

EE579 Group 4 Documentation

This site provides documentation to allow integration with our API and MQTT Broker.

If you would like to learn the web interface and functionality provided by the system you can start by [Creating an Account](#). Then you can get started adding [Devices](#) and [Rules](#)

If you would like information on how to develop your own IoT device and integrate with our system, start by reading the [Register a Device](#) page.

System Overview

Admin Interface

Our admin interface is available here: <https://www.ee579-group4.net>. This site enables users to login and configure rules, devices, and device groups.

Our system is multi-tenant, with users able to create multiple tenants and invite other users. Rules and devices are then scoped to these tenants, meaning users only have permission to access and edit resources belonging one of their tenants.

API

Our API is written in C# using ASP.NET Core. Full Swagger OpenAPI documentation is available here: <https://ee579-dev-api.azurewebsites.net>

MQTT Broker

We are using [Azure IoT Hub](#) as our MQTT Broker. Information on connecting and receiving/publishing messages can be found [here](#)

GitHub Repos

- [ee579-api](#) - C# API repository
- [ee579-web](#) - React web interface repository
- [ee579-msp430](#) - Repository containing msp430 and ESP8266 code
- [ee579-docs](#) - Repository containing the markdown files used to create these docs

Getting Started

Registering a device

In order for your device to connect to IoT Hub, it must first retrieve credentials. This can be done by sending a `POST` request to `https://ee579-dev-api.azurewebsites.net/devices/register`. This will register your device with our system and return credentials to allow connection over MQTT. The device should make this request when first powered on, but the same endpoint can be used even if a device is already registered in the system. This accounts for situations where the device may have lost its credentials, and allows the device to make the request every time it powers on, if desired.

Request Format

Example Body

```
{
  "deviceId": "<string>"
}
```

Parameters

- **deviceId**: This should be a unique identifier. A MAC address is recommended, but it can take any form - guid, uuid, etc.

Response Format

Example Body

```
{
  "connectionString": "HostName=IFTTT-Iot-Hub.azure-devices.net;DeviceId={deviceId};SharedAccessKey=El50EkQ/S/9qp5dd/V3VpsizIvSv+SA8TVF3QiGc93A=",
  "host": "IFTTT-Iot-Hub.azure-devices.net",
  "port": 8883,
  "topic": "devices/{deviceId}/messages/devicebound/#",
  "password": "SharedAccessSignature sr=IFTTT-Iot-Hub.azure-devices.net%2Fdevices%2F{deviceId}&sig=KOYS9LgCJ9eH7TTL1MIGvedxIVI3cXmha7uU6yB4Bs6M%3D&se=88018657115"
}
```

Parameters

- **connectionString**: The connection string the broker. {deviceId} should be replaced with a unique identifier. A MAC address is recommended, but it can take any form - guid, uuid, etc.
- **host**: The MQTT broker url.

- **topic:** The topic that the device should subscribe to, to receive cloud-to-device messages.
- **password:** The password that should be used when connecting to the broker, in the form of a [SAS token](#).

Debugging

It may be necessary to make this request on a computer, rather than IoT device when debugging. The [API Swagger Docs](#) can be used to make this request and provides a prefilled request body. Any other method of making HTTP requests can also be used - cURL, [Postman](#), etc.

Next Steps

- [Connecting to the broker](#)

MQTT Broker

Connecting

To connect to IoT Hub there are multiple different protocols that can be used, you can either directly connect to these protocols or preferably use one of the IoT Hub device SDKs.

Connecting using the IoT Hub SDKs

Connecting to IoT Hub using one of Microsoft's SDKs use a connection string to connect. The connection string can be garnered from the response from our API's register request. Using the SDKs is the recommended way of connecting for its ease of use, especially regarding certificate handling as IoT Hub forces TLS/SSL.

Some of the available SDKs include:

- [C SDK](#)
- [Python SDK](#)
- [Node.js SDK](#)
- [Java SDK](#)
- [.NET SDK](#)

Using our Example Code for an ESP8266

We have provided some example code for registering a device with the API and sending/receiving messages from IoT Hub. This example code is for the ESP8266 Wifi Module, but the methods will be similar on alternative devices. The code can be pulled from this [repository](#). Cd into the esp_iotHub_client directory for the example code. By following the instructions [here](#), you can setup the Arduino IDE with functionality for flashing our example code to the ESP8266.

Connecting directly to the protocols

As MQTT is usually implemented on memory restricted devices, it is possible that the memory required to implement broker connection/interaction using an SDK is unavailable. For this reason, directly connecting to the protocol might be required. Below is a mosquitto subscribe command showing the parameters required to manually connect to the cloud device topic.

```
mosquitto_sub \  
  -h IFTTT-Iot-Hub.azure-devices.net \  
  -p 8883 \  
  -t "devices/00:0a:95:9d:68:16/messages/devicebound/#" \  
  -i 00:0a:95:9d:68:16 \
```

```
-u "IFTTT-Iot-Hub.azure-devices.net/00:0a:95:9d:68:16" \  
-P "SharedAccessSignature sr=IFTTT-Iot-Hub.azure-  
devices.net%2Fdevices%2F00%3A0a%3A95%3A9d%3A68%3A16&sig=ulc08e%2FYp%2FMLJdyMLxsV  
\  
--cafile C:\Users\Fraser\GitHub\ee579-  
api\EE579\EE579.Core\Slices\Devices\Simulate\iotHubCert.pem -d -V mqttv311 -  
q 1
```

- **-h**: The host name for the broker.
- **-p**: The port number of the broker.
- **-t**: The topic the cloud to device messages will be received on.
- **-i**: The unique identifier for the device. Recommended as the device MAC address.
- **-u**: The MQTT username which here is just set to the hostname/deviceId.
- **-P**: The password for the broker. Can be retrieved from the register API response.
- **--catfile**: Path to the IoT Hub Certificate.
- **-V**: The MQTT version. IoT Hub only supports MQTT version 3.1.1.
- **-q**: This is for the quality of service. IoT Hub only supports a quality service of 1.

