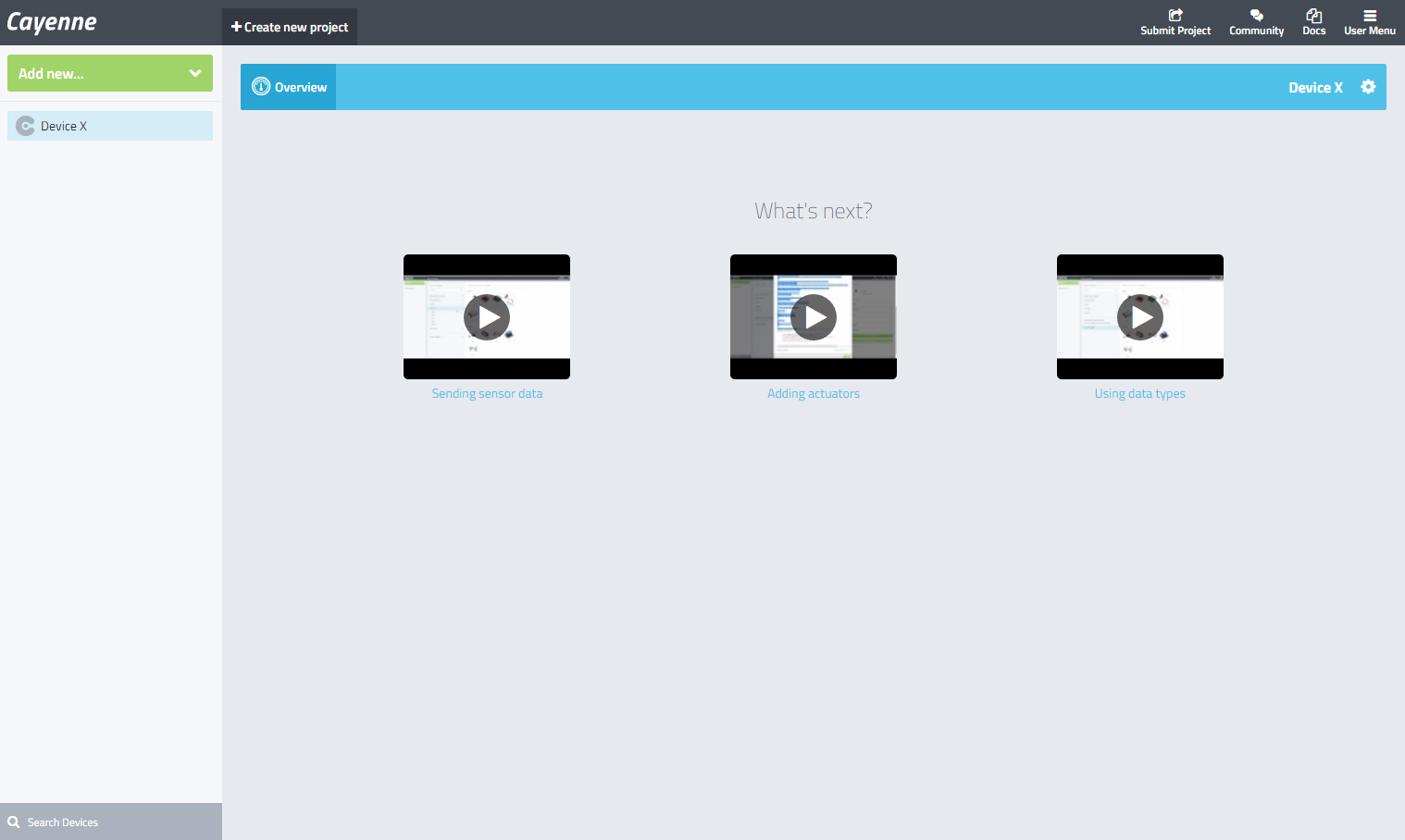
[](https://camo.githubusercontent.com/8540f601f7ed9d93f9146d417e312a5e3fafa5c8/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4d5154542e66782d322d436f6e6e656374696f6e2d50726f66696c652e706e67)

We can now make use of our profile to connect to the Cayenne MQTT server and test out publishing and subscribing to data.

**Connect board to Cayenne**

Now that our profile is setup, we can connect to Cayenne. To do so, click on the **Connect** button. This will establish a connection to Cayenne, and it will also mimic our board coming online. Switching back to examining our browser, you'll find that the *Choose SDK and connect your device* screen will disappear and the default dashboard for our device will appear.

[](https://camo.githubusercontent.com/d5a7422a8cca751f9fddcbd59dbf8e4a1883d567/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4d5154542e66782d332d70726f66696c652d637265617465642e706e67)

[](https://camo.githubusercontent.com/ea0ed6d9f1d3a06dc200e77992617728fd419f05/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4d5154542e66782d352d6465766963652d64617368626f6172642e706e67)

**Congrats! You are now connected to the Cayenne cloud using MQTT.**

**Send sensor data to Cayenne**

Now that we have a connection to the Cayenne MQTT server, let's put our connection to use by simulating sending sensor data to our dashboard. For our example, we will simulate having a Temperature sensor connected by publishing a sensor reading up to our dashboard. To do this, switch to MQTT.fx and make sure the *Publish* tab is selected. From here we can enter the MQTT details to publish sample sensor data to Cayenne.

