Lesson 6 –
Mobile app development (part 2)

- S.P. Chong

Objectives

- In this lesson, you will learn to create a **mobile app**. for an **Android** phone.

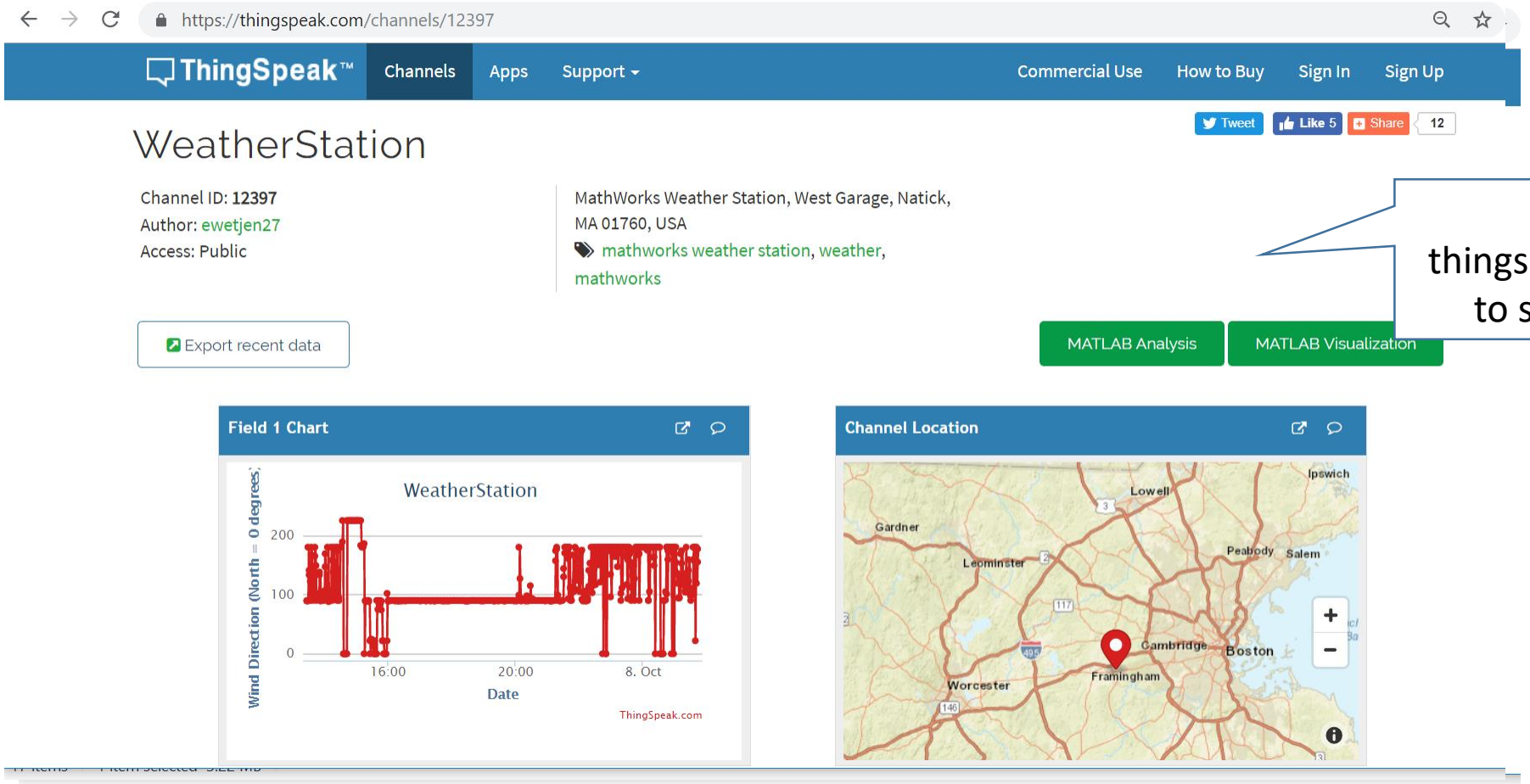
- The programming will be "**graphical**", which means you don't really need to learn a new programming language.

- You will learn to create **user interface**, handle **events** and use selected **phone features** such as location sensor.

- The example apps will allow a user to **register & login**, to do **remote monitoring & control**.

Monitoring sensors (via Thingspeak)

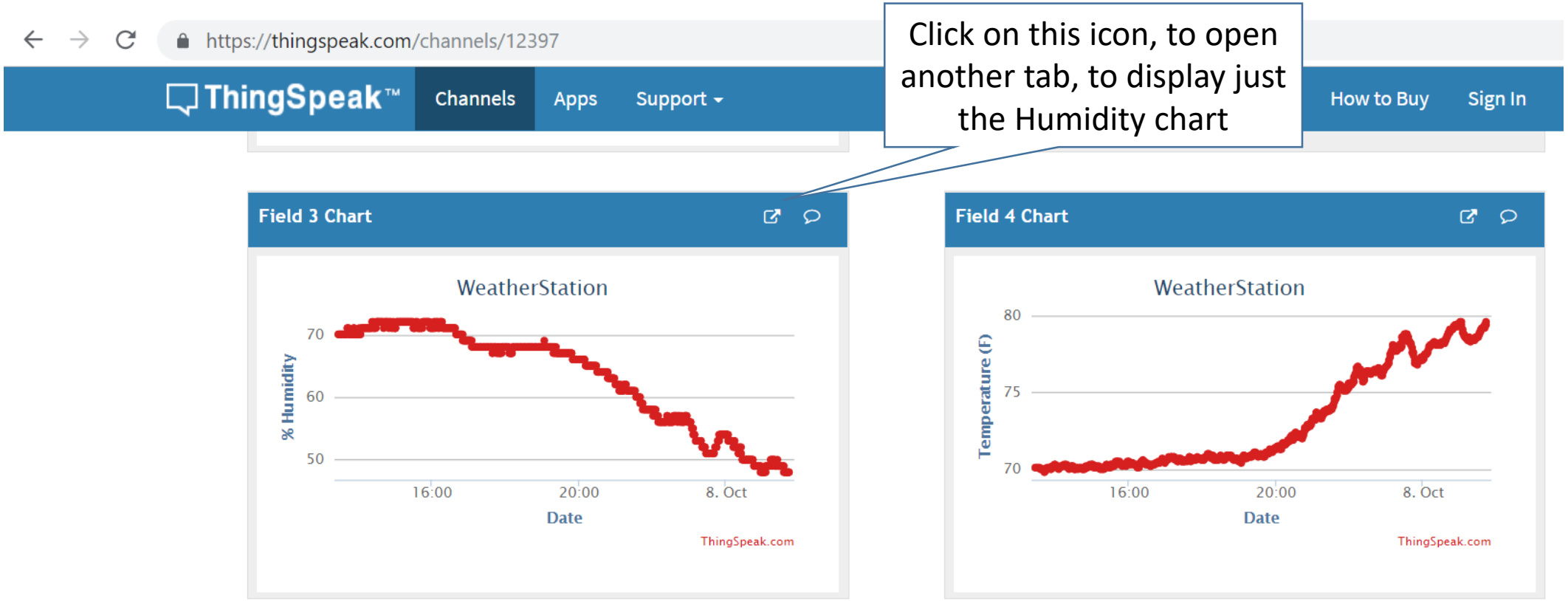
- Let's develop an app to **monitor** the temperature & humidity **sensor** data uploaded to a **Thingspeak** channel.
- We will use the data from the **public** Thingspeak channel 12397: a "weather station" in US.



Browse to thingspeak.com/channels/12397 to see this weather station.

Monitoring sensors (via Thingspeak) (cont.)

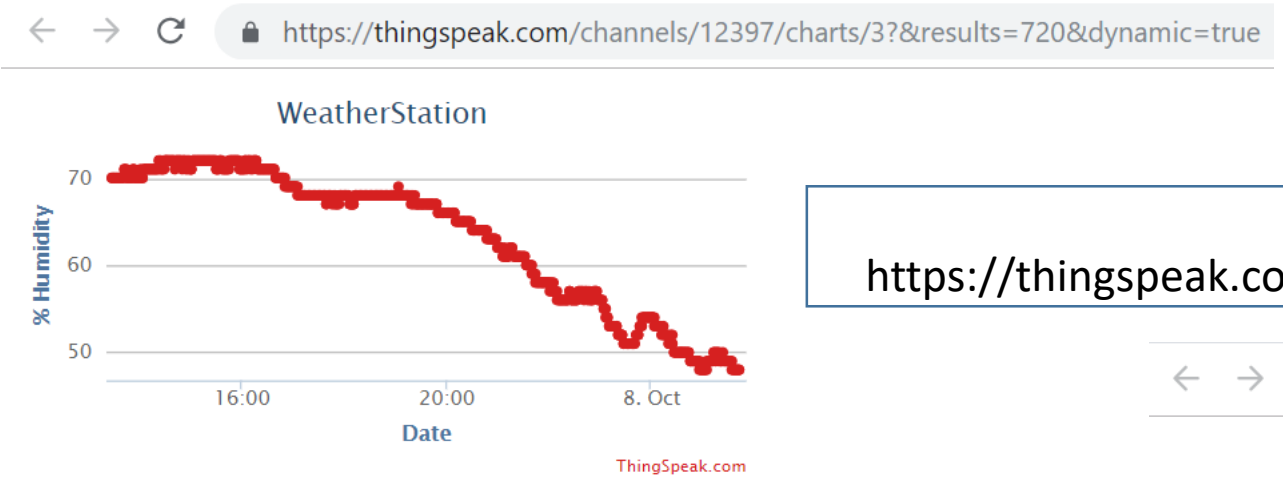
- We need the **url's** of the individual **charts**, to develop the app.



