

Documentation

Group 08:

Roberto Gomez Pomares

Leonhard Keller

Andreas Sauerwein

Advisors:

Tapio Wikström

Marko Uusitalo

Table of Contents

1 Task	1
2 Set Up.....	1
2.1 Controlling the Raspberry Pi	1
2.2 Sensor and RedBoard.....	1
3 Gathering Data	3
4 Storing Data	4
4.1 Local Publishing Method	4
4.2 Data Retention	6
5 Sending to Cloud	6
6 Main Code	8

1 Task

For this project, we were task to gather data from a sensor(s) using a Raspberry Pi and store it in a local database. The same data is also supposed to be send to a cloud. Other considerations for the project would be adding features like security and data retention. For this project we decided to focus on the temperature sensor.

2 Set Up

2.1 Controlling the Raspberry Pi

In order to work with the Raspberry Pi (RP), we used a VNC Server to remote control it from our own computers. We set up VNC Server on the RP and VNC Viewer on our own laptops. An HDMI cable was plugged into the RP to a monitor. We established a direct connection from our devices by discovering our RP's private IP address and entering it into the VNC Viewer.

Each member could then connect to and remote control the Raspberry Pi using VNC Viewer. This allowed us to work on the project efficiently.

2.2 Sensor and RedBoard

First, we set up the temperature sensor on the breadboard.

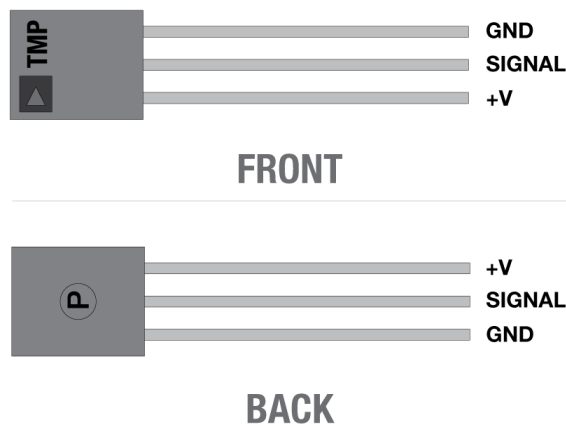
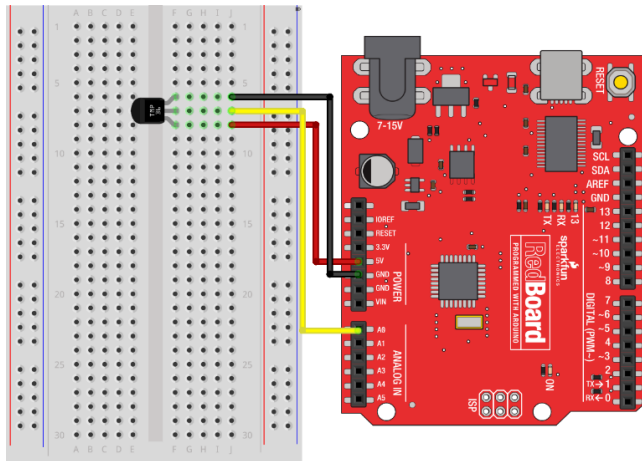


Figure 1 Pin outs of the temperature sensor

(Source: <https://learn.sparkfun.com/tutorials/activity-guide-for-sparkfun-tinker-kit/circuit-9-temperature-sensor>)

We hooked up the sensor on the breadboard and connected it to the RedBoard as shown on the SparkFun Tinker Kit guide. Using jumper wires, we connected the Signal leg to Analog Pin 0, GND leg to GND on the RedBoard, and +V to 5V.



fritzing

Figure 2 Circuit diagram

(Source: <https://learn.sparkfun.com/tutorials/activity-guide-for-sparkfun-tinker-kit/circuit-9-temperature-sensor>)

