Sheffield Hallam University
Department of Engineering
BEng (Hons) Computer Systems Engineering

Activity ID	**Activity Title**	**Laboratory Room No.**	**Level**
Lab 103	Developing IoT applications with MQTT	4302	6

Semester	**Duration [hrs]**	**Group Size**	**Max Total Students**	**Date of approval/ review**	**Lead Academic**
1	8	2	25	09-20	Alex Shenfield

Equipment (per student/group)

Number	**Item**
1	Arduino kit
1	Ethernet shield

Learning Outcomes

	Learning Outcome
1	Understand the benefits, challenges and pitfalls of developing internet enabled embedded systems.
3	Implement multi-component distributed embedded systems using a flexible network architecture.

Developing IoT applications with MQTT

Introduction

Computing is about more than the PC on your desktop! Embedded devices are everywhere – from wireless telecommunications infrastructure points to electronic point of sale terminals. One definition of an embedded system is:

> "An embedded system is a computer system designed to perform one or a few dedicated functions often with real-time computing constraints."
>
> (http://en.wikipedia.org/wiki/Embedded_system)

In this series of labs you are going to be introduced to the open source Arduino platform – a cheap, simple to program, well documented prototyping platform for designing electronic systems. The heart of this prototyping system is the Arduino microcontroller board itself which is based on the commonly used ATMega328 chip.

The purpose of this lab session is to build on the network functionality explored during the last lab session to develop Internet-of-Things applications using the popular MQTT protocol.

Bibliography

There are no essential bibliographic resources for this laboratory session aside from this tutorial sheet. However the following websites and tutorials may be of help (especially if you haven't done much electronics previously or your digital logic and/or programming is a bit rusty):

- http://www.arduino.cc/
- http://www.ladyada.net/learn/arduino/index.html
- http://tronixstuff.wordpress.com/tutorials/

Methodology

Check that you have all the necessary equipment (see Figure 1)!

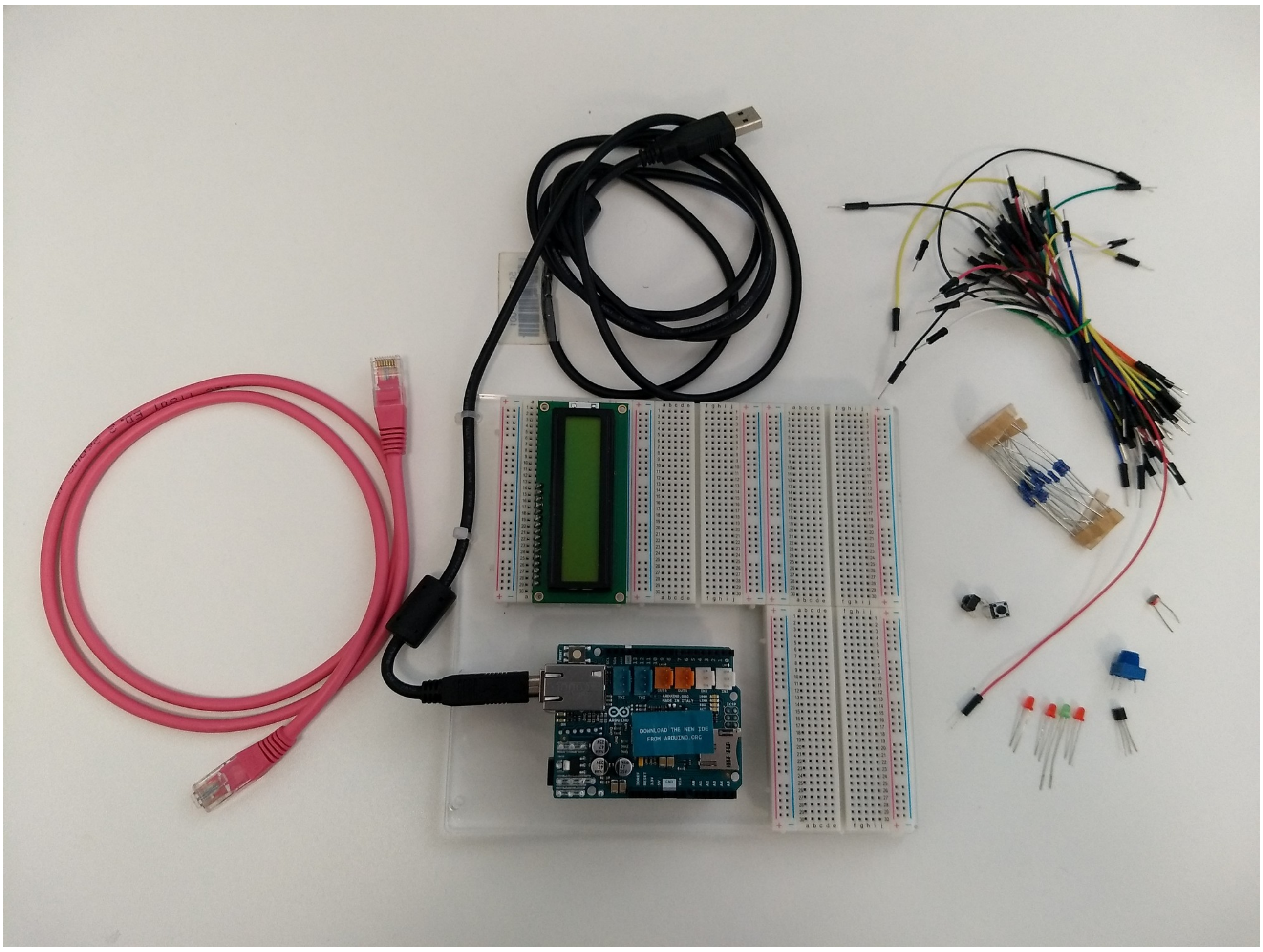


Figure 1 – The necessary equipment for this lab

Task 1

Last week we looked at using the Arduino Uno and Ethernet shield with the HTTP protocol – creating both a simple web server and RESTful webservices using HTTP's **request – response** paradigm. However, as we have discussed in the lectures, even the relatively light-weight REST web service architecture has limitations when it comes to embedded networked devices in the Internet-of-Things. For this reason, many Internet-of-Things devices (particularly those with strict resource constraints such as small micro-controllers) avoid HTTP and use alternative protocols such as CoAP and MQTT.

The official MQTT 3.1.1 specification[1] says:

> "MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium."

One of the key benefits in comparison to HTTP is that MQTT uses 10s of bytes in protocol headers, whereas HTTP can use 100s – 1000s of bytes for headers. Stanford-Clark and Nipper (1999) specified the following goals for MQTT:

- Simple to implement
- Provide a Quality of Service Data Delivery
- Lightweight and Bandwidth Efficient
- Data Agnostic
- Continuous Session Awareness

Another benefit of MQTT is the **publish – subscribe** model which reduces network traffic by allowing a server to push updates to subscribed clients when there is new information of interest available. Figure 2 illustrates this interaction.

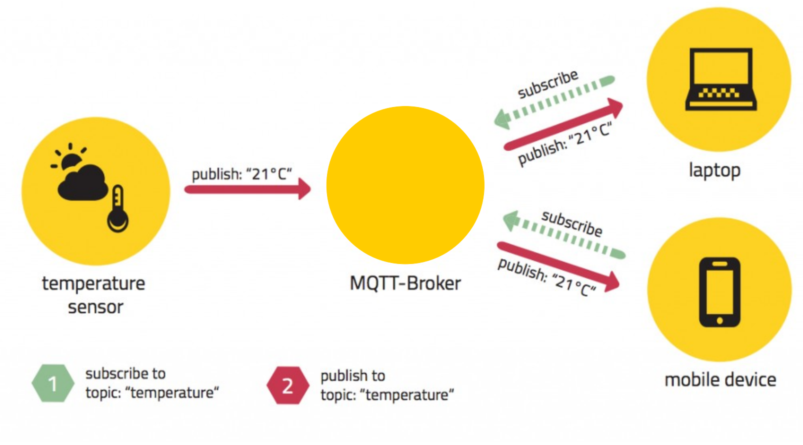


Figure 2 – MQTT's publish – subscribe model

1 http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

In this task we are going to write a simple Arduino application capable of posting a status message to a remote MQTT broker and receiving status updates back. We are going to use the Adafruit IO data analysis platform to send and receive our data – it is a user friendly way of developing web interfaces to visualise the data coming from Internet-of-Things devices.

First you will need to head over to https://io.adafruit.com/ and sign up for an Adafruit account. You can then join Adafruit IO and log in with your account. After logging in you will see a welcome dashboard (as in Figure 3).