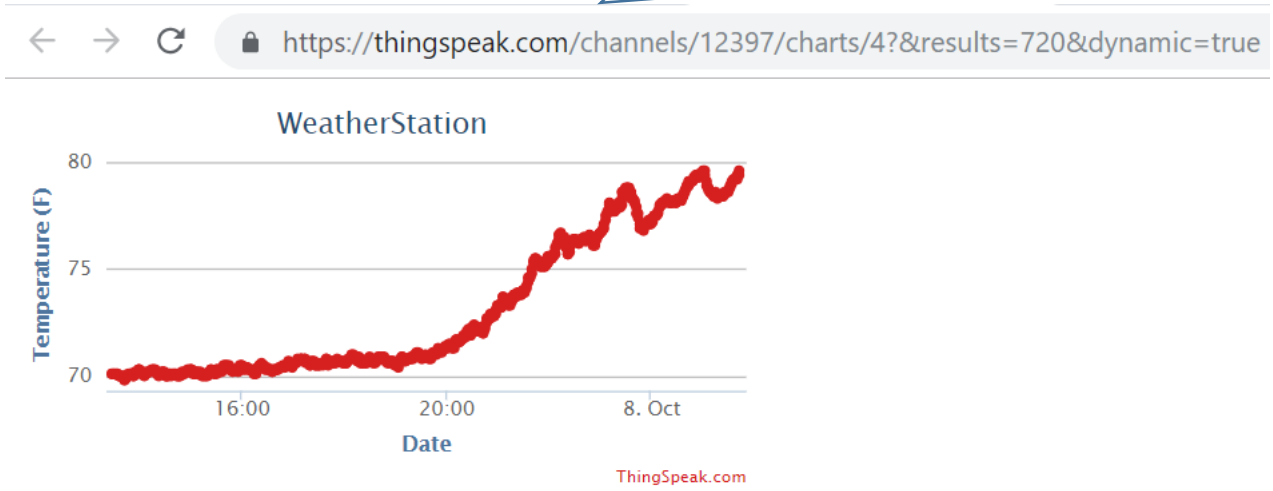
Monitoring sensors (via Thingspeak) (cont.)

url of the Humidity chart:
https://thingspeak.com/channels/12397/charts/3?&results=720&dynamic=true

results=720 means displays the last 720 data points in the chart.

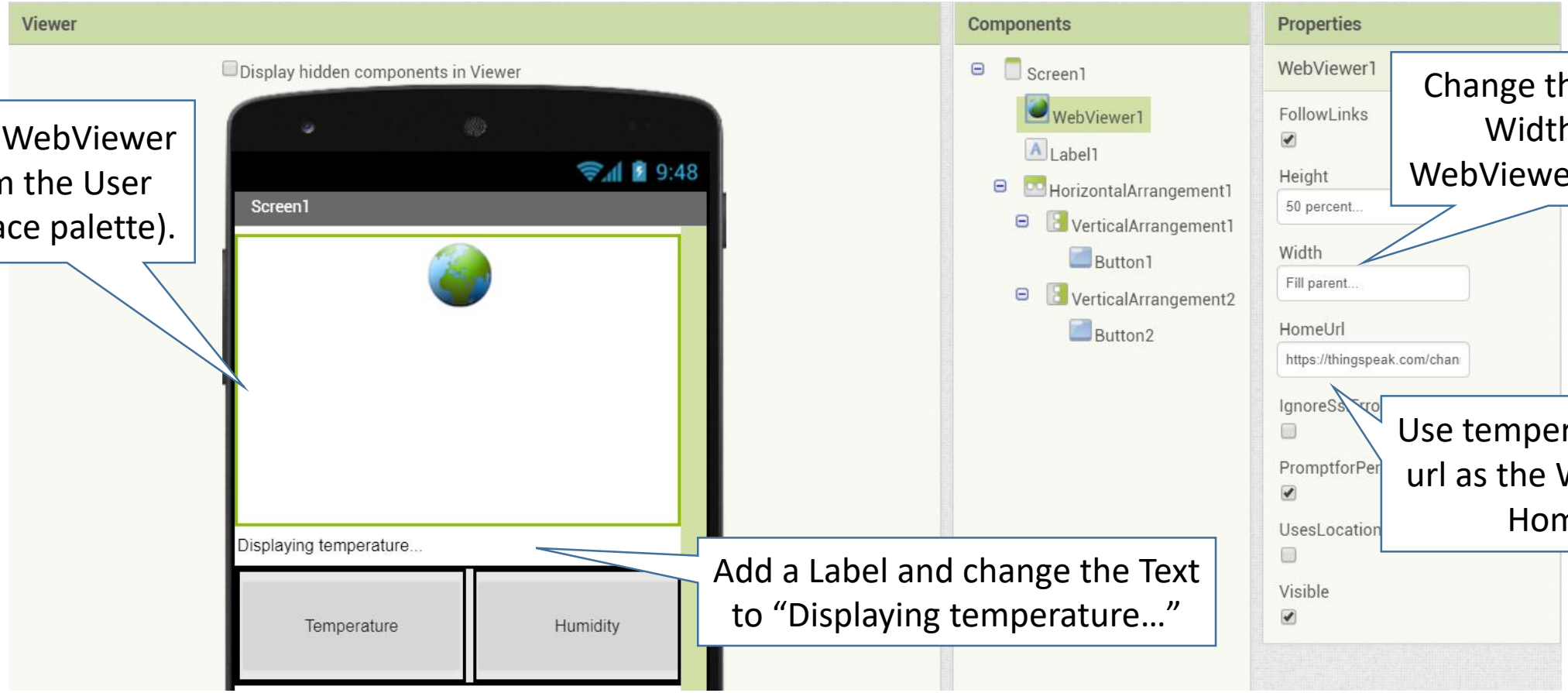


url of the Temperature chart:
https://thingspeak.com/channels/12397/charts/4?&results=720&dynamic=true



Monitoring sensors (via Thingspeak) (cont.)

- Use a **WebViewer** to display a chart. Use 2 buttons to select which chart to display.



Add a WebViewer (from the User Interface palette).

Add a Label and change the Text to "Displaying temperature..."

Change the Height & Width of the WebViewer as follows:

Use temperature chart's url as the WebViewer's HomeUrl.

Monitoring sensors (via Thingspeak) (cont.)

- Use **Horizontal & Vertical Arrangements** to position/layout the buttons.

Viewer

Display hidden components in Viewer

9:48

Screen1

Displaying temperature...

Temperature

Humidity

Components

Screen1

WebViewer1

Label1

HorizontalArrangement1

VerticalArrangement1

Button1

VerticalArrangement2

Button2

Properties

HorizontalArrangement1

AlignHorizontal

Left : 1

AlignVertical

Top : 1

BackgroundColor

Default

Height

20 percent...

Width

Fill parent...

Image

None...

Visible

The VerticalArrangements are placed inside the HorizontalArrangement.

Change the Heights & Widths as follows:

Components

Screen1

WebViewer1

Label1

HorizontalArrangement1

VerticalArrangement1

Button1

VerticalArrangement2

Button2

Properties

VerticalArrangement1

AlignHorizontal

Left : 1

AlignVertical

Top : 1

BackgroundColor

Default

Height

Fill parent...

Width

50 percent...