MQTT Messaging Schema

This page shows schema for messages between device and cloud. All message parameters should be sent in the message body, in JSON format as detailed below. It should be noted that the JSON is case-sensitive so all JSON key and values must be formatted as shown in the 'Message Body' schema.

Device to Cloud Messages

Device-to-cloud messages should be sent when an input has been triggered. These messages will be used to trigger rules if the conditions are met.

Button Pushed

This message is sent to the broker whenever a button is pushed on the microcontroller.

Message Body

```
{
  "InputType": enum, // [Button1, Button2]
  "Value": int // The length the button is held in ms
}
```

Potentiometer Input

This message is sent every time the analogue value from the potentiometer is read. The value is sent with this message.

Message Body:

```
{
  "InputType": "Potentiometer",
  "Value": int // (0 - 1023) - the analogue value of the potentiometer
}
```

Temperature Input

This message is sent every time the analogue value from the temperature sensor is read. The value is sent with this message.

Message Body:

```
{
  "InputType": "Temperature",
  "Value": int // (-50 to 100) - the temperature in degrees celsius
}
```

Cloud to Device Messages

Cloud-to-device messages are used to perform outputs on the device as a result of a rule being triggered. There is also a message used to configure the device's inputs to reduce the number of messages sent.

LED Output

This message is sent to notify the microcontroller to turn an LED on/off and with the specified colour.

Message Body:

```
{
  "OutputType": "LedOutput",
  "Peripheral": enum, // [Led1, Led2, Led3]
  "Value": bool, // the desired state of the LED.
  "Colour" enum // (Led3 only) [Red/Green/Blue/Purple/Yellow/White]
}
```

Breath LED

This message is sent to notify the microcontroller to breathe an LED at a specified speed and colour.

Message Body:

```
{
  "OutputType": "LedBreathe",
  "Peripheral": enum, // [Led1, Led2, Led3]
  "Period": int, // how long the period of the breathing should be in ms.
  "Colour": enum // (Led3 only) [Red/Green/Blue/Purple/Yellow/White]
}
```

Blink LED

This message is sent to notify the microcontroller to blink an LED at a specified speed and colour.

Message Body:

```
{
  "OutputType": "LedBlink",
  "Peripheral": enum, // [Led1, Led2, Led3]
  "Period": int, // how long the period of the blinking should be in ms.
  "Colour": enum // (Led3 only) [Red/Green/Blue/Purple/Yellow/White]
}
```

Fade LED

This message is sent to notify the microcontroller to fade an LED to a desired state at a specified speed and colour.

Message Body:

```
{
  "OutputType": "LedFade",
  "Peripheral": enum, // [Led1, Led2, Led3]
  "Value": bool, // the desired state of the LED.
  "Duration": int, // the duration the fade should take in ms.
  "Colour": enum // (Led3 only): [Red/Green/Blue/Purple/Yellow/White]
}
```

Cycle LED 3

This message is sent to notify the microcontroller to cycle the LED colour in the specified direction.

Message Body:

```
{
  "OutputType": "LedCycle",
  "Direction": bool, // cycle forwards or backwards - [true/false]
  respectively
  "Period": int // how long the period of the cycle should be in ms
}
```

Buzzer On

This message is sent to notify the microcontroller to turn the piezo buzzer on for the specified length of time.

Message Body:

```
{
  "OutputType": "BuzzerOn",
  "Duration": int, // the duration the buzzer should be on in ms
}
```

Beep Buzzer

This message is sent to make the microcontroller toggle a buzzer at the desired rate.

Message Body:

```
{
  "OutputType": "BuzzerBeep",
  "OnDuration": int, // the duration the buzzer should be on in ms
  "OffDuration": int // the duration the buzzer should be off in ms
}
```

Configure Device

This message is sent to notify the microcontroller which input devices are involved in rules. This is used to limit the number of messages sent by devices - if an input device is not involved in any rules, messages should not be sent to the broker when its value changes.

This message is sent whenever a device connects to the broker, or if a rule is created, updated, or deleted involving the device.

Message Body:

```
{
  "MessageType": "DeviceConfig",
  "Button1": bool,
  "Button2": bool,
  "Potentiometer": bool,
  "Temperature": bool
}
```


Web Hooks

What is a Web Hook

Web hooks allow one application to integrate with another over HTTP. This is commonly done by the external application making a post request to an endpoint provided by the application in which the functionality is performed.

Using Web Hooks in our System

In the context of our system, web hooks can be used for both inputs to rules, and outputs from rules.

Inputs

Using a web hook as a rule input allows a 3rd party application to trigger the web hook rule input by making an HTTP request to our API. When a rule is created with a web hook input, a unique URL is generated. In order to trigger the rule, a post request must be made to this URL. Inherently, the logic that determines when the web hook URL should be invoked must be implemented by the 3rd party developer. For example, if a 3rd party wanted to trigger a rule using a moisture sensor, the 3rd party must implement their own low level code to handle the desired analogue thresholds for the sensor and then send the HTTP request to the web hook URL, which will trigger the rule.

