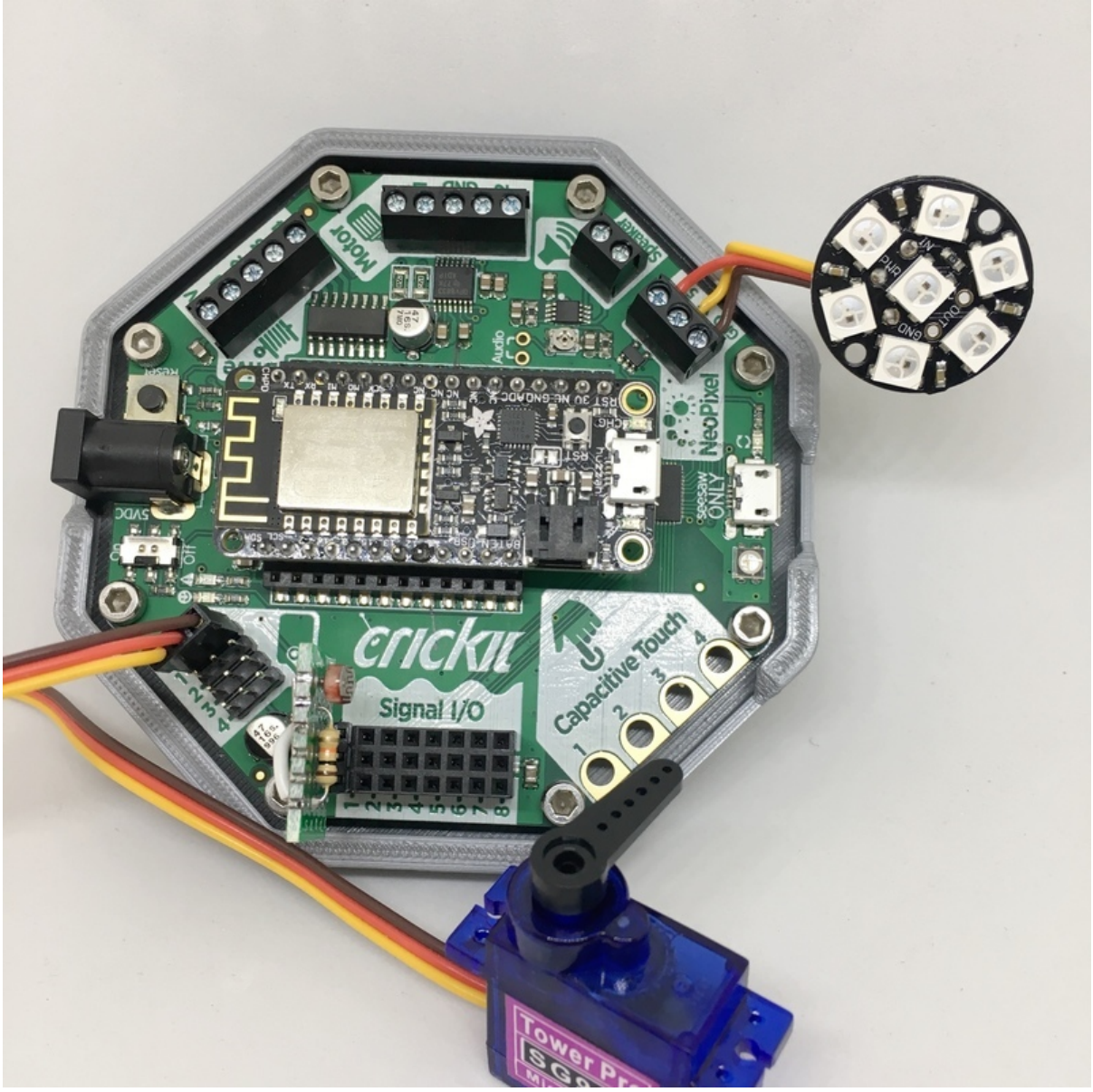
Using Crickit and Adafruit IO together

Created by Dave Astels

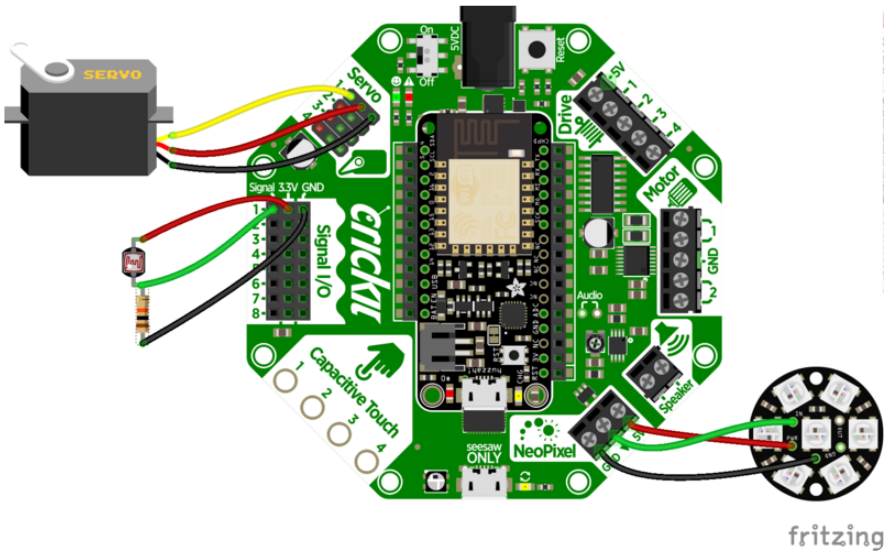


https://learn.adafruit.com/crickit-and-adafruitio

Last updated on 2025-11-11 05:07:17 PM EST

Table of Contents

Overview



This guide will explore using Crickit with Adafruit IO together. Crickit provides various ways to take measurements and control other hardware. It's a natural fit with Adafruit IO which lets you store data and manage events.

If you haven't used Crickit before, we have a guide to get you started (https://adafru.it/BD7). If you aren't familiar with Adafruit IO, we have a great introduction (https://adafru.it/BRB). Log into Adafruit IO and set up an account if you haven't.

We need to use a board that has WiFi capabilities. Since we'll be using a Crickit, this means using the FeatherWing Crickit and a WiFi capable Feather. In this guide we'll use the Feather HUZZAH ESP8266. If you haven't used this Feather before, or haven't used it with Adafruit IO, we have a guide that walks you through getting it up and running (https://adafru.it/ufK).