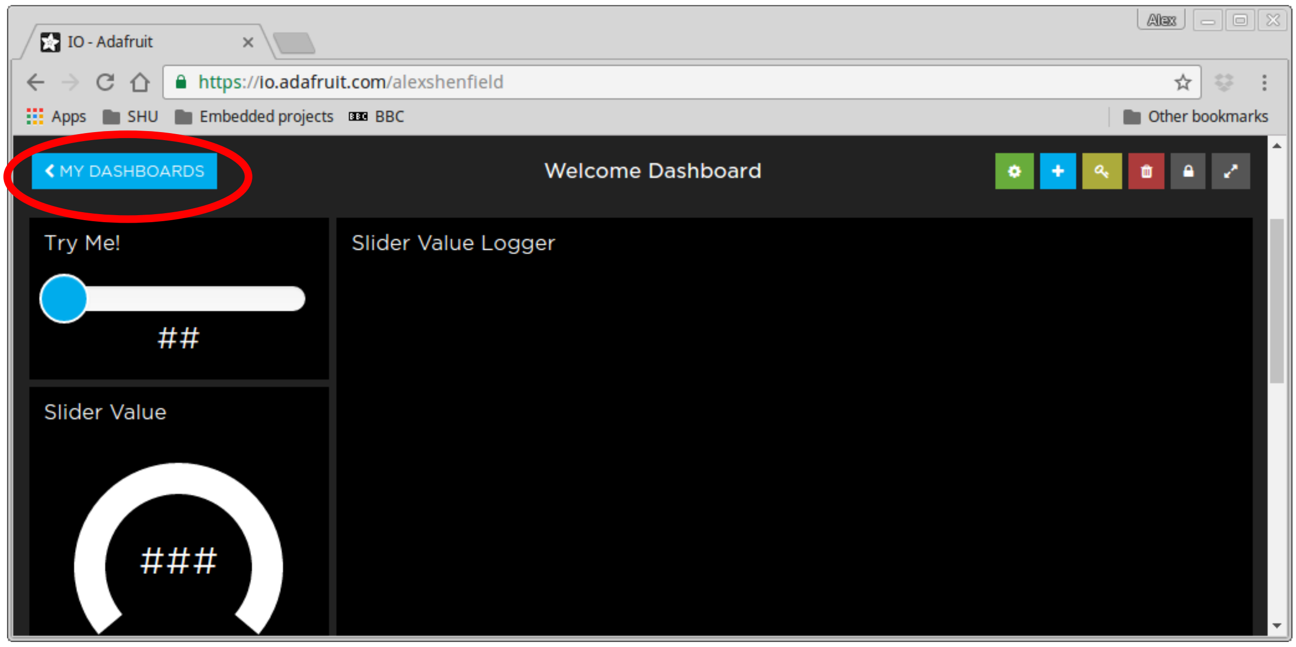
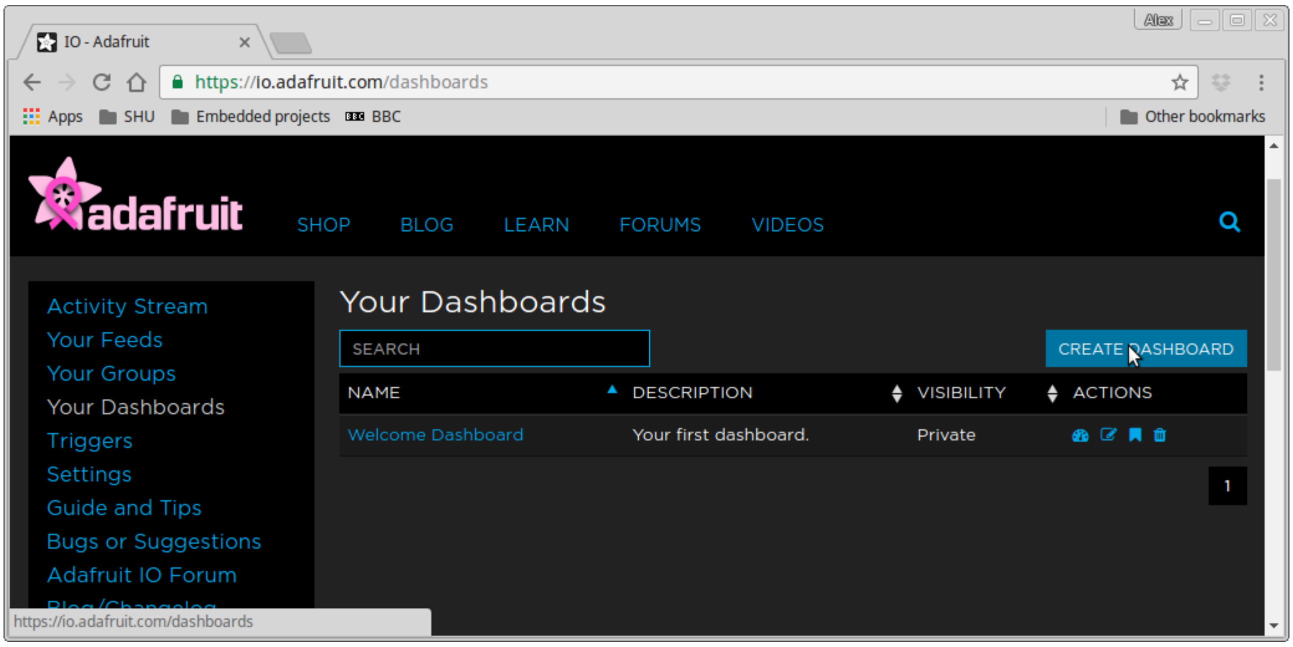


Figure 3 – The Adafruit IO Welcome Dashboard

This welcome dashboard demonstrates some of the controls and GUI elements available to us using the Adafruit IO platform. However, we will create some new feeds and a new dashboard for receiving the data in this task. For more details on creating feeds and dashboards, see the getting started guide available at https://learn.adafruit.com/adafruit-io/getting-started

Note, the free version of Adafruit.io has some limitations in terms of the number of dashboards and feeds (i.e. MQTT topics) we can use. These should be fine for the work we are doing this semester – though it is not very hard to create your own MQTT broker and dashboard if needed.

Go back to "My Dashboards" (highlighted in Figure 2, above) and select "Create Dashboard" as shown in Figure 4.



NAME	DESCRIPTION	VISIBILITY	ACTIONS
Welcome Dashboard	Your first dashboard.	Private	

Figure 4 – Creating a new dashboard

Give the new dashboard a sensible name (I have chosen to call mine "My first IoT application") and then add a new block to it (see Figure 5).

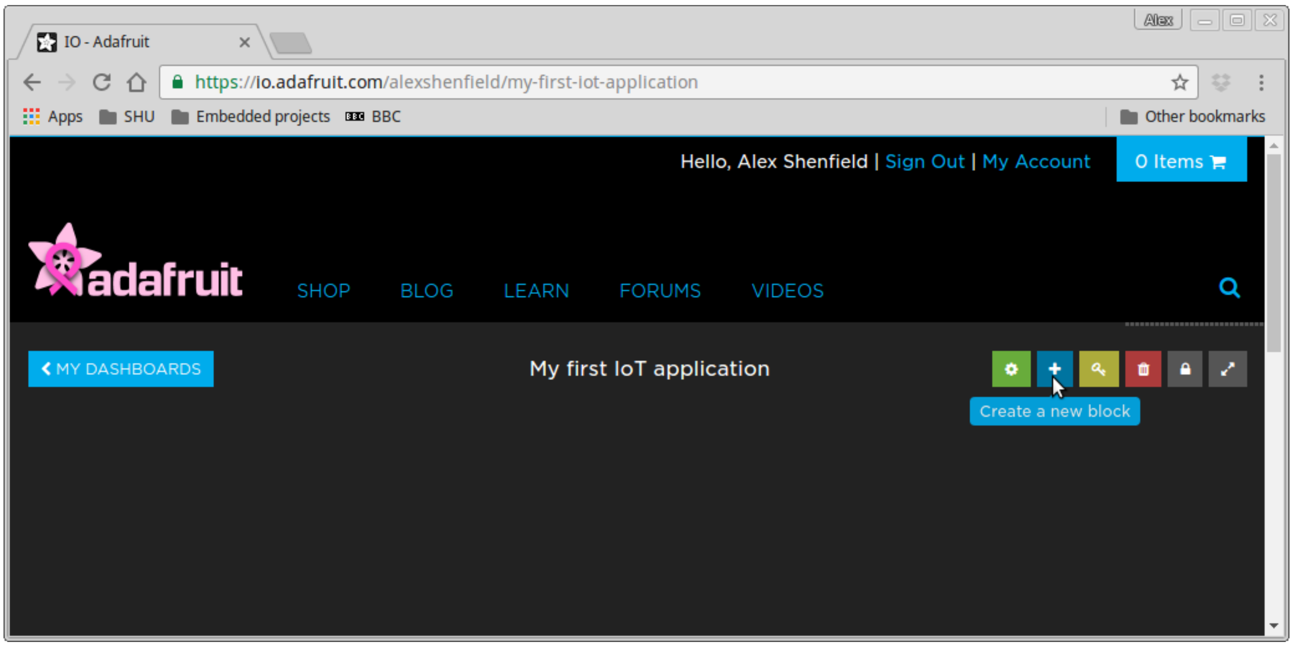
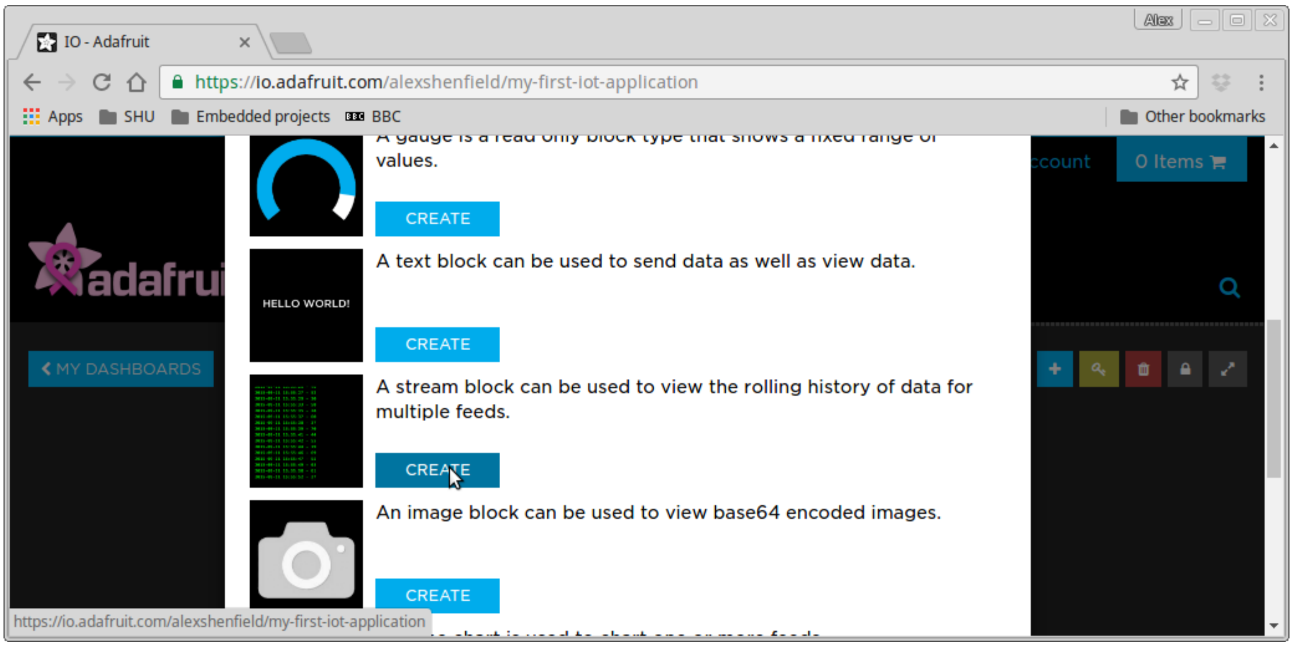


Figure 5 – A blank dashboard

We want to create a new stream block that we can use for displaying text sent to a feed (or multiple feeds). This will allow us to display our status messages. See Figure 6.


IO - Adafruit
https://io.adafruit.com/alexshenfield/my-first-iot-application
Apps
SHU
Embedded projects
BBC
Other bookmarks
Alex
adafruit
MY DASHBOARDS
0 Items
CREATE
HELLO WORLD!
CREATE
CREATE
CREATE
https://io.adafruit.com/alexshenfield/my-first-iot-application

A gauge is a read only block type that shows a fixed range of values.