An example URL is shown below:

```
http://ee579-dev-api.azurewebsites.net/webhooks/trigger/{unique_code}
```

This is a callback URL to our API where `{unique_code}` is the code used to identify the specific rule input.

Outputs

Using a web hook as a rule output allows our system to trigger events on a 3rd party application. This works by our API sending an HTTP post request to your specified URL upon the rule's inputs being triggered. The URL must be public facing and have an appropriate CORS policy to allow the receipt of our request. That is, POST request should be allowed and the cross origin policy should allow for requests from our API.

Why use Web Hooks?

Our web hook implementation allows our system to be controlled by, or control, a completely different system, assuming that the different system is capable of sending or receiving HTTP post requests. For a web hook as a rule input it means a rule from any connected device on our system, can be triggered by invoking the web hook URL (that we generate for you) by sending a post request to it. This gives almost limitless potential for input interoperability thereby making our system not subject to the hardware or input implementation limitations on certain microcontrollers such as the MSP430. That is, any complex inputs that we do not support can still be used where their HIGH event on the 3rd party application, can then invoke the web hook URL, thus triggering a rule where our system need not know of the complex input that might have caused it, only that the web hook has been invoked. Likewise, with using a web hook as a rule output, it can trigger functionality on a different system where the triggered functionality is controlled by the rule logic in our own system.

Examples and Use Cases

IFTTT

[IFTTT](#) (If This Then That) is a popular software platform that connects apps, devices and services from different developers in order to trigger one or more automations involving those apps, devices and services. IFTTT offers web hooks as both logic inputs and outputs. On our system, by specifying the web hook output URL as your IFTTT web hook input URL, you have potential to control hundreds of different services and devices all from our own system. An example of this could be using a device on our system with a rule setup with a button pressed input event, then with a web hook output from the rule that points to a web hook input on IFTTT, you could automatically order a Dominoes, send an SMS for you, turn your heating up, post a tweet and hundreds more options. Likewise, this system interaction can work in reverse, with devices on our system being influenced by your logic on IFTTT such as sounding a buzzer on one of our devices and a specific time of day. The possibilities are endless.

Azure Logic Apps

The corporate equivalent to IFTTT, is [Logic Apps](#). A Logic App allows for automation of business workflow tasks and can be controlled with input triggers and produce some output such as sending an email. In a similar fashion to how our system might interact with IFTTT, there is vast potential for controlling your business logic using devices on our system.

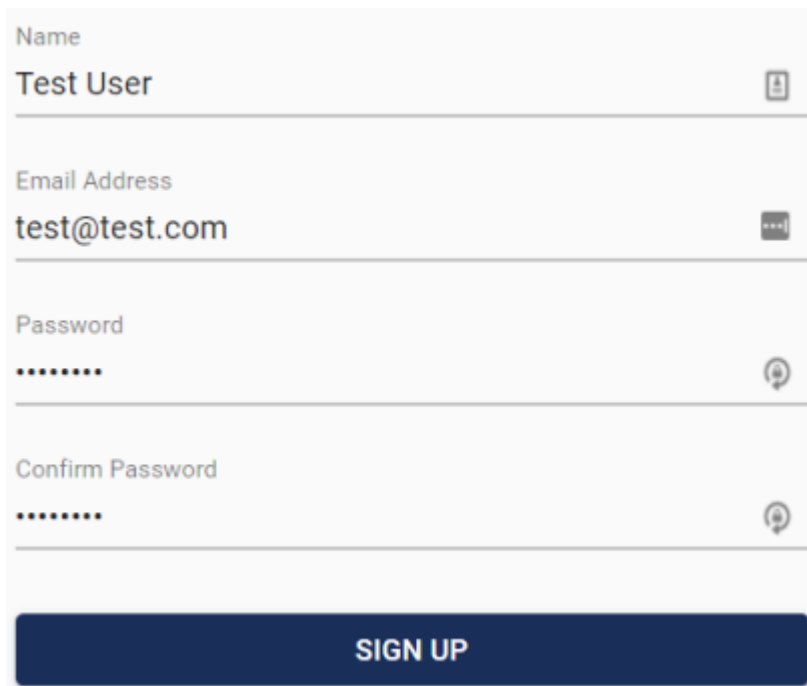
Create an Account

Visit <https://www.ee579-group4.net/signup> to get started.

There are several methods available for signing in:

Username and Password

To create an account with a username and password, fill out the form on the sign up page.



Name
Test User

Email Address
test@test.com

Password
.....

Confirm Password
.....

SIGN UP

You will then be sent an email to verify that you have access to the email address. You must then click the link in the email before you can sign in.

External Providers

Logging in with an external provider allows you to reduce the number of passwords and accounts you have to manage. This site allows you to sign in with your Google or Microsoft account. Click the respective button to start the external sign in process. You will then be redirected to the external providers site where you can sign in.

After signing in to the external providers account, you will be redirected back to this site and signed in. Verifying your email is not required when using external sign in as signing into the account verifies that you have access to the email.

Next Steps

- Learn about how [Multi-Tenancy](#) works in this application

Multi-Tenancy

What is Multi-Tenancy?