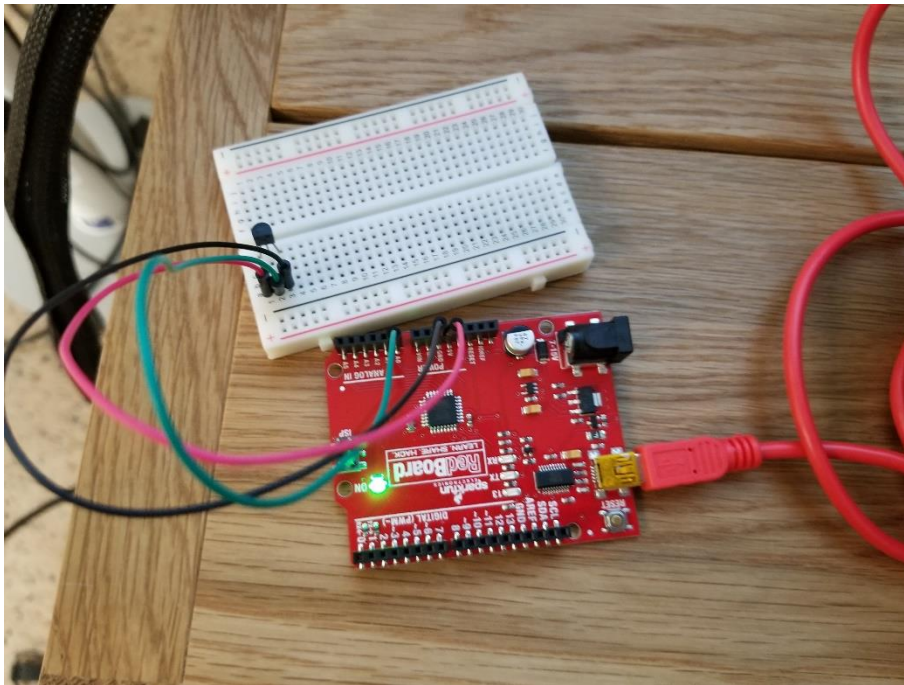


Figure 3 Final set up

Once that was done, we connected the RedBoard to the Raspberry Pi to start recording data.

3 Gathering Data

The code to record the data is written into Arduino IDE.

The code to read and write the output was gotten from the SparkFun Tinker Kit guide. The results are given into four values: raw value of temperature, value converted into voltage, value converted into Celsius, and value converted into Fahrenheit.

In a python code, we import serial and the commands .Serial() and .readline() to get the current data.

```
import serial

ser = serial.Serial('/dev/ttyUSB0',9600)

read_serial = ser.readline()

print read_serial
```

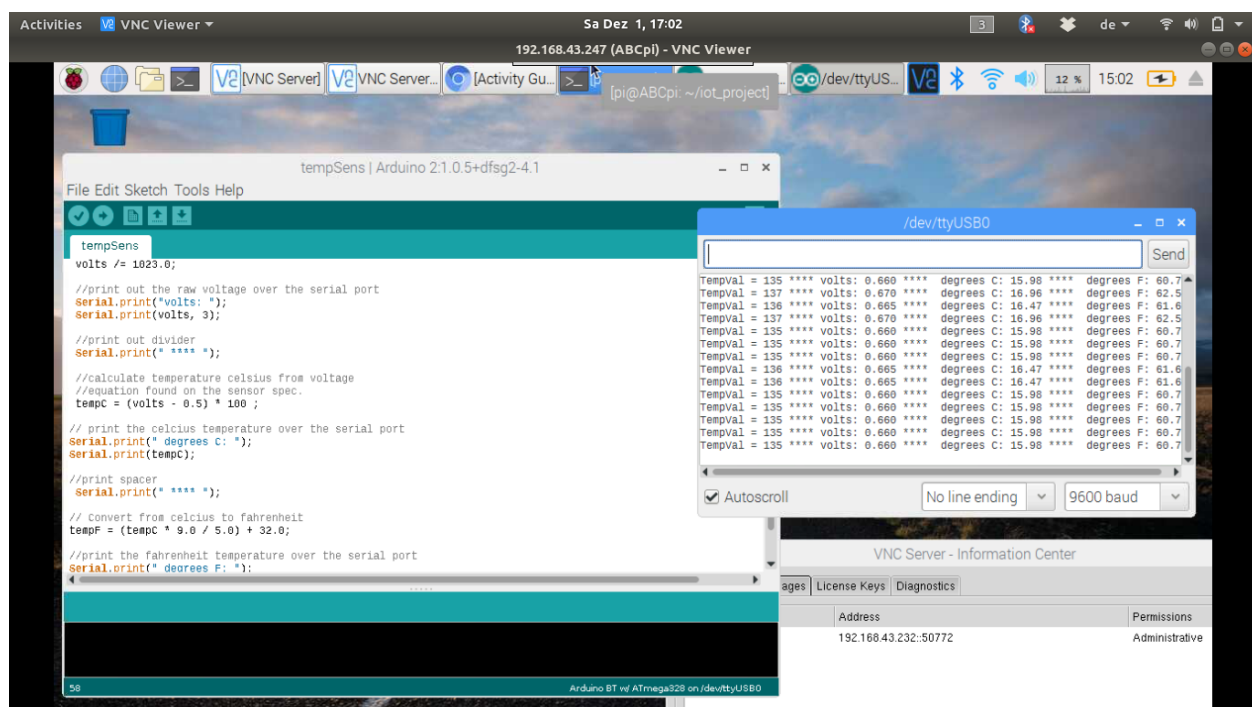


Figure 4 The Python code and the printed results

In the current implementation, the RP will record data continuously until told to stop.

4 Storing Data

To store data locally, we open a file called “tempdata.log” or create one if the file does not exist yet. We then write onto the log using the string `read_serial` from above. We also imported `datetime` in order to tell the reading apart by knowing when each reading was taken.

```
import time
import datetime

logfile = open('tempdata.log', 'a+')
~~~

timeCap = str (datetime.datetime.now())
read_serial = timeCap + ": " + ser.readline()
logfile.write(read_serial)
```

Like with the collection of data, the log will continuously be updated with each new reading until the program is told to stop.

4.1 Local Publishing Method

Next, we used the following two commands to install Apache2 and start a server to upload our data to a web page.:

```
sudo apt install apache2
```

```
sudo systemctl start apache2
```

Websocketd will be used in order to update the information on the server without the need for user input and provide a simple way to publish file content. Websocketd was installed on the Raspberry Pi and its executable place in `/usr/bin/`. A folder named “static” with the link to our log file and the following `index.html` file was put in `/var/www/html/`:

```
<!DOCTYPE
html>

<html>
  <head>
    <script type="text/javascript">
      var ws = new WebSocket('ws://ABCPi:8080/');
      ws.onmessage = function(event) {
        //document.getElementById('msgBox').innerHTML =
        event.data;

        console.log(event.data)
      }
    </script>
```

```
</head>  
</html>
```

The following command is then used to start publishing from the static folder and make it accessible on port 8080:

```
websocketd --port=8080 --staticdir=static/ tail -f tempdata.log
```



Figure 5 Data file accessible from webpage

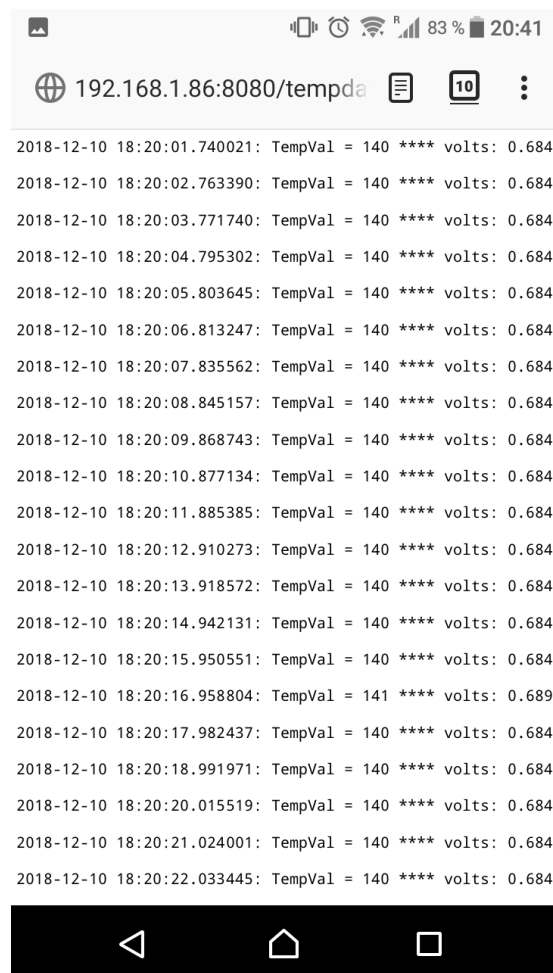


Figure 6 Inside the tempdata.log file

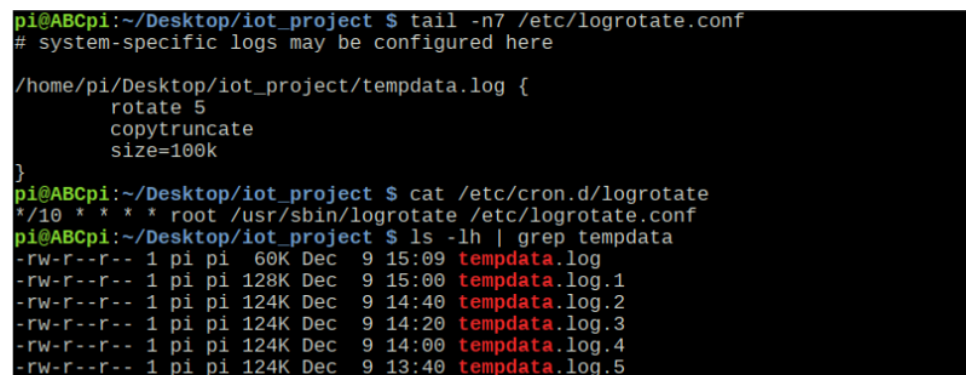
4.2 Data Retention

To avoid having a large file full of data, we made our logfile rotate once it passes a certain file size. We made a rotation rule for the file in `/etc/logrotate.conf` that there can only be a maximum of 6 rotation files and that once a file reaches 100 kb or higher it should rotate to the next one.

By default, the logrotation would check the file once daily, so we had to change the schedule otherwise the file would exceed pass 100 kb way before 24 hours. We replaced `/etc/cron.daily/logrotate` with a new rule in `/etc/cron.d/` so it would check the file after 10 minutes. In our implementation, the size does not exceed the limit after the first 10 minutes, so the rotation takes place every 20 minutes. The command we used to change the schedule is:

```
*/10 * * * * root /usr/sbin/logrotate /etc/logrotate.conf
```

By having the `*/10` and the rest of the time slots left as `*`'s, the new schedule is to check the files every 10 minutes every day.



```
pi@ABCpi:~/Desktop/iot_project $ tail -n7 /etc/logrotate.conf
# system-specific logs may be configured here

/home/pi/Desktop/iot_project/tempdata.log {
    rotate 5
    copytruncate
    size=100k
}
pi@ABCpi:~/Desktop/iot_project $ cat /etc/cron.d/logrotate
*/10 * * * * root /usr/sbin/logrotate /etc/logrotate.conf
pi@ABCpi:~/Desktop/iot_project $ ls -lh | grep tempdata
-rw-r--r-- 1 pi pi 60K Dec 9 15:09 tempdata.log
-rw-r--r-- 1 pi pi 128K Dec 9 15:00 tempdata.log.1
-rw-r--r-- 1 pi pi 124K Dec 9 14:40 tempdata.log.2
-rw-r--r-- 1 pi pi 124K Dec 9 14:20 tempdata.log.3
-rw-r--r-- 1 pi pi 124K Dec 9 14:00 tempdata.log.4
-rw-r--r-- 1 pi pi 124K Dec 9 13:40 tempdata.log.5
```

Figure 7 The logrotation rule and the files with their sizes and times rotated out

5 Sending to Cloud

In order to send the collected data to the cloud, we register the Raspberry Pi to Azure. In the IoT hub navigation menu on Azure, we register the device by adding a new device and gave it an ID. We then ran the temperature application by using:

```
sudo node index.js '<YOUR AZURE IOT HUB DEVICE CONNECTION STRING>'
```

where the string was the Hub's hostname, the Device ID, and the Share Access Key.

From the Cloud shell, we could then start and view the program on the shell by using the following command:

```
az iot hub monitor-events --device-id Group08 --hub-name iotprojecthub
```



```
time.sleep(1)
KeyboardInterrupt
pi@ABCPi:~/iot_project $ cat tempdata.log
TempVal = 136 **** volts: 0.665 **** degrees C: 16.47 **** degrees F: 61.65
TempVal = 136 **** volts: 0.665 **** degrees C: 16.47 **** degrees F: 61.65
TempVal = 136 **** volts: 0.665 **** degrees C: 16.47 **** degrees F: 61.65
TempVal = 136 **** volts: 0.665 **** degrees C: 16.47 **** degrees F: 61.65
TempVal = 135 **** volts: 0.660 **** degrees C: 15.98 **** degrees F: 60.77
TempVal = 136 **** volts: 0.665 **** degrees C: 16.47 **** degrees F: 61.65
TempVal = 136 **** volts: 0.665 **** degrees C: 16.47 **** degrees F: 61.65
2018-12-01 14:23:15.856089
2018-12-01 14:24:08.511907
2018-12-01 14:24:09.528032TempVal = 135 **** volts: 0.660 **** degrees C: 15.98 **** degrees F: 60.77
2018-12-01 14:24:11.161256TempVal = 136 **** volts: 0.665 **** degrees C: 16.47 **** degrees F: 61.65
pi@ABCPi:~/iot_project $ rm tempdata.log
pi@ABCPi:~/iot_project $ vim azure_test.py
pi@ABCPi:~/iot_project $ python azure_test.py
Message transmitted to IoT Hub
Confirmation received for message with result = OK
Confirmation received for message with result = OK
Confirmation received for message with result = OK
Confirmation received for message with result = OK
^CTraceback (most recent call last):
  File "azure_test.py", line 28, in <module>
    time.sleep(1)
KeyboardInterrupt
pi@ABCPi:~/iot_project $ cat tempdata.log
2018-12-01 14:25:09.781350
2018-12-01 14:25:10.796023 TempVal = 135 **** volts: 0.660 **** degrees C: 15.98 **** degrees F: 60.77
2018-12-01 14:25:12.429533 TempVal = 136 **** volts: 0.665 **** degrees C: 16.47 **** degrees F: 61.65
2018-12-01 14:25:13.453014 TempVal = 136 **** volts: 0.665 **** degrees C: 16.47 **** degrees F: 61.65
pi@ABCPi:~/iot_project $ vim azure_test.py
pi@ABCPi:~/iot_project $
```

Figure 8 Data is stored in the tempdata.log and send to the cloud

For testing purposes, we created our own IoT hub, like described above. “Group08” is the name of our device and “iotprojecthub” is the name of our Azure Cloud Hub. For the assignment we will use the given connection string.

6 Main Code

The program starts by initializing a client with the host, device ID, and share access keys as a single string and the protocol. It then enters a loop where it will start recording data from the serial port and converting it to a string that can be sent to the cloud and a local file. Here is what our main looks like:

```
# IoT Project Group08 ABC

from iothub_client import IoTHubClient, IoTHubTransportProvider, IoTHubMessage
import time
import serial
import sys
import datetime

CONNECTION_STRING = "HostName=iotprojecthub.azure-
devices.net;DeviceId=Group08;SharedAccessKey=cVe0Z/aRyhebmXI1KzuSaBR9YlNui+hprfaIYsI35A="

PROTOCOL = IoTHubTransportProvider.MQTT
ser = serial.Serial('/dev/ttyUSB0', 9600)
logfile = open('tempdata.log', 'a+')

def send_confirmation_callback(message, result, user_context):
    print("Confirmation received for message with result = %s" % (result))

if __name__ == '__main__':
    client = IoTHubClient(CONNECTION_STRING, PROTOCOL)
    print("Message transmitted to IoT Hub")

    while True:
        # Capture timestamp
        timeCap = str(datetime.datetime.now())
        # Read data from serial port
        read_serial = timeCap + ": " + ser.readline()
        # Convert into IoTHubMessage and send to Hub
        message = IoTHubMessage(read_serial)
        client.send_event_async(message, send_confirmation_callback, None)
        # Log message
        logfile.write(read_serial)
        time.sleep(1)
```