First we need some hardware to play with. Let's add a few things to the Crickit. To start, we can use the capacitive touch inputs as they are. We can use one of the analog inputs to make a light sensor with a 10K resistor and a photoresistor. For output we can hook up a servo and some NeoPixels. You can use any number or configuration of NeoPixels; remember to adjust the code to reflect the number of pixels in the product you use.

If you aren't familiar with NeoPixels, they're quite handy. See our NeoPixel Überguide (https://adafru.it/dhw) for everything you need to know about them.

Parts

Below are the parts to make one HUZZAH-CRICKIT node. For the final exercise you'll need to build two, but the second doesn't need the servo or NeoPixel jewel.

1 x Feather HUZZAH ESP8266 https://www.adafruit.com/product/2821
An 'all-in-one' ESP8266 WiFi development board with built in USB and battery charging.

---

1 x Crickit FeatherWing https://www.adafruit.com/product/3343
Crickit robotics board for any Feather board.

---

1 x Micro Servo https://www.adafruit.com/product/169
TowerPro SG92R micro servo

---

1 x NeoPixel Jewel https://www.adafruit.com/product/2226
NeoPixel Jewel - 7 x 5050 RGB LED with Integrated Drivers

---

1 x Photo cell https://www.adafruit.com/product/161
CdS photoresistor (not RoHS compliant)

---

1 x 10K ohm resistor https://www.adafruit.com/product/2784
25 Pack of 10K Ω Resistors

Adafruit IO Configuration

Configuration

Each Arduino sketch for this guide uses a single **config.h** file that stores your WiFi and Adafruit IO credentials. There are placeholders in that file that need to be replaced with values specific to your WiFi network and Adafruit IO account. Make changes and use that file with each of the sketches on subsequent pages in this guide: copy it into the directory for each sketch. The ESP/Adafruit IO guide (https://adafru.it/ufK) walks you through making those changes.

```
// SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/************************ Adafruit IO Config *******************************/

// visit io.adafruit.com if you need to create an account,
// or if you need your Adafruit IO key.
#define IO_USERNAME    "your_username"
#define IO_KEY         "your_key"
```

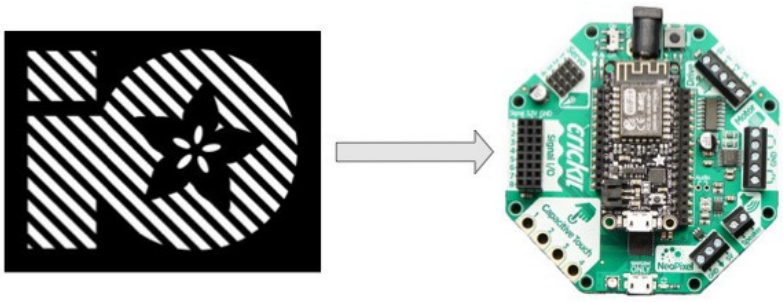
```
/************************** WIFI *****************************************/

// the AdafruitIO_WiFi client will work with the following boards:
//   - HUZZAH ESP8266 Breakout -> https://www.adafruit.com/products/2471
//   - Feather HUZZAH ESP8266 -> https://www.adafruit.com/products/2821
//   - Feather M0 WiFi -> https://www.adafruit.com/products/3010
//   - Feather WICED -> https://www.adafruit.com/products/3056

#define WIFI_SSID       "your_ssid"
#define WIFI_PASS       "your_pass"

// comment out the following two lines if you are using fona or ethernet
#include "AdafruitIO_WiFi.h"
AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);


/************************** FONA *****************************************/

// the AdafruitIO_FONA client will work with the following boards:
//   - Feather 32u4 FONA -> https://www.adafruit.com/product/3027

// uncomment the following two lines for 32u4 FONA,
// and comment out the AdafruitIO_WiFi client in the WIFI section
// #include "AdafruitIO_FONA.h"
// AdafruitIO_FONA io(IO_USERNAME, IO_KEY);


/************************** ETHERNET *****************************************/

// the AdafruitIO_Ethernet client will work with the following boards:
//   - Ethernet FeatherWing -> https://www.adafruit.com/products/3201

// uncomment the following two lines for ethernet,
// and comment out the AdafruitIO_WiFi client in the WIFI section
// #include "AdafruitIO_Ethernet.h"
// AdafruitIO_Ethernet io(IO_USERNAME, IO_KEY);
```

Libraries

To build the code in this guide you will need to use the Arduino Library Manager to install the latest versions of the `Adafruit_Seesaw`, `Adafruit_IO_Arduino`, and `ArduinoHTTPClient` libraries. If you haven't installed Adafruit libraries into your Arduino environment before, see [this guide](https://adafru.it/dit).

Controlling Hardware From a Dashboard

In this section we'll set up a dashboard on Adadfruit IO to control a servo and neopixels on the Feather/Crickit.



Setting up Adafruit IO

Let's start by connecting the servo and NeoPixels to Adafruit IO. If you aren't familiar with feeds, we have a guide for that (https://adafru.it/ioA).

We start by creating a new feed group called `crickit` (Feeds -> Actions -> Create New Group). We'll use that to contain the feeds for this guide. Now we can add the feeds: `neopixel-control` and `servo1-control` via Feeds -> Actions -> Create a new Feed. When you create each, you have the option of adding them to a group. Do so, adding them to the `crickit` group.

Arduino Code

The first program is called **dashboard_control.ino**. This program will allow you to control a servo and NeoPixels connected on an Adafruit Crickit Featherwing via Adafruit IO.

The key parts of the code are setting up the feeds and handling events from them.

Code Highlights

Setting up the feeds is split into two parts: declaring them, and initializing them. The declarations are simple: make variables that are pointers to `AdafruitIO_Feed` instances.

```
AdafruitIO_Feed *servo1_control;
AdafruitIO_Feed *neopixel_control;
```

There's a function to allocate/initialize the feeds. It's called near the start of the `setup()` function.

```
void setup_feeds()
{
  servo1_control = io.feed("crickit.servo1-control");
```

```
  neopixel_control = io.feed("crickit.neopixel-control");
}
```

Notice that the feed names have a `crickit.` prefix. That's because they are in the `crickit` feed group. Now the connection to Adafruit IO can be started.