Image

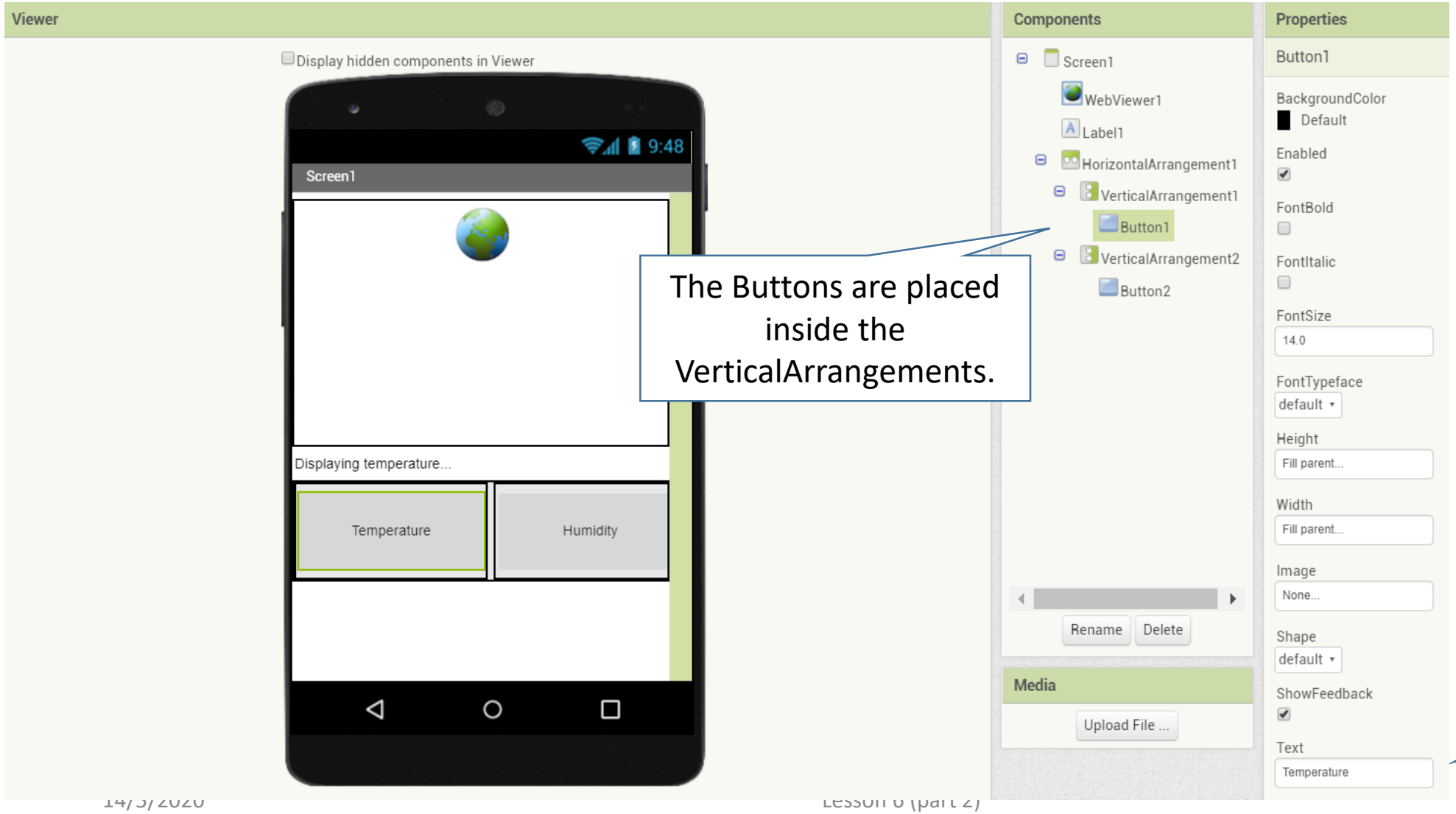
None...

Visible

Do likewise for VerticalArrangement2.

Think of a Horizontal Arrangement as a “row”, and a Vertical Arrangement as a “column”.

Monitoring sensors (via Thingspeak) (cont.)



The Buttons are placed inside the VerticalArrangements.

Change the Buttons' Texts accordingly.

Monitoring sensors (via Thingspeak) (cont.)

- The **Blocks** view is relatively simple: when a button is clicked, change the WebViewer's url (and the Label's Text) accordingly.

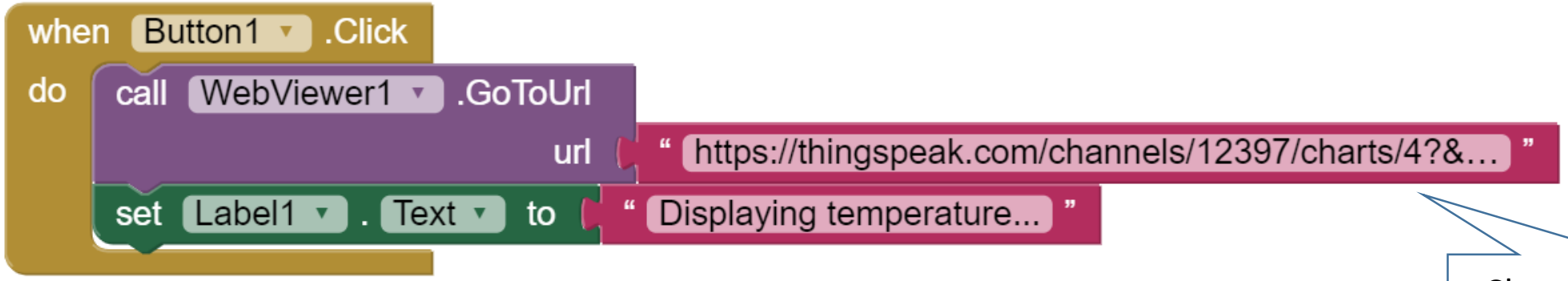
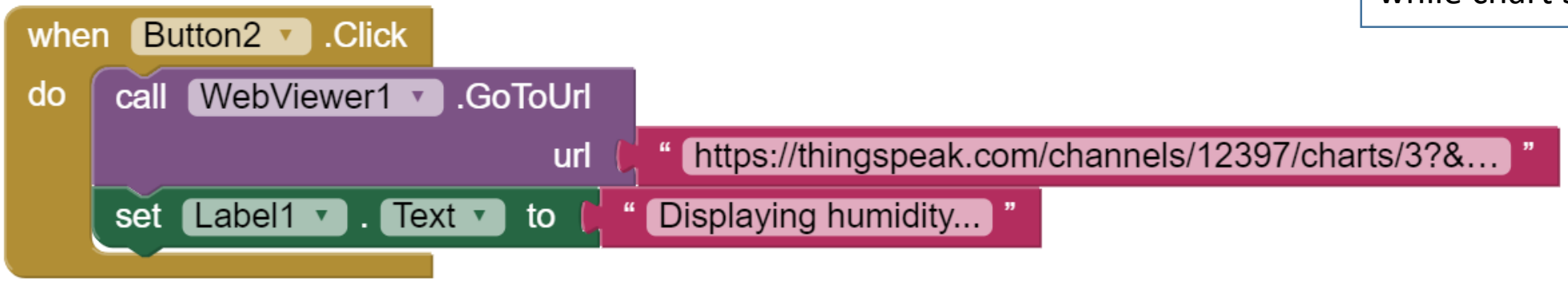
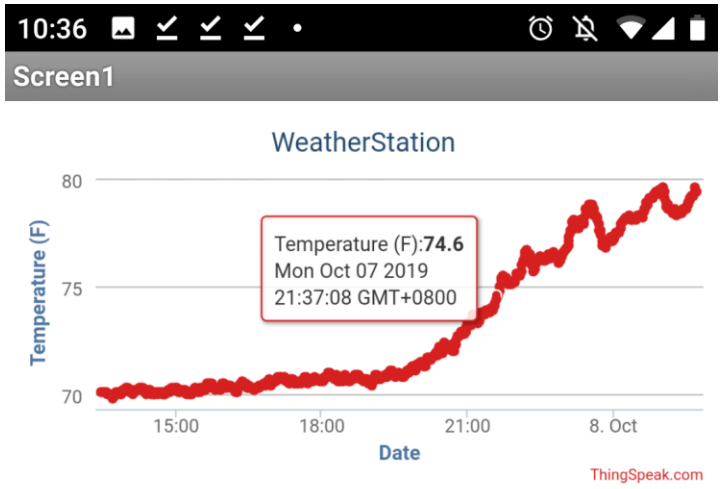


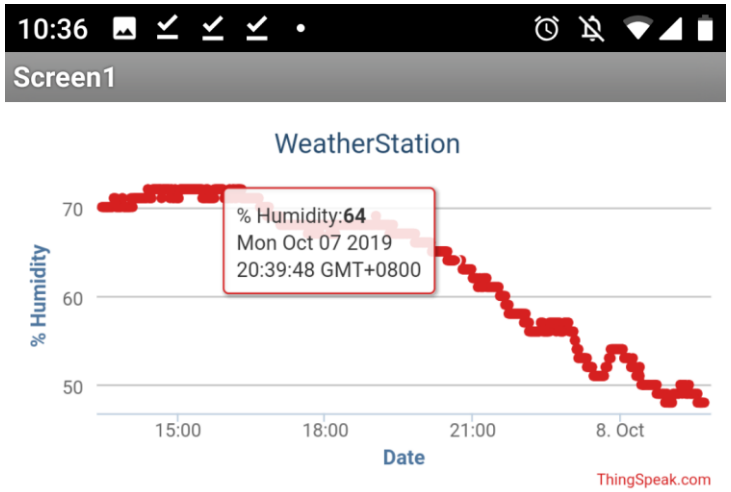
Chart 4 is temperature while chart 3 is humidity.



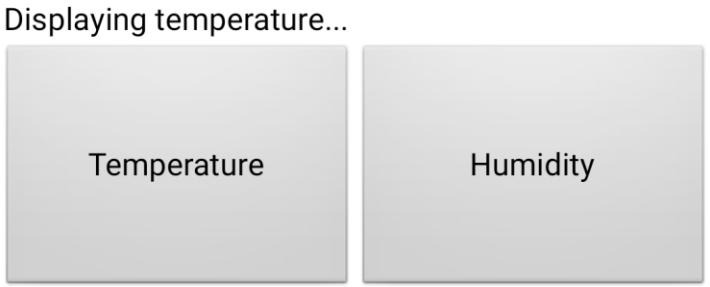
Monitoring sensors (via Thingspeak) (cont.)

- Once you are done, test the app in your Android phone as before:





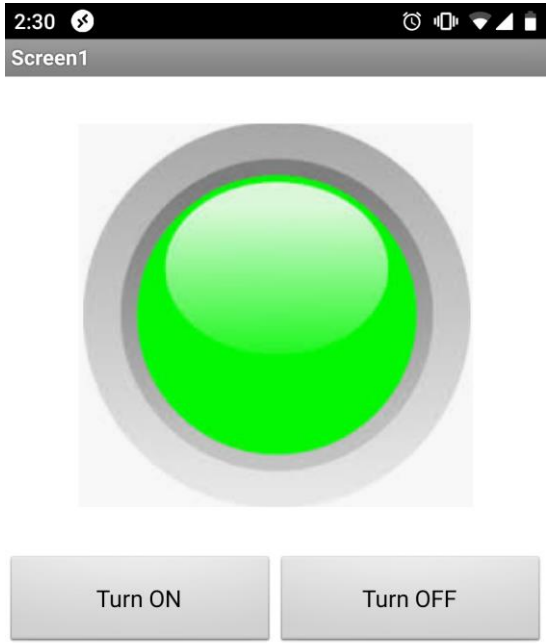
If you pinch the phone screen, you will see these icons which allow zooming.



