Dashboard with Arduino

Tutorial 2: Listen for Sensor Value

GNG1103 F19

1. Objective

In this tutorial, you will learn how to set up a listening server on Dashboard and use it to wait for events from an Arduino. This will be accomplished with some basic code in Dashboard as well as modifying an example in the Arduino IDE. This tutorial will expose students to basic TCP communications, basic programming, and user interface design.

2. Equipment

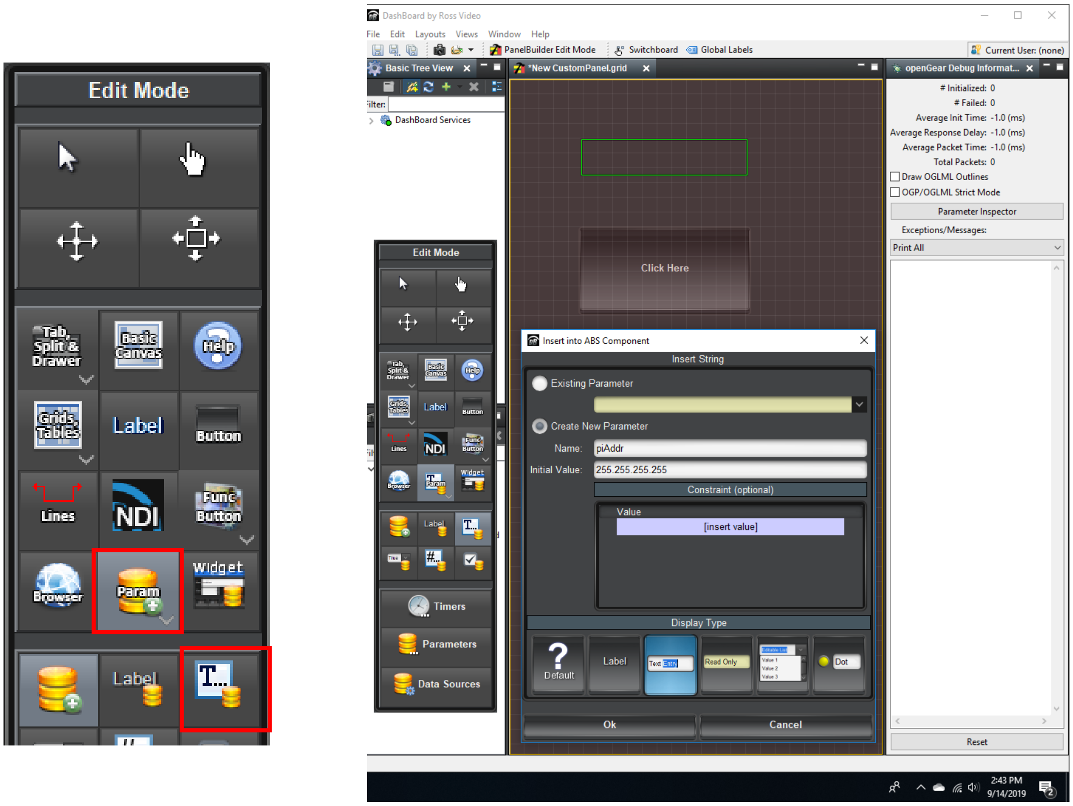
- 1x Personal Computer
- 1x NodeMCU
  - Alternatively, you can use any esp8266 module with Arduino but these instructions may be slightly different
- 1x Micro USB cable
- 1x Breadboard
- 1x Sensor of your choice with necessary instrumentation parts (in this case a thermistor + 1 2200 Ω resistor)

3. Downloads and Setup

1. Do Tutorial 1: Hello Arduino! before trying this tutorial. All the set-up is the same.

4. Set up the Panel

1. Open Dashboard and create a new panel by using File > New > New CustomPanel File. Set the folder where you want to save your panel and give the file a name that makes sense. Keep the template as “Blank Self-Contained Data Source Panel (XPression)”.
2. Click the PanelBuilder Edit Mode button to begin making the panel.
3. Next, click the “Param+” button and then the text entry button.
4. Click and drag in the panel area to create the text entry space.
5. In the menu that comes up, select “Create New Parameter” and give it a name and an initial value. In this case, the name is what we will use in the code, and the initial value is what will initially appear in the text. We have called this parameter “*ardMessage*”, and given it the initial value “N/A”, because this is where the sensor values will show up.
6. Change the display type to “Read Only” and press OK