In config.h the `AdafruitIO` interface object is created, in our case using WiFi:

```
AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);
```

Once the feeds have been created, the Adafruit IO interface object can be initialized. This will make the connection to Adafruit IO and get it running.

```
  io.connect();

  while(io.status() &lt; AIO_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
```

Setting up the hardware

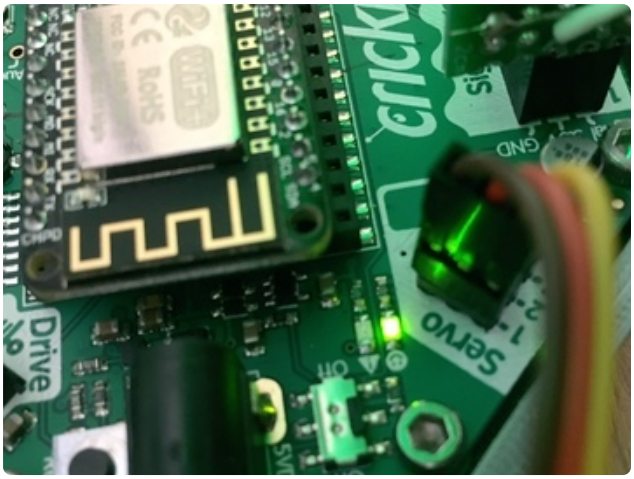
Now that Adafruit IO is set up, we can turn to the Crickit and it's connected hardware.

```
Adafruit_Crickit crickit;
seesaw_Servo servo_1(&amp;crickit);
seesaw_NeoPixel strip = seesaw_NeoPixel(NEOPIX_NUMBER_OF_PIXELS, NEOPIX_PIN, NEO_GRB
+ NEO_KHZ800);
```

And we initialize them at the end of setup():

```
void setup()
{
  ...
  servo1_control-&gt;get();
  servo_1.attach(CRICKIT_SERVO1, 600, 2400);
}
```

The attach function literally attaches the servo object to a specific servo port on the Crickit, servo 1, in this case.



The attach function 'attaches' the servo object to a specific servo port on the Crickit. `servo 1`, in this case.

Be sure to plug in your servo into the right slot, and with the white/yellow wire next to the **1** symbol

Loop()

In this sketch, the `loop()` function is does the bare minimum: it keeps the connection to Adafruit IO alive and well and routes any incoming events to the associated handler function.

```
void loop()
{
  // io.run(); is required for all sketches.
  // it should always be present at the top of your loop
  // function. it keeps the client connected to
  // io.adafruit.com, and processes any incoming data.
  io.run();
}
```

Event handlers

Back in `setup()`, after the connection to Adafruit IO and the feeds are in place the event handlers can be set.

```
  servo1_control-&gt;onMessage(handle_servo_message);
  neopixel_control-&gt;onMessage(handle_neopixel_message);
```

This attaches an event handler function to each of the Adafruit IO feeds that we want to respond to: `handle_servo_message` handles servo angle events, and `handle_neopixel_message` handles NeoPixel color changes.

```
void handle_servo_message(AdafruitIO_Data *data)
{
  Serial.print("received servo control &lt;- ");
  Serial.println(data-&gt;value());
  int angle = data-&gt;toInt();
  if(angle &lt; 0) {
    angle = 0;
  } else if(angle &gt; 180) {
```

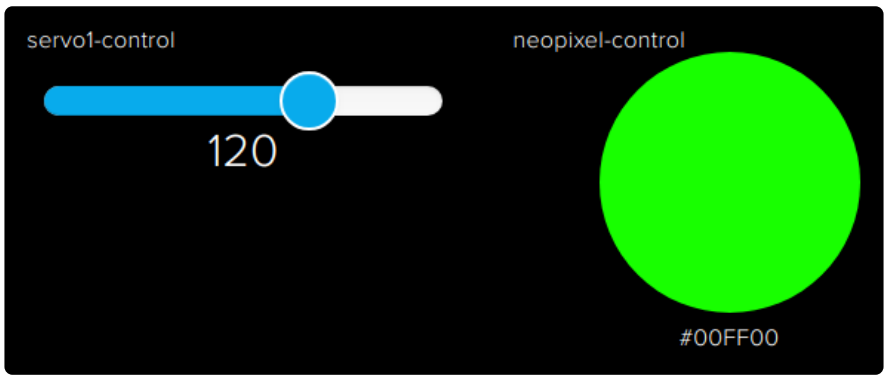
```
    angle = 180;
  }
  servo_1.write(angle);

}

void handle_neopixel_message(AdafruitIO_Data *data)
{
  Serial.print("received neopixel control &lt;- ");
  Serial.println(data-&gt;value());
  long color = data-&gt;toNeoPixel();
  for (int pixel = 0; pixel &lt; NEOPIX_NUMBER_OF_PIXELS; pixel++) {
    strip.setPixelColor(pixel, color);
  }
  strip.show();
}
```

That's it: set everything up and attach even handlers. This sketch is completely reactive, responding to events by making changes to the hardware.

Generating events

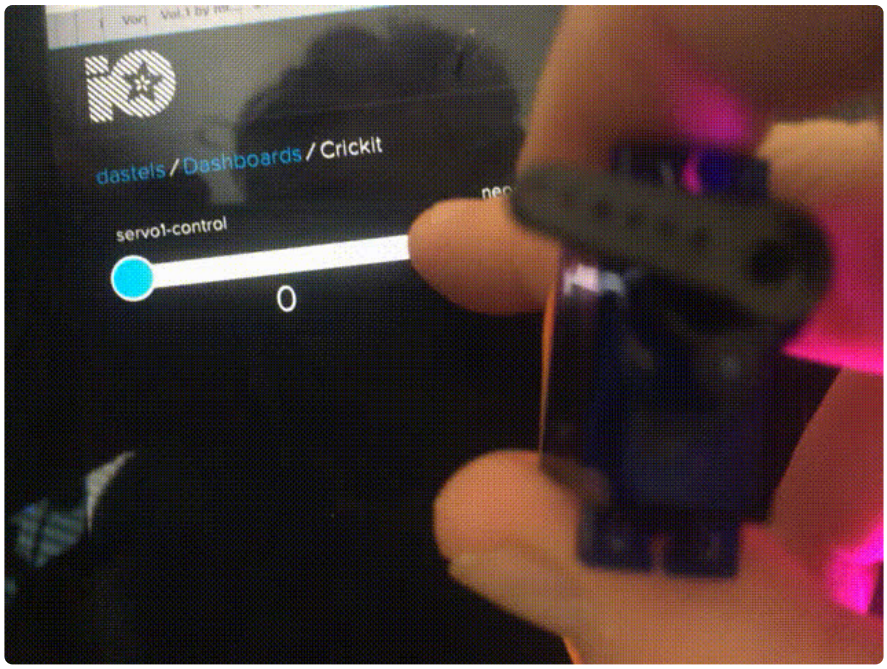
Now that the Feather and Crickit are set up and the sketch in place, we need to send it events to handle. We'll make an Adafruit IO dashboard for this. If you haven't worked with dashboards, there's a guide to get you started (https://adafru.it/f5m).