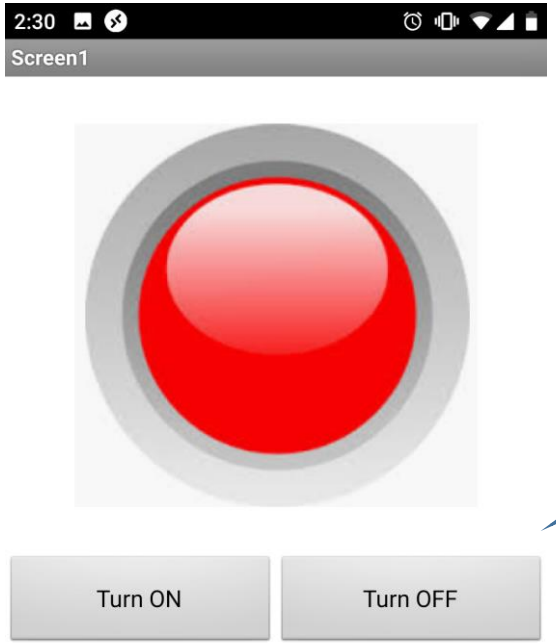
Use the buttons to select which chart to display.

Controlling actuators (via Thingspeak)

- Let's develop an app to **control** an actuator (e.g. a motor) / an indicator (e.g. an LED) via a **Thingspeak** channel.
- The user interface looks like the following. The components used are shown on the next slide.



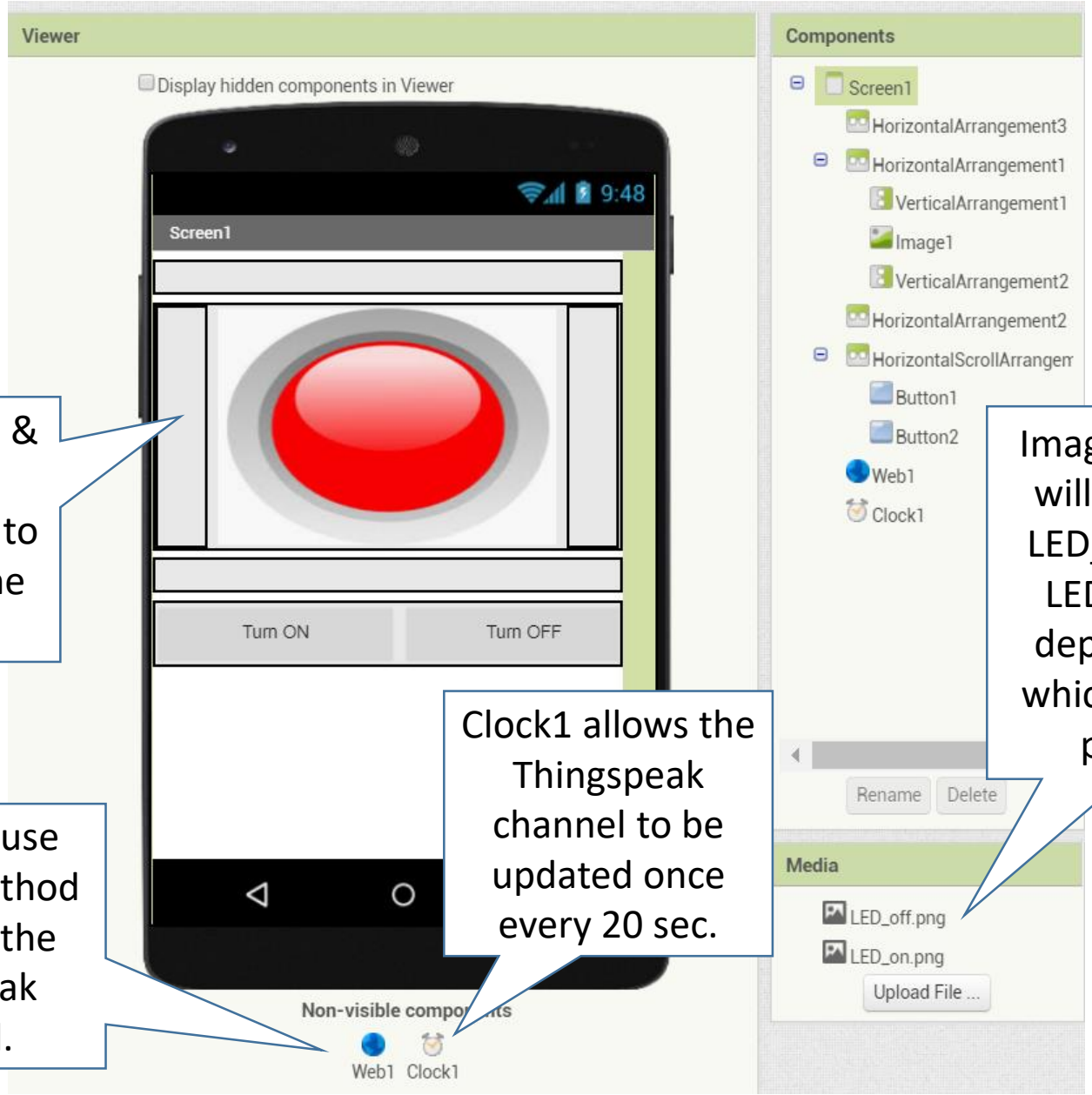
Press the Turn ON button to turn the LED on (RED).



Press the Turn OFF button to turn the LED off (GREEN).

Controlling actuators (via Thingspeak) (cont.)

- Note that this app only allows the user to send on & off commands to the Thingspeak channel.
- An Arduino UNO must be programmed to read from this Thingspeak channel and to turn the physical LED on / off.



Use Horizontal & Vertical Arrangements to improve on the layout.

Web1 will use the GET method to update the Thingspeak channel.

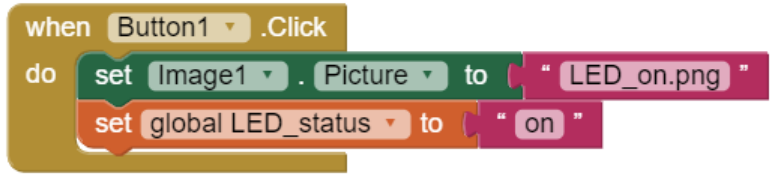
Clock1 allows the Thingspeak channel to be updated once every 20 sec.

Image1' Picture will change to LED_off.png or LED_on.png, depending on which button is pressed.

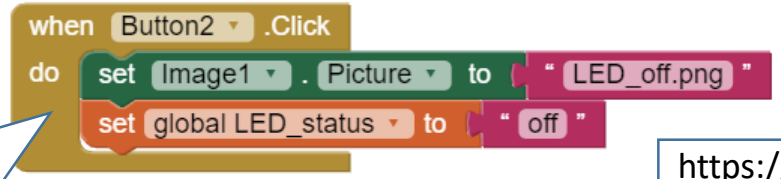
Controlling actuators (via Thingspeak) (cont.)

A global variable to store the LED status, i.e. whether it should be on or off.

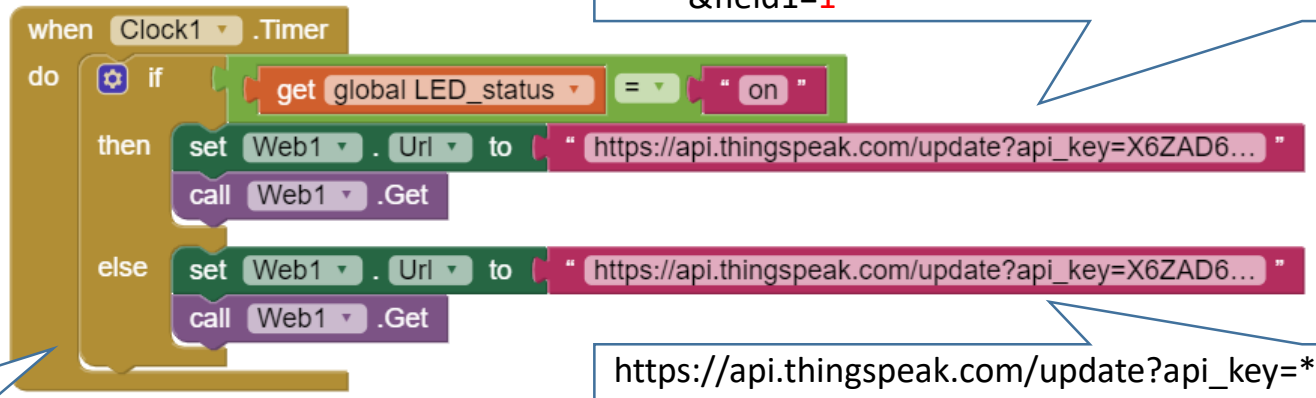
When Button1 (Turn ON) is clicked, Image1's Picture & LED_status are both changed.



