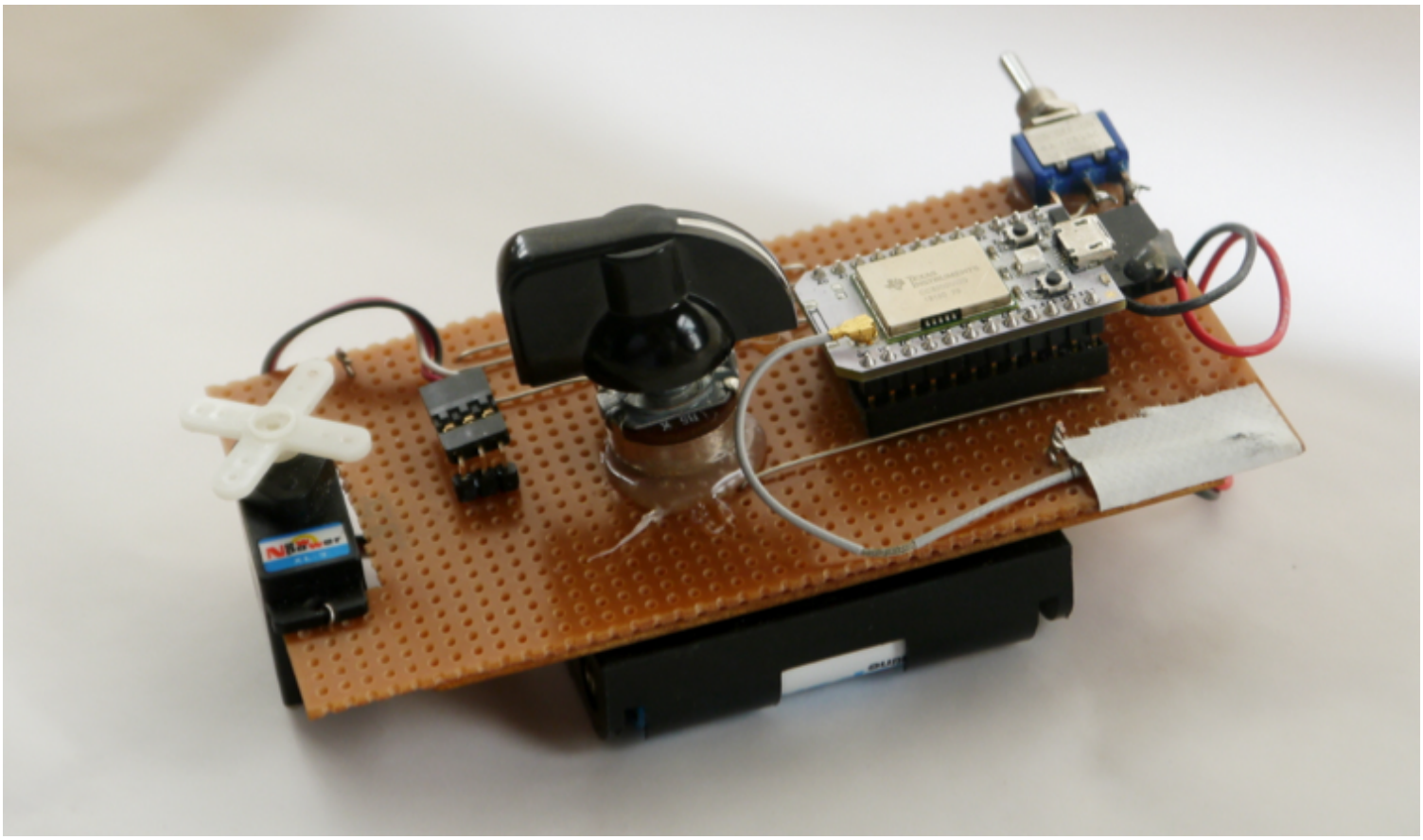
MIKE KARLINER

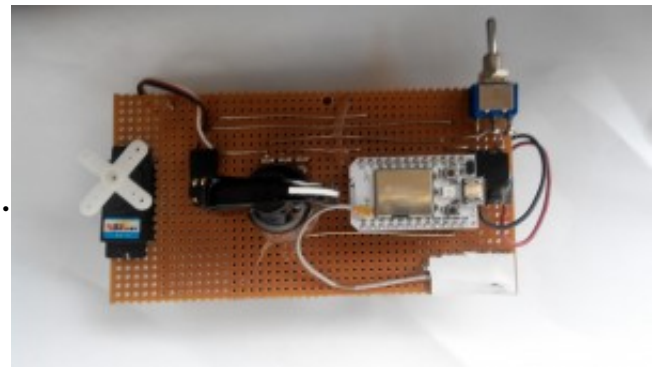
APRIL 29, 2015

Using a Spark Core with ThingStudio



Last week, I was walking around Maker Faire UK with my walkabout ThingStudio demo. The hardware was based around the Spark Core Wifi dev board. I thought I'd go through the demo hardware and software so you can see how to use Spark with ThingStudio. This recipe should also work with the Spark Photon and Electron when they come out, especially if Spark send us some early units to test (hint, hint).

Let's start with the hardware, this was pretty basic, just a potentiometer on A0, a servo attached to A1, and the RGB led on the board. This collection of bits was just intended to demonstrate the UI features of ThingStudio clearly, rather than to do anything in particular. A pack of AA's on the back made the whole thing pocketable

.  (http://blog.thingstud.io/wp-content/uploads/pocketspark.jpg)

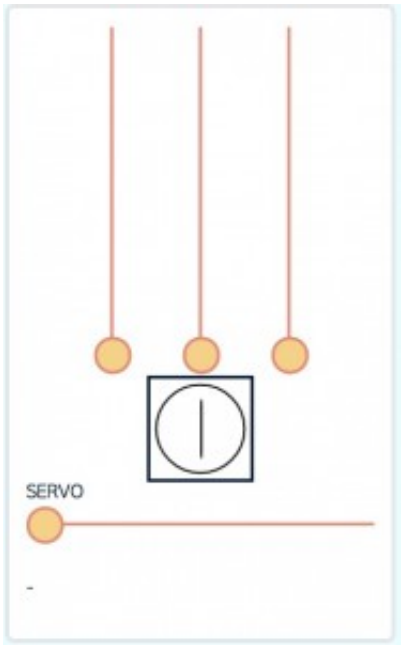
So far, so obvious.

What does it do?

The goal of this demo is to show two-way, real-time control and monitoring of IoT devices.

The three vertical sliders control the red, green and blue of the RGB led on the SparkCore.

The circular indicator moves when you turn the potentiometer, and the horizontal slider moves the servo. The UI is shown below.



Programming the Spark

As always, the prerequisite for working with ThingStudio is to have some way of using the MQTT protocol. Fortunately, the nice people at Spark have already provided an MQTT library, so we are in good shape. You can download the Spark example from github here: https://github.com/mkarliner/ThingStudioExamples.git. Let's work through the code pretty much line by line.

```
1  // This #include statement was automatically added by the Spark IDE.
2  #include "MQTT/MQTT.h"
3
4  Servo myservo;
5
6  void callback(char* topic, byte* payload, unsigned int length);
7  MQTT client("mqtt.modern-industry.com", 1883, callback);
8
9  // receive message
10
11
12 int red=0;
13 int blue=0;
14 int green=0;
```

At the top of the file we declare the Servo the signature for the MQTT callback (more about that later), and the connection details for the MQTT client.

Note the top lines, inserted by the Spark IDE. To include the MQTT library, bring up the libraries tab (the bookmark-like icon fourth from the bottom of the left hand side), search for MQTT, and add it to the project.

We also declare some global ints to store the red, green and blue values as they come in from the ThingStudio UI.