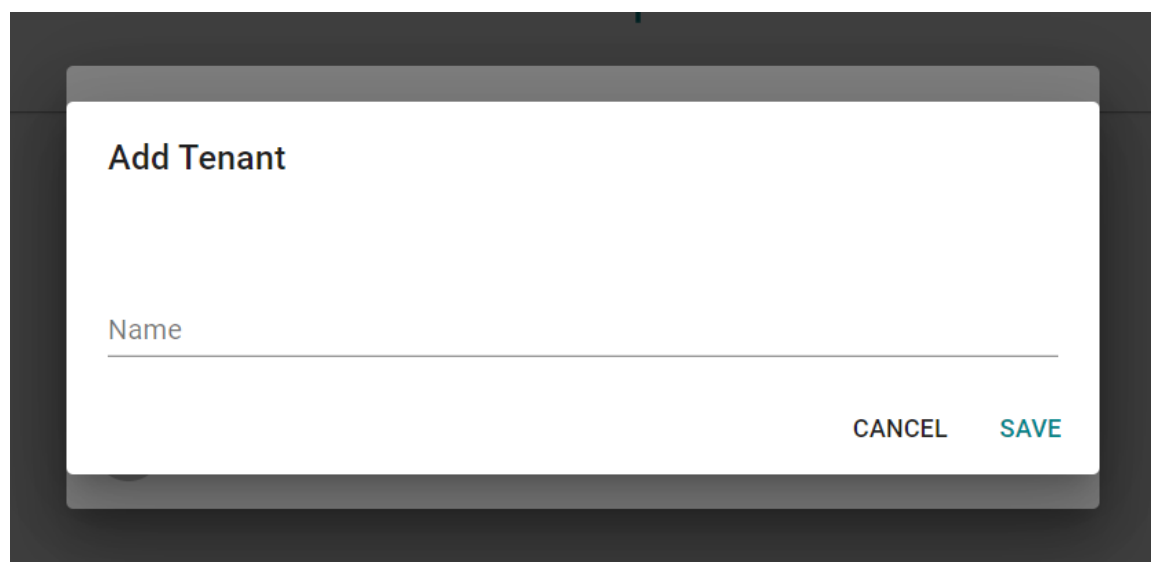
Multi-tenancy is an architecture that allows a single application to be used by multiple customers. This means that data in the application is scoped to the current customer and not visible to other customers of the application. An example of this is the university's use of Microsoft's suite of applications. In this case, the university has their own tenant. This tenant contains all of the data specific

Multi-Tenancy in this application

Multi-tenancy in this application involves scoping rules, devices, and device groups to tenants. This allows multiple customers to use the application without other customers seeing their data. Users can then belong to multiple tenants, allowing them to view and manage their data.

Creating a Tenant

When first creating an account a new tenant is created for you, allowing you to get started creating rules and adding devices. You may want to add another tenant in order to keep unrelated data separate - e.g., keeping rules and devices for each building in their own tenant. This can be done by clicking the 'Switch Tenant' button in the left drawer and then selecting 'Create a new tenant'.

A screenshot of a web application interface showing a modal dialog box titled "Add Tenant". The dialog box is white with a dark gray border and is centered on a dark gray background. Inside the dialog, there is a text input field labeled "Name" with a light gray placeholder text. Below the input field, there are two buttons: "CANCEL" in dark gray and "SAVE" in teal. The dialog box has a subtle drop shadow.

Next Steps

- Learn how to [manage users](#) within tenants
- Find out how to [add and manage devices](#)



Users

The screenshot shows the 'Users' management interface for a tenant named 'Fraser Bell (Strath)'. The interface includes a sidebar with navigation options and a main content area with a table of users.

Navigation Sidebar:

- EE579 IFTTT
- Dashboard
- Rules
- Devices
- Users**
- Device Groups
- Switch Tenant
- Sign Out
- Manage Account
- Fraser Bell (Stra... ▼

Users Table:

Name	Email	Role	Status	Actions
Fraser Bell (Strath)	fraser.bell.2017@uni.strath.ac.uk	Owner	Active	-
-	test@test.com	User	Invited	 

1-2 of 2 < >

Roles

Users can have one of two roles within a tenant:

- **User** - Users with this role can manage devices, rules, and device groups. However, they cannot manage users belonging to the tenant or edit the tenant itself.
- **Owner** - This user has full control over the tenant. They can manage everything a user can, along with inviting and removing users and editing the tenant.

Inviting Users

The screenshot shows a web application interface for managing users in a tenant named 'Fraser Bell's Tenant'. The main section is titled 'Users +'. Below the title is a table with columns: Name, Email, Role, and Status. The table contains several rows of user data. A modal window titled 'Invite User' is open in the center, allowing a user to add a new user. The modal has fields for 'Email' and 'Role' (a dropdown menu currently set to 'User'). Below these fields is a note: 'Users can manage rules and devices but cannot invite or remove other users'. At the bottom of the modal are two buttons: 'CANCEL' and 'INVITE'.

Name	Email	Role	Status
Fraser Bell (Strath)	fraser.bell.2017@uni.strath.ac.uk	User	Active
-	-	User	Invited
-	-	User	Invited
-	-	User	Invited
-	-	User	Invited
-	-	User	Invited
-	-	User	Invited
-	-	User	Invited
-	-	User	Invited
-	fraser.bell.2017+9@uni.strath.ac.uk	User	Invited
-	fraser.bell.2017+1@uni.strath.ac.uk	User	Invited

Users can be invited to tenants by their email address. If the user already has an account, they will have immediate access to the tenant and receive an email telling them that they have been invited to your tenant. If they have not yet created an account they will receive an email informing them they have been invited to your tenant, including a link to sign up.