Likewise, when Button2 is clicked.



https://api.thingspeak.com/update?api_key=**********&field1=1



https://api.thingspeak.com/update?api_key=**********&field1=0

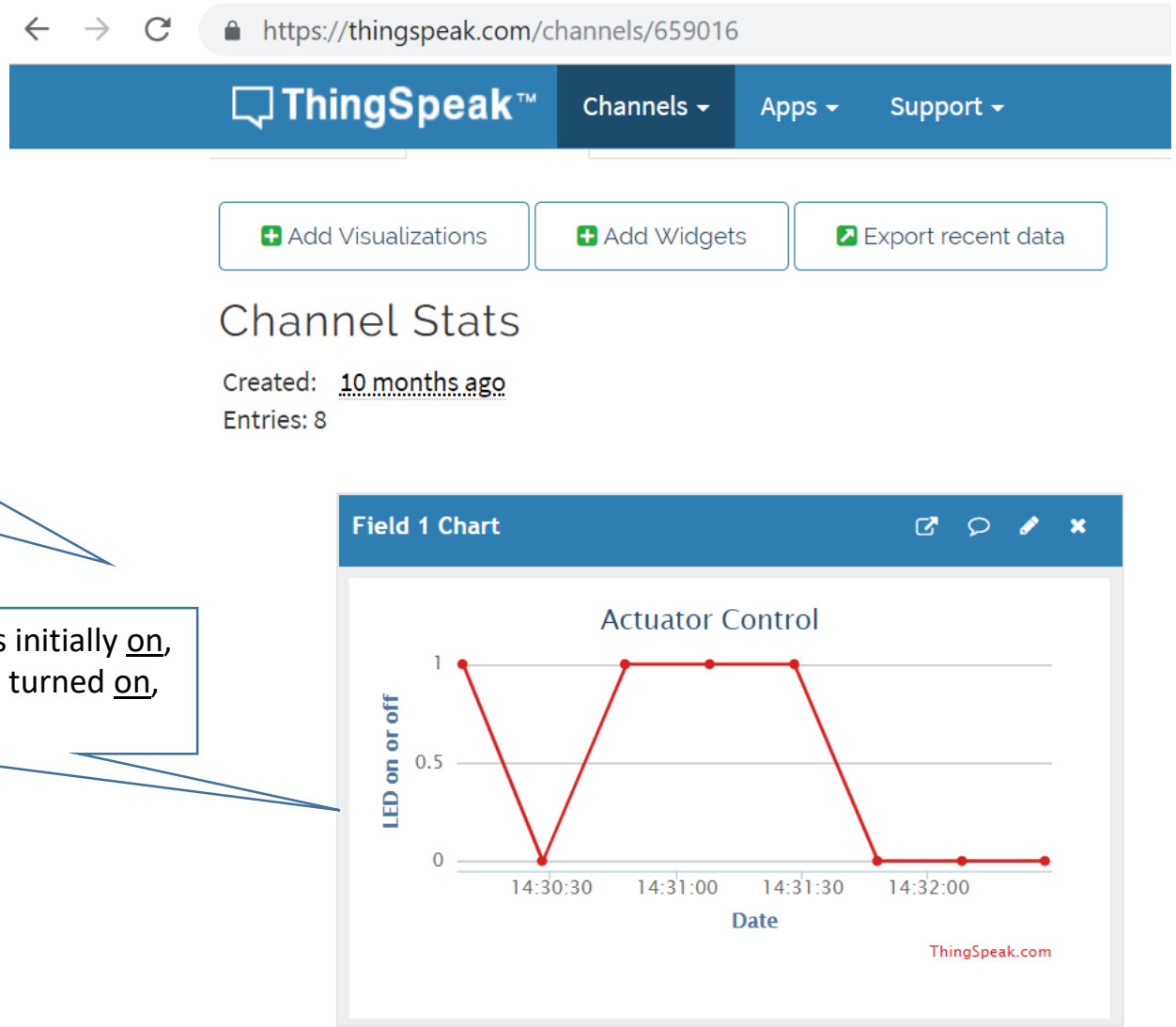
Every 20 seconds, the LED status is sent to the Thingspeak channel.

Controlling actuators (via Thingspeak) (cont.)

- Sample run.

LED status is sent once every 20 sec. as there is a limit to how regularly a channel can be updated.

For the period shown, the LED was initially on, turned off after a while, and then turned on, and then off again.



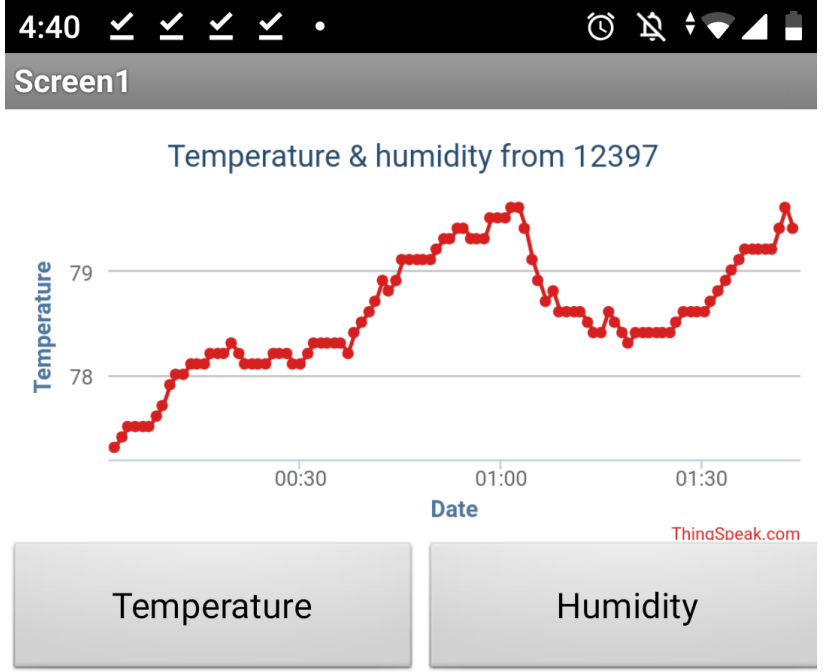
We will next learn how an Arduino UNO can be programmed to read from this Thingspeak channel and to turn the physical LED on / off.

Lab Exercises

- Exercise 6.4 – Monitoring Sensors (enhanced)
- Exercise 6.5 – Controlling Actuators (enhanced)

Exercise 6.4 – Monitoring Sensors

- We will enhance the “Monitoring Sensor via Thingspeak” app in the lecture:
  - to display the **latest** temperature & humidity **values** as labels, and
  - to alert the viewer, when a pre-defined threshold has been exceeded.



{"channel":{"id":400360,"name":"Temperature \u0026 humidity from 12397","description":"For App Inventor use","latitude":"0.0","longitude":"0.0","field1":"Temperature","field2":"Humidity","created_at":"2018-01-10T03:46:15Z","updated_at":"2019-10-18T08:36:20Z","last_entry_id":100},"feeds":[{"created_at":"2019-10-07T17:43:45Z","entry_id":100,"field1":"79.4","field2":"48"}]}

Temperature in Fahrenheit, Humidity in %

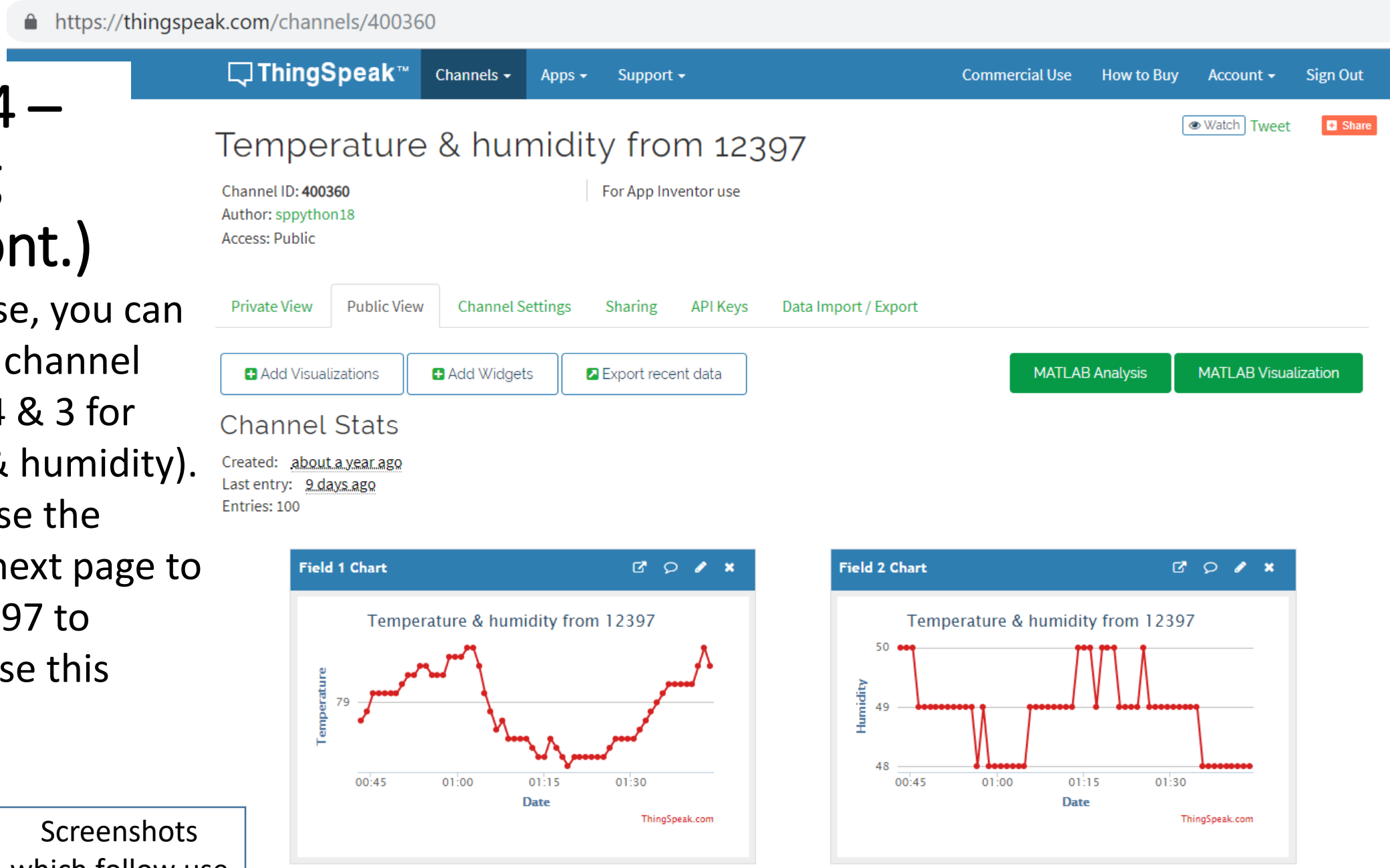
When Temperature / Humidity exceeds pre-defined threshold e.g. 75°F / 50%, it will be shown in red.