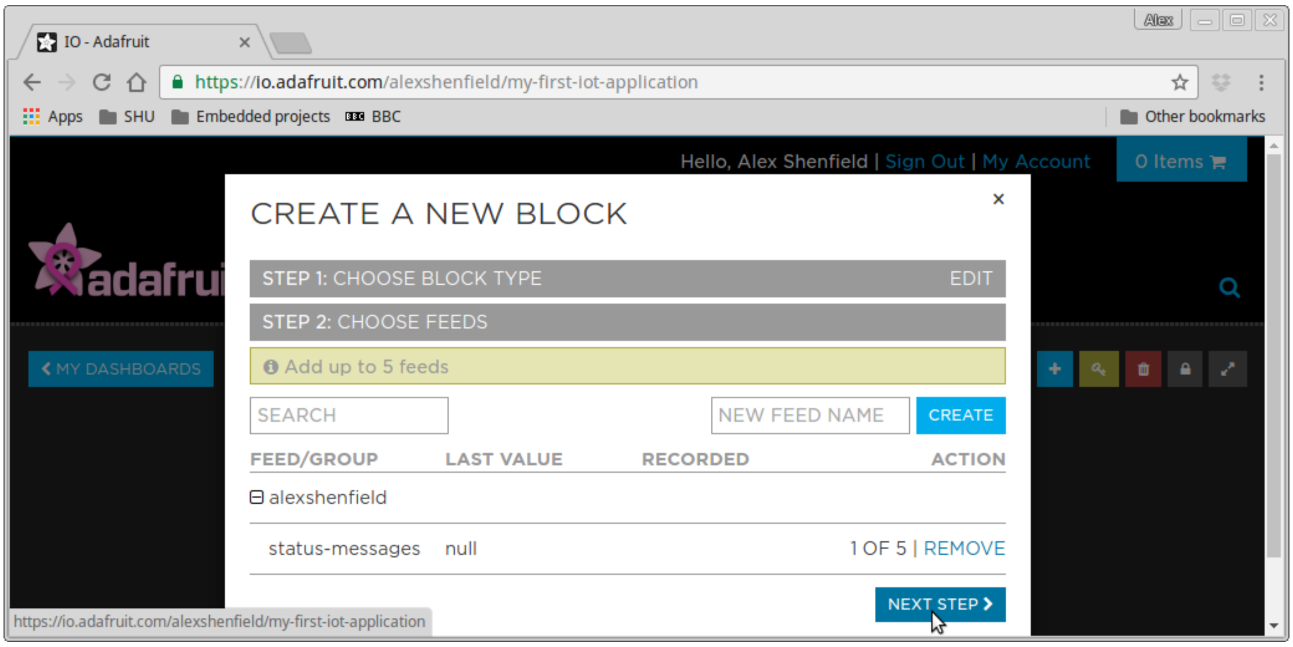
A text block can be used to send data as well as view data.

A stream block can be used to view the rolling history of data for multiple feeds.

An image block can be used to view base64 encoded images.

Figure 6 – Adding elements to our dashboard

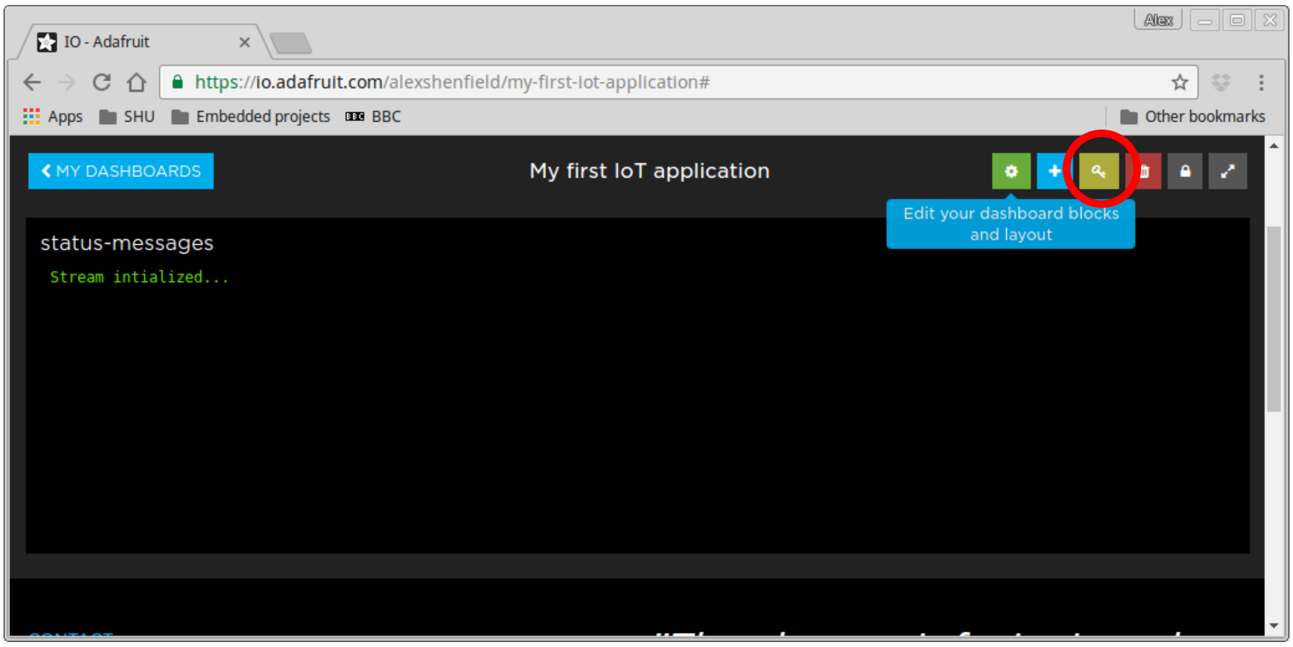
In the wizard that follows, we want to set the stream block up to be attached to a "status-messages" feed. To do this you will need to type the feed name into the appropriate box and click the "Create" button. You can then add the feed to the stream block as in Figure 7.


IO - Adafruit
https://io.adafruit.com/alexshenfield/my-first-iot-application
Apps
SHU
Embedded projects
BBC
Other bookmarks
Alex
Hello, Alex Shenfield | Sign Out | My Account
0 Items
adafruit
MY DASHBOARDS
CREATE A NEW BLOCK
STEP 1: CHOOSE BLOCK TYPE
EDIT
STEP 2: CHOOSE FEEDS
Add up to 5 feeds
SEARCH
NEW FEED NAME
CREATE
NEXT STEP
https://io.adafruit.com/alexshenfield/my-first-iot-application

FEED/GROUP	LAST VALUE	RECORDED	ACTION
⊟ alexshenfield			
status-messages	null		1 OF 5 | REMOVE

Figure 7 – Attaching a feed / topic to our stream block

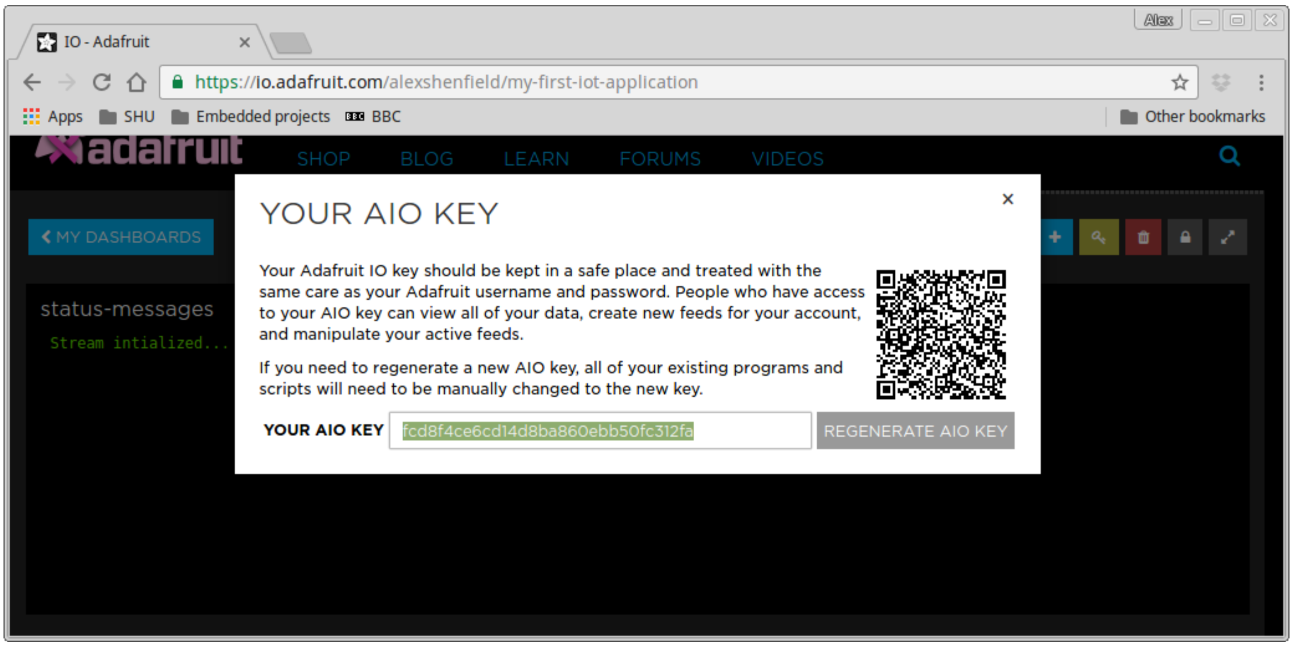
Once you have done this you can alter the settings for the stream block – I like the defaults (who doesn't like green text on a black background?!) but I also want the feed to show timestamps. Once you have created the block you can resize it appropriately – see Figure 8.



```
Stream intialized...
```

Figure 8 – Our simple dashboard

The last thing we need to do is to get the Adafruit IO key that serves as a password to access your feeds. To do this, click on the key icon (shown highlighted in Figure 7, above) and copy the resulting key as shown in Figure 9.



YOUR AIO KEY

Your Adafruit IO key should be kept in a safe place and treated with the same care as your Adafruit username and password. People who have access to your AIO key can view all of your data, create new feeds for your account, and manipulate your active feeds.

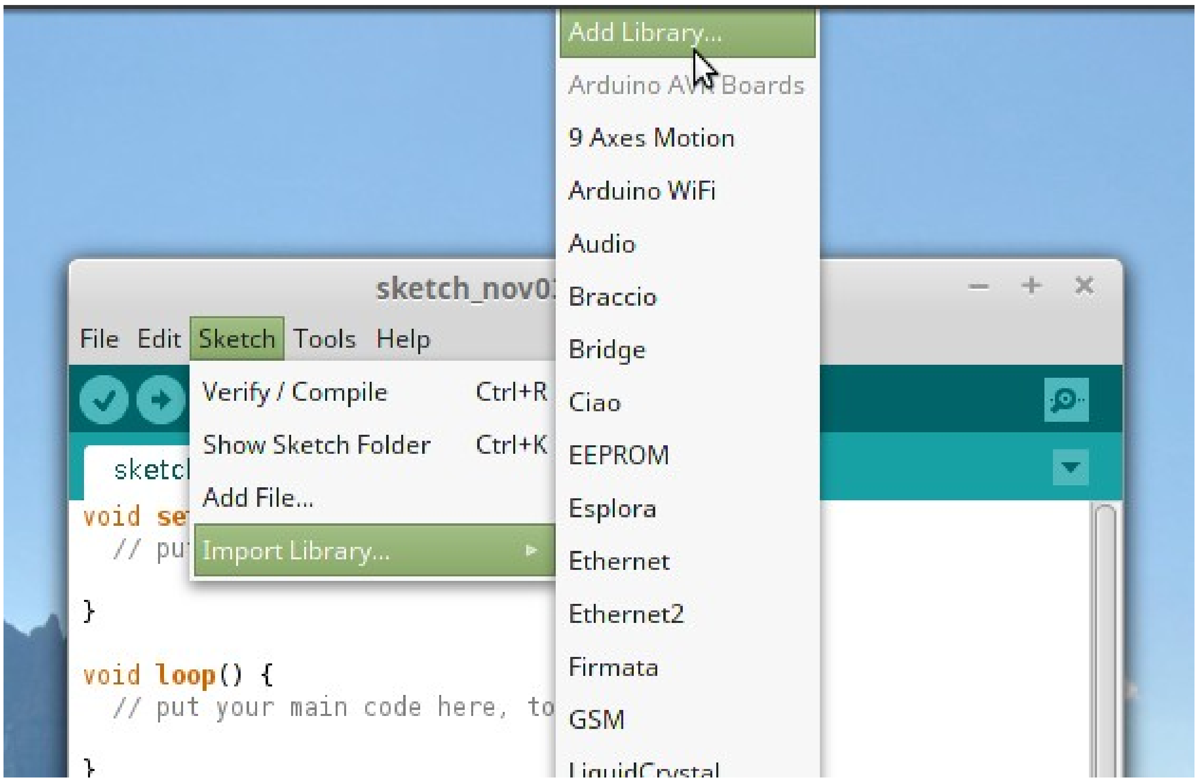
If you need to regenerate a new AIO key, all of your existing programs and scripts will need to be manually changed to the new key.