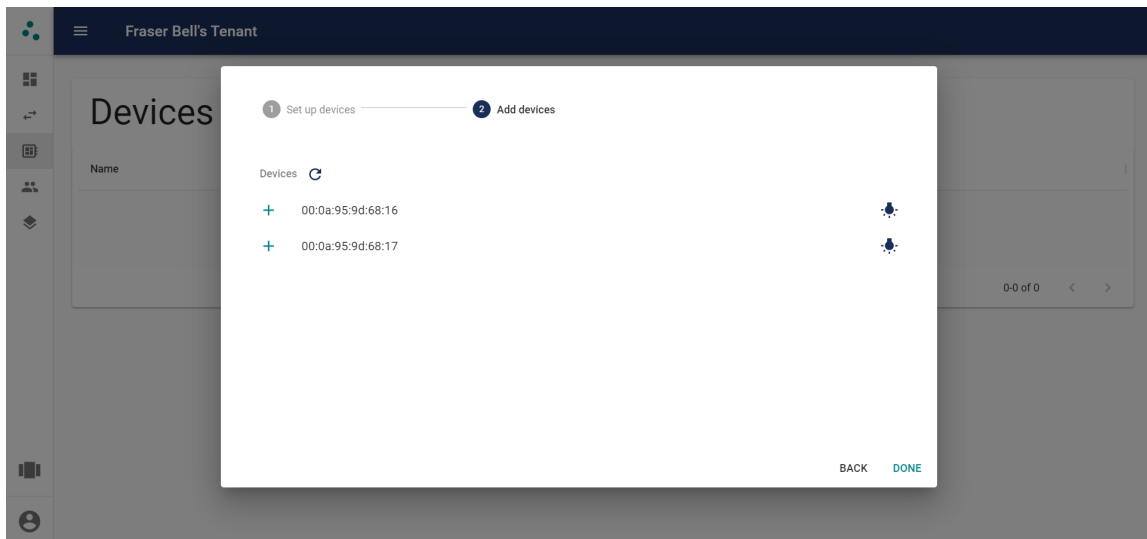
Devices

Devices are the devices that will be used in rules. They send inputs to the API and receive messages to perform outputs.

Adding Devices

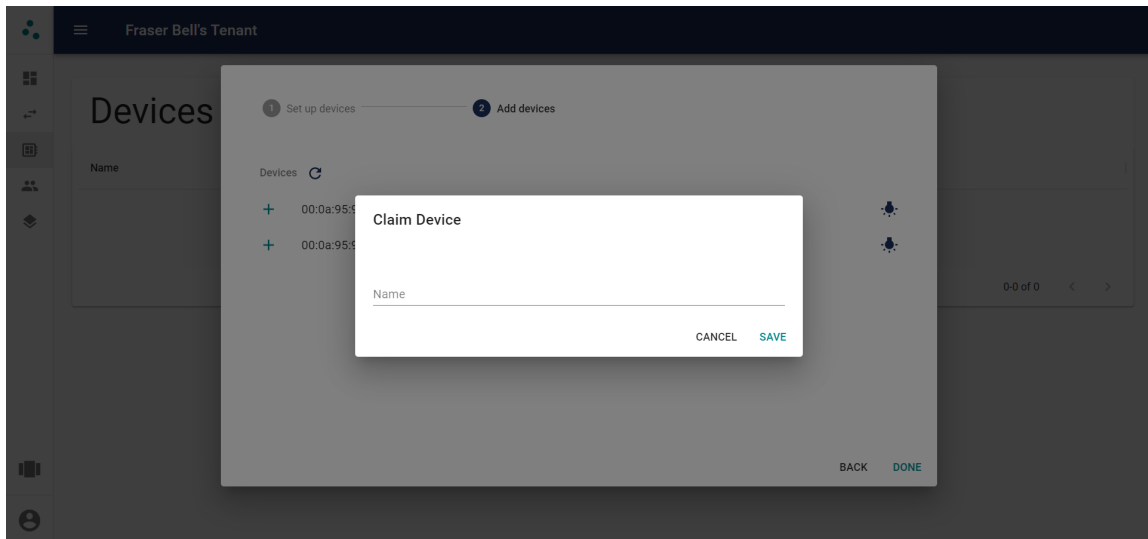
Before devices can be used to create rules, they must be claimed by a tenant. The first step is to power the device on. When powered on for the first time, the device will register itself with the system, making it possible to claim it in the web interface.

After the device has been powered on, navigatet to the devices page and click the add button above the table. This shows the modal below.