You can explore other forms of alert, e.g. sound effect, animation, or even sending a message or email.

GET response for debugging (more on this later).

Exercise 6.4 – Monitoring Sensors (cont.)

- For this exercise, you can use the public channel 12397 (fields 4 & 3 for temperature & humidity).
- You can also use the MatLab code next page to copy from 12397 to 400360, and use this instead.



Screenshots which follow use 400360.

Exercise 6.4 – Monitoring Sensors (cont.)

Apps / MATLAB Analysis / Copy temperature & humidity from 12397 / Edit

Name

Copy temperature & humidity from 12397

MatLab Analysis to copy data from a public channel into our own channel.

MATLAB Code

```
% Save read & write channel IDs and Write API Key into variables.
readChId = 12397;
writeChId = 400360;
writeKey = '4QQATSBOSV6V2TC2';

% Read latest temperature & humidity data with timestamp
%  from public WeatherStation channel into variables.
[temp,time] = thingSpeakRead(readChId,'Fields',4,'NumPoints',100);
humi = thingSpeakRead(readChId,'Fields',3,'NumPoints',100);

% Write data with timestamp to your own channel.
thingSpeakWrite(writeChId,[temp,humi],'Fields',[1,2],...
'TimeStamps',time,'Writekey',writeKey);
```

Read the last 100 points.

For Public Channel 12397, Field 4 is temperature, Field 3 is humidity.

For our Channel 400360, Field 1 is temperature, Field 2 is humidity.

Save and Run | Save*

Save and Run to update the charts.

Help

My Channels | Documentation

New Channel

Channel Info

Name: Temperature & humidity from 12397
Channel ID: 400360
Access: Public
Read API Key: 5JEAMOCZ11WYQ7U9
Write API Key: 4QQATSBOSV6V2TC2
Fields:
1: Temperature
2: Humidity

Exercise 6.4 – Monitoring Sensors (cont.)

- Create the user interface such as this:

WebViewer to display the chart.

Buttons and Labels are organised into 2 rows & 2 columns in a Table Arrangement.

Buttons to select which chart to display.

Labels to display the latest values.

A Label is used to display the GET's response – this is for debugging only.

A Web component (from Connectivity palette) is used to read the latest values from our channel.

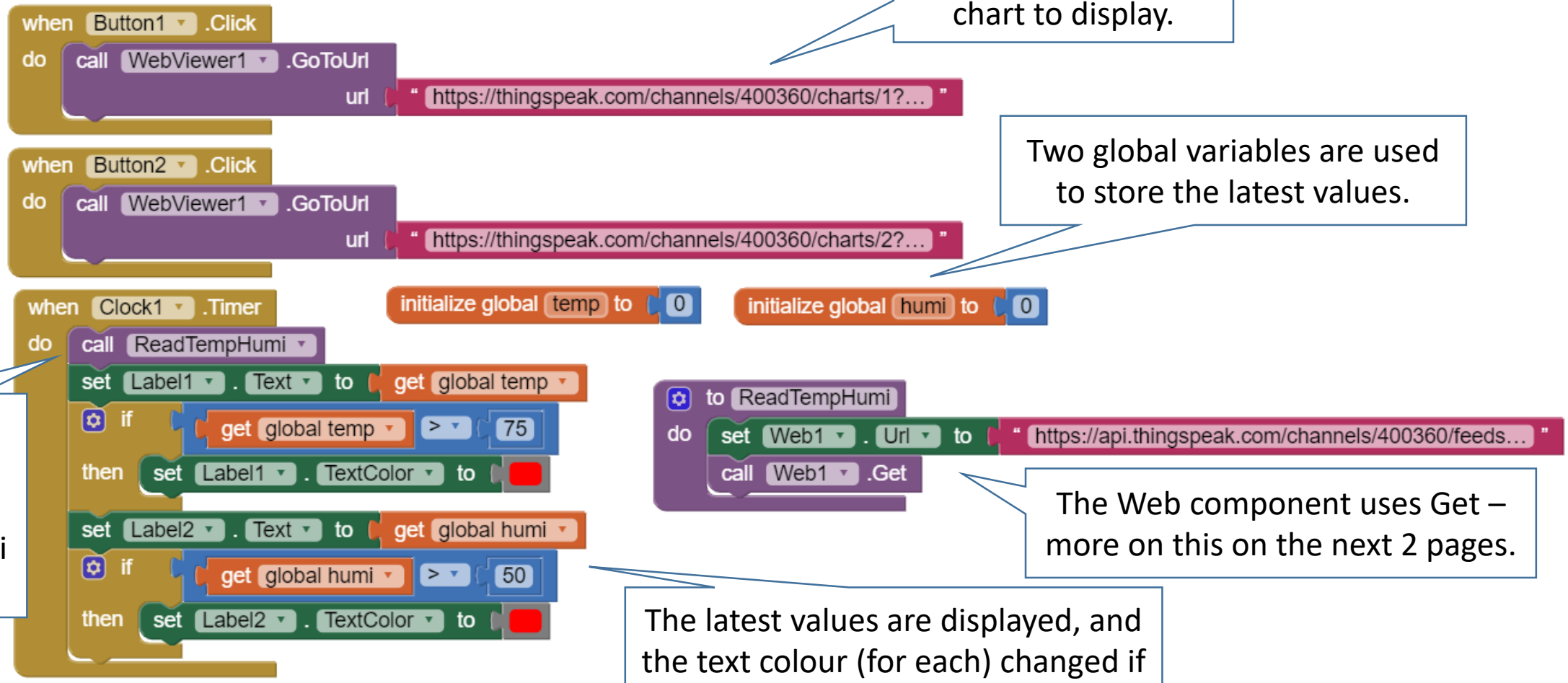
A Clock component (from Sensors palette) is used to refresh the display once every second, for example.

Exercise 6.4 – Monitoring Sensors (cont.)

- The Blocks view is shown here and on the next 3 slides.

Buttons select which chart to display.

Two global variables are used to store the latest values.



At regular interval, the function "ReadTempHumi" is called.

The Web component uses Get – more on this on the next 2 pages.

The latest values are displayed, and the text colour (for each) changed if the threshold is exceeded.

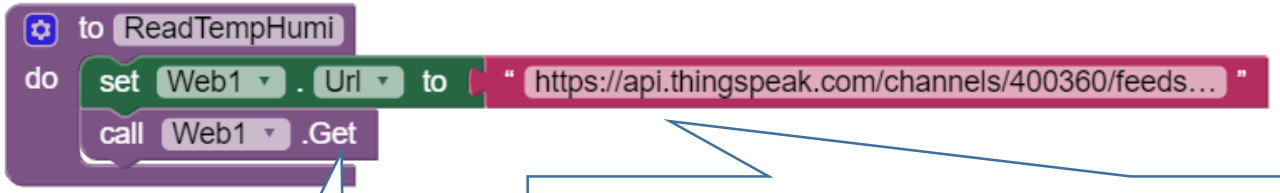
Exercise 6.4 – Monitoring Sensors (cont.)

Thingspeak tells you how to read a channel feed.

Read a Channel Feed

```
GET https://api.thingspeak.com/channels/400360/feeds.json?results=2
```

Change to results=1 if you only want the latest temperature and humidity.



GET

https://api.thingspeak.com/channels/400360/feeds.json?results=1

Exercise 6.4 – Monitoring Sensors (cont.)

{"channel":{"id":400360,"name":"Temperature \u0026 humidity from 12397","description":"For App Inventor use","latitude":"0.0","longitude":"0.0","field1":"Temperature","field2":"Humidity","created_at":"2018-01-10T03:46:15Z","updated_at":"2019-10-18T08:41:20Z","last_entry_id":100},"feeds":[{"created_at":"2019-10-07T17:43:45Z","entry_id":100,"field1":"79.4","field2":"48"}]}

The Web1's Get returns this, which consists of a json, inside a list, which in turn is inside another json.