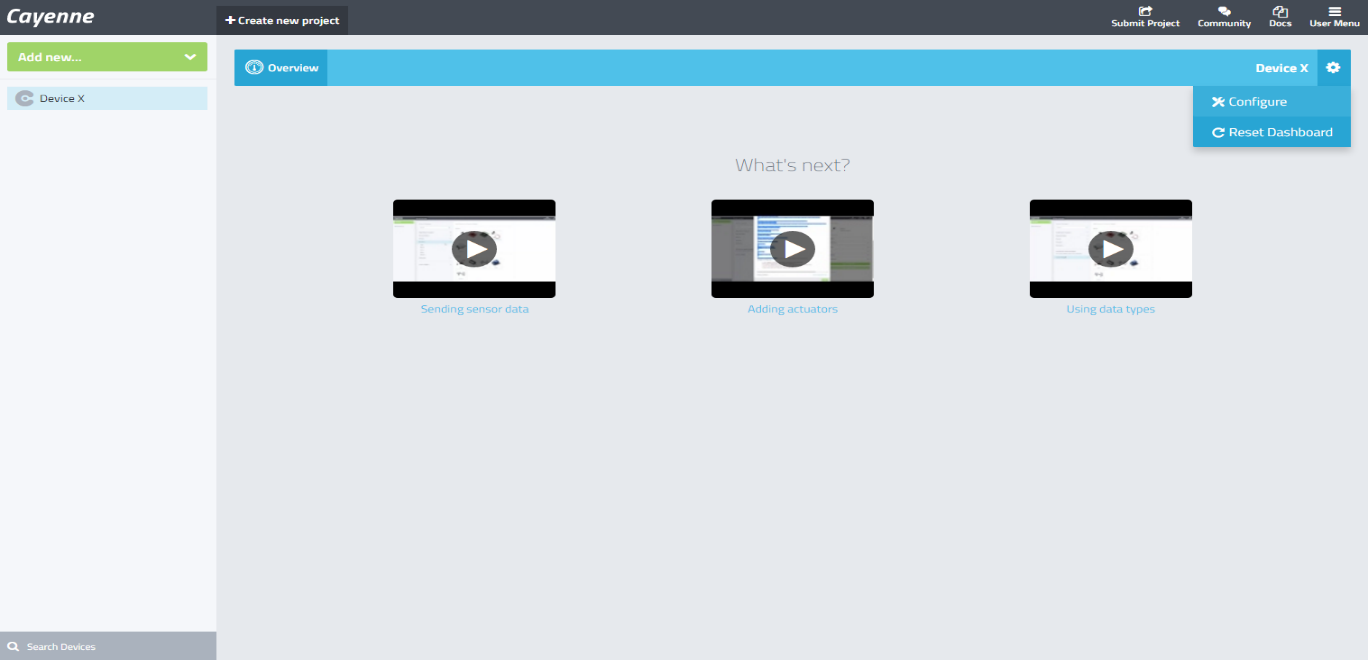
To publish sensor data to Cayenne using MQTT, we refer to the [MQTT Messaging Topics - Send sensor data](https://github.com/myDevicesIoT/cayenne-docs/blob/master/docs/MQTTAPIS.md#cayenne-mqtt-api-mqtt-messaging-topics-send-sensor-data) section of the docs. There, we find the details on which MQTT call to make. According to the docs, sending sensor data expects:

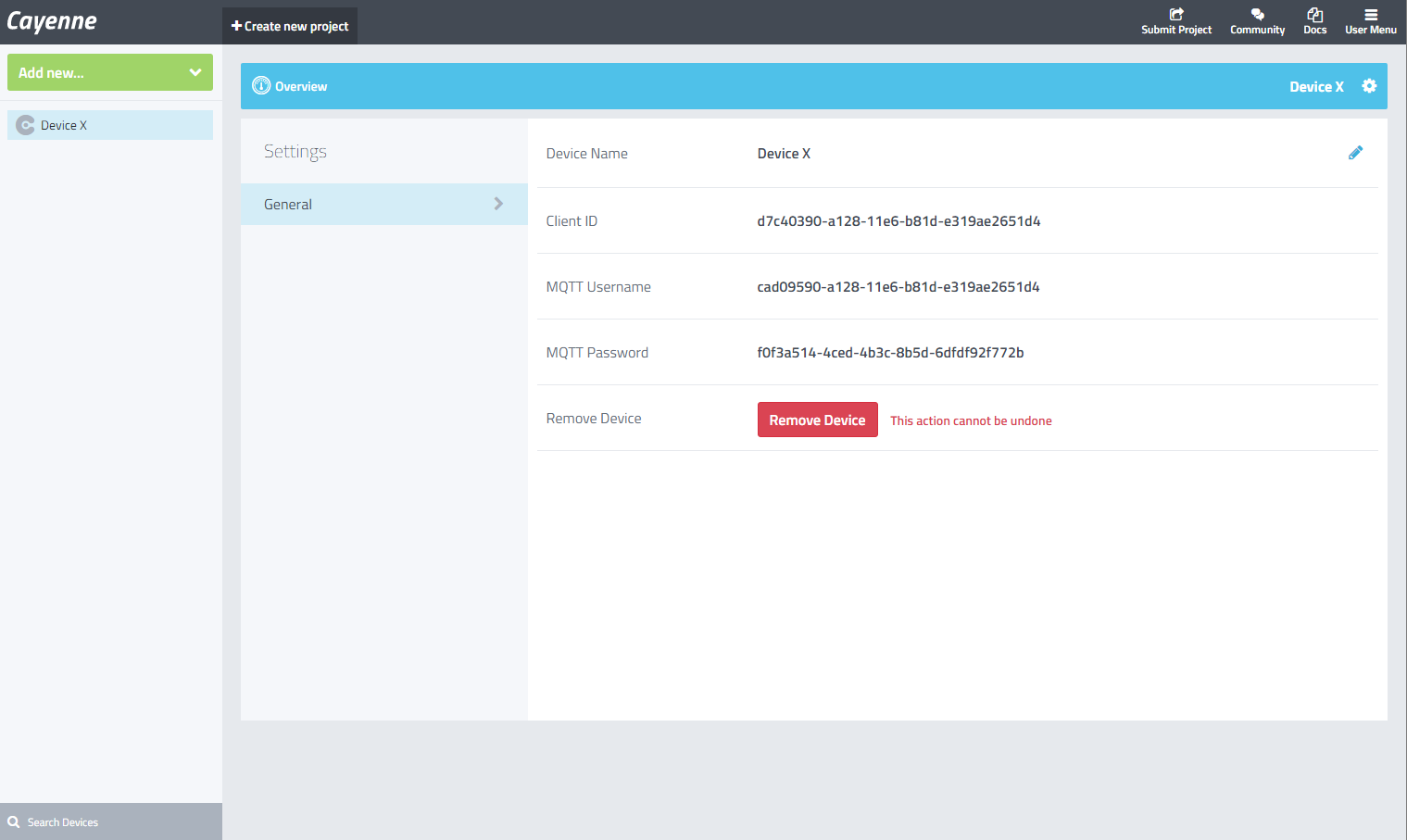
v1/username/things/clientID/data/channel

We then substitute the values for our account, board and sensor.

* Replace **username** with the **MQTT Username** for your account.
* Replace **ClientID** with the **Client ID** for your board.
* Replace **channel** with the appropriate channel that this sensor is connected to. For this example, we'll assume that we have a temperature sensor connected and using Channel 0.