Next, skip on to line 52 where the setup function is.

```
1  void setup() {
2      RGB.control(true);
3       RGB.color(0, 0, 255);
4      Serial.begin(9600);
5      delay(10000);
6      Serial.println("Hello Mike");
7      myservo.attach(A1);
8      pinMode(A0, INPUT);
9
10     // connect to the server
11     client.connect("sparkclient", "guest", "guest");
12
13     // publish/subscribe
14     if (client.isConnected()) {
15         Serial.println("Connected");
16         RGB.color(0, 255, 0);
17         client.subscribe("/mike/sparkred");
18         client.subscribe("/mike/sparkblue");
19         client.subscribe("/mike/sparkgreen");
20         client.subscribe("/mike/sparkservo");
21     }
22 }
```

We start by initialising the RGB led, the Serial port (for debugging) the Servo and the potentiometer input pin. Next, we actually attempt to connect to the MQTT broker. Finally, we wait for an MQTT connection, and when it is valid, we set the led color to green (again for debugging), and subscribe to the topics that our ThingStudio UI will send commands on.

Note: For this demo, I was using our open MQTT broker at mqtt.thingstud.io. In general, you should move to your own broker, but if you are going to use our broker, it's good practice to prefix your topics, by a unique string, like your username, so you don't accidentally get messages from someone else who is also using a topic like "/test".

OK, now let's turn to line 16, where the MQTT callback is declared, and where most of the actual work is done.

```
1  void callback(char* intopic, byte* payload, unsigned int length) {
2      //Payloads are JSON strings, so increment past the first
3      //quote to get to the first message character.
4      payload++;
5      //And now the payload is 1 character shorter, so copy
6      // one less character.
7      char p[length + 1];
8      memcpy(p, payload, length-1);
9      p[length] = NULL;
10
11     char t[strlen(intopic)+1];
12     memcpy(t, intopic, strlen(intopic));
13     t[strlen(intopic)] = NULL;
14
15
16     String message(p);
17     String topic(t);
18     Serial.println(topic);
19     Serial.println(message);
```

```
20      if(topic.equals("/mike/sparkred")) {
21          red = atoi(p);
22          Serial.println("red");
23      }
24
25      else if (topic.equals("/mike/sparkblue"))
26          blue = atoi(p);
27      else if (topic.equals("/mike/sparkgreen"))
28          green = atoi(p);
29      else if (topic.equals("/mike/sparkservo"))
30          myservo.write(atoi(p));
31      else
32          RGB.color(255, 255, 255);
33      RGB.color(red, green, blue);
34      delay(10);
35  }
```

This function is called everytime a message is received from ThingStudio. We've just made the change that all messages to and from ThingStudio are in JSON format, which means that simple values are enclosed in quotes. You could fire up a full JSON parse to do this, and the Arduino/Spark one is available, but for now, its overkill, so all we do is, in the very first line of the callback, increment the char pointer to the payload by one, thereby stepping past the first quote mark.

The next few lines convert the 'C' strings to C++ *Strings*, which gives us the *equals* method.

We now test the topic of the incoming message.

For each of the RGB topics, we convert the R,G or B value and set one of the globals.

If the topic is the servo topic, we convert the position and write to the servo.

That's incoming dealt with. Now to the output, publishing side, from line 77.

```
1   int oldval =0;
```

```
2
3  void loop() {
4      int val;
5      char buf[80];
6
7      if (client.isConnected()) {
8          client.loop();
9          //  Serial.println("LOOP");
10     }
11
12     val = analogRead(A0);
13     int diff = abs(val - oldval);
14         if(diff &gt; 10) {
15             Serial.println(val);
16             sprintf(buf, "%d", val);
17             client.publish("/mike/sparkpot", buf);
18             oldval = val;
19         }
20         delay(10);
21
22 }
```

The first few lines are just a loop to keep handling incoming messages from MQTT/ThingStudio.

Next, we are going to send the value of the potentiometer. There are two issues here.

Firstly, there is always some noise on sensors like potentiometers, thermocouples, etc. and if we just tested to see if the value had changed before sending the value, we'd been sending thousands of messages a second to the browser, which would probably break it. So, we keep track of the previous readings and only send a message if the the difference between the two is greater than the noise level of the pot, which is typically in the region of 3 or 4.