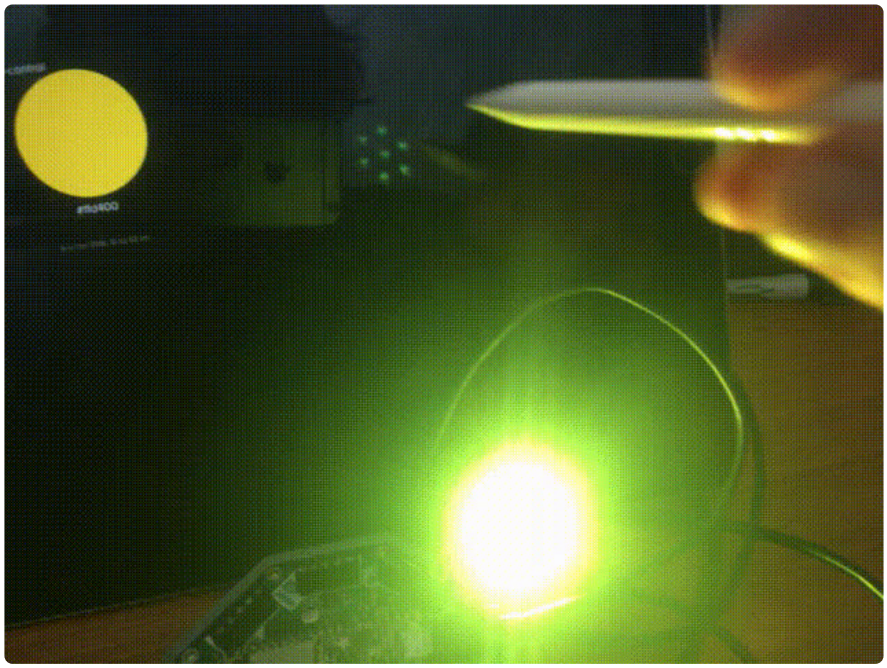
We'll want a number slider block connected to the `servo1-control` feed that has a value range of 0 to 180. For `neopixel-control` we need to make a color picker block.



If everything worked, you should be able to power up the Feather/Crickit and change the dashboard controls to see the servo rotate and the NeoPixels change color.

You can use the Arduino serial monitor to monitor the status messages from the sketch, just in case the board does not make a connection first to the WiFi network and then to Adafruit IO.





The complete sketch is below:

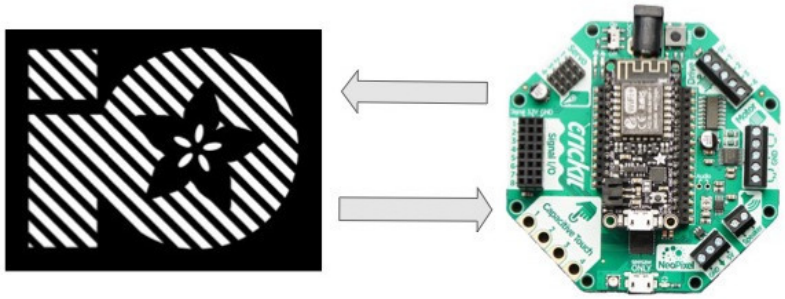
```
// SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
//
// SPDX-License-Identifier: MIT

// Crickit + Adafruit IO Subscribe Example
//
// Adafruit invests time and resources providing this open source code.
// Please support Adafruit and open source hardware by purchasing
// products from Adafruit!
//
// Written by Dave Astels for Adafruit Industries
// Copyright (c) 2018 Adafruit Industries
```

```
// Licensed under the MIT license.
//
// All text above must be included in any redistribution.

/************************ Configuration ***********************************/

// edit the config.h tab and enter your Adafruit IO credentials
// and any additional configuration needed for WiFi, cellular,
// or ethernet clients.
#include "config.h"
#include <Adafruit_Crickit.h>
#include <seesaw_servo.h>
#include <seesaw_neopixel.h>

#define NEOPIX_PIN (20)                  /* Neopixel pin */
#define NEOPIX_NUMBER_OF_PIXELS (7)


// set up the feeds
AdafruitIO_Feed *servo1_control;
AdafruitIO_Feed *neopixel_control;

// set up the Crickit

Adafruit_Crickit crickit;
seesaw_Servo servo_1(&crickit);  // create servo object to control a servo

// Parameter 1 = number of pixels in strip
// Parameter 2 = Arduino pin number (most are valid)
// Parameter 3 = pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
//   NEO_RGBW    Pixels are wired for RGBW bitstream (NeoPixel RGBW products)
seesaw_NeoPixel strip = seesaw_NeoPixel(NEOPIX_NUMBER_OF_PIXELS, NEOPIX_PIN, NEO_GRB
+ NEO_KHZ800);

void setup_feeds()
{
  servo1_control = io.feed("crickit.servo1-control");
  neopixel_control = io.feed("crickit.neopixel-control");
}


void setup()
{
  // start the serial connection
  Serial.begin(115200);

  // wait for serial monitor to open
  while(! Serial);

  setup_feeds();
  Serial.println("Feeds set up");

  Serial.println("Connecting to Adafruit IO");

  // connect to io.adafruit.com
  io.connect();

  // set up message handlers for the servo and neopixel feeds.
  // the handle_*_message functions (defined below)
  // will be called whenever a message is
  // received from adafruit io.

  servo1_control->onMessage(handle_servo_message);
  neopixel_control->onMessage(handle_neopixel_message);
```

```
  // wait for a connection
  while(io.status() < AIO_CONNECTED) {
    Serial.print(".");
    //    Serial.println(io.statusText());
    delay(500);
  }

  // we are connected
  Serial.println();
  Serial.println(io.statusText());

  if (!crickit.begin()) {
    Serial.println("Error starting Crickit!");
    while(1);
  } else {
    Serial.println("Crickit started");
  }

  if(!strip.begin()){
    Serial.println("Error starting Neopixels!");
    while(1);
  } else {
    Serial.println("Neopixels started");
  }

  servo1_control->get();

  servo_1.attach(CRICKIT_SERVO1);

  Serial.println("setup complete");
}


void loop()
{

  // io.run(); is required for all sketches.
  // it should always be present at the top of your loop
  // function. it keeps the client connected to
  // io.adafruit.com, and processes any incoming data.
  io.run();
}


void handle_servo_message(AdafruitIO_Data *data)
{
  Serial.print("received servo control <- ");
  Serial.println(data->value());
  int angle = data->toInt();
  if(angle < 0) {
    angle = 0;
  } else if(angle > 180) {
    angle = 180;
  }
  servo_1.write(angle);

}


void handle_neopixel_message(AdafruitIO_Data *data)
{
  Serial.print("received neopixel control <- ");
  Serial.println(data->value());
  long color = data->toNeoPixel();
  for (int pixel = 0; pixel < NEOPIX_NUMBER_OF_PIXELS; pixel++) {
    strip.setPixelColor(pixel, color);
  }
  strip.show();
}
```

Sending Measurements to Adafruit IO

It's time to turn it around and send data to Adafruit IO. The Crickit has builtin capacitive touch sensors which are easy to use, and we can add a simple photocell circuit to measure light level.