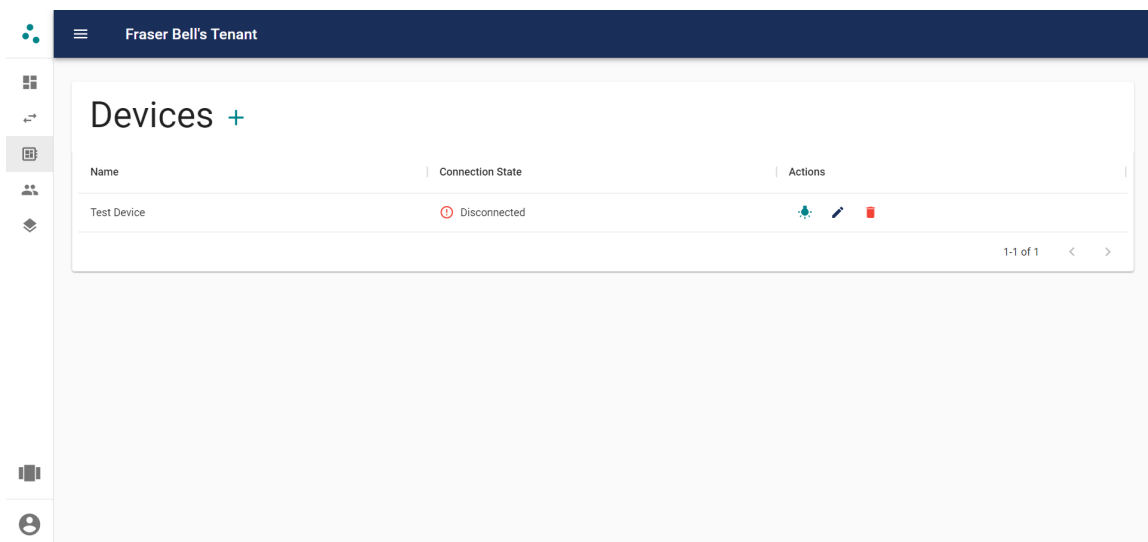
This shows the devices available for you to claim. You must be on the same network as a device in order to claim it. This prevents other users from claiming devices that do not belong to them. The devices are listed by their MAC address which you may use to identify specific devices when multiple are available. If you do not know the MAC address of your device you can press the lightbulb button. This will flash an LED on the device for 5 seconds, allwoing you to identify which device the list item corresponds to.

To claim a device press the add button next to its MAC address. This will bring up a form allowing you to give the device a user friendly name before claiming it.



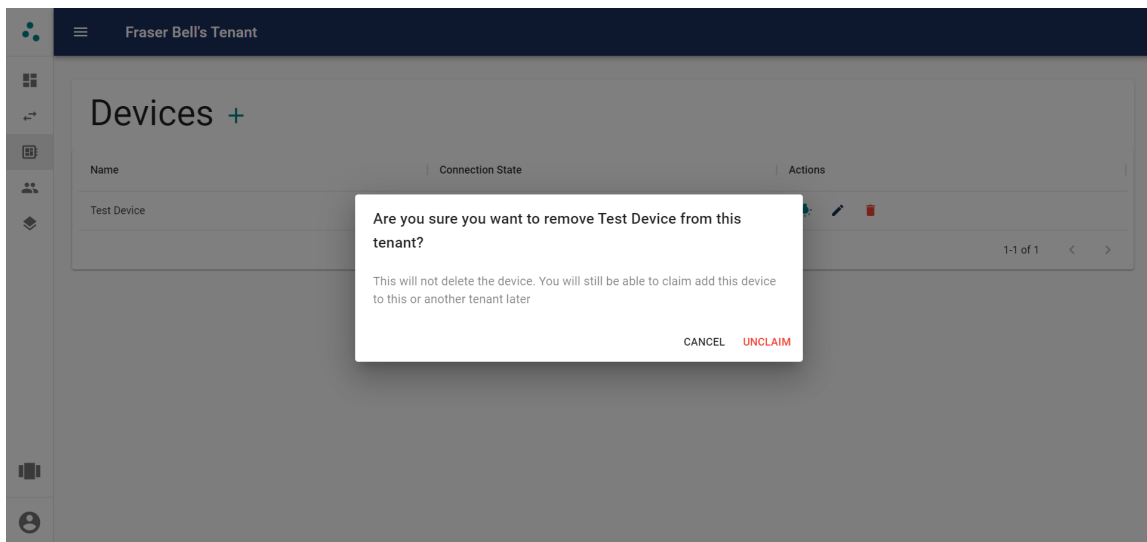
Managing Existing Devices

After devices have been added, they will be visible on the devices page as shown below.



This page allows you to view all of your devices and their current state. The 'Connection State' column shows whether the device is currently connected to the MQTT broker and able to send and receive inputs and outputs. It also allows you to perform some actions on the device. Here you may identify a device in the same way as when claiming it, by pressing the LED button. You can also change the device's name and unclaim it from the current tenant.