YOUR AIO KEY fcd8f4ce6cd14d8ba860ebb50fc312fa

Figure 9 – Retrieving the AIO key

So we have now set up a simple dashboard in Adafruit IO!

However, before we write our Arduino application we will need to download and install the PubSubClient library for Arduino. There is a link to the library zip file on blackboard[2], and this can be installed via the Arduino IDE library manager (see Figure 10).



```
void se
  // pu

}

void loop() {
  // put your main code here, to

}
```

Figure 10 – Installing a library

2 With the latest version available from github - https://github.com/knolleary/pubsubclient

As with lab 2, you will also need to make a note of the MAC address on the bottom of your Ethernet shield (see Figure 11).

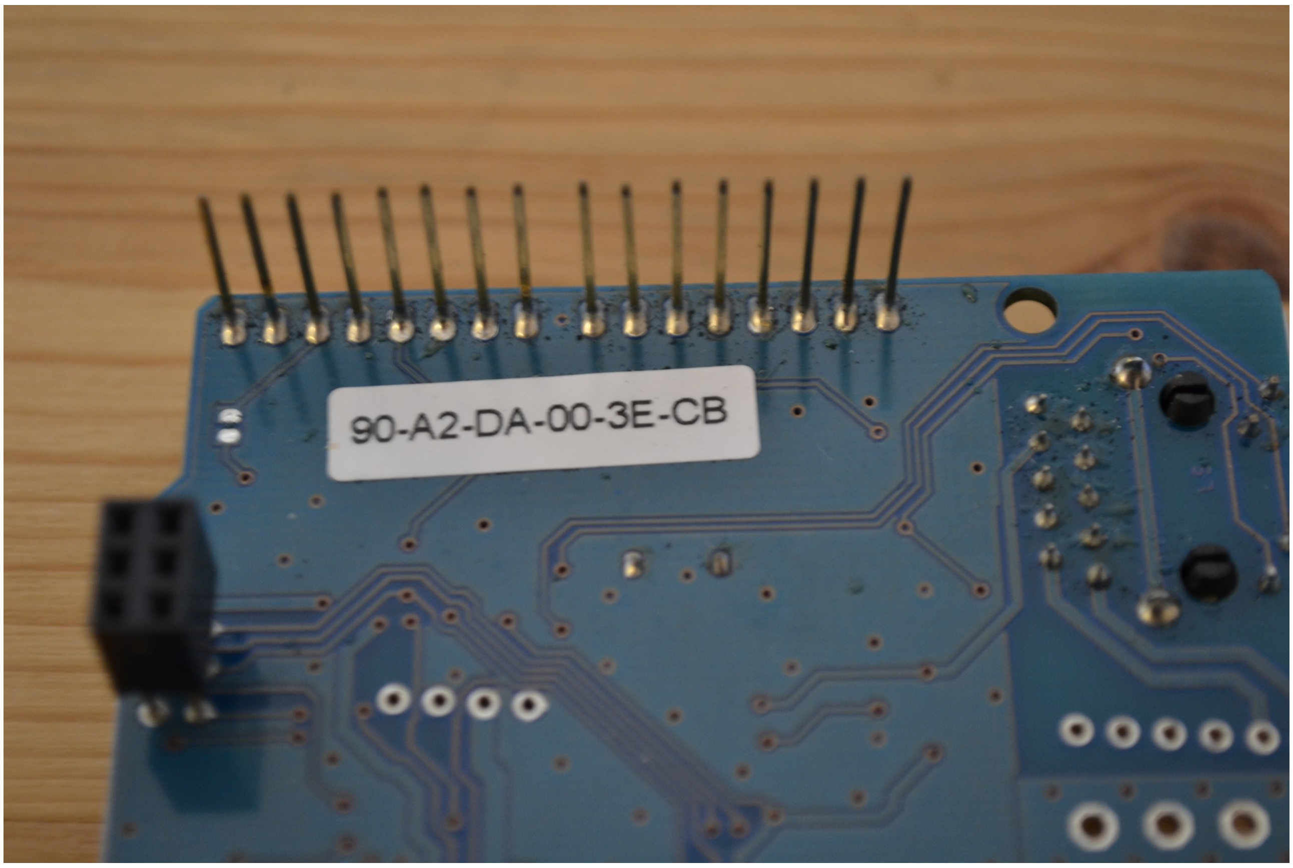


Figure 11 – Ethernet shield MAC address

The MAC address of your Ethernet shield is:

You will need to change this in the code for your MQTT client (otherwise we might not be able to get a DHCP address).

You will also need to connect the Arduino Uno and Ethernet shield together and connect them to the network port (see Figure 12).

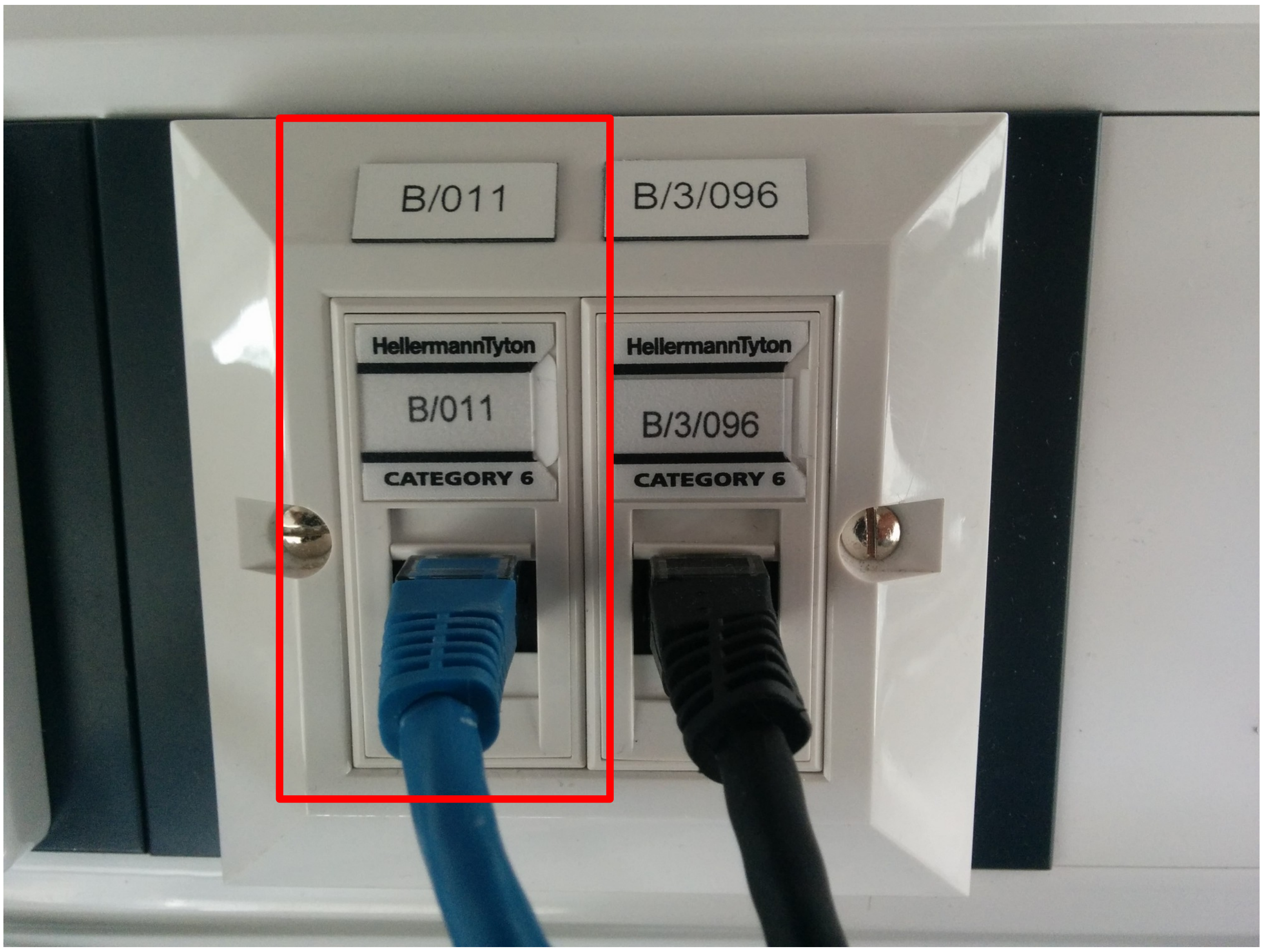


Figure 12 – The networks ports

Don't forget to connect up the patch panel to the switch!

We can now proceed to write the code for our Arduino application! The point of this application is to make a connection to our MQTT broker, post a status message to the remote MQTT broker, and receive status updates back. If the connection is broken for any reason then the Arduino MQTT client will keep trying to reconnect and then send a status message when it does. Code listing 1 shows this code.

Note – Adafruit.io limits the amount of messages you can send to **30** per minute. If you try to send more than this you will be throttled initially and repeated offenders will potentially be blocked from using the service! It is therefore very important that the timing code works properly! Pay particular attention to the highlighted code in the main loop.

As an early warning of this – you can see that we are subscribing to the user's "throttle" topic. This should let us know if we are exceeding the rate limit for messages in adafruit.io.