A json (Java Script Object Notation) consists of name:value pairs, separated by commas, enclosed by curly brackets.

The same Get response formatted nicely.

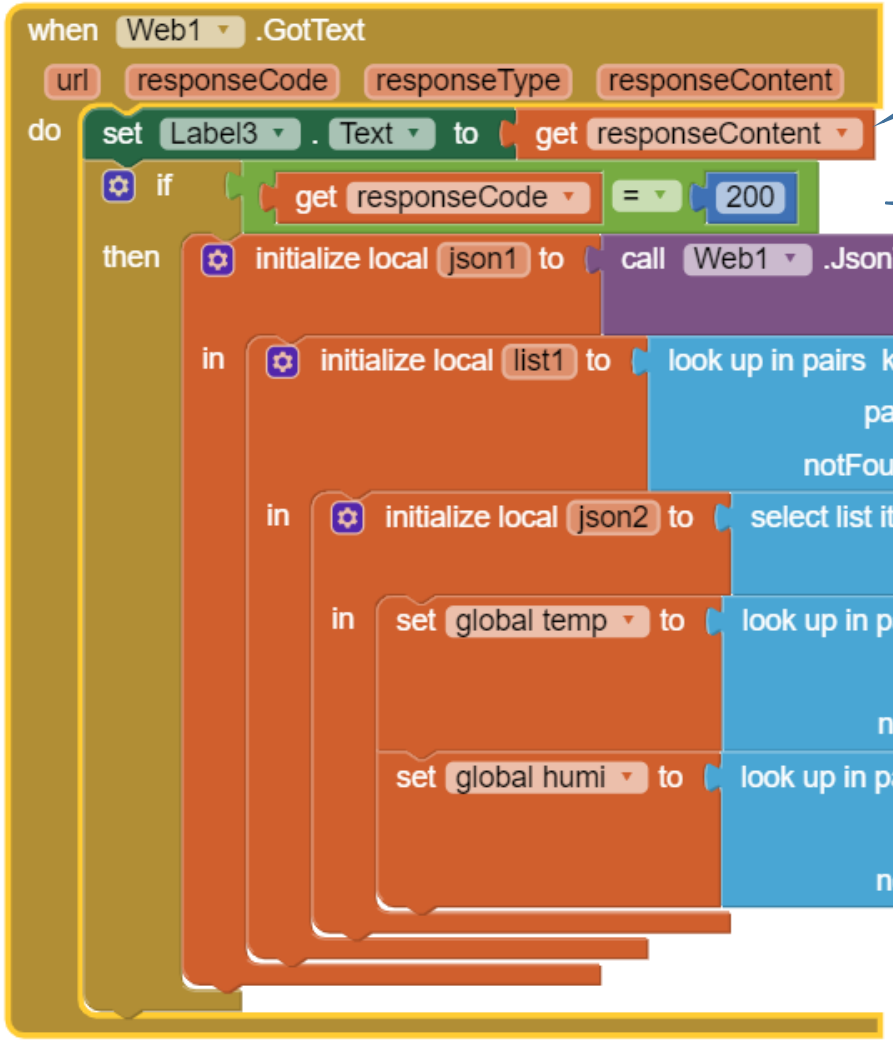
A list consists of one or more items, separated by commas, enclosed by square brackets.

```
{ "channel" : {"id":400360,….“last_entry_id”:100},
  “feeds” : [ { “created_at” : ”…”,
                “entry_id” : 100,
                “field1” : “79.4”,
                “field2” : “48” } ] }
```

Our objective is to get to the 2 numbers 79.4 and 48 – see next page.

Red brackets enclose the “outer” json, blue brackets enclose the list of ONE item, green brackets enclose the “inner” json.

Exercise 6.4 – Monitoring Sensors (cont.)



Label3 shows the Response Content, for debugging.

Response Code of 200 indicates a successful Web1's Get.

We use json1 for the "outer" json, and json2 for the "inner" json.

The list starts after "feeds".

The "inner" json is item 1 in the list.

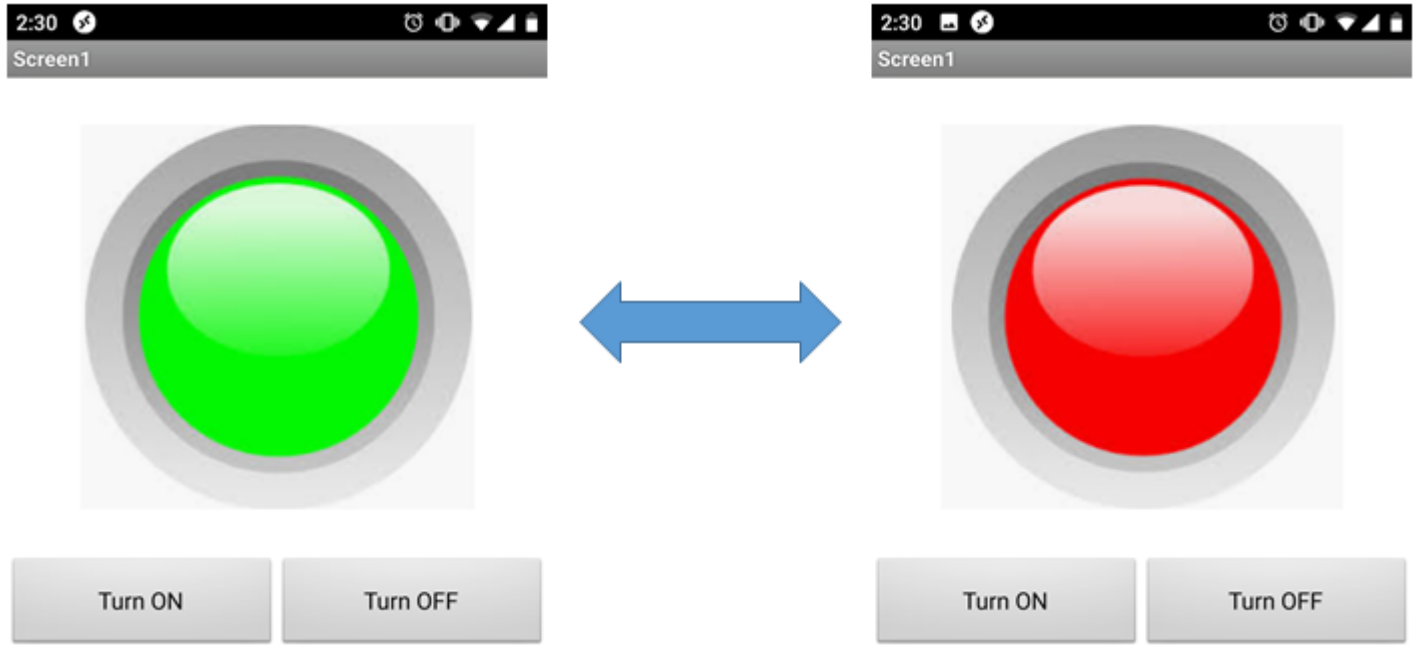
Temperature is field1

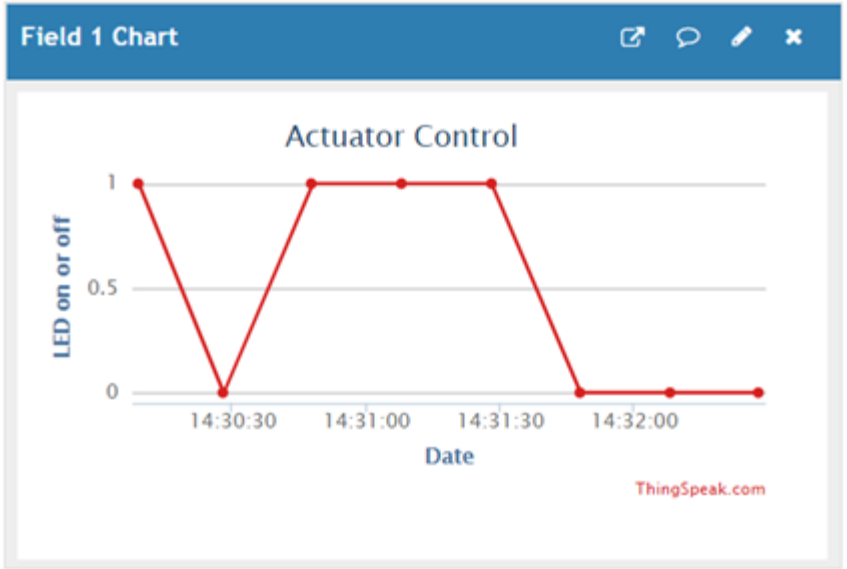
Humidity is field2

{ "channel" : {"id":400360,…."last_entry_id":100},
"feeds" : [{ "created_at" : "…",
"entry_id" : 100,
"field1" : "79.4",
"field2" : "48" }] }

Exercise 6.5 – Controlling Actuators

- The "actuator control" app discussed in the lecture allows the user to send on & off commands to the Thingspeak channel.
- The next few slides show an Arduino UNO can be programmed to read from this Thingspeak channel and to turn the physical LED on / off.
- The ESP01 WiFi module will be used for internet access.