Initialized: 0
Failed: 0
Average Init Time: -1.0 (ms)
Average Response Delay: -1.0 (ms)
Average Packet Time: -1.0 (ms)
Total Packets: 0
☐ Draw OGLML Outlines
☐ OGP/OGLML Strict Mode

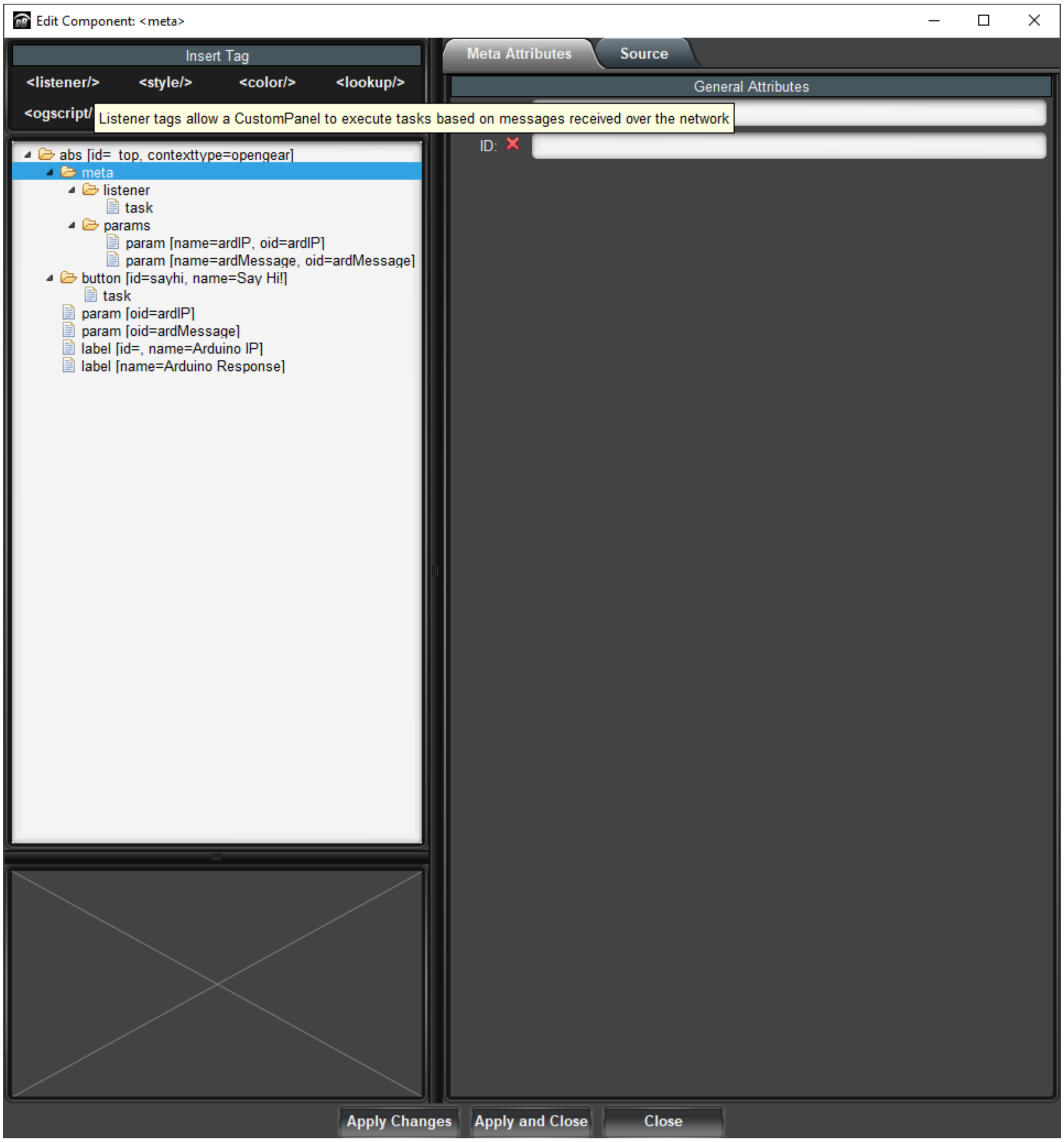
○ Existing Parameter
● Create New Parameter
Name: piAddr
Initial Value: 255.255.255.255

Figure 1: Creating a text entry section

5. Set up Listener

A listener is a program that watches a specific port on your network and registers when data is sent to that port. Using this, we can periodically get a measurement from a sensor.