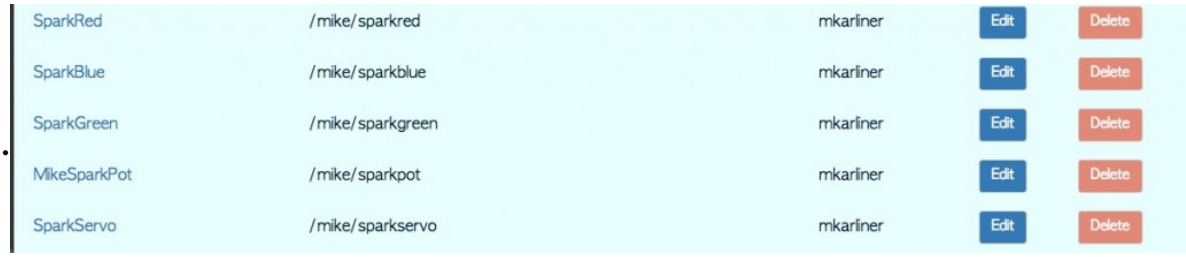
The second issue is just one of a bit of safety if you happen to forget to connect the pot or connect it wrong. In this case the value of the ADC readings will swing wildly all over the place because the line would be floating, so we put in a small delay at the

end of the function to limit the number of callbacks to 100/sec (10ms delay).

Designing the UI

I'll assume that you have a working MQTT broker, or you are using our open one (if not, you can get one and learn how to set it up here (http://blog.thingstud.io/recipes/how-to-use-a-plain-ol-arduino-with-thingstudio/)).

First create the feeds for your app. When you have finished, your list of feeds should look like this, but with your username inserted in the topics



(http://blog.thingstud.io/wp-content/uploads/pocketSparkFeeds.jpg)

Next, let's review the screen code.

```
<div class="rotate-90" >

    <input class="knob-slider" type="range" data-feed="SparkRed" data-continuous="true"  min=

<input class="knob-slider" type="range" data-feed="SparkGreen" data-continuous="true"    min=

<input class="knob-slider" type="range" data-feed="SparkBlue" data-continuous="true"    min="

</div>


<svg width="100" height="100" viewBox="0 0 50 50">
    <circle cx="25" cy="25" r="22" stroke="black" stroke-width="1" fill="white"/>
    <rect id="hour"  transform="rotate({{message "MikeSparkPot" }} 25 25) " x="25" y="10" wid
</svg>
```

```
16
17
18 <div>SERVO
19 <input type="range" class="knob-slider" data-feed="SparkServo" data-continuous="true"  min="0
20 </div>
```

The RGB sliders are enclosed in a a div with a CSS class that rotates them through 90 degrees to make them vertical.

Each of the of the sliders has a maximum value of 255 (for the RGB values), and the data attributes data-feed, which specifies which feed values go to and data-continuous="true", which means that they will output a continuous stream of MQTT messages while the slider is being moved.

The next section gives us our rudimentary SVG dial. It specifies an SVG viewport and draws a circle and a rectangle within it. When we receive messages on the MikeSparkPot feed, the rectangle will be rotated by the number of degrees specified by the messsage.

Finally, the servo slider works much the same what as the RGB slide , but outputting to the SparkServo feed.

A bit of style

I also made a custom Theme for this demo which does the 90 rotate and styles the SVG viewport here.

```
1 svg {
2     border: solid;
3             margin-right: auto;
4     margin-left: auto;
5     display: block;
6
7 }
8
```

```
9
10 .rotate-90 {
11     transform: rotate(-90deg);
12     margin-top: 4em;
13 }
```

And that's it!

Have fun, and give some feedback on the forum or comments here.

About Mike Karliner (http://blog.thingstud.io/author/mkarliner/)

I make stuff.

TAGS: mqtt (http://blog.thingstud.io/tag/mqtt/) - spark (http://blog.thingstud.io/tag/spark/)

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Comment

Post Comment

LINKS

Home (http://www.thingstud.io)
Blog (http://blog.thingstud.io)
Forum (http://forum.thingstud.io)
Sign In (http://www.thingstud.io/sign-in/)

ABOUT THINGSTUDIO

Create real-time user interfaces for the Internet of Things. Learn more about ThingStudio (http://www.thingstud.io).