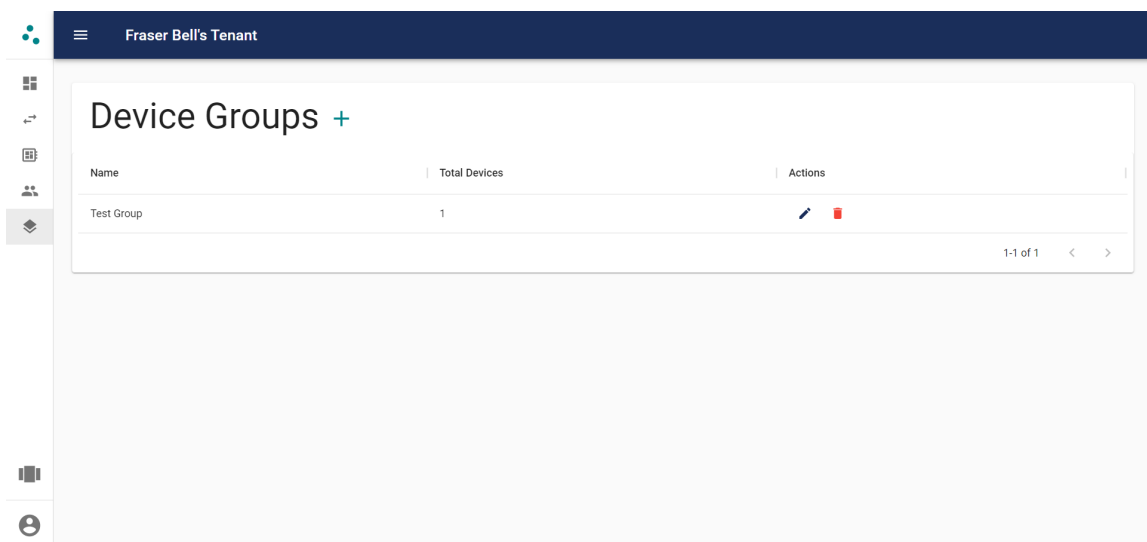
Unclaiming a Device



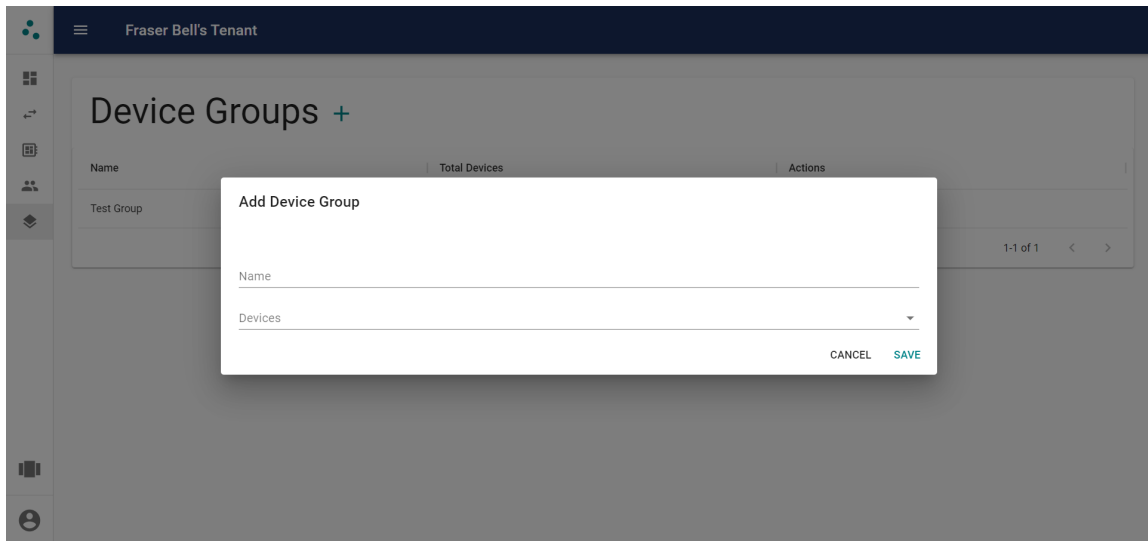
You may wish to unclaim a device from a tenant. This removes the device from its current tenant, and removes it from any rules it may have been involved in. This does not delete the device completely, only resets it, allowing it to be claimed by another tenant. If you want to move a device from one tenant to another this would be the way of doing that.

Device Groups

Device groups provide a method of grouping devices with similar functions to improve the process of adding and managing rules.



Device groups can be added by specifying a name, and list of devices that should be long to the group. The name and devices can also be edited later.



Next Steps

- Find out how [Rules](#) work

Rules

Rules are the core functionality of the system, allowing inputs from devices to trigger outputs.

Creating Rules

Navigate to the rules page and press the add button to create a new rule. The rule form is shown below:

The screenshot shows the 'Add Rule' form in the Fraser Bell's Tenant system. The form is divided into three main sections: Rule Name, Trigger, and Outputs. The Trigger section includes Input Properties (Input Type: Button Pushed, Button Pressed Duration: Short Press) and Trigger Device(s) (Trigger Device or Trigger Group). The Outputs section includes Output Properties (Output Type: Buzzer - On, Buzzer On Duration (ms): 2000) and Output Device(s) (Trigger Device or Trigger Group). A blue button labeled 'ADD ANOTHER OUTPUT' is at the bottom right.

The form is composed of 3 sections: Name, Trigger, and Outputs.

Trigger

The Rule trigger is the device input that will trigger the rule and cause the outputs to be performed. This is composed of the input type and parameters desired, and the device or device group the input will come from. If a device group is chosen, an input matching the desired type and parameters from any of the groups devices will trigger the rule.

There are several input types that can be chosen:

Input Types

Button Pushed

Input Properties



Input Type

Button Pushed



Button Pressed Duration

Short Press

This input is sent when the button is pushed on a device. The duration desired to trigger the rule can be set to one of three values:

- **Short Press** - $0s < 2s$
- **Medium Press** - $2s < 10s$
- **Long Press** - $> 10s$

Switch Flipped

Input Properties



Input Type

Switch Flipped

Switch Peripheral

Switch 1

Switch State

On

This input allows rules to be triggered when one of the 2 switches on the device are flipped. The peripheral (switch 1 or 2) can be chosen and the switch state. The switch state is used to specify whether the switch should be triggered when the switch is flipped to an on or off state.

Potentiometer

Input Properties



Input Type

Potentiometer

256

768

Greater Than

256



Less Than

768

This input allows you to specify a range of potentiometer values that will trigger the rule. The default configuration is a range between two values but this can also be inverted to allow for

specifying a range of values less than the lower value *or* greater than the higher value. This can be done by pressing the button between the values, or manually setting the greater than value to be higher than the less than.

Input Properties




The interface shows a configuration for a Potentiometer input. At the top, there is a circular refresh icon and a dropdown menu labeled 'Input Type' with 'Potentiometer' selected. Below this is a horizontal slider with two teal circular markers. The left marker is labeled '256' and the right marker is labeled '768'. Under the slider, there are two input fields: 'Greater Than' with the value '768' and 'Less Than' with the value '256'. A double-headed arrow icon is positioned between these two fields, indicating that the range can be inverted.