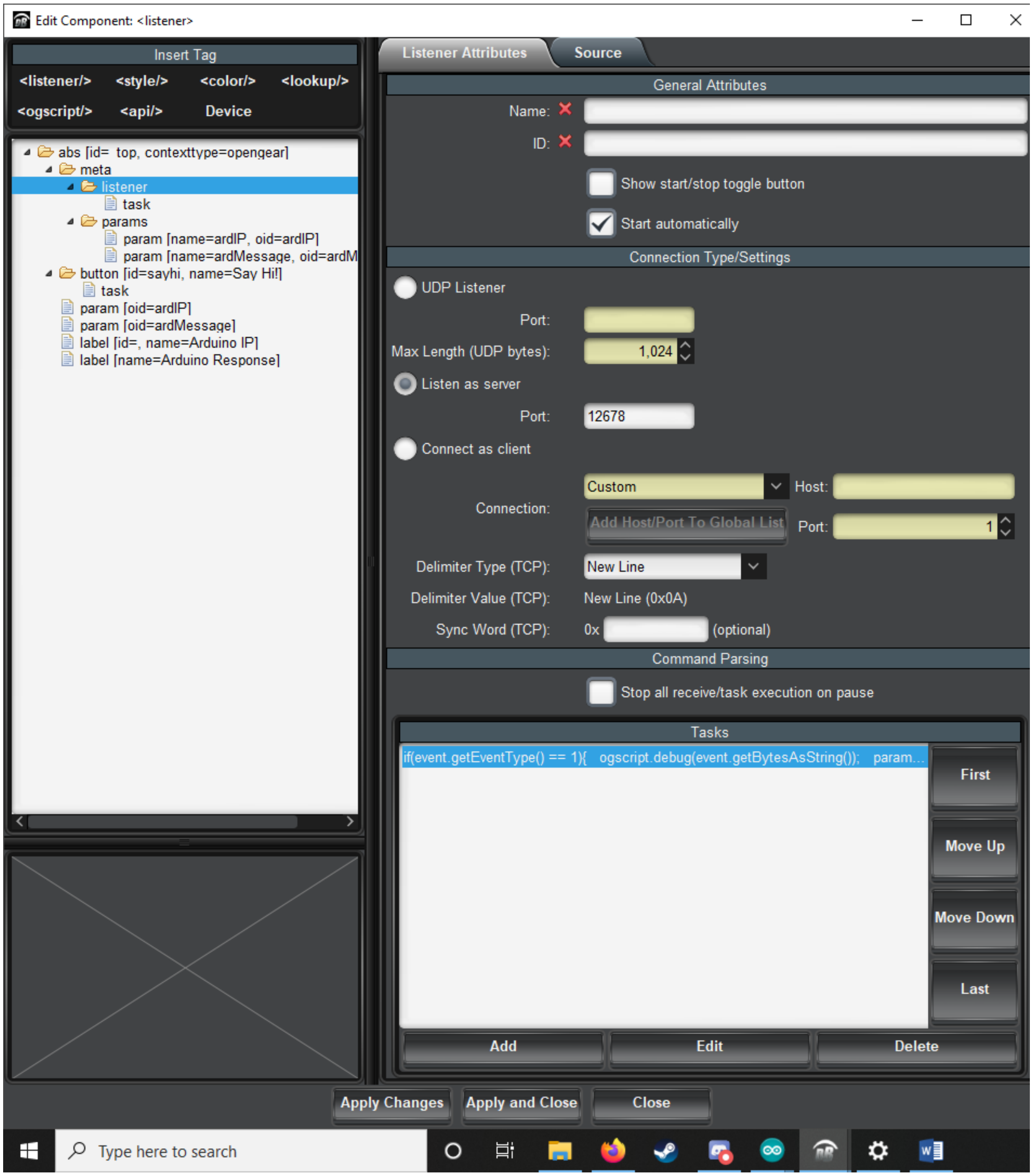
1. Double click anywhere in your panel to open up the general edit window.
2. On the left side, you'll see a list of all the objects you have in your panel. Click the "meta" folder to highlight it.
3. Now, in the top left, click "<listener/>" to create your server.