Back in your Adafruit IO account add a `touch_0` feed to the `crickit` group. Then it can be added to the code along with the existing feeds.

The next program is called **crickit_io.ino** and will read the Crickit capacitive touch pads and send data back toan Adafruit IO Dashboard.

Code Highlights

```
AdafruitIO_Feed *touch;
...
touch = io.feed("crickit.touch-0");
```

Since this is an outgoing feed, we don't need a handler for it. We do, however, need to send values to it. We do this in the loop function. We could just do something like:

```
void loop()
{
  io.run();
  uint16_t val = crickit.touchRead(0);

  if (val &gt;= CAPTOUCH_THRESH) {
    touch-&gt;save(1);
  } else if (val &lt; CAPTOUCH_THRESH) {
    touch-&gt;save(0);
  }
}
```

The problem with this is that it would spam the feed with generally redundant data. That's bad just for bandwidth reasons, but there is also constraints on the rate at which you can send data to a feed.

We can do a couple things to prevent that. The first is to throttle our data collection and reporting. We can wrap our collection/reporting code in an `if` with a condition that is only true every so often. In the code it's set to 1000 ms. This way, the capacitive touch sensors are read and information sent to Adafruit IO at most once per second.

```
#define IO_LOOP_DELAY (1000)
unsigned long lastUpdate = 0;

void loop()
{
  io.run();

  if (millis() &gt; (lastUpdate + IO_LOOP_DELAY)) {
    // measure/update code here
    lastUpdate = millis();
  }

}
```

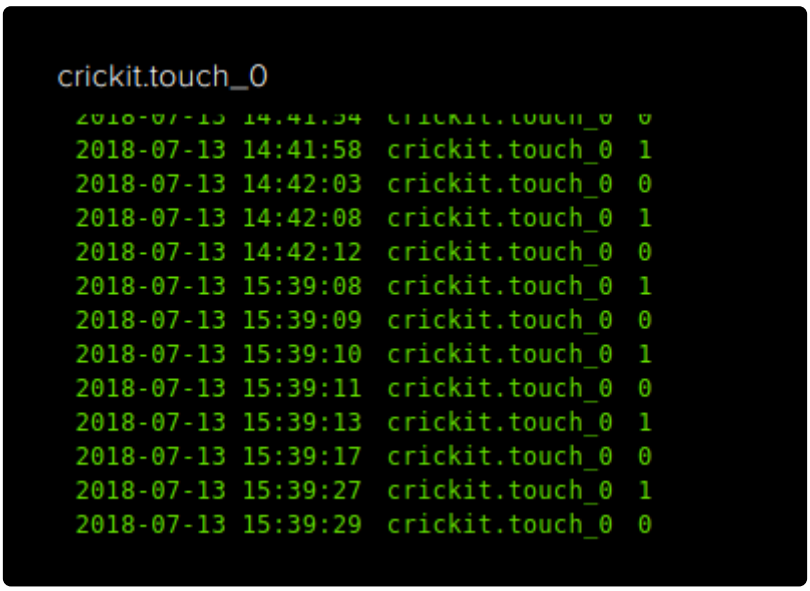
Another approach (which works well combined with the above) is to only send an update when there's a good reason to. In this case we can send an update only when the state changes. To do this we keep track of the last thing we sent and only send an update when it would be different.

The final cap-touch code is below.

```
boolean last_touch = false;
...

void loop()
{
  io.run();

  if (millis() &gt; (lastUpdate + IO_LOOP_DELAY)) {

    uint16_t val = crickit.touchRead(0);

    if (val &gt;= CAPTOUCH_THRESH &amp;&amp; !last_touch) {
      touch-&gt;save(1);
      last_touch = true;
      Serial.println("CT 0 touched.");
    } else if (val &lt; CAPTOUCH_THRESH &amp;&amp; last_touch) {
      touch-&gt;save(0);
      last_touch = false;
      Serial.println("CT 0 released.");
    }

    // after publishing, store the current time
    lastUpdate = millis();
  }
```

```
}
```

Back in our Adafruit IO dashboard we can add a block to show the cap-touch values that get sent. A stream block will work well for this data:



We can do something very similar for the light sensor, starting with setting up the feed in Adafruit IO and the code.

```
AdafruitIO_Feed *light;
uint16_t last_reported_light = 0;
...
light = io.feed("crickit.light")
```

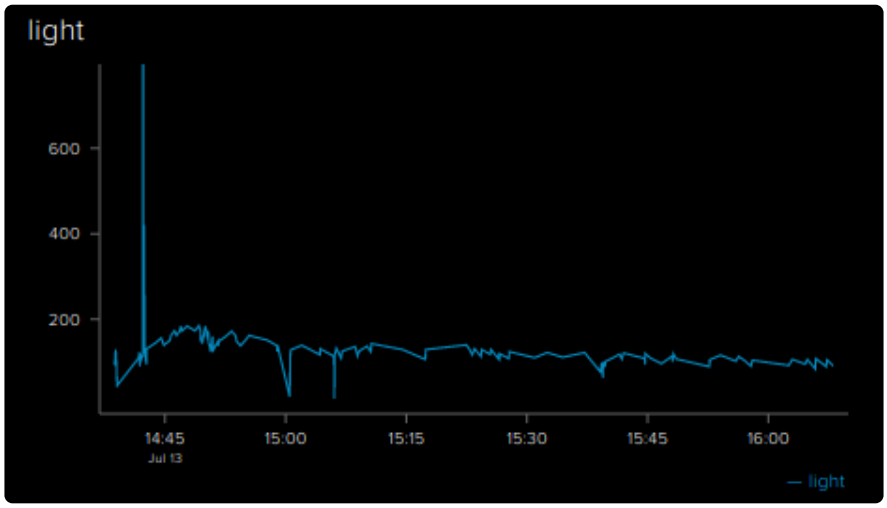
Now back the the `loop()` function where we will read the sensor and send the readings to IO.

```
void loop()
{
  ...
  if (millis() &gt; (lastUpdate + IO_LOOP_DELAY)) {

    uint16_t light_level = crickit.analogRead(CRICKIT_SIGNAL1);
    uint16_t light_delta = abs(light_level - last_reported_light);

    if (light_delta &gt; 10) {
      light-&gt;save(light_level);
      last_reported_light = light_level;
      Serial.print("Sending ");
    }
    Serial.print("Light: ");
    Serial.println(light_level);
```

```
    ...
    // after publishing, store the current time
    lastUpdate = millis();
  }

}
```

Instead of sending every time the light reading changes, which would be quite often, we can send only when it changes by some amount (10 in the code above). This way, minor variations are filtered out and only significant changes are sent.