Code listing 1:

```
/*
 * mqtt_message_test.ino
 *
 * basic mqtt sketch to send a simple status message to an adafruit.io
 * feed and then subscribe to all the feeds. it then periodically sends
 * status update messages (once every 10 seconds) containing the current
 * uptime of the system.
 *
 * author:  alex shenfield
 * date:    10/09/2018
 */

// include necessary libraries
#include <SPI.h>
#include <Ethernet2.h>
#include <PubSubClient.h>

// ETHERNET CONNECTION DECLARATIONS

// mac address
byte mac[] = {0x90, 0xA2, 0xDA, 0x0F, 0xF5, 0xE7};

// the ethernet shields and ethernet libraries support DHCP as long as
// you are plugged into a device with a DHCP server - it is always a
// good idea to have a static address to fall back on though!
IPAddress ip(192, 168, 0, 11);
IPAddress gateway(192, 168, 0, 254);
IPAddress subnet(255, 255, 255, 0);

// MQTT DECLARATIONS

// mqtt server details
const char userid[] = "<your adafruit.io username>";
const char apikey[] = "<your adafruit.io api key>";
const char server[] = "io.adafruit.com";
const long port     = 1883;

// get an ethernet client and pass it to the mqtt client
EthernetClient ethernet_client;
PubSubClient client(ethernet_client);

// timing variables - note: adafruit.io allows a maximum of 30 messages
// per minute - any more than this and your account will be throttled
// and then blocked!
long previous_time = 0;
long connection_interval = 10000;
```

```
// CODE

// set up code
void setup()
{
  // set up serial comms for debugging and display of DHCP allocated ip
  // address
  Serial.begin(9600);
  Serial.println("starting mqtt client on arduino ...");

  // mqtt server and subscription callback
  client.setServer(server, port);
  client.setCallback(callback);

  // start the ethernet shield comms - initially try to get a DHCP ip
  // address
  if (Ethernet.begin(mac) == 0)
  {
    // if DHCP fails, allocate a static ip address
    Serial.println("failed to configure ethernet using DHCP");
    Ethernet.begin(mac, ip);
  }

  // allow the hardware to sort itself out
  delay(1500);

  // print the ip address to the serial monitor
  Serial.print("mqtt client is at: ");
  Serial.println(Ethernet.localIP());
}

// main loop
void loop()
{
  // if the client isn't connected then try to reconnect
  if (!client.connected())
  {
    reconnect();
  }
  else
  {
    // handle subscriptions to topics (i.e. incoming messages)
    client.loop();

    // periodically publish a message to a feed (note, this uses the
    // same non blocking timing mechanism as in blink_without_delay.ino
    // from lab 1)
    unsigned long current_time = millis();
    if(current_time - previous_time > connection_interval)
    {
      previous_time = current_time;
      send_message();
    }
  }
}
```

```
// MQTT FUNCTIONS

// mqtt message received callback
void callback(char* topic, byte* payload, unsigned int length)
{
  // print the feed the message comes from
  Serial.print("message arrived [");
  Serial.print(topic);
  Serial.print("] ");

  // print the message
  for (int i = 0; i < length; i++)
  {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}

// reconnect to server
void reconnect()
{
  // loop until we're reconnected
  while (!client.connected())
  {
    Serial.println("attempting mqtt connection...");

    // try to connect to adafruit.io
    if (client.connect("my_arduino", userid, apikey))
    {
      Serial.println("... connected");

      // once connected, publish an announcement and subscribe to all
      // feeds
      client.publish("<your adafruit.io username>/f/status-messages", "we are alive!");
      client.subscribe("<your adafruit.io username>/f/#");
      client.subscribe("<your adafruit.io username>/throttle");
    }
    else
    {
      // print some error status
      Serial.print("connection failed, rc = ");
      Serial.print(client.state());
      Serial.println();
      Serial.println("we will try again in 5 seconds");

      // wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

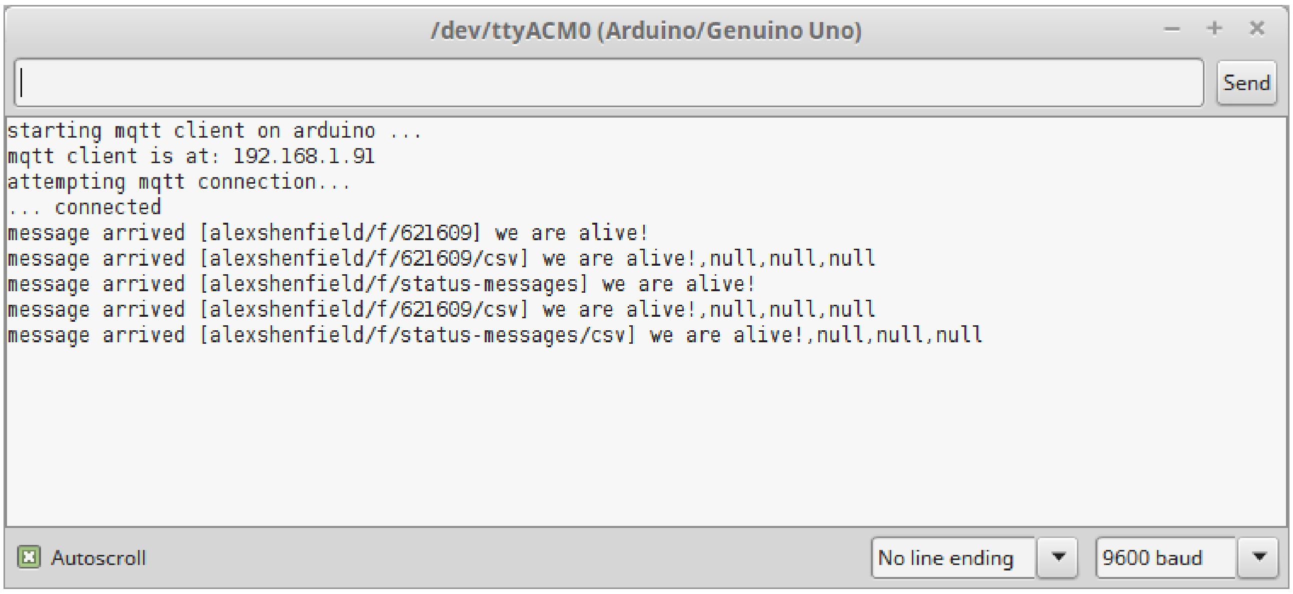
```
// APPLICATION FUNCTIONS

// convert the current uptime to a string and publish to the status-messages
// feed
void send_message()
{
  char time_str[40];
  sprintf(time_str, "up for %lu s", (millis() / 1000));
  client.publish("<your adafruit.io username>/f/status-messages", time_str);
}
```

Enter this code and upload it to your Arduino.

Don't forget:

1. to change the code to the MAC address of your Ethernet shield
2. to use the appropriate fall-back static IP address
3. to use your own user name and API key!

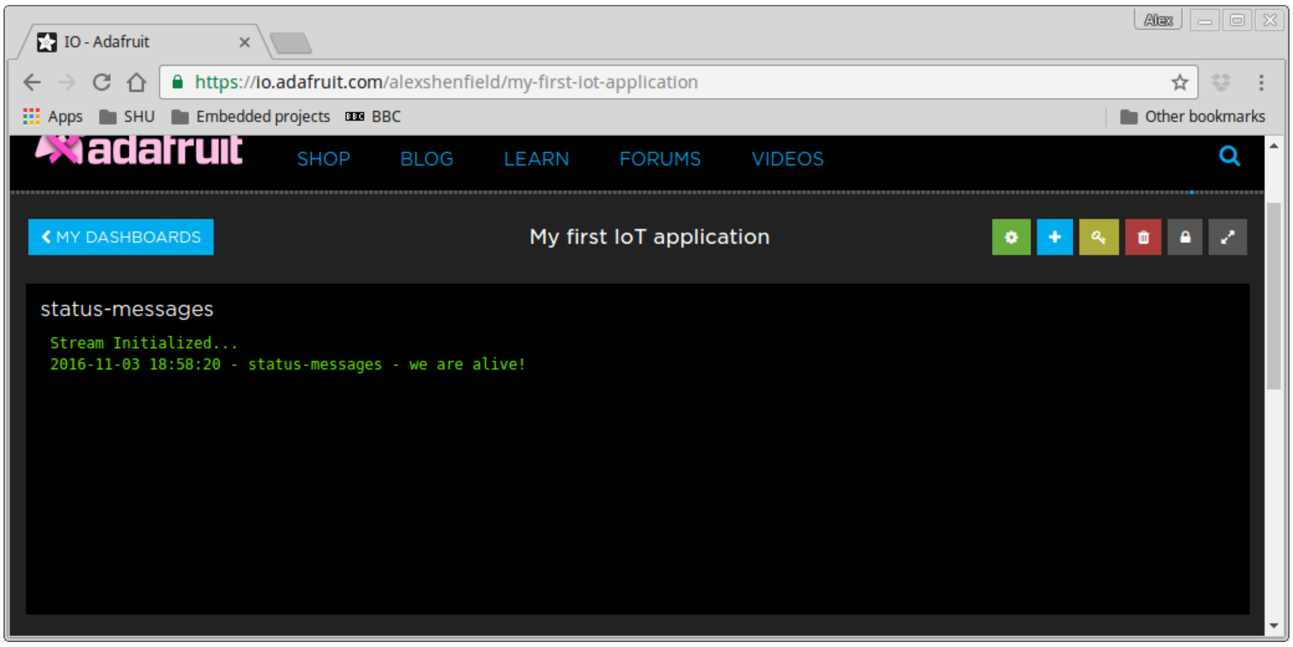
Once you have uploaded this code, you should see the following output in the serial monitor window (see Figure 13).



```
starting mqtt client on arduino ...
mqtt client is at: 192.168.1.91
attempting mqtt connection...
... connected
message arrived [alexshenfield/f/621609] we are alive!
message arrived [alexshenfield/f/621609/csv] we are alive!,null,null,null
message arrived [alexshenfield/f/status-messages] we are alive!
message arrived [alexshenfield/f/621609/csv] we are alive!,null,null,null
message arrived [alexshenfield/f/status-messages/csv] we are alive!,null,null,null
```

Figure 13 – Serial monitor output

And this should update the stream view on Adafruit IO (as in Figure 14).



```
Stream Initialized...
2016-11-03 18:58:20 - status-messages - we are alive!
```

Figure 14 – MQTT message sent to Adafruit IO

Exercises

Now you are going to make some modifications to the program to add functionality:

1. At the moment you are subscribed to every topic in your feed (which is why you see a lot of repetition when a status message is updated). Alter your program to only subscribe to the status-message topic.

2. Add some additional controls (e.g. sliders and / or buttons) to your Adafruit IO dashboard and subscribe your Arduino application to those feeds / topics.

3. Adapt the method that periodically publishes status messages to send random integer values to the Adafruit IO service. Tie these in to a specific topic (e.g. `"<your aio username>/f/random_numbers"`).

<u>Task 2</u>

We have now introduced the basics of publishing and subscribing to feeds / topics using MQTT on the Arduino. In this task we are going to create a system that monitors ambient light levels within a room and reports them back to the MQTT broker. We will then visualise these light levels using the graph block functionality of the Adafruit IO platform.

Build the circuit shown in Figure 15.

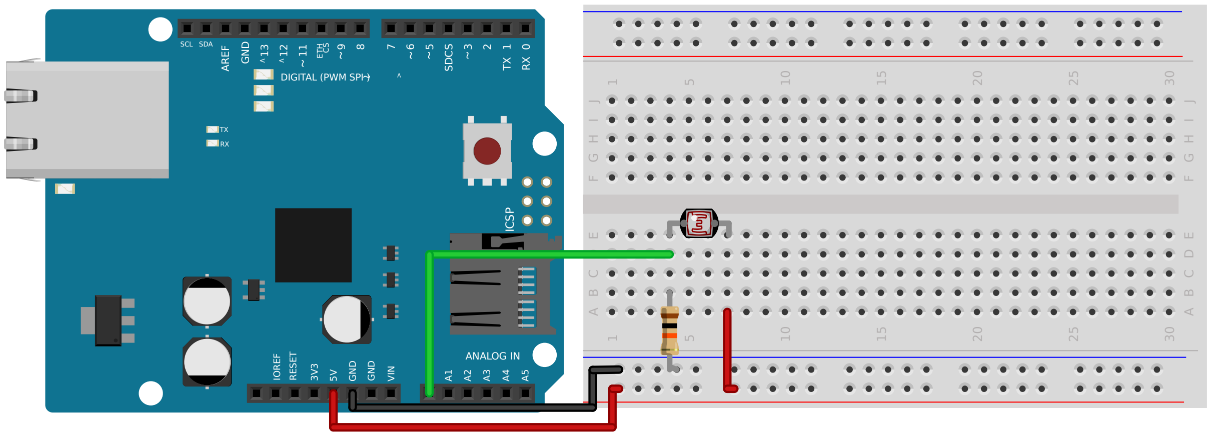


Figure 15 – Light sensor circuit

Code listing 2 provides the full code for this MQTT based light monitoring system. This program reuses a lot of the code from the previous task in setting up the MQTT connection to Adafruit IO and handling message received callbacks.

Don't forget to change the code to your own username and AIO key!

Code listing 2:

```
/*
 * iot_data_node.ino
 *
 * mqtt sketch to send ambient light levels to the adafruit io data
 * analytics platform for visualisation
 *
 * author:  alex shenfield
 * date:    10/09/2018
 */

// include necessary libraries
#include <SPI.h>
#include <Ethernet2.h>
#include <PubSubClient.h>

// ETHERNET CONNECTION DECLARATIONS

// mac address
byte mac[] = {0x90, 0xA2, 0xDA, 0x0F, 0xF5, 0xE7};

// the ethernet shields and ethernet libraries support DHCP as long as
// you are plugged into a device with a DHCP server - it is always a
// good idea to have a static address to fall back on though!
IPAddress ip(192, 168, 0, 11);
IPAddress gateway(192, 168, 0, 254);
IPAddress subnet(255, 255, 255, 0);

// MQTT DECLARATIONS

// mqtt server details
const char userid[] = "<your adafruit.io username>";
const char apikey[] = "<your adafruit.io api key>";
const char server[] = "io.adafruit.com";
const long port     = 1883;

// get an ethernet client and pass it to the mqtt client
EthernetClient ethernet_client;
PubSubClient client(ethernet_client);

// timing variables - note: adafruit.io allows a maximum of 30 messages
// per minute - any more than this and your account will be throttled
// and then blocked!
long previous_time = 0;
long connection_interval = 10000;
```

```
// CODE

// set up code
void setup()
{
  // set up serial comms for debugging and display of DHCP allocated ip
  // address
  Serial.begin(9600);
  Serial.println("starting mqtt client on arduino ...");

  // initialise pins
  pinMode(A0, INPUT);

  // mqtt server and subscription callback
  client.setServer(server, port);
  client.setCallback(callback);

  // start the ethernet shield comms - initially try to get a DHCP ip
  // address
  if (Ethernet.begin(mac) == 0)
  {
    // if DHCP fails, allocate a static ip address
    Serial.println("failed to configure ethernet using DHCP");
    Ethernet.begin(mac, ip);
  }

  // allow the hardware to sort itself out
  delay(1500);

  // print the ip address to the serial monitor
  Serial.print("mqtt client is at: ");
  Serial.println(Ethernet.localIP());
}

// main loop
void loop()
{
  // if the client isn't connected then try to reconnect
  if (!client.connected())
  {
    reconnect();
  }
  else
  {
    // handle subscriptions to topics (i.e. incoming messages)
    client.loop();

    // periodically send light levels to adafruit.io (note, this uses the
    // same non blocking timing mechanism as in blink_without_delay.ino
    // from lab 1)
    unsigned long current_time = millis();
    if(current_time - previous_time > connection_interval)
    {
      previous_time = current_time;
      send_light();
    }
  }
}
```

```
// MQTT FUNCTIONS

// mqtt message received callback
void callback(char* topic, byte* payload, unsigned int length)
{
  // print the feed the message comes from
  Serial.print("message arrived [");
  Serial.print(topic);
  Serial.print("] ");

  // print the message
  for (int i = 0; i < length; i++)
  {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}

// reconnect to server
void reconnect()
{
  // loop until we're reconnected
  while (!client.connected())
  {
    Serial.println("attempting mqtt connection...");

    // try to connect to adafruit.io
    if (client.connect("my_arduino", userid, apikey))
    {
      Serial.println("... connected");

      // once connected, publish an announcement and subscribe to all
      // feeds
      client.publish("<your adafruit.io username>/f/status-messages", "we are alive!");
      client.subscribe("<your adafruit.io username>/f/status-messages");
      client.subscribe("<your adafruit.io username>/f/light");
      client.subscribe("<your adafruit.io username>/throttle");
    }
    else
    {
      // print some error status
      Serial.print("connection failed, rc = ");
      Serial.print(client.state());
      Serial.println();
      Serial.println("we will try again in 5 seconds");

      // wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

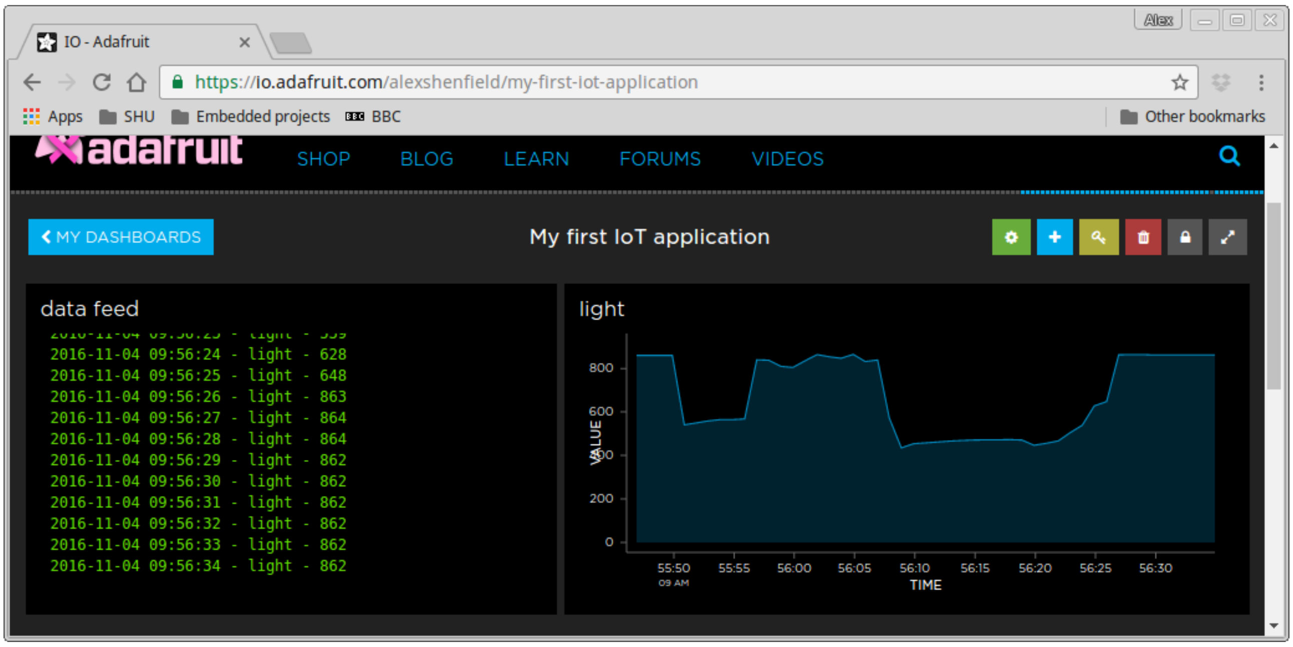
```
// send the light level value
void send_light()
{
  // if the client is already connected then send data
  if (client.connected())
  {
    // get the light level
    int light = analogRead(A0);

    // convert the temperature to a string of characters
    char light_level[5];
    sprintf(light_level, "%i", light);
    client.publish("<your adafruit.io username>/f/light", light_level);
  }
}
```

Enter this code, but **before you upload it** you should configure your Adafruit IO dashboard to get this data and process it!

To do this we need to add a graph block to our Adafruit IO dashboard and link it to the light level topic. I have also edited the stream block so that the light level messages appear there too (this helps us with debugging and making sure messages are getting through, etc.).

We can then upload the code to our Arduino and start receiving data on our dashboard!

My dashboard now looks like Figure 16.



```
2016-11-04 09:56:24 - light - 628
2016-11-04 09:56:25 - light - 648
2016-11-04 09:56:26 - light - 863
2016-11-04 09:56:27 - light - 864
2016-11-04 09:56:28 - light - 864
2016-11-04 09:56:29 - light - 862
2016-11-04 09:56:30 - light - 862
2016-11-04 09:56:31 - light - 862
2016-11-04 09:56:32 - light - 862
2016-11-04 09:56:33 - light - 862
2016-11-04 09:56:34 - light - 862
```

Figure 16 – My Adafruit IO dashboard for light level logging

Exercises

Now you are going to make some modifications to the program to add functionality:

1. Provide a calibration routine for the light sensor during the setup() function that means that the light value is remapped as a percentage before sending to Adafruit IO.

2. Add a potentiometer to your system. Read the data from that and send to the Adafruit IO platform for visualisation on the same graph as the light readings.

3. Add an automatic monitoring routine in your Arduino code that turns on an LED when the light level goes below a certain set point.

Task 3

We are now going to build our first complete Internet-of-Things application – an IoT enabled light switch! This application will have 3 parts to it: the hardware, the Arduino application, and the web interface. The hardware is very simple – it is just a button and an LED. This circuit is shown in Figure 17, below.

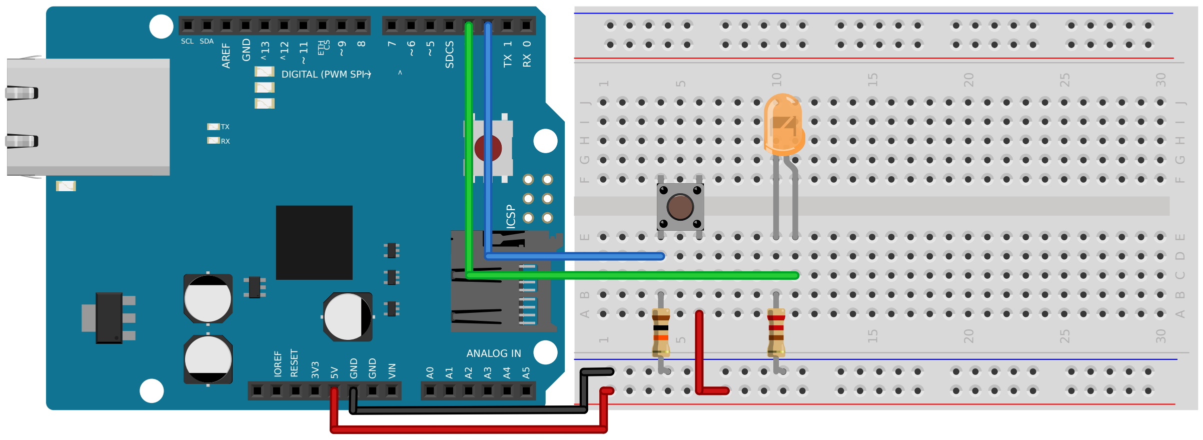


Figure 17 – A light switch

The code for the Arduino application is the most complicated bit of this system – so we will develop it in stages. The first stage is to develop and test the debouncing code for toggling our led when the button is pressed. Remember, when a button is pressed, the mechanical contacts in the switch may "bounce" around – potentially triggering several button presses in quick succession. To avoid this, we can use debouncing to remove these effects. The simplest way of debouncing is to add **delay()** statements into our program – however, this will interfere with the MQTT client functions dealing with subscription to topics. To overcome this, we will have to develop a debouncing routine that doesn't rely on delays.