Listener tags allow a CustomPanel to execute tasks based on messages received over the network

Figure 2 Add a Listener

4. Now the listener menu will appear. In this menu
    a. Check “Start Automatically”. This will run the listener when your panel is active. You can also make a button to start the server if you want.
    b. Select “Listen as server” and give your server a port number (something between 1000 to 50000 should work). On Windows open a terminal and type netstat –an to check your active ports. Choose a number that isn’t on this list. We have used port 12678 in this example.
    c. In “Delimiter Type (TCP)” select “New Line” from the drop down menu. This means we will end our messages with a newline character “\n” from the Arduino.



```
abs [id=  top, contexttype=opengear]
    meta
        listener
            task
        params
            param [name=ardIP, oid=ardIP]
            param [name=ardMessage, oid=ardM
    button [id=sayhi, name=Say Hi!]
        task
    param [oid=ardIP]
    param [oid=ardMessage]
    label [id=, name=Arduino IP]
    label [name=Arduino Response]
```

Name: ☒
ID: ☒
☐ Show start/stop toggle button
☑ Start automatically

○ UDP Listener
Port:
Max Length (UDP bytes): 1,024
● Listen as server
Port: 12678
○ Connect as client
Connection: Custom
Host:
Port: 1
Delimiter Type (TCP): New Line
Delimiter Value (TCP): New Line (0x0A)
Sync Word (TCP): 0x (optional)

☐ Stop all receive/task execution on pause

```
if(event.getEventType() == 1){   ogscript.debug(event.getBytesAsString());   param...
```

Figure 3 Listener menu

5. Now add a task. Here we are going to have to write code instead of using visual logic unfortunately. There are two functions in the listener that will of use to us here: event.getEventType() and event.getBytesAsString(). The first function will tell us what's happening on the server:
    a. Type 0 is a connection
    b. Type 1 is available data
    c. Type 2 is a disconnection

The second function will read available TCP data as a string so we can use it for stuff. Write the following code in your task:

```
if(event.getEventType() == 1){
  var rawData = event.getBytesAsString();
  ogscript.debug(rawData);
  params.setValue('ardMessage',0,rawData);
}
```

If there is data available, this code will save it in the variable “rawData” then print this data to the debug menu and set the parameter associated with our text box. For some more advanced code involving actually parsing the message, check out the listener in the foosball example.

6. Make your circuit

In this tutorial we have put together a basic voltage divider circuit with a thermistor. This is a device that changes its resistance in response to a change in temperature. You don’t need to understand the minutia of this circuit, it’s only provided for your reference. The wiring from your sensor will almost certainly be different. Note that we have actually used a NodeMCU, not a UNO as pictured in the cartoon drawing of the circuit. The wiring is equivalent though.