We can add a block to our dashboard to visualize the light readings. A line chart would be a good choice.



The complete sketch is below.

```
// SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
//
// SPDX-License-Identifier: MIT

// Crickit + Adafruit IO Publish & Subscribe Example
//
// Adafruit invests time and resources providing this open source code.
// Please support Adafruit and open source hardware by purchasing
// products from Adafruit!
//
// Written by Dave Astels for Adafruit Industries
// Copyright (c) 2018 Adafruit Industries
// Licensed under the MIT license.
//
// All text above must be included in any redistribution.

/************************ Configuration ******************************/

// edit the config.h tab and enter your Adafruit IO credentials
// and any additional configuration needed for WiFi, cellular,
// or ethernet clients.
#include "config.h"
```

```
#include <Adafruit_Crickit.h>
#include <seesaw_servo.h>
#include <seesaw_neopixel.h>

#define NEOPIX_PIN (20)                  /* Neopixel pin */
#define NEOPIX_NUMBER_OF_PIXELS (7)

#define CAPTOUCH_THRESH 500

#define IO_LOOP_DELAY (1000)
unsigned long lastUpdate = 0;

// set up the feeds
AdafruitIO_Feed *servo1_control;
AdafruitIO_Feed *neopixel_control;
AdafruitIO_Feed *light;
uint16_t last_reported_light = 0;

AdafruitIO_Feed *touch;
boolean last_touch = false;

// set up the Crickit

Adafruit_Crickit crickit;
seesaw_Servo servo_1(&crickit);  // create servo object to control a servo

// Parameter 1 = number of pixels in strip
// Parameter 2 = Arduino pin number (most are valid)
// Parameter 3 = pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
//   NEO_RGBW    Pixels are wired for RGBW bitstream (NeoPixel RGBW products)
seesaw_NeoPixel strip = seesaw_NeoPixel(NEOPIX_NUMBER_OF_PIXELS, NEOPIX_PIN, NEO_GRB
+ NEO_KHZ800);

void setup_feeds()
{
  servo1_control = io.feed("crickit.servo1-control");
  neopixel_control = io.feed("crickit.neopixel-control");
  light = io.feed("crickit.light");
  touch = io.feed("crickit.touch-0");
}


void setup()
{
  setup_feeds();
  Serial.println("Feeds set up");

  // start the serial connection
  Serial.begin(115200);

  // wait for serial monitor to open
  while(! Serial);

  Serial.println("Connecting to Adafruit IO");

  // connect to io.adafruit.com
  io.connect();

  // set up a message handler for the count feed.
  // the handleMessage function (defined below)
  // will be called whenever a message is
  // received from adafruit io.
  // setup handlers
  servo1_control->onMessage(handle_servo_message);
  neopixel_control->onMessage(handle_neopixel_message);
```

```
  // wait for a connection
  while(io.status() < AIO_CONNECTED) {
    Serial.println(io.statusText());
    delay(500);
  }

  // we are connected
  Serial.println();
  Serial.println(io.statusText());

  if (!crickit.begin()) {
    Serial.println("Error starting Crickit!");
    while(1);
  } else {
    Serial.println("Crickit started");
  }

  if(!strip.begin()){
    Serial.println("Error starting Neopixels!");
    while(1);
  } else {
    Serial.println("Neopixels started");
  }

  servo1_control->get();
  servo_1.attach(CRICKIT_SERVO1);

  Serial.println("setup complete");
}


void loop()
{

  // io.run(); is required for all sketches.
  // it should always be present at the top of your loop
  // function. it keeps the client connected to
  // io.adafruit.com, and processes any incoming data.
  io.run();

  if (millis() > (lastUpdate + IO_LOOP_DELAY)) {

    uint16_t light_level = crickit.analogRead(CRICKIT_SIGNAL1);
    uint16_t light_delta = abs(light_level - last_reported_light);

    if (light_delta > 10) {
      light->save(light_level);
      last_reported_light = light_level;
      Serial.print("Sending ");
    }
    Serial.print("Light: ");
    Serial.println(light_level);

    uint16_t val = crickit.touchRead(0);

    if (val >= CAPTOUCH_THRESH && !last_touch) {
      touch->save(1);
      last_touch = true;
      Serial.println("CT 0 touched.");
    } else if (val < CAPTOUCH_THRESH && last_touch) {
      touch->save(0);
      last_touch = false;
      Serial.println("CT 0 released.");
    }

    // after publishing, store the current time
    lastUpdate = millis();
  }
```

```
}

void handle_servo_message(AdafruitIO_Data *data)
{
  Serial.print("received servo control <- ");
  Serial.println(data->value());
  int angle = data->toInt();
  if(angle < 0) {
    angle = 0;
  } else if(angle > 180) {
    angle = 180;
  }
  servo_1.write(angle);

}


void handle_neopixel_message(AdafruitIO_Data *data)
{
  Serial.print("received neopixel control <- ");
  Serial.println(data->value());
  long color = data->toNeoPixel();
  for (int pixel = 0; pixel < NEOPIX_NUMBER_OF_PIXELS; pixel++) {
    strip.setPixelColor(pixel, color);
  }
  strip.show();
}
```

Setting up Actions

> ɔns were previously called Triggers. Some images may show the old ɟers terminology.

So now that we are sending cap-touch and light data to Adafruit IO and reacting to servo and NeoPixel events from it let's link them up.

Adafruit IO has a really cool feature for adding some logic to your data collection: actions. A action is a response to an event getting posted, I.e. a new piece of data being added (aka published) to a feed. When data gets added to a feed, any associated triggers compare the new value against a constant or the current value of another feed.