The full debouncing application is shown in code listing 3, below.

Code listing 3:

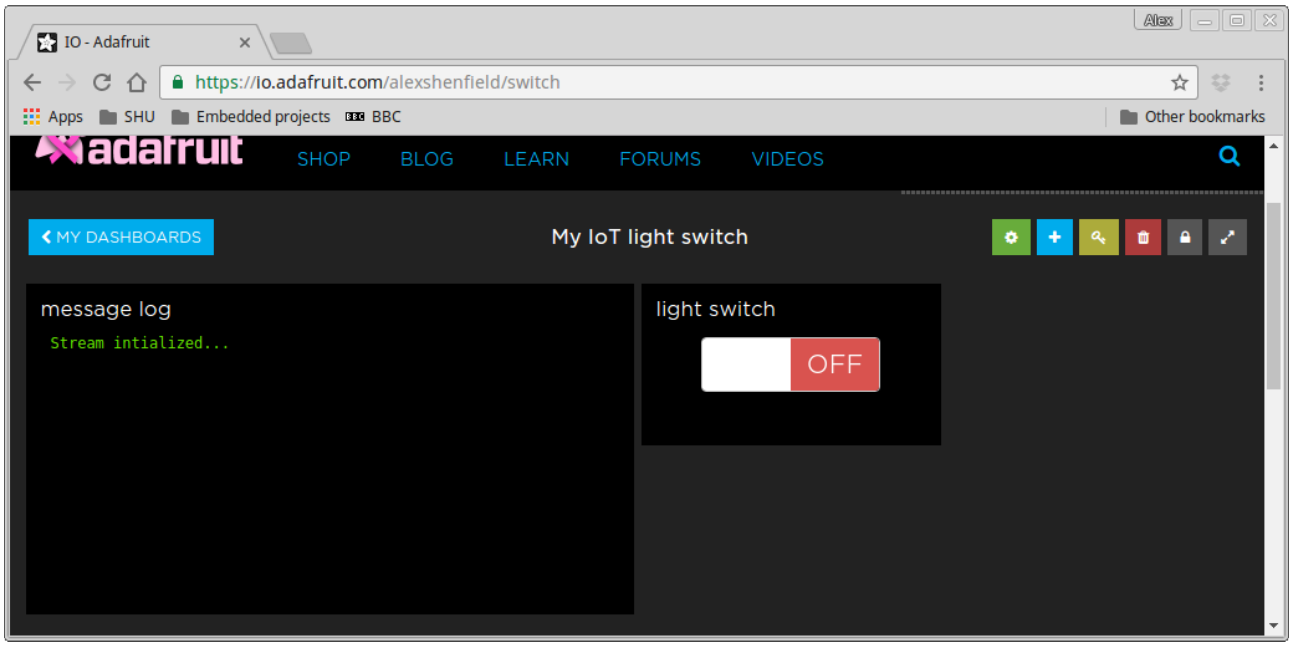
```
/*
 * debouncing.ino
 *
 * debouncing code that doesn't rely on delays
 *
 * author:  alex shenfield
 * date:    27/10/18
 */

// set some debounce variables for our button
int curr_button_state = LOW;
int last_button_state = LOW;

unsigned long last_debounce_time = 0;
unsigned long debounce_delay = 50;

int led_state = LOW;

// set up code
void setup()
{
  // declare appropriate pins
  pinMode(2, INPUT);
  pinMode(3, OUTPUT);
}

// main program loop
void loop()
{
  check_button();
  digitalWrite(3, led_state);
}
```

```
// check button function
void check_button()
{
  // get the state of the button
  int reading = digitalRead(2);

  // if the button state changed then reset the deboucing timer
  if (reading != last_button_state)
  {
    last_debounce_time = millis();
  }

  // if the reading has been there longer than the debounce delay,
  // then it is a valid state
  if ((millis() - last_debounce_time) > debounce_delay)
  {
    // so if the button state has changed
    if (reading != curr_button_state)
    {
      // then update it
      curr_button_state = reading;

      // if the button has been pressed then toggle the led
      if (curr_button_state == HIGH)
      {
        led_state = !led_state;
      }
    }
  }

  // update last button state
  last_button_state = reading;
}
```

Build the circuit in Figure 16, enter the code above, and test that everything works!

As with the previous applications, much of the code that sets up the MQTT connection and connects and subscribes to topics will remain the same. However, as we will now be reading some data from the remote MQTT broker we will have to work on handling data in our MQTT callback – to do this we will have to extract the data from the message payload and then process it. This means that we need to know what data we are sending from our web interface!

The next step is to design our Adafruit IO dashboard for this application. My dashboard is shown in Figure 18 – it simply has a stream for status messages and a toggle switch for the light switch.



```
Stream intialized...
```

Figure 18 – My IoT light switch dashboard

We can develop a simple MQTT Arduino application to connect to these feeds and display the data on the serial monitor so that we can understand exactly what data is being sent from our web interface. This code is given in code listing 4, below.

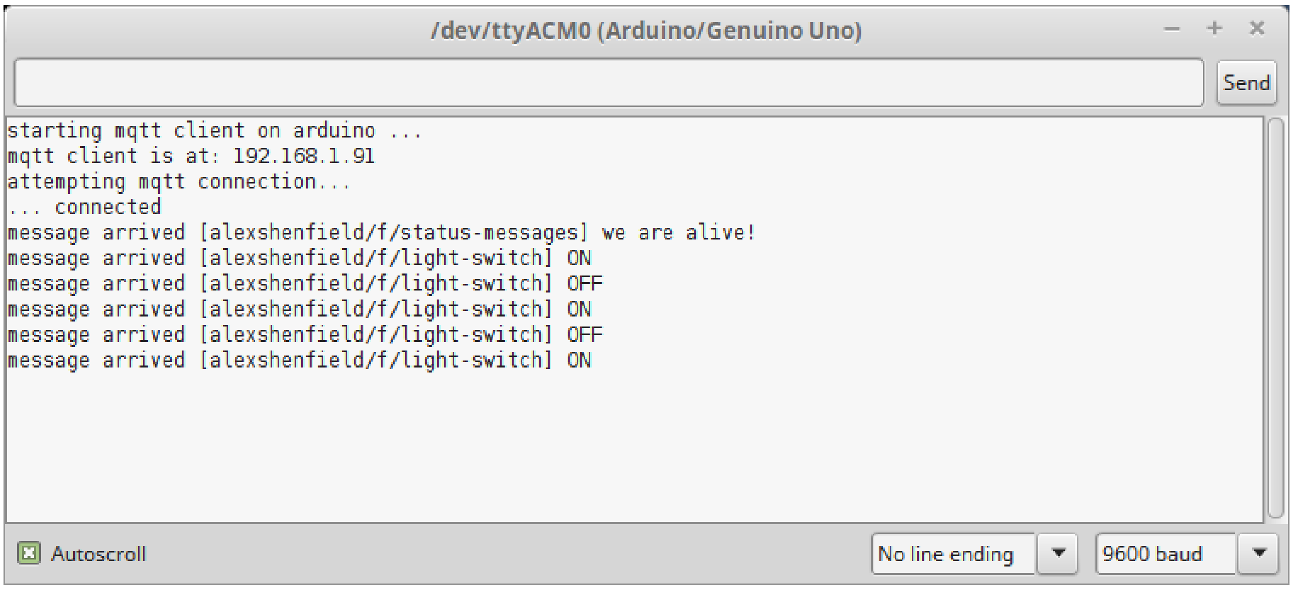
Code listing 4:

```
/*
 * iot_messages.ino
 *
 * a simple mqtt application to parse mqtt messages on subscribed
 * topics
 *
 * author:  alex shenfield
 * date:    10/09/2018
 */

// include necessary libraries
#include <SPI.h>
#include <Ethernet2.h>
#include <PubSubClient.h>

// ETHERNET CONNECTION DECLARATIONS

// mac address
byte mac[] = {0x90, 0xA2, 0xDA, 0x0F, 0xF5, 0xE7};

// the ethernet shields and ethernet libraries support DHCP as long as
// you are plugged into a device with a DHCP server - it is always a
// good idea to have a static address to fall back on though!
IPAddress ip(192, 168, 0, 11);
IPAddress gateway(192, 168, 0, 254);
IPAddress subnet(255, 255, 255, 0);

// MQTT DECLARATIONS

// mqtt server details
const char userid[] = "<your adafruit.io username>";
const char apikey[] = "<your adafruit.io api key>";
const char server[] = "io.adafruit.com";
const long port     = 1883;

// get an ethernet client and pass it to the mqtt client
EthernetClient ethernet_client;
PubSubClient client(ethernet_client);
```

```
// CODE

// set up code
void setup()
{
  // set up serial comms for debugging and display of DHCP allocated ip
  // address
  Serial.begin(9600);
  Serial.println("starting mqtt client on arduino ...");

  // mqtt server and subscription callback
  client.setServer(server, port);
  client.setCallback(callback);

  // start the ethernet shield comms - initially try to get a DHCP ip
  // address
  if (Ethernet.begin(mac) == 0)
  {
    // if DHCP fails, allocate a static ip address
    Serial.println("failed to configure ethernet using DHCP");
    Ethernet.begin(mac, ip);
  }

  // allow the hardware to sort itself out
  delay(1500);

  // print the ip address to the serial monitor
  Serial.print("mqtt client is at: ");
  Serial.println(Ethernet.localIP());
}

// main loop
void loop()
{
  // if the client isn't connected then try to reconnect
  if (!client.connected())
  {
    reconnect();
  }
  else
  {
    // handle subscriptions to topics (i.e. incoming messages)
    client.loop();
  }
}
```

```
// MQTT FUNCTIONS

// mqtt message received callback
void callback(char* topic, byte* payload, unsigned int length)
{
  // get the topic
  String t = String(topic);

  // get the value of the message
  char data[length + 1];
  for (int i = 0; i < length; i++)
  {
    data[i] = payload[i];
  }
  data[length] = '\0';

  // print the message to the serial window
  Serial.print("message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  Serial.println(data);
}

// reconnect to server
void reconnect()
{
  // loop until we're reconnected
  while (!client.connected())
  {
    Serial.println("attempting mqtt connection...");

    // try to connect to adafruit.io
    if (client.connect("my_arduino", userid, apikey))
    {
      Serial.println("... connected");

      // once connected, publish an announcement and subscribe to all
      // feeds
      client.publish("<your adafruit.io username>/f/status-messages", "we are alive!");
      client.subscribe("<your adafruit.io username>/f/light-switch");
      client.subscribe("<your adafruit.io username>/throttle");
    }
    else
    {
      // print some error status
      Serial.print("connection failed, rc = ");
      Serial.print(client.state());
      Serial.println();
      Serial.println("we will try again in 5 seconds");

      // wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

Entering and running this code shows the following output in our serial window when we toggle the switch on our web interface (see Figure 19).