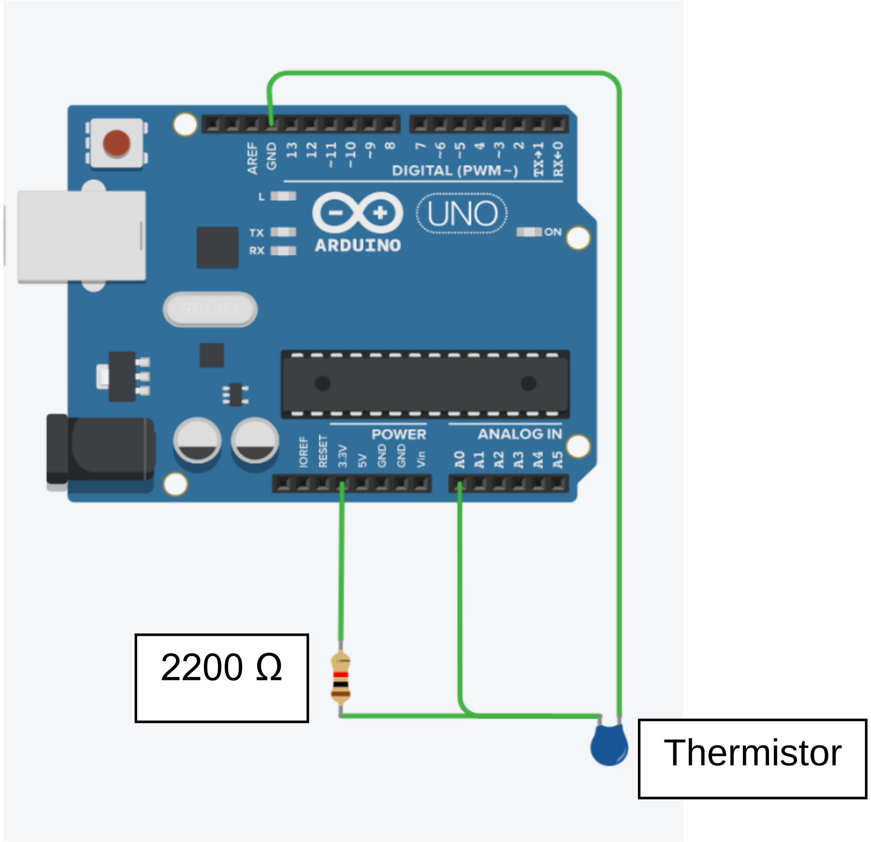
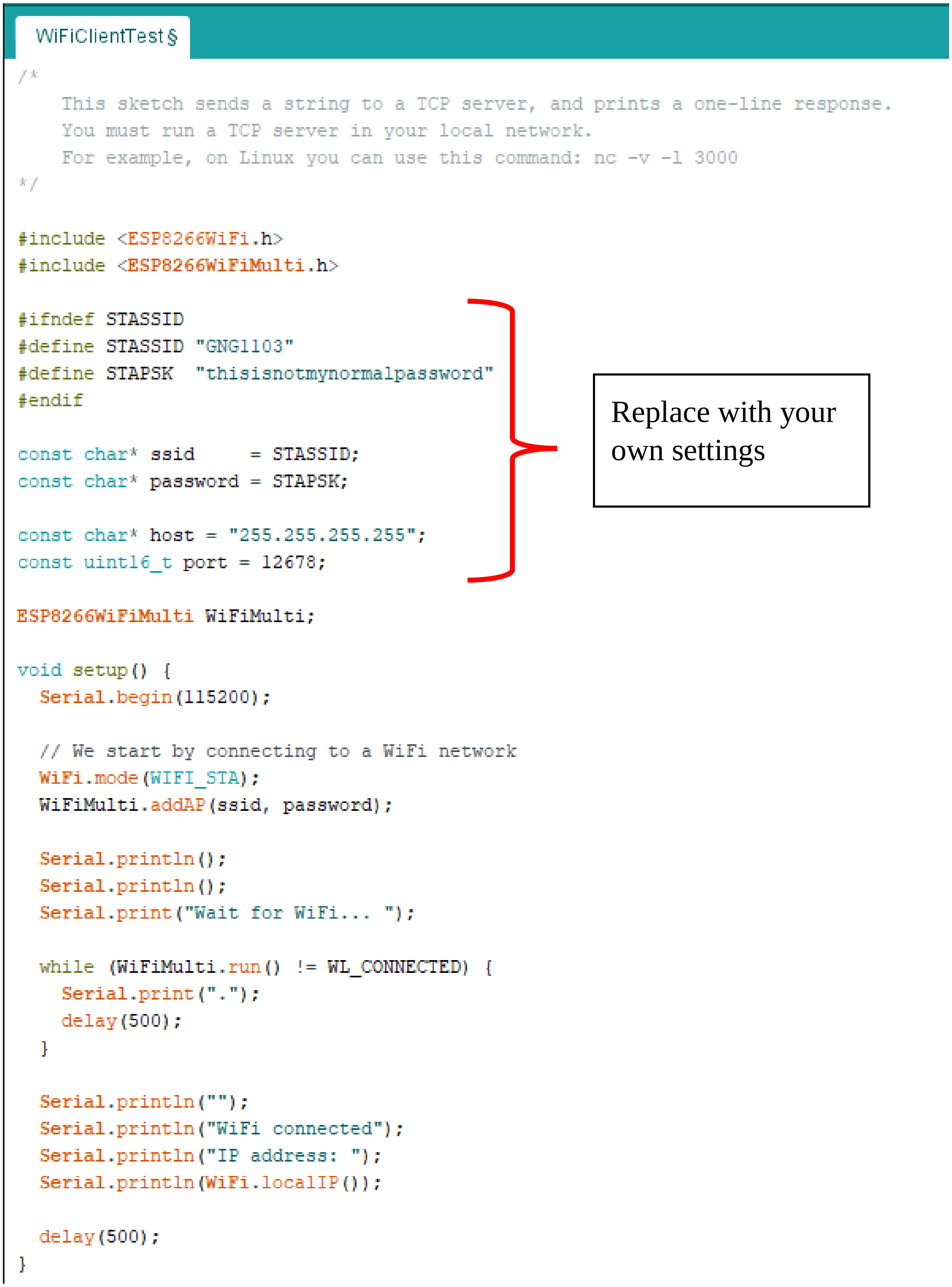