If the condition is satisfied it can take one of several actions:

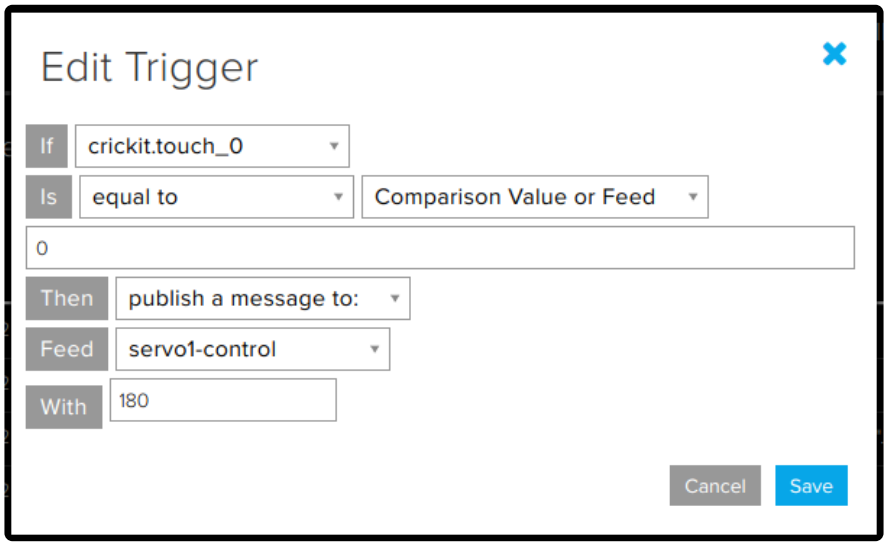
- email you a message,
- send a webhook message, or

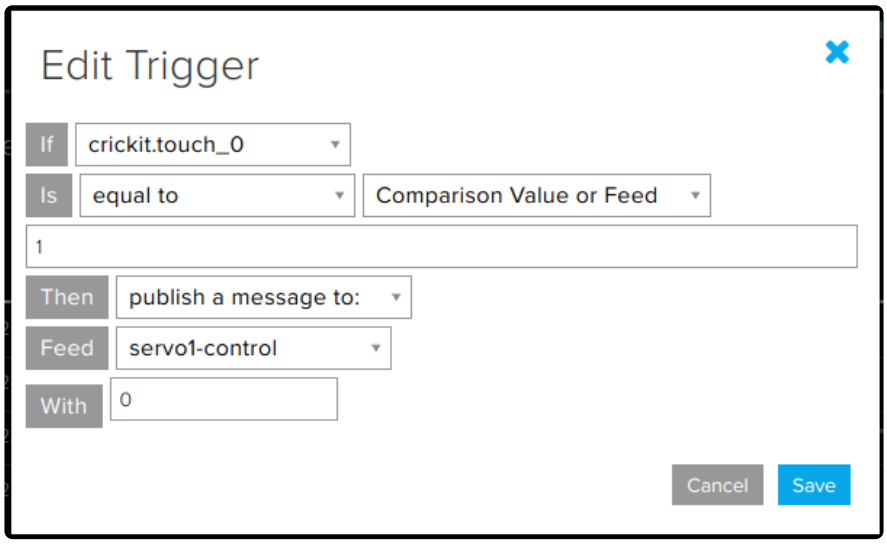
- publish a specific value to feed.

It's the last of these that is of particular interest.

Let's make the cap-touch move the servo between 0 and 180 degrees. Since we send a 1 and 0 when the cap-touch is touched and released, respectively, we can use that in two triggers. One for each state.

In the main Adafruit IO screen, on the left side, you'll see a menu item **Actions**. Click that and create the two Reactive Actions below. They look similar but have slightly different values.



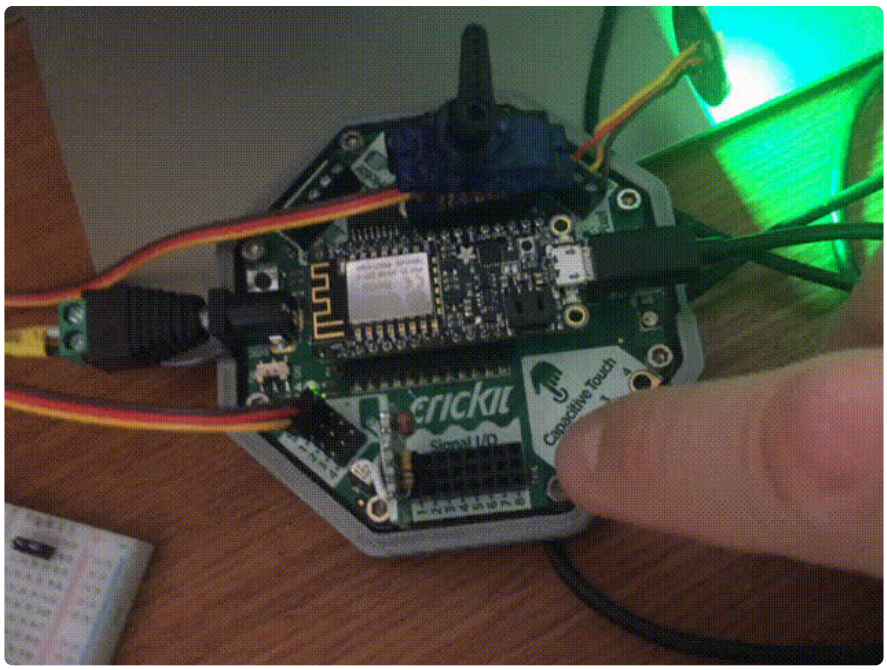


We can do something similar with the light and NeoPixel feeds. In all we have four actions:

```
If crickit.touch_0 is equal to "0" then set servo1-control to "180".
If crickit.touch_0 is equal to "1" then set servo1-control to "0".
If light is less than or equal to "500" then set neopixel-control to "#00FF00".
If light is greater than "500" then set neopixel-control to "#FF0000".
```

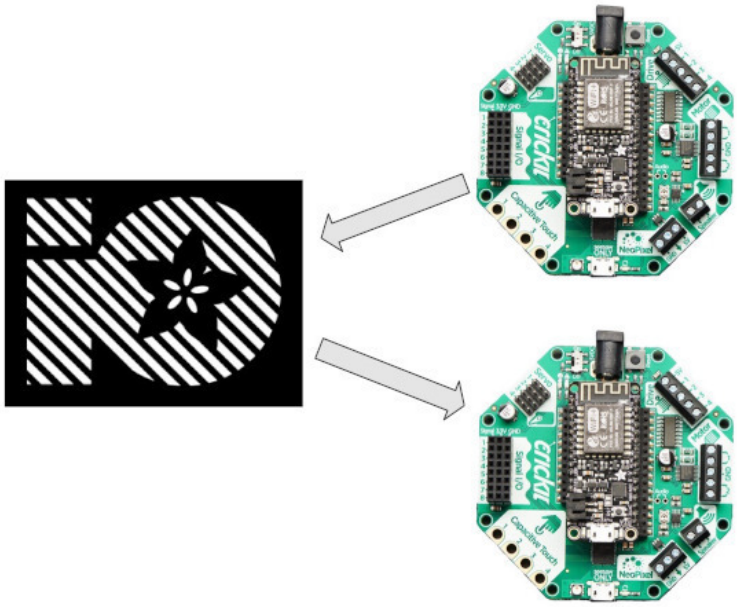
With these in place, load up the code below in **remote_monitor.ino**. This program has both the touch input code and the servo/NeoPixel output code.