*TIP: If you ever need to refer to the MQTT Credentials needed for operations such as this, you can refer to the Configuration screen for your board. To do so, select the****cogwheel****menu for your board and then the****Configure****option. In the configuration screen that appears, you'll find the values that you need.*

[](https://camo.githubusercontent.com/cce290a41201d5ef50a3b10efadab9f4d2dc9c09/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4150492d436f6e6669677572652d6d656e752e706e67)

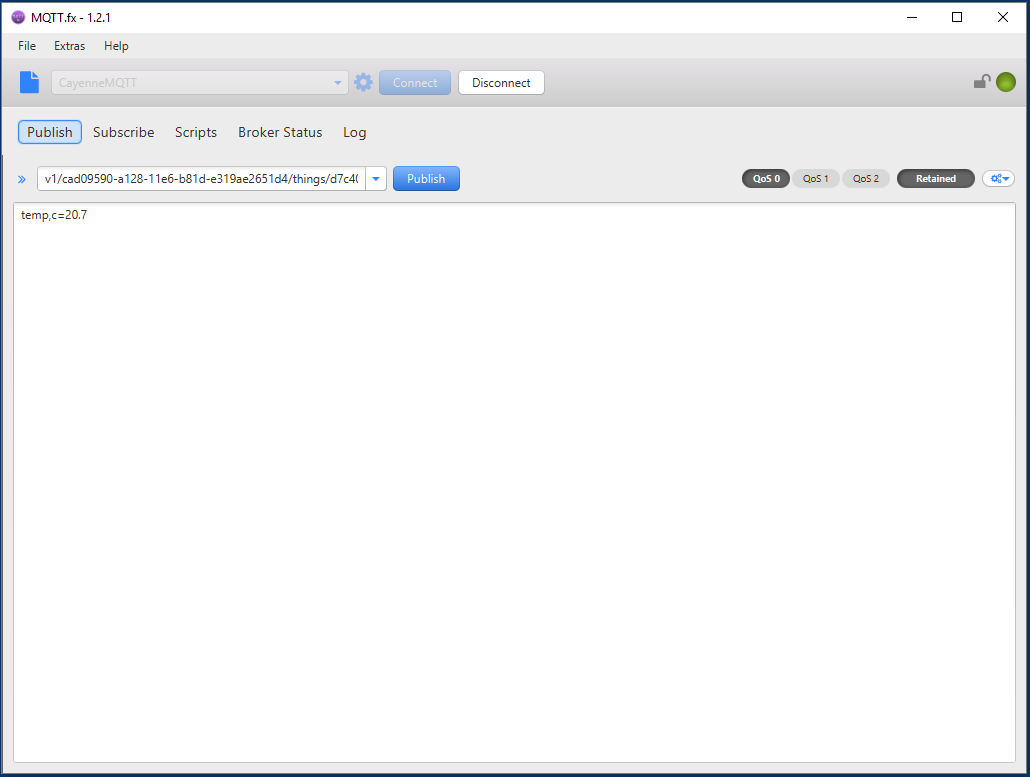
[](https://camo.githubusercontent.com/90c3a350d797101faa620ba7076a8e32be7881b1/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4150492d436f6e6669677572652d73637265656e2e706e67)

Next, we need to send some actual data. According to the documentation, we need to send this in the following form:

type,unit=value

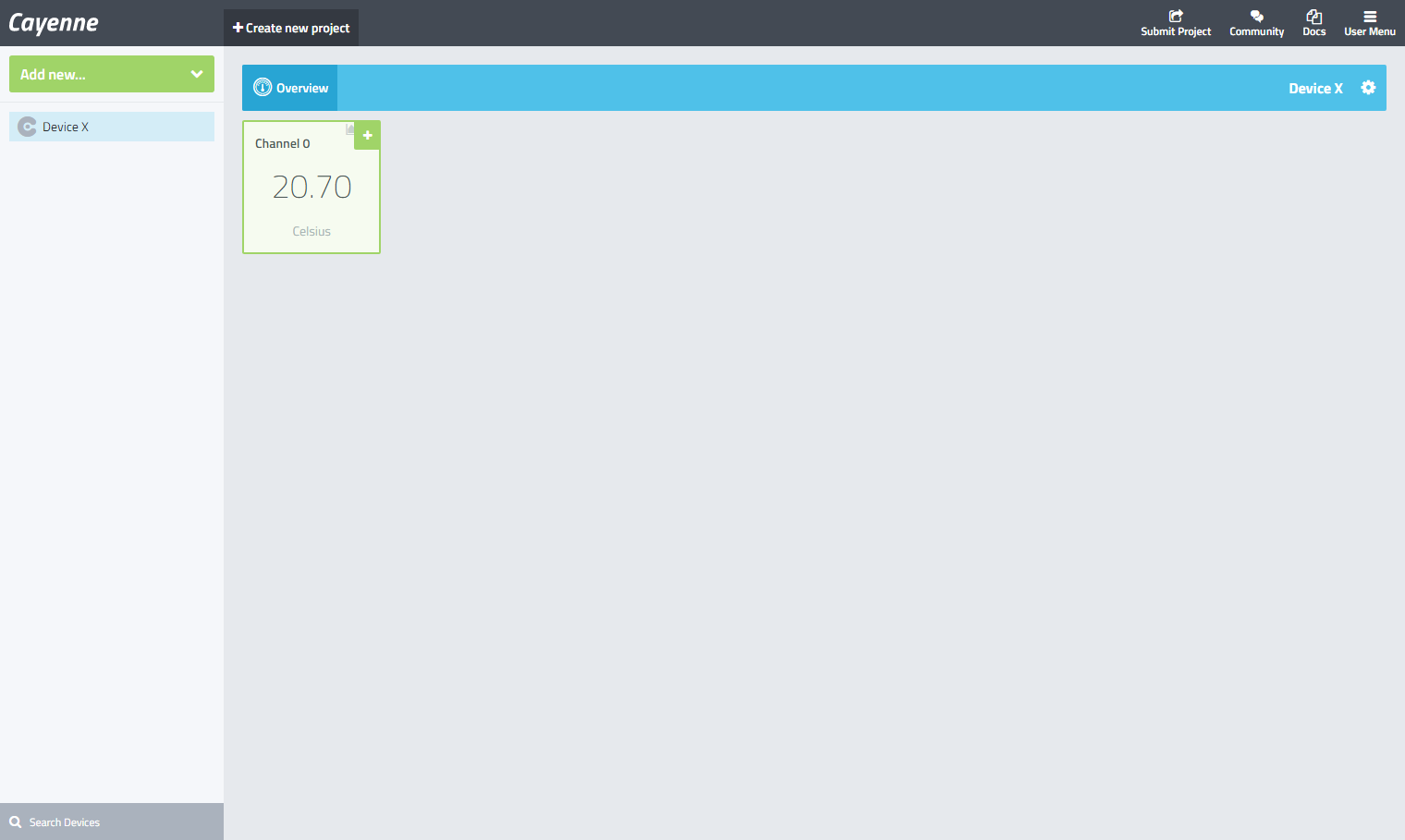
The Cayenne documentation provides a complete list of [Supported Data Types](https://github.com/myDevicesIoT/cayenne-docs/blob/master/docs/MQTTAPIS.md#cayenne-mqtt-api-supported-data-types) that can be referred to when you need to know what details to put here. We want to send Temperature data, so we find **Temperature** in the supported data types section. For our example, let's assume that we want to send our temperature to Celsius. We'll use a sample sensor value, say 20.7 Celsius to be sent to Cayenne. With this in mind, and after examining the chart for the Temperature data type, we determine the following values should be used:

* **Type (Temperature):** temp
* **Unit (Celsius):** c
* **Value:** 20.7

[](https://camo.githubusercontent.com/a3bf16d5e8af3bf9eb398b42e96def2fe0ef03a1/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4d5154542e66782d382d73656e642d74656d702d73656e736f722d646174612e706e67)

*TIP: If you want the Cayenne Cloud to retain the last published value for a sensor, be sure to click on the****Retained****button in MQTT.fx. Without this, you may find that your widget disappears when you refresh the page.*

After entering in the MQTT details for publishing our sample sensor data, click on the **Publish** button to send the data to Cayenne. Cayenne will receive this data and automatically add a widget for it! Cayenne will do this automatically for any new MQTT data that you send it. Widgets created in this way are temporary by default, giving you an easy way to test sending new data to Cayenne. If you want to keep this widget permanently, simply click on the widget tile and it will become a permanent widget in your dashboard.

[](https://camo.githubusercontent.com/2c29b34b34e761f455ecf8272cfd54b258829062/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4d5154542e66782d392d74656d702d7769646765742d637265617465642e706e67)

**Congrats! You are now sending data to the Cayenne Cloud!**

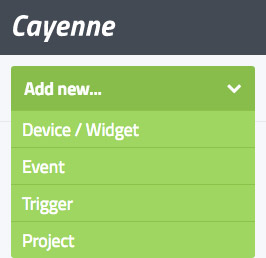
**Control a Light actuator**

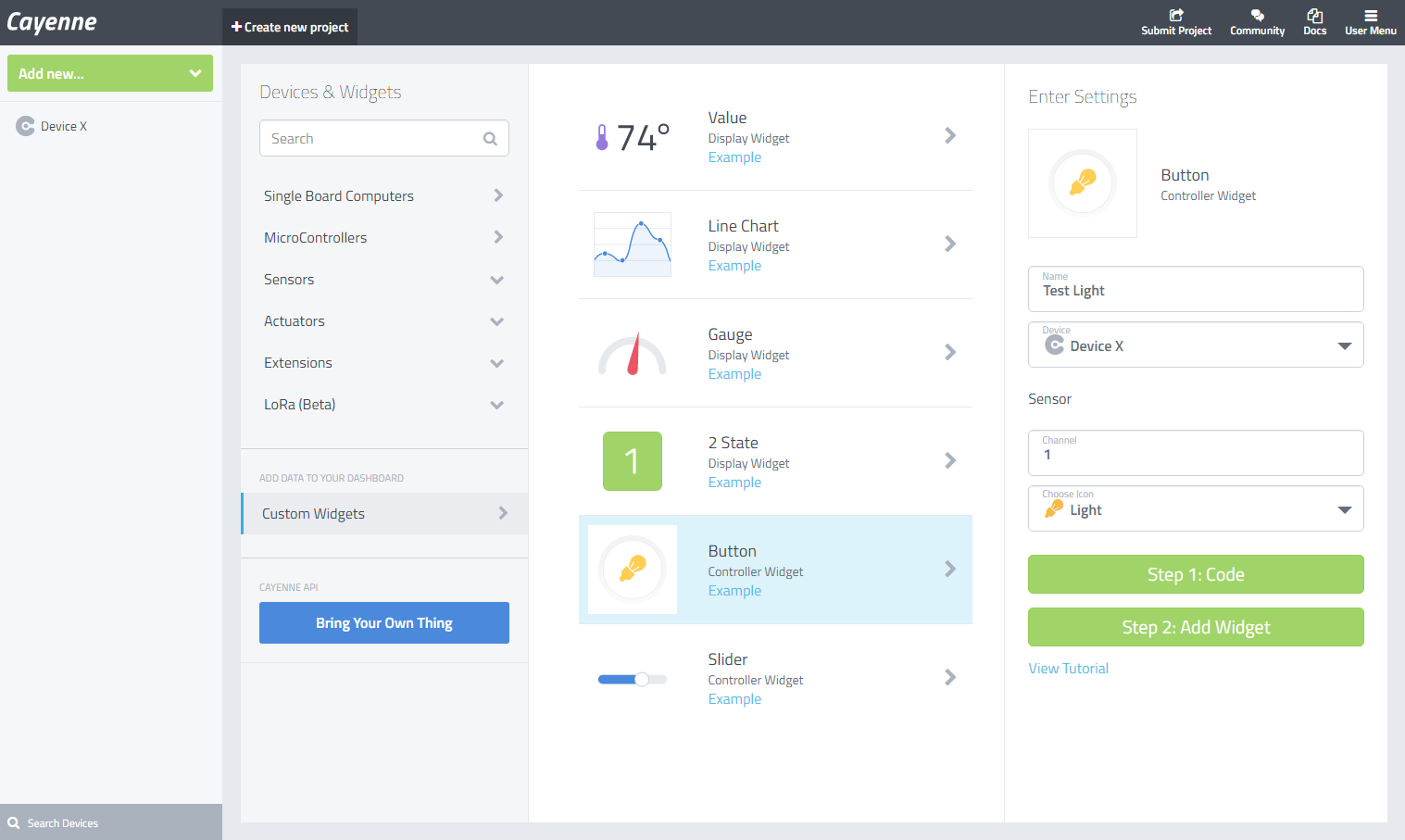
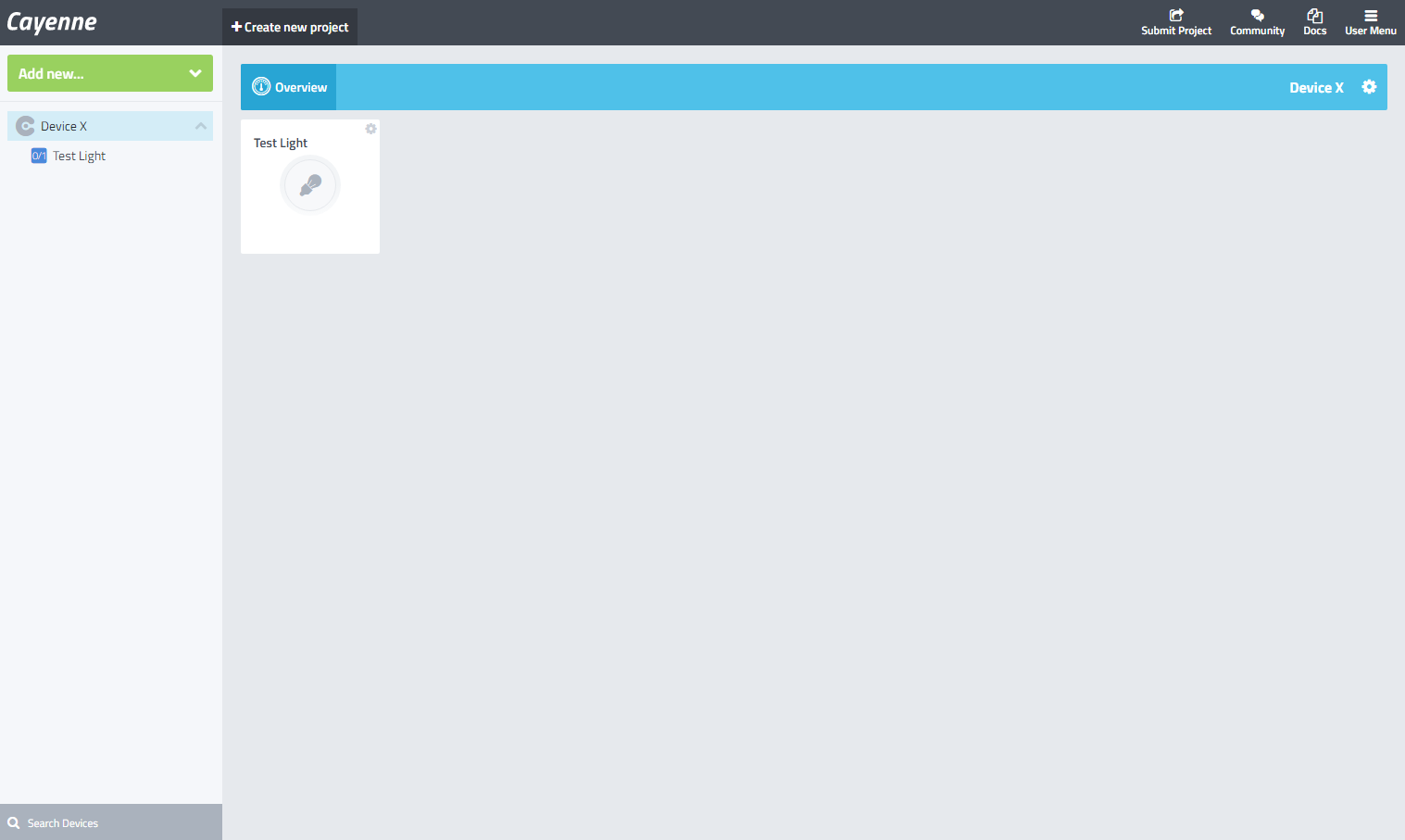
Now that we have our connection to Cayenne and successfully sent test data to our dashboard, let’s look at how easy it is to add an actuator. When users change the state of actuators using the dashboard widgets, Cayenne publishes *COMMAND* messages. By subscribing to these messages, you will be informed when our actuator's state was changed.