Figure 4 Circuit used in this example

7. Code on Arduino

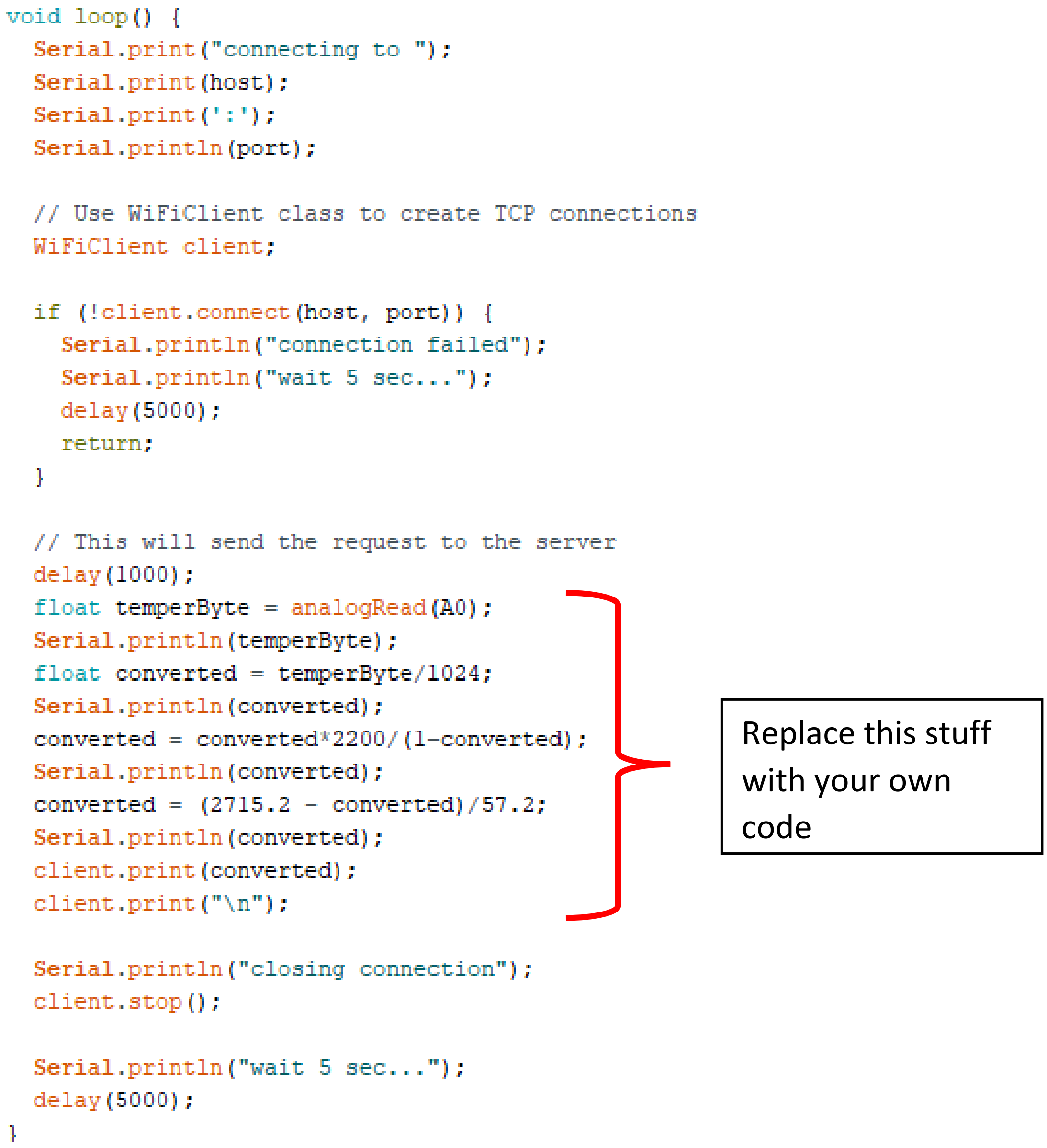
Now we need to write the Arduino side of things. This will be done by modifying an existing example in the new esp8266 libraries that were downloaded when we added the new boards.