```
starting mqtt client on arduino ...
mqtt client is at: 192.168.1.91
attempting mqtt connection...
... connected
message arrived [alexshenfield/f/status-messages] we are alive!
message arrived [alexshenfield/f/light-switch] ON
message arrived [alexshenfield/f/light-switch] OFF
message arrived [alexshenfield/f/light-switch] ON
message arrived [alexshenfield/f/light-switch] OFF
message arrived [alexshenfield/f/light-switch] ON
```

Autoscroll: ✓ | No line ending | 9600 baud

Figure 19 – Serial window output from toggling a switch on my dashboard

So this means that we can interact with the toggle switch by sending and receiving “ON” and “OFF”. We can use the **strcmp** function to check for these values on the light-switch topic / feed in our MQTT subscription callback.

The code for the complete application is shown in code listing 5.

Code listing 5:

```
/*
 * iot_light_switch.ino
 *
 * an internet of things light switch that integrates with adafruit io
 * to allow remote control of lighting
 *
 * WARNING: if you toggle the light switch too fast, you are likely to
 * hit the rate limits of adafruit.io - pay attention to the throttle
 * topic in the serial window
 *
 * author:  alex shenfield
 * date:    10/09/2018
 */

// include necessary libraries
#include <SPI.h>
#include <Ethernet2.h>
#include <PubSubClient.h>

// ETHERNET CONNECTION DECLARATIONS

// mac address
byte mac[] = {0x90, 0xA2, 0xDA, 0x0F, 0xF5, 0xE7};

// the ethernet shields and ethernet libraries support DHCP as long as
// you are plugged into a device with a DHCP server - it is always a
// good idea to have a static address to fall back on though!
IPAddress ip(192, 168, 0, 11);
IPAddress gateway(192, 168, 0, 254);
IPAddress subnet(255, 255, 255, 0);

// MQTT DECLARATIONS

// mqtt server details
const char userid[] = "<your adafruit.io username>";
const char apikey[] = "<your adafruit.io api key>";
const char server[] = "io.adafruit.com";
const long port     = 1883;

// get an ethernet client and pass it to the mqtt client
EthernetClient ethernet_client;
PubSubClient client(ethernet_client);
```

```
// DEBOUNCING AND LED DECLARATIONS

// light bulb and light switch pins
const int light_bulb = 3;
const int light_switch = 2;

// set some debounce variables for our button
int curr_button_state = LOW;
int last_button_state = LOW;

unsigned long last_debounce_time = 0;
unsigned long debounce_delay = 50;

int led_state = LOW;
```

```
// CODE

// set up code
void setup()
{
  // set up serial comms for debugging and display of DHCP allocated ip
  // address
  Serial.begin(9600);
  Serial.println("starting mqtt client on arduino ...");

  // mqtt server and subscription callback
  client.setServer(server, port);
  client.setCallback(callback);

  // start the ethernet shield comms - initially try to get a DHCP ip
  // address
  if (Ethernet.begin(mac) == 0)
  {
    // if DHCP fails, allocate a static ip address
    Serial.println("failed to configure ethernet using DHCP");
    Ethernet.begin(mac, ip);
  }

  // allow the hardware to sort itself out
  delay(1500);

  // print the ip address to the serial monitor
  Serial.print("mqtt client is at: ");
  Serial.println(Ethernet.localIP());
}

// main loop
void loop()
{
  // if the client isn't connected then try to reconnect
  if (!client.connected())
  {
    reconnect();
  }
  else
  {
    // handle the light switch
    update_light_state();
    digitalWrite(light_bulb, led_state);

    // handle mqtt functions (e.g. subscriptions to topics)
    client.loop();
  }
}
```

```
// MQTT FUNCTIONS

// mqtt message received callback
void callback(char* topic, byte* payload, unsigned int length)
{
  // get the topic
  String t = String(topic);

  // get the value of the message
  char data[length + 1];
  for (int i = 0; i < length; i++)
  {
    data[i] = payload[i];
  }
  data[length] = '\0';

  // print the message to the serial window
  Serial.print("message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  Serial.println(data);

  // parse the light switch topic to work out what the light bulb
  // should be doing
  if (t.indexOf("light-switch") > 0)
  {
    if (strcmp(data, "ON") == 0)
    {
      led_state = HIGH;
    }
    else if (strcmp(data, "OFF") == 0)
    {
      led_state = LOW;
    }
  }
}
```

```
// reconnect to server
void reconnect()
{
  // loop until we're reconnected
  while (!client.connected())
  {
    Serial.println("attempting mqtt connection...");

    // try to connect to adafruit.io
    if (client.connect("my_arduino", userid, apikey))
    {
      Serial.println("... connected");

      // once connected, publish an announcement ...
      client.publish("<your adafruit.io username>/f/status-messages", "we are alive!");

      // also publish the light switch state (so that we reset any UI controls) ...
      send_light_status(led_state);

      // ... and subscribe to the appropriate feeds (including the
      // "throttle" topic which will let us know if we are bumping up
      // against the adafruit.io rate limit)
      client.subscribe("<your adafruit.io username>/f/status-messages");
      client.subscribe("<your adafruit.io username>/f/light");
      client.subscribe("<your adafruit.io username>/throttle");
    }
    else
    {
      // print some error status
      Serial.print("connection failed, rc = ");
      Serial.print(client.state());
      Serial.println();
      Serial.println("we will try again in 5 seconds");

      // wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

```
// APPLICATION FUNCTIONS

// send the light switch status to the mqtt broker
void send_light_status(int led)
{
  // if the client is already connected then send data
  if (client.connected())
  {
    // send "ON" if the led is on, and "OFF" if the led is off
    if (led == HIGH)
    {
      client.publish("<your adafruit.io username>/f/light-switch", "ON");
    }
    else
    {
      client.publish("<your adafruit.io username>/f/light-switch", "OFF");
    }
  }
}

// update the state of the light by reading the light switch and debouncng it
void update_light_state()
{
  // get the state of the light switch
  int reading = digitalRead(light_switch);

  // if the light switch state has changed then reset the deboucing
  // timer
  if (reading != last_button_state)
  {
    last_debounce_time = millis();
  }

  // if the reading has been there longer than the debounce delay,
  // then it is a valid state
  if ((millis() - last_debounce_time) > debounce_delay)
  {
    // so if the light switch state has changed
    if (reading != curr_button_state)
    {
      // then update it
      curr_button_state = reading;

      // if the button has been pressed then toggle the led state and
      // update the mqtt broker
      if (curr_button_state == HIGH)
      {
        led_state = !led_state;
        send_light_status(led_state);
      }
    }
  }

  // update last button state
  last_button_state = reading;
}
```

You can see from the above code that we have restructured the MQTT subscription callback so that we extract the data from the MQTT message payload and use that to decide what to do about the state of the light bulb.

We also make sure that when we operate the physical button to turn the light on, that we notify the web interface that the light switch has been turned on or off. This is so the web interface remains consistent with the rest of our system.

Enter this code and test it to make sure everything works. You should see the led turning on and off with both the physical button and the toggle switch on the web interface. My completed system is shown in Figure 20.

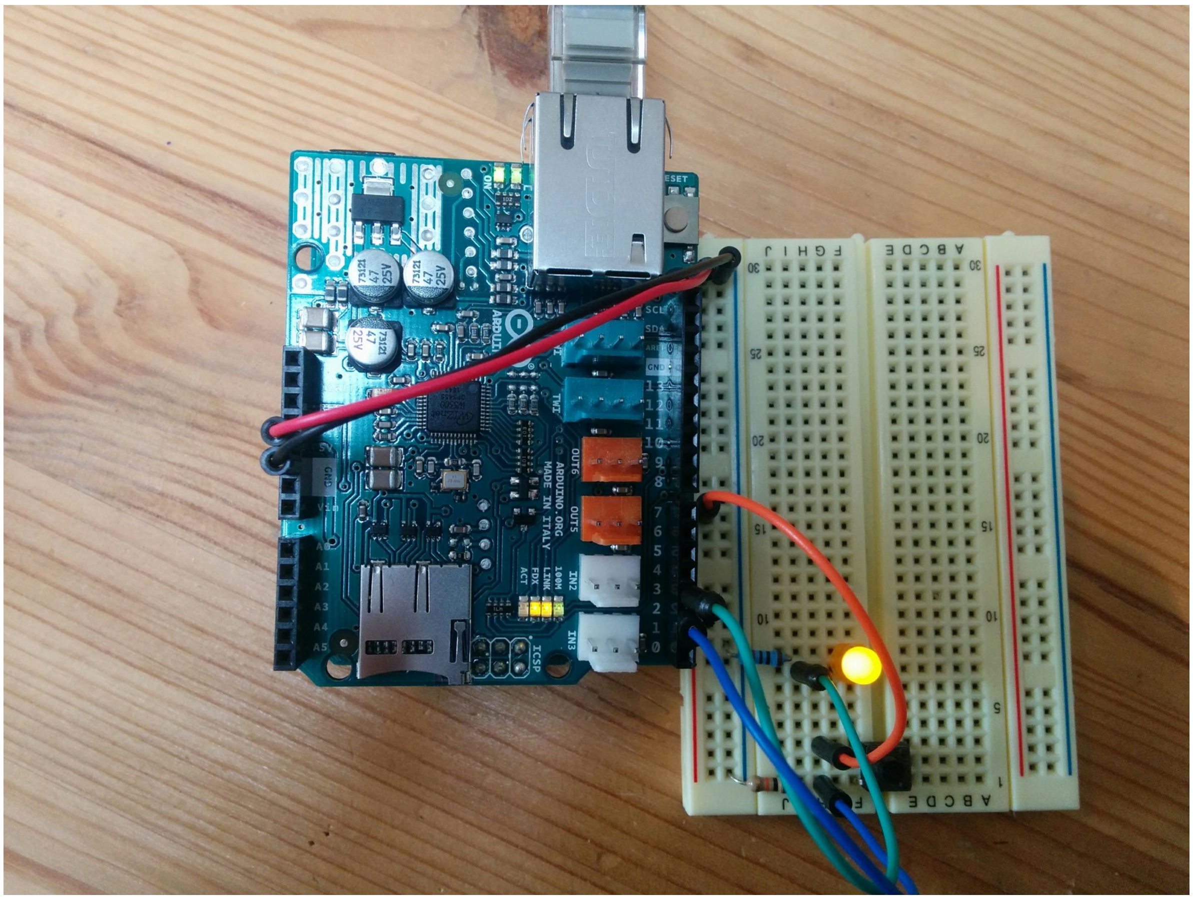


Figure 20 – My IoT light switch!

Exercises

Now you are going to make some modifications to the program to add functionality:

1. Add a calibrated light sensor (as in task 2) to send the ambient light levels to the Adafruit IO platform and chart these on a graph block in your dashboard.

2. Add the ability to turn an automatic monitoring routine on or off from the web interface (e.g. using another toggle button). This automatic monitoring routine should turn the LED on when the light level goes below a certain set point and off when it goes above the set point.

3. Add the ability to change the lighting set point from the web interface (e.g. using a slider control).

Check list

Task 1 – Program	
Task 1 – Additional functionality 1	
Task 1 – Additional functionality 2	
Task 1 – Additional functionality 3	
Task 2 – Program	
Task 2 – Additional functionality 1	
Task 2 – Additional functionality 2	
Task 2 – Additional functionality 3	
Task 3 – Program 1	
Task 3 – Program 2	
Task 3 – Program 3	
Task 3 – Additional functionality 1	
Task 3 – Additional functionality 2	
Task 3 – Additional functionality 3	

Feedback