For this example, we will set up a [Button widget](https://github.com/myDevicesIoT/cayenne-docs/blob/master/docs/MQTTAPIS.md#custom-widgets-button) on our dashboard and use it to send actuator commands to an imaginary actuator connected to our board. In doing so, we will also cover the steps that Cayenne expects well-behaved clients to perform when interacting with actuators via MQTT.

**Add dashboard widget**

Let's start by adding a Button widget on the dashboard. From the Cayenne dashboard, click **Add New** > **Device / Widget**.

[](https://camo.githubusercontent.com/f987545f3d70a6b61fa7814e0e37c530a91ab59c/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4164644e65772e6a7067)

1. Choose **Custom Widget** > **Button**.
2. Give your actuator a name, for example enter “Test Light” into the **Name** field.
3. We’ll be adding this actuator to our custom device, so make sure your device is selected in the **Device** field.
4. Select 1 from the **Channel** field.
5. We can specify an **Icon** for our actuator. Say we’re using it to control a Light, so let’s select a Light icon here.
6. [](https://camo.githubusercontent.com/62e61e372bdb8d3815ba2cbc8b55d55f4d5e9216/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4d5154542e66782d31302d6163747561746f722d73657474696e67732e706e67)Click the **Step 2: Add Actuator** button. The light widget will then be added to our dashboard.  
   [](https://camo.githubusercontent.com/7d43bd6aed00294e43dc9bf815ae1caf2b776430/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4150492d6163747561746f722d6f66662e706e67)

**Testing the actuator**

To test out our actuator, it first helps to understand a bit about how Cayenne expects a well-behaved client to react to actuator commands. Cayenne expects a client to:

1. Subscribe to *COMMAND* messages from Cayenne.
2. When a new *COMMAND* arrives, handle changing the status of the actuator connected to the board.

**Note:** Cayenne will inform the listener which Channel was effected as well as what the new state is.

1. After changing (or attempting) to change the actuator state, inform Cayenne what the current status now is.

**Note:** It's very important to inform Cayenne what the correct state is. This ensures that the dashboard properly reflects the correct current state of the device.

1. Inform Cayenne whether the event was handled OK or had an Error.

**Note:** If there was an error, Cayenne will handle showing the error to the user so that they're aware of it.

With that background in place, let's go over-performing each of these steps using MQTT.fx.

**Receive Actuator command**

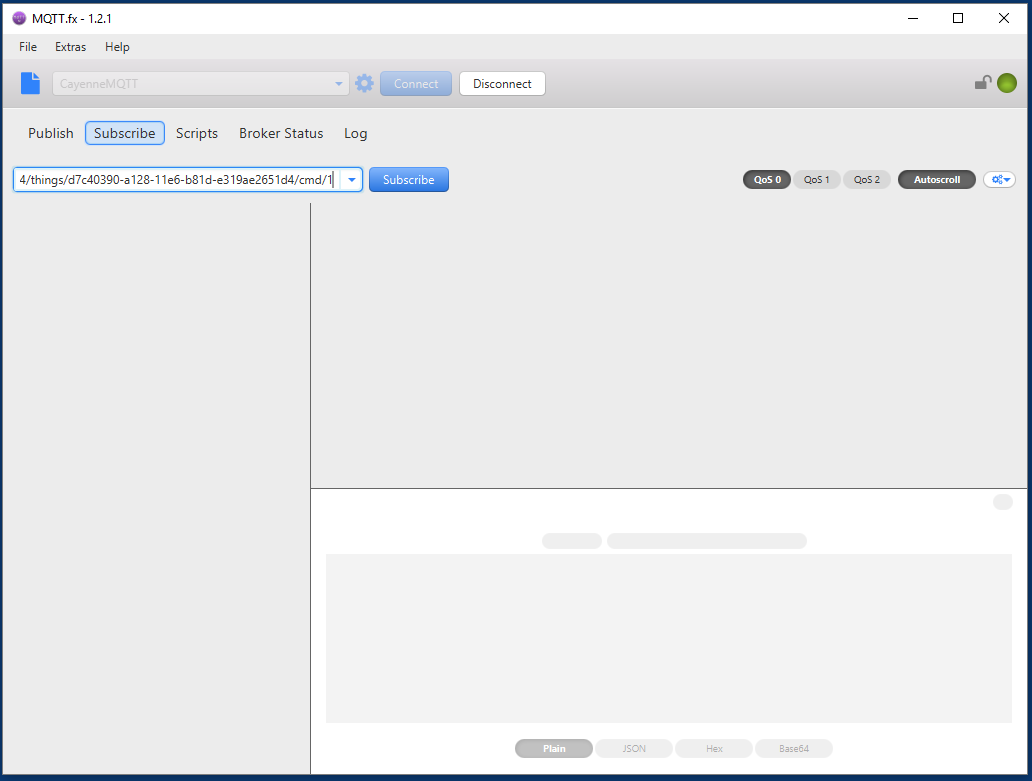
To receive the *COMMAND* messages that Cayenne sends for changes to our actuator, we must first Subscribe to them. To subscribe to actuator command messages, we refer to the [MQTT Messaging Topics - Receive actuator command](https://github.com/myDevicesIoT/cayenne-docs/blob/master/docs/MQTTAPIS.md#cayenne-mqtt-api-mqtt-messaging-topics-receive-actuator-command) section of the docs. There, we find the details on what MQTT call to make. According to the docs, subscribing to actuator commands expects:

v1/username/things/clientID/cmd/channel

We then substitute the values for our account, board and actuator.

* Replace **username** with the **MQTT Username** for your account.
* Replace **ClientID** with the **Client ID** for your board.
* Replace **channel** with the appropriate channel that this actuator is connected to. For this example, we've already chosen Channel **1** when our Light actuator widget was setup.

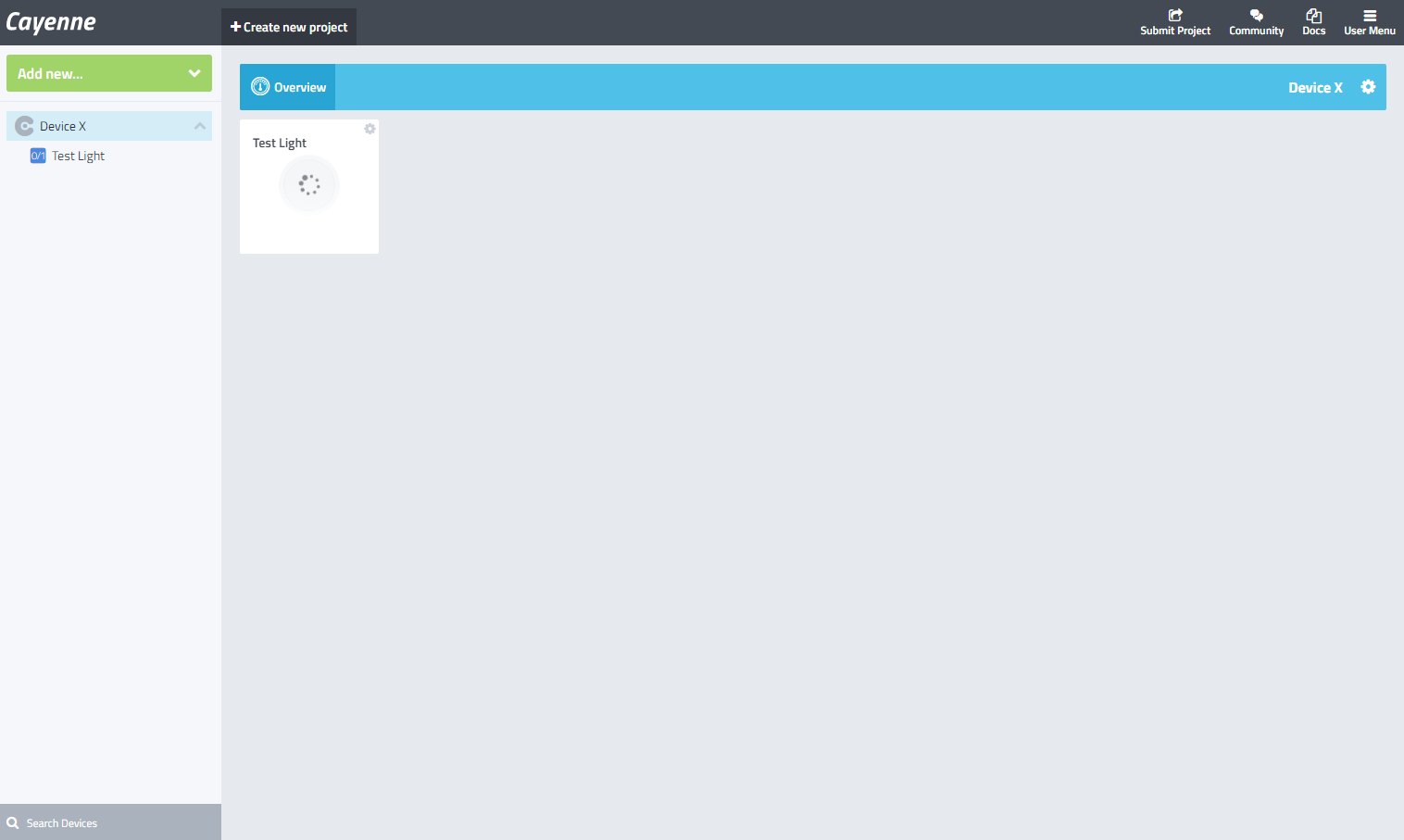
*TIP: For the purposes of testing with MQTT, you could also use the wildcard****#****which would subscribe to all channels. Cayenne supports all the usually filtering and control from MQTT that you'd expect. But since we know the specific Channel that we want to subscribe to in our example, we'll use that.*

[](https://camo.githubusercontent.com/e949866e586ded68ed439595651be9974b5280d7/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4d5154542e66782d31312d7375627363726962652d636f6d6d616e642d6d657373616765732e706e67)

After entering in the details needed to subscribe to the Command messages for our actuator, click on the **Publish** button in MQTT.fx. We are now subscribed to the *COMMAND* messages that Cayenne sends when our actuator state is changed.