1. Open up the IDE and navigate to File > Examples > ESP8266WebServer > WiFiClientBasic.
2. Add the name and password for your internet (mobile hotspot) to STASSID and STAPSK in the code.
3. Add the IP address of the computer with Dashboard on it along with the chosen port (12678 in this case) to the host and port variables. There are a number of ways to find the IP address of your computer. Remember to use the IP for the connection you are using, some methods will show you IP addresses for Ethernet even if you are connected via Wi-Fi.
4. Delete the existing client.println("hello from ESP8266"). We will replace this with our own code. This function, client.print() or client.println(), will send a TCP message to the server containing whatever we give to the function. In this case we will send our sensor reading using this function. Remember that we have told Dashboard that a newline character will terminate our messages. So, if you use println() the message will end, but we can use a bunch of print() functions to build up a message before using a final println().
5. In this example we will use analogRead(A0) to read the voltage drop across the thermistor. Then (after converting this value to the temperature) we will send it to the server with client.println(). Replace this code, with whatever is necessary to read from your sensor. My code looks like this:



```
/*
    This sketch sends a string to a TCP server, and prints a one-line response.
    You must run a TCP server in your local network.
    For example, on Linux you can use this command: nc -v -l 3000
*/

#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>

#ifndef STASSID
#define STASSID "GNG1103"
#define STAPSK  "thisisnotmynormalpassword"
#endif

const char* ssid     = STASSID;
const char* password = STAPSK;

const char* host = "255.255.255.255";
const uint16_t port = 12678;

ESP8266WiFiMulti WiFiMulti;

void setup() {
  Serial.begin(115200);

  // We start by connecting to a WiFi network
  WiFi.mode(WIFI_STA);
  WiFiMulti.addAP(ssid, password);

  Serial.println();
  Serial.println();
  Serial.print("Wait for WiFi... ");

  while (WiFiMulti.run() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  delay(500);
}
```

Replace with your own settings



```
void loop() {
  Serial.print("connecting to ");
  Serial.print(host);
  Serial.print(':');
  Serial.println(port);

  // Use WiFiClient class to create TCP connections
  WiFiClient client;

  if (!client.connect(host, port)) {
    Serial.println("connection failed");
    Serial.println("wait 5 sec...");
    delay(5000);
    return;
  }

  // This will send the request to the server
  delay(1000);
  float temperByte = analogRead(A0);
  Serial.println(temperByte);
  float converted = temperByte/1024;
  Serial.println(converted);
  converted = converted*2200/(1-converted);
  Serial.println(converted);
  converted = (2715.2 - converted)/57.2;
  Serial.println(converted);
  client.print(converted);
  client.print("\n");

  Serial.println("closing connection");
  client.stop();

  Serial.println("wait 5 sec...");
  delay(5000);
}
```

Replace this stuff with your own code

6. Under Tools > Boards, select the appropriate board with WiFi module. In this case it's the NodeMCU 1.0 (ESP-12E Module).