When using a potentiometer value as an input it should be noted that the rule will not be triggered every time the potentiometer value changes if it is within the trigger range. Instead, the system keeps track of the last potentiometer value sent by the device and only triggers the rule if the last value was outwith the range and the new value is within it.

Temperature

Input Properties

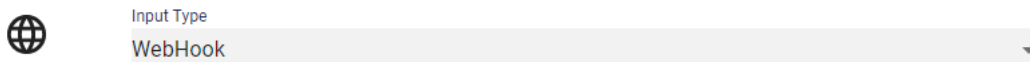


The interface shows a configuration for a Temperature input. At the top, there is a flame icon and a dropdown menu labeled 'Input Type' with 'Temperature' selected. Below this is a horizontal slider with two teal circular markers. The left marker is labeled '-9°C' and the right marker is labeled '50°C'. Under the slider, there are two input fields: 'Greater Than' with the value '-9' and 'Less Than' with the value '50'. A double-headed arrow icon is positioned between these two fields, indicating that the range can be inverted.

This input allows you to use the temperature sensor on your device as a trigger for the rule. It functions in a similar way to the potentiometer trigger, with a range picker. Again this range can be inverted, and the rule is only triggered when the last value did not satisfy the conditions.

Web Hooks

Input Properties

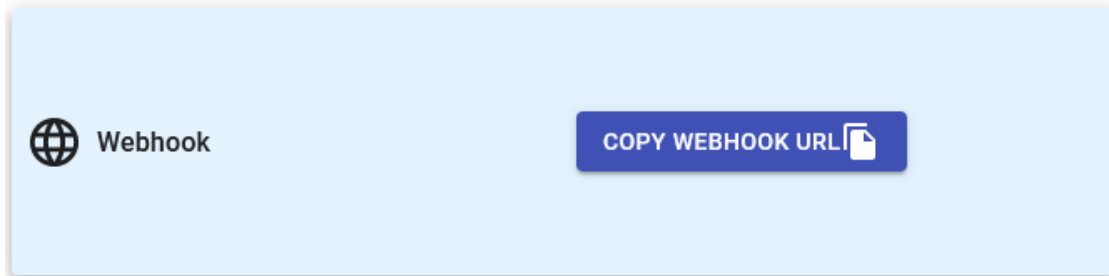


The interface shows a configuration for a WebHook input. At the top, there is a globe icon and a dropdown menu labeled 'Input Type' with 'WebHook' selected.

This is an input that can be triggered via the invocation of a generated URL, commonly known as a web hook. It works similar to a physical input such as a button press, but instead of being triggered upon the button being pressed, the trigger happens upon invocation of the URL. After adding a rule with a web hook as an input, from the rules page on the website, find the added rule

from which the invoke URL can be garnered. Below is an example of what this looks like. For more information for developers and using web hooks for interoperability please see [here](#).

Inputs



Outputs

These are the outputs that will be performed when a rule is triggered. Multiple outputs can be performed for each rule.

OUTPUT TYPES

Buzzer - On


Output Properties

	<div>Output Type</div> <div>Buzzer - On</div>
	<div>Buzzer On Duration (ms)</div> <div>2000</div>

This output is used to sound a continuous buzzer tone for the specified duration.

Buzzer - Beep

Output Properties

	<div>Output Type</div> <div>Buzzer - Beep</div>	
	<div>Buzzer On Duration (ms)</div> <div>1000</div>	<div>Buzzer Off Duration (ms)</div> <div>1000</div>

This output beeps the buzzer with the duty cycle specified. The minimum values for the on and off duration are 200ms.

LED - Output

Output Properties



Output Type

Led - Output

Led

Led 1

Value

On

This output can be used to turn an LED on or off. There are 3 LEDs that can be selected, corresponding to the LEDs on the MSP430G2553. As such, LEDs 1 and 2 are single colour, but a colour - *Red, Green, Blue, Purple, Yellow, White* - can be chosen when LED 3 is selected.

Output Properties



Output Type

Led - Output

Led

Led 3

Led Colour

Red

Value

On

This choice of colour also applies to the other output types involving LED 3.

LED - Blink

Output Properties



Output Type

Led - Blink

Led

Led 1

Period (ms)

1000

This output blinks the LED with the specified period.

LED - Breathe

Output Properties



Output Type

Led - Breathe



Led

Led 1



Period (ms)

2000

This output makes the LED 'breathe' by smoothly changing the brightness from zero to full brightness and back again, and then repeats. The period of this breathing can be configured.

LED - Fade

Output Properties



Output Type

Led - Fade



Led

Led 1



Period (ms)

2000

This output fades the LED off over a given period of time.

LED - Cycle

Output Properties



Output Type

Led - Cycle



Direction

Forwards - RGBPYW



Period (ms)

2000

This output applies only to LED 3 as it cycles through all possible colours. The direction of the cycle can be chosen along with the period - the total amount of time it will take to cycle through all colours

Web Hooks

Output Properties



Output Type

Webhook

Url

{any_public_url}

☐ Forward Message 

This output sends an HTTP post request to the specified URL. The `Forward Message` option allows for the message body of the input on this rule to be forwarded to the specified URL. For example, if the input was also a web hook that posted some payload in the body, this will be retained and posted to the specified URL. If it was a button press as the input, the MQTT JSON message also gets forwarded. For more information for developers and using web hooks for interoperability please see [here](#).

Outputs



Webhook

Url: https://api.huli.life

Forward Message: False