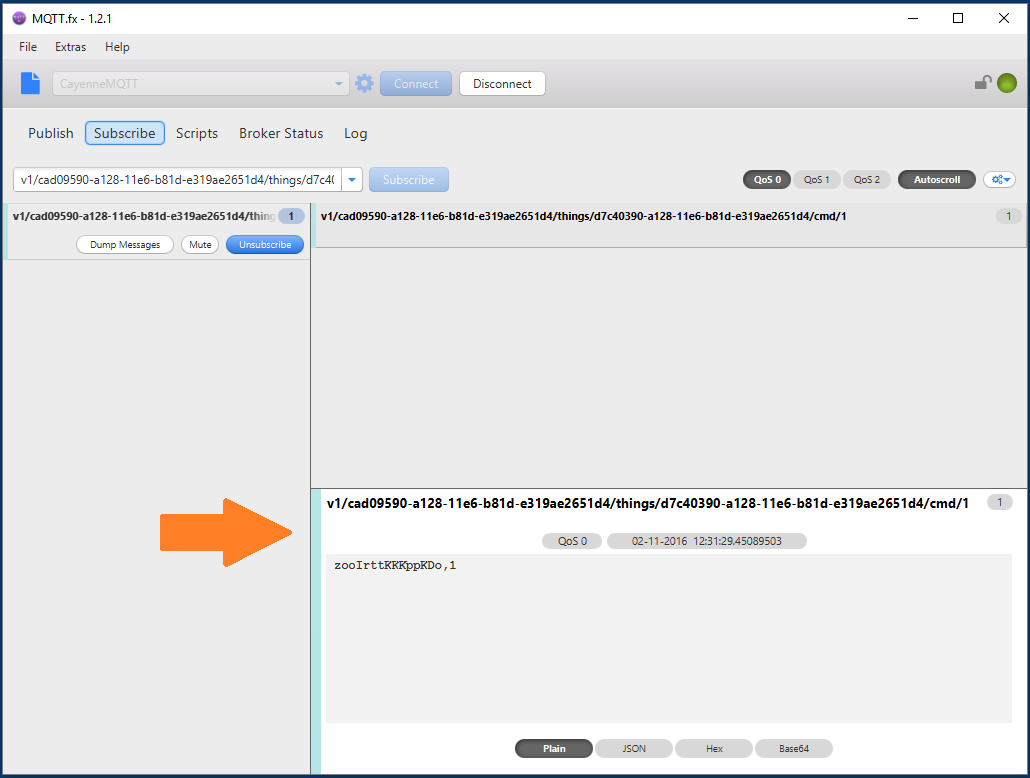
**Send Actuator Updated Value**

Now that we'll be notified when our actuator's state is changed on the dashboard, let's try it out. Using the Cayenne dashboard, click on the Button for your Light actuator. Cayenne publishes a *COMMAND* message and the dashboard widget will enter a *Waiting* state as it awaits confirmation that the actuator's state was changed.

[](https://camo.githubusercontent.com/c3d578d49316814f96f1ec9423affc9a73114ba2/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4150492d6163747561746f722d77616974696e672e706e67)

Examining MQTT.fx, we can see the Command message arrive from Cayenne. The message from Cayenne includes two parts:

1. A **Sequence** identifier. This is a randomly generated string that is used by the Cayenne Cloud to tie which widget this command is associated with. Keep this value in mind for now as we will need this identifier in the next step when we send a command response back to Cayenne.
2. A **Value**. This indicates what the new value of the actuator should be. In the case of our Light actuator, Cayenne is informing us that the new state should be 1 - "On".

[](https://camo.githubusercontent.com/fed3b8c50a0308a11926e52beb9bdfc654e98ee4/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4d5154542e66782d3132622d636f6d6d616e642d6d6573736167652d66726f6d2d636179656e6e652e706e67)

**Note:** Normally, at this point, we would handle changing the state of our actuator. For example, our code would interact with the actuator and change its state. Since we're just faking some data, there's nothing for us to change, but we do need to inform Cayenne that the actuator's state was changed. We do so by sending a value to Cayenne for what the new state is. Cayenne wanted us to turn on the Light, so let's simply tell Cayenne that's what happened.

To tell Cayenne what the updated value is, we refer to the [Send Actuator Updated Value](https://github.com/myDevicesIoT/cayenne-docs/blob/master/docs/MQTTAPIS.md#cayenne-mqtt-api-mqtt-messaging-topics-send-actuator-updated-value) section of the docs. There, we find the details on what MQTT call to make. According to the docs, Cayenne expects:

v1/username/things/clientID/data/channel

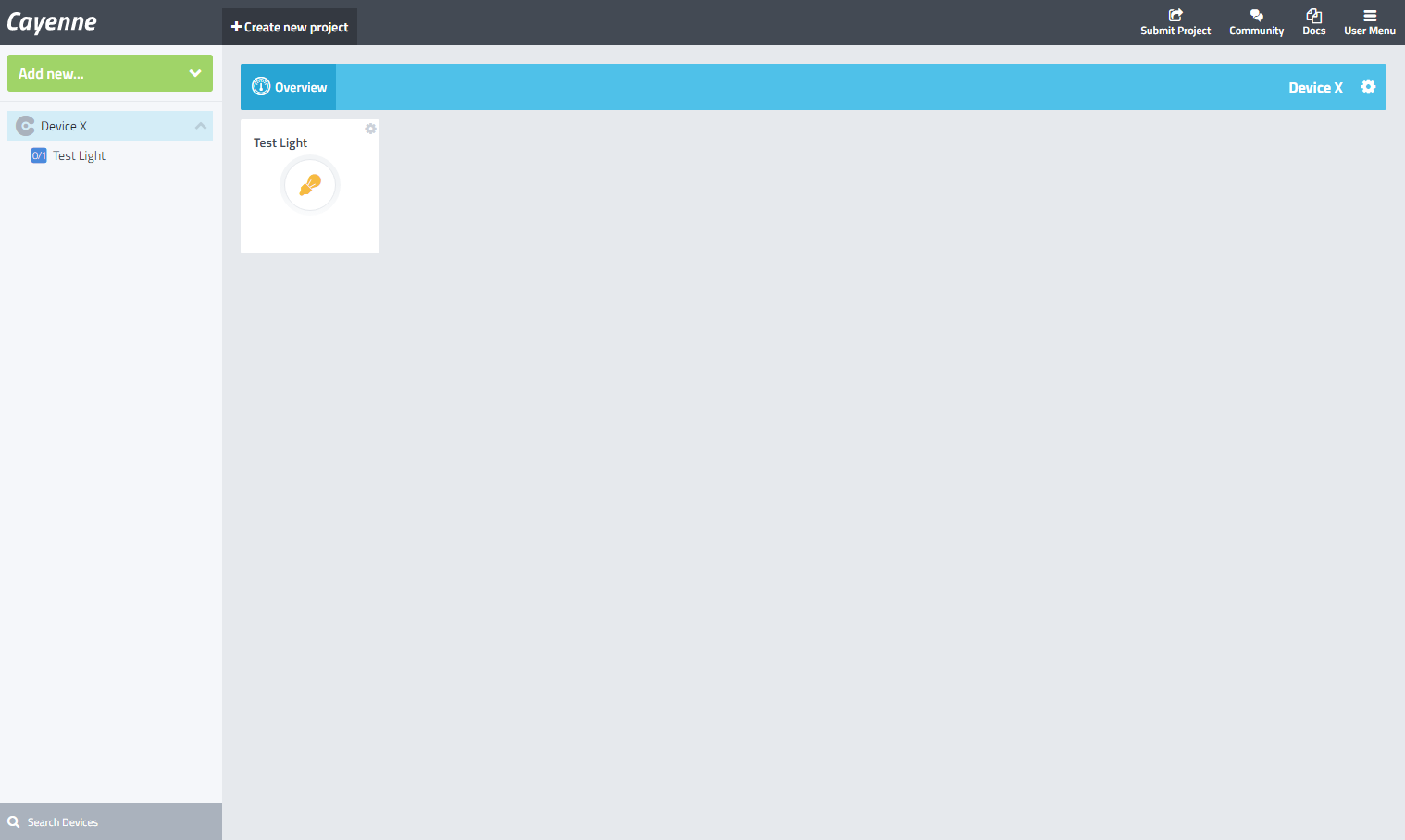
Once again, we then substitute the values for our account, board and actuator.