Exercise 6.5 – Controlling Actuators (cont.)

sketch_test_download | Arduino 1.8.2

File Edit Sketch Tools Help

sketch_test_download

```
#include <SoftwareSerial.h>

#define DEBUG true

// LED at D13
int ledPin = 13;

// UNO's D10 connected to ESP's TX
// UNO's D11 connected to ESP's RX via resistor network
SoftwareSerial ESP01(10, 11); // RX, TX

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  while (!Serial){
  }
  Serial.println("Starting...");
  ESP01.begin(9600);
}
```

This is modified from "Sketch_test_upload" used in Exercise 3.4

The ESP01 has to be connected to the UNO this way – with level shifting!

The Serial Monitor has to be opened, to continue…

Exercise 6.5 – Controlling Actuators (cont.)

```
void loop() {
  // Reset ESP8266, put it into mode 1 i.e. STA only, make it join hotspot / AP, establish single connection
  Serial.println("--- Reset ESP8266 ---");
  sendData("AT+RST\r\n",2000,DEBUG);
  Serial.println("--- Put it into mode 1 i.e. STA only ---");
  sendData("AT+CWMODE=1\r\n",2000,DEBUG);
  Serial.println("--- Make it join hotspot / AP ---");
  sendData("AT+CWJAP=\"[redacted]\",\"[redacted]\"\r\n",4000,DEBUG); // Change these!!!
  Serial.println("--- Establish single connection ---");
  sendData("AT+CIPMUX=0\r\n",2000,DEBUG);

  // Blink LED on board
  digitalWrite(ledPin, HIGH);
  delay(200);
  digitalWrite(ledPin, LOW);

  // Make TCP connection
  Serial.println("--- Make TCP connection ---");
  String cmd = "AT+CIPSTART=\"TCP\",\"184.106.153.149\",80\r\n"; // 184.106.153.149 = Thingspeak.com's IP address
  sendData(cmd,2000,DEBUG);
```

Connecting to the hotspot / access point in the same way.

You can see the Serial Monitor sample response 3 slides later.

Make TCP connection to thingspeak.com, port 80.

Exercise 6.5 – Controlling Actuators (cont.)

```
// Prepare GET string
String getStr = "GET /channels/659016/fields/1/last.csv\r\n";

// Send data length & GET string
Serial.println("--- Send data length & GET string ---");
ESP01.print("AT+CIPSEND=");
ESP01.println (getStr.length());
Serial.print("AT+CIPSEND=");
Serial.println (getStr.length());
delay(500);
if( ESP01.find( ">" ) )
{
  Serial.print(">");
  String reply = sendData(getStr,2000,DEBUG);
  char num = reply.length();
  Serial.println("--- Actuator value from Thingspeak: ---");
  Serial.println(reply[num-10]);
  if(reply[num-10]=='0') {
     Serial.println("--- 0 => Turning off ---");
     digitalWrite(ledPin, LOW);
  }
  else {
     Serial.println("--- 1 => Turning on ---");
     digitalWrite(ledPin, HIGH);
  }
}

// Close connection, wait a while before repeating...
Serial.println("--- Close connection, wait a while before repeating... ---");
sendData("AT+CIPCLOSE",16000,DEBUG);
}
```

Modified the channel number used, if necessary.

This reads the channel 659016, field 1, last value using the csv format.

Other formats: json, xml, txt.

Same as Exercise 3.4, the number of bytes in the getStr is sent first. When the thingspeak server responds with >, the getStr is sent.

Sending the getStr will make the Thingspeak server reply with this:
+IPD,52:created_at,entry_id,field1
2019-11-01T06:32:48Z,9,0
CLOSED

By trial and error, reply[num-10] will give the last field 1 value

If the last value is 0, the LED is turned OFF. Otherwise, it is turned ON.

Close TCP connection.

Exercise 6.5 – Controlling Actuators (cont.)

```
String sendData(String command, const int timeout, boolean debug)
{
    String response = "";
    ESP01.print(command);
    long int time = millis();

    while( (time+timeout) > millis()) {
      while(ESP01.available()) {
        // "Construct" response from ESP01 as follows
         // - this is to be displayed on Serial Monitor.
        char c = ESP01.read(); // read the next character.
        response+=c;
      }
    }

    if(debug) {
      Serial.print(response);
    }
    return (response);
}
```

sendData function – same as Exercise 3.4.

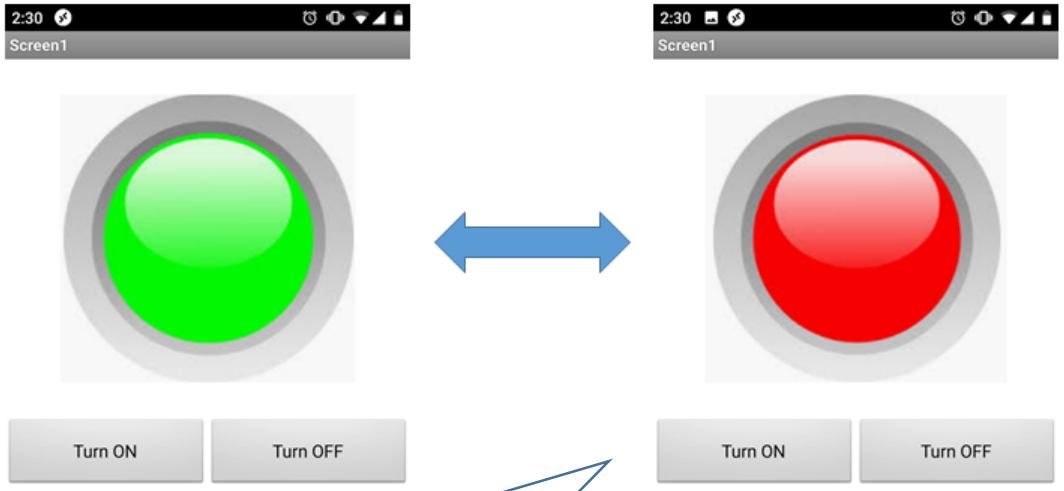
Exercise 6.5 – Controlling Actuators (cont.)

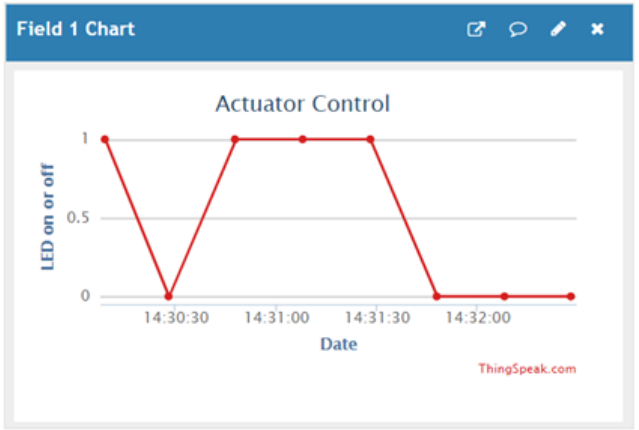
```
Starting...
--- Reset ESP8266 ---
AT+RST


OK
--- Put it into mode 1 i.e. STA only ---
AT+CWMODE=1


OK
--- Make it join hotspot / AP ---
AT+CWJAP=“yyyyyyyy",“xxxxxxxx"

WIFI CONNECTED
WIFI GOT IP
--- Establish single connection ---
AT+CIPMUX=0

busy p...

OK
```

```
--- Make TCP connection ---
AT+CIPSTART="TCP","184.106.153.149",80

CONNECT

OK
--- Send data length & GET string ---
AT+CIPSEND=40
>
Recv 40 bytes

SEND OK

+IPD,52:created_at,entry_id,field1
2019-11-01T06:32:48Z,9,0
CLOSED
--- Actuator value from Thingspeak: ---
0
--- 0 => Turning off ---
--- Close connection, wait a while before repeating... ---
```

You can see something similar to this in the Serial Monitor when the program is run.

Exercise 6.5 – Controlling Actuators (cont.)

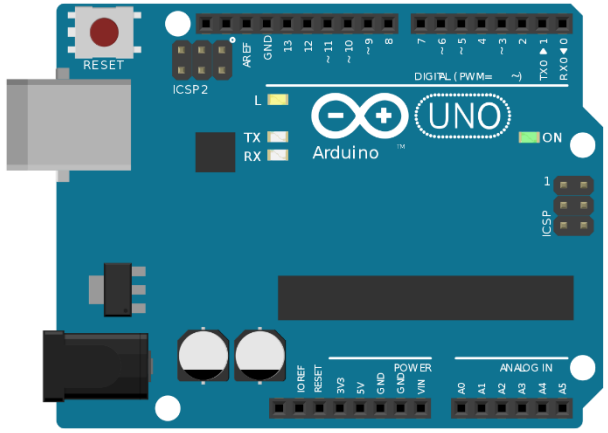


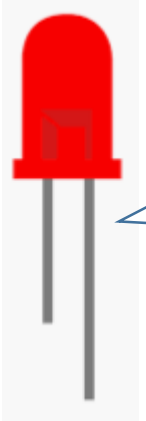


1. The App is used to send 0 or 1 to a Thingspeak channel, field 1.

2. The UNO + ESP01 are used to read 0 or 1 from the Thingspeak channel, field 1.

+IPD,52:created_at,entry_id,field1
2019-11-01T06:32:48Z,9,0





3. A ‘0’ will cause the LED to be turned OFF, while a ‘1’ will cause it to be turned ON.


Thank you!!!