**CIT 595 Final Report**

Team Members: Jue Liu, Ao Sun, Xiaojun Sun

**Introduction**

**Project Goals:**

For the final project, our goal is to develop a Pebble smartwatch application that is able to get data from and send control information to remote sensors and display driven by an Arduino microcontroller. In addition to basic functional requirements, we decide to add the following features. First, another proximity sensor is added to the system and we are going to realize the control from user interface to the display, and process/display the data sent from the sensor to the user interface. Besides, we will add additional features to the user interface for it to display daily movie list shown in Cinemark University City Penn 6. The third one is grab the current weather of Philadelphia from weather forecast website and send/display in arduino.

**High-level Approach:**

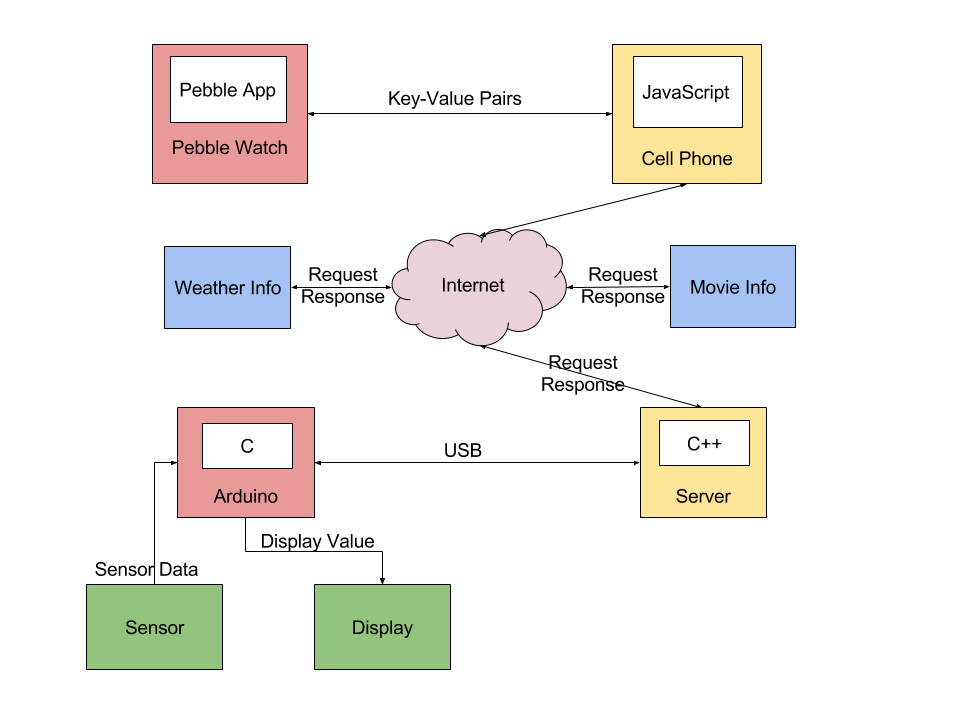
The system is consisted of three major components, described as follows.

* The sensor and display component includes a temperature sensor and a proximity sensor, a seven segment display and a RGB light. We program to Arduino microcontroller and enable it to reads temperature or distance from relevant sensor depending on the command sent from the server, and writes to serial port once a second.
* The server, which works as a middleware, interacts with both Pebble watch and Arduino. It not only reads and store the data from Arduino, but also writes commanding key word through serial port to Arduino according to the request send from pebble. Besides, server also send key value pair information to pebble when it’s requested.
* The pebble app is used to generate a user interface, send messages to the cell phone and retrieve messages from the cell phone. The cell phone will send requests to servers and retrieve responses by JavaScript after it get messages from the pebble watch.

Work Routine Explanation

In general, sensor and display part and user interface communicate to each other through a cpp program which acts as a server. On the one hand, the Arduino is connected via USB to Mac machine which will run the cpp server program. On the other hand, the Pebble smartwatch application communicates with the middleware over the Internet via Android or IOS phone by sharing the same local network with server.

**Project Architecture**



**How do the components communicate through key value pair:**

* Pebble with Middleware:

/current: request current temperature from server.

/average: request average temperature from server.

/high: request highest temperature so far from server.

/low: request lowest temperature so far from server.

/Celsius: request temperature show in Celsius from server.

/Fahrenheit: request temperature show in Fahrenheit from server.

/stop: tell server to make Arduino in a standby mode.

/resume: tell server to make Arduino out of a standby mode.

/proximity: tell server to read data from proximity sensor in Arduino.

/distance: request distance data from server.

/temp: tell server to read data from temperature sensor in Arduino.

* Middleware with Arduino:

‘c’ : tell Arduino to make the temperature on seven-segment display in Celsius.

‘f’ : tell Arduino to make the temperature on seven-segment display in Fahrenheit.

‘s’ : tell Arduino to stop the sensor from sending current temperature to the server.

‘r’: tell Arduino to resume displaying temperature.

‘p’: tell Arduino to send data from proximity sensor.