* Replace **username** with the **MQTT Username** for your account.
* Replace **ClientID** with the **Client ID** for your board.
* Replace **channel** with the appropriate channel that this actuator is connected to. For this example, we chose Channel 1 for our Light actuator.

Next, we need to inform Cayenne what the actual value for our actuator is. Cayenne asked for the new status to be 1 (meaning on), so we'll give it the value of **1**.

**Note:** Be sure to inform Cayenne what the **actual** status of the actuator is. For example, if changing the actuator state failed, be sure to tell Cayenne what the correct current value is. Cayenne needs to receive the current value so that the dashboard correctly informs the user what the actual state of their device is.

Once the dashboard receives an updated value for the widget, our Light actuator switches out of the *Waiting* state and now reflects the fact that our Light is On.

[](https://camo.githubusercontent.com/b3b7784ad217e504731c08705c44202ba131f16b/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4150492d6163747561746f722d6f6e2e706e67)

**Send command response**

The final step in completing our actuator change is to inform Cayenne as to whether the actuator state change was handled OK, or if there was a problem. This gives Cayenne an opportunity to display an error message to the user if needed. To send this to Cayenne, we use the [Send Command response](https://github.com/myDevicesIoT/cayenne-docs/blob/master/docs/MQTTAPIS.md#cayenne-mqtt-api-mqtt-messaging-topics-send-command-response) call. Examining the docs for this call, we see Cayenne expects:

v1/username/things/clientID/response

Once again, we then substitute the values for our account and board.

* Replace **username** with the **MQTT Username** for your account.
* Replace **ClientID** with the **Client ID** for your board.

Next, we need to tell Cayenne if there was an error or if things were handled OK. According to the docs, we need to send this in one of the following forms (depending on OK vs Error):

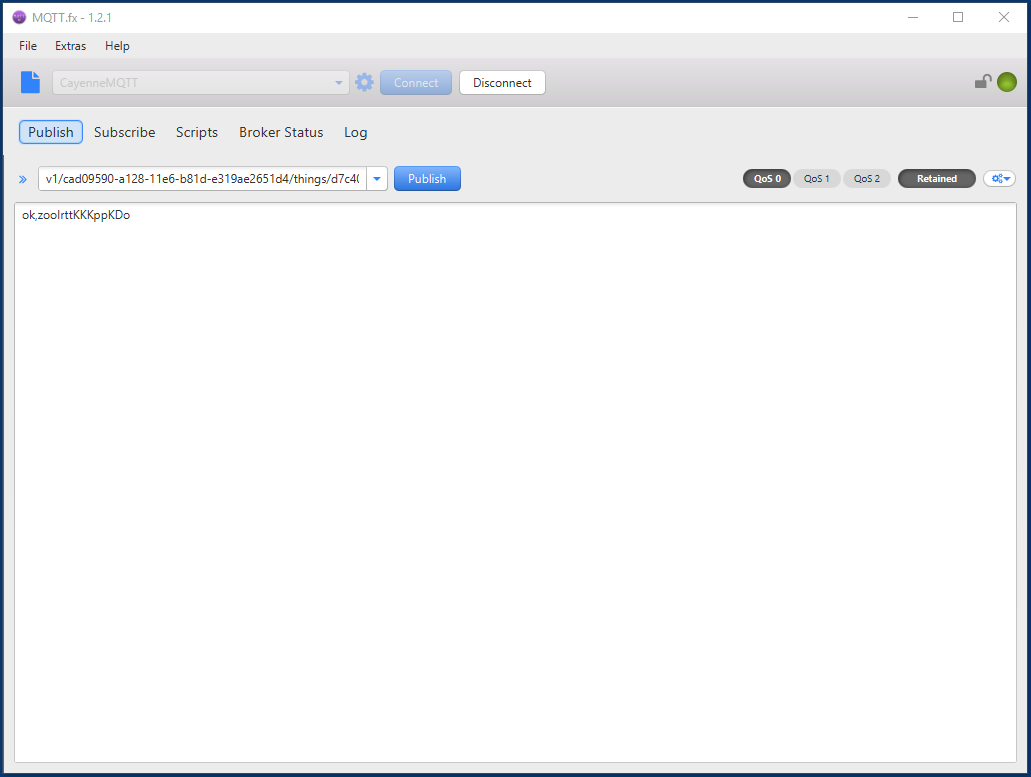
ok,seq

error,seq=message

We're using a make-believe actuator, so changing our actuator's state can't fail. So, in this case, let's just tell Cayenne that everything was handled without incident. We need to provide the following two pieces of information for this:

1. The string **Value** "ok". This lets Cayenne know things were handled OK, no errors to display to the user on the dashboard.
2. The **Sequence** identifier. If you'll recall, this value was provided to us when we received the *COMMAND* message from Cayenne. We provide it here so that Cayenne knows what we're responding to.

Click on the **Publish** button to send the command response to Cayenne. If you send an error message, Cayenne will display it in the dashboard.

[](https://camo.githubusercontent.com/dfbaa7e197fd34f7646a02e39cb476f278e23a1f/687474703a2f2f7777772e6d79646576696365732e636f6d2f636179656e6e652f75706c6f6164732f4d5154542e66782d31342d726573706f6e73652d746f2d636179656e6e652e706e67)

# **Cayenne Agent**

The Cayenne agent is a full-featured client for the [Cayenne IoT project builder](https://mydevices.com/). It sends system information as well as sensor and actuator data and responds to actuator messages initiated from the Cayenne dashboard and mobile apps. The Cayenne agent currently supports Raspbian on the Raspberry Pi, but it can be extended to support additional Linux flavours and other platforms.