7. Check your port

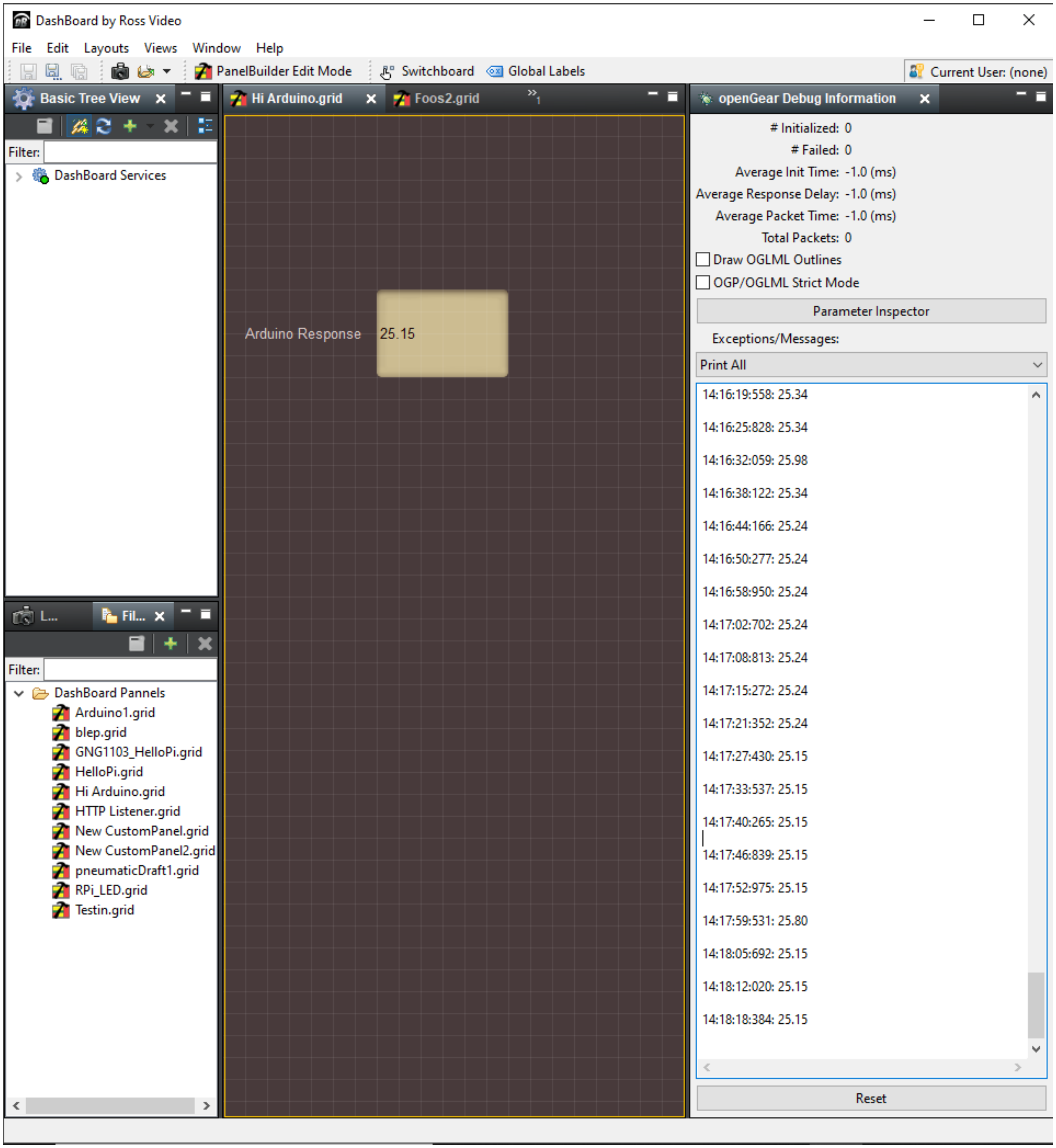
8. Upload the code to your board.

9. When the upload is completed, open up the Serial and wait for the connection message to show up.

8. Test Your System

Now, with both the Arduino and Dashboard running we can now test the system that we've just made.

1. With the Arduino connected to the mobile hotspot, you will see the Arduino in the list of connected devices
2. With both the Arduino and Dashboard running you should see sensor values appear



Arduino Response 25.15

Initialized: 0
Failed: 0
Average Init Time: -1.0 (ms)
Average Response Delay: -1.0 (ms)
Average Packet Time: -1.0 (ms)
Total Packets: 0
☐ Draw OGLML Outlines
☐ OGP/OGLML Strict Mode

Exceptions/Messages: Print All

```
14:16:19:558: 25.34
14:16:25:828: 25.34
14:16:32:059: 25.98
14:16:38:122: 25.34
14:16:44:166: 25.24
14:16:50:277: 25.24
14:16:58:950: 25.24
14:17:02:702: 25.24
14:17:08:813: 25.24
14:17:15:272: 25.24
14:17:21:352: 25.24
14:17:27:430: 25.15
14:17:33:537: 25.15
14:17:40:265: 25.15
14:17:46:839: 25.15
14:17:52:975: 25.15
14:17:59:531: 25.80
14:18:05:692: 25.15
14:18:12:020: 25.15
14:18:18:384: 25.15
```

Figure 5 Arduino sensor readings appearing in both the parameter section as well as the debugger .As you can see, it’s way too warm in my room.

3. **Success!** Good luck! ☺.