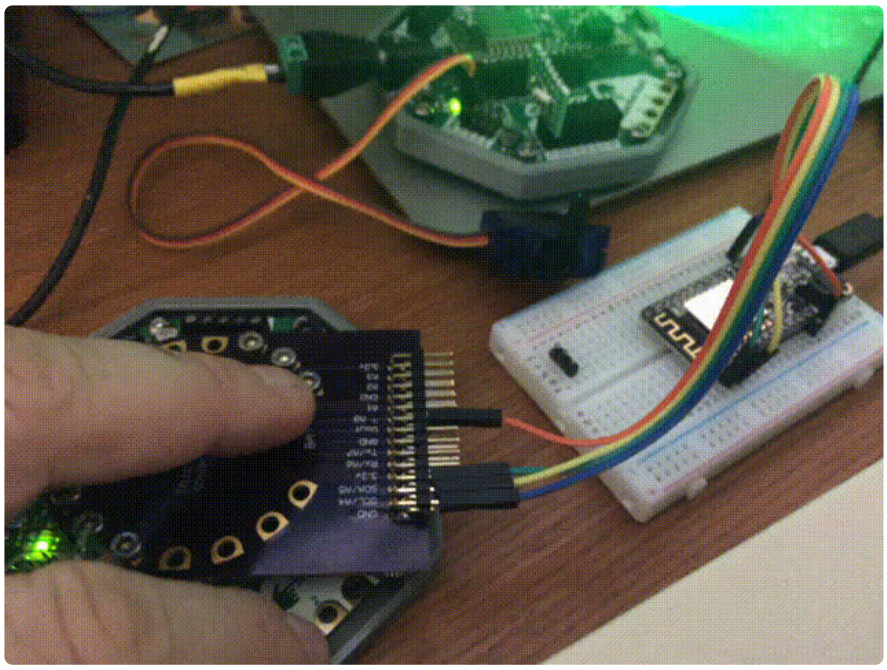
When the sketch is running, you should be able to touch and release the capacitive touch sensor 1 to control the servo, as well as vary the lighting on the photoresistor to control the color of the NeoPixels.



Multiple Data Sources

Things get interesting when you take into account that multiple devices can connect to Adafruit IO and publish to feeds. If we build another Feather + Crickit and put a input only version of the sketch on it (below) we can have it publish readings that the actions then use. The original Feather + Crickit consumes the output from the actions. Conceivably (possibly limited by the data limits on your Adafruit IO account) you could have any number of nodes feeding data to the system.





```
// SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
//
// SPDX-License-Identifier: MIT

// Crickit + Adafruit IO Publish Example
//
// Adafruit invests time and resources providing this open source code.
// Please support Adafruit and open source hardware by purchasing
// products from Adafruit!
//
// Written by Dave Astels for Adafruit Industries
// Copyright (c) 2018 Adafruit Industries
// Licensed under the MIT license.
//
// All text above must be included in any redistribution.

#include "config.h"
```

```
#include <Adafruit_Crickit.h>

#define CAPTOUCH_THRESH 500

#define IO_LOOP_DELAY (1000)
unsigned long lastUpdate = 0;

// set up the feeds
AdafruitIO_Feed *light;
uint16_t last_reported_light = 0;

AdafruitIO_Feed *touch;
boolean last_touch = false;

// set up the Crickit

Adafruit_Crickit crickit;

void setup_feeds()
{
  light = io.feed("crickit.light");
  touch = io.feed("crickit.touch-0");
}


void setup()
{
  setup_feeds();
  Serial.println("Feeds set up");

  // start the serial connection
  Serial.begin(115200);

  // wait for serial monitor to open
  while(! Serial);

  Serial.println("Connecting to Adafruit IO");

  // connect to io.adafruit.com
  io.connect();

  // wait for a connection
  while(io.status() < AIO_CONNECTED) {
    Serial.print(".");
    //    Serial.println(io.statusText());
    delay(500);
  }

  // we are connected
  Serial.println();
  Serial.println(io.statusText());

  if (!crickit.begin()) {
    Serial.println("Error starting Crickit!");
    while(1);
  } else {
    Serial.println("Crickit started");
  }

  Serial.println("setup complete");
}


void loop()
{

  // io.run(); is required for all sketches.
  // it should always be present at the top of your loop
  // function. it keeps the client connected to
```

```
  // io.adafruit.com, and processes any incoming data.
  io.run();

  if (millis() > (lastUpdate + IO_LOOP_DELAY)) {

    uint16_t light_level = crickit.analogRead(CRICKIT_SIGNAL1);
    uint16_t light_delta = abs(light_level - last_reported_light);

    if (light_delta > 10) {
      light->save(light_level);
      last_reported_light = light_level;
      Serial.print("Sending ");
    }
    Serial.print("Light: ");
    Serial.println(light_level);

    uint16_t val = crickit.touchRead(0);

    if (val >= CAPTOUCH_THRESH && !last_touch) {
      touch->save(1);
      last_touch = true;
      Serial.println("CT 0 touched.");
    } else if (val < CAPTOUCH_THRESH && last_touch) {
      touch->save(0);
      last_touch = false;
      Serial.println("CT 0 released.");
    }

    // after publishing, store the current time
    lastUpdate = millis();
  }

}
```

Robot Applications

Crickit is designed to make it easier to build robotics projects. See Ladyada's Crickit Manifesto (https://adafru.it/BPS) for the product's background story.

When used with a WiFi capable Feather like the HUZZAH ESP8266, your robot project can communicate with Adafruit IO. If you build a controller that can do so as well, it's easy to see how you could use Adafruit IO to link the two, providing remote control of your project.

A bit more involved idea would be to have a more capable computer (like your desktop, laptop, or even a cloud based instance) read data from feeds that your robot publishes to, and send back commands on other feeds. What software could be doing that on the bigger computer? It could be something as simple as a small Python script, or as elaborate as something driven by deep learning and AI.

Adafruit IO lets you do all that, while at the same time storing the data that gets sent to feeds. For instance you could have soil-condition monitoring devices throughout a garden that send measurements to Adafruit IO. Using actions, that data could be used to send commands back to the devices to control water valves and the like. However

you would be collecting historical data at the same time that can be used to perform higher level analysis.