‘t’: tell Arduino to send data from temperature sensor.

**Implementation**

**Sensor and Display:**

This component acts as command executor. It follows commands sent from user interface via cpp server and displays demanded temperature or distance information from respective sensor. It may also enter a standby mode if is asked to do so.

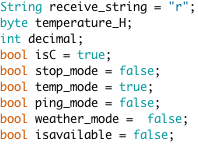
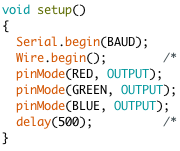
For this part, we use a Gravitech 7-segment Shield attached to an Arduino microcontroller to control the behavior of sensors, display and RGB light. Temperature sensor has already been integrated to the Arduino microcontroller board. For proximity sensor, we need to connect its three pins to Arduino MCU for the program to read sensor data. The MCU is well programmed to read in command sent from cpp server, and display the right information.

In more detailed description, there are three modes in general. The three modes are temp\_mode which will display temperature information, ping\_mode which will display distance data, and weather\_mode which will display current temperature sent by user interface. If no command is sent to Arduino, the Arduino will be in its default state and displays current temperature sent by temperature sensor in Celsius.

Code Realization

Programming is done in Arduino IDE in a C pattern.

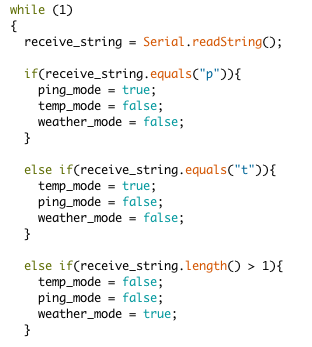
1. First there are some important bool type variables to define specific mode the program is in. A setup function is also needed to start I2C communication and enable RGB lights. Initially, it is in temp\_mode and displays temperature in Celsius. Below is the configuration and basic setup.

2. All the functional design is included in a loop function.

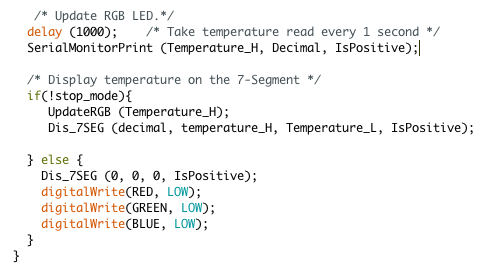
All the code related to serial communication initial configuration is neglected here. The most important part is included in a while loop.

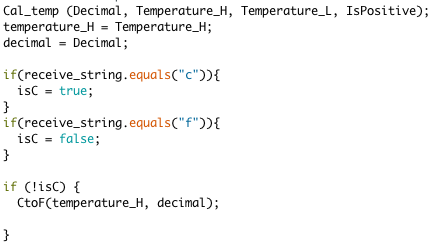
The arduino program will continuously read data from sensor and the data is stored in a global String type variable called receive\_string. *receive\_string* is initialized to be “r” which indicates the program is in work mode.



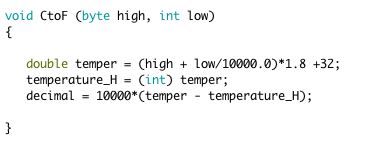
The program is set in different mode based on the command sent from server.

* In temp\_mode, there is more detailed classification. If the server sends “s”, display will enter into standby mode. But the program keeps sending data to serial port for the server to keep track of temperature and calculate average, highest and lowest temperature.

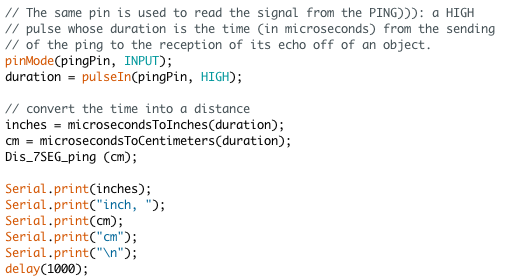




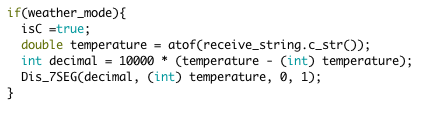
It is also able to switch temperature from Celsius to Fahrenheit if server sends a specific command to arduino. Below CtoF function is used to convert Celsius temperature to Fahrenheit. Notice from the code that the input high byte to the function CtoF is not the copy of the read-in value, therefore it will not influence other functionality.



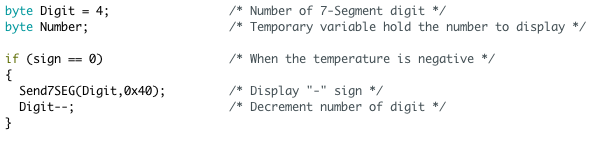
* For one of the **additional feature**, we use ping sensor to detect distance variations. In the ping\_mode, arduino will display the data sent from proximity sensor and send it to server through serial communication. The proximity sensor directly detects duration time read from pingPin and uses two functions to transform to distance in inches and cm. It will finally display distance in cm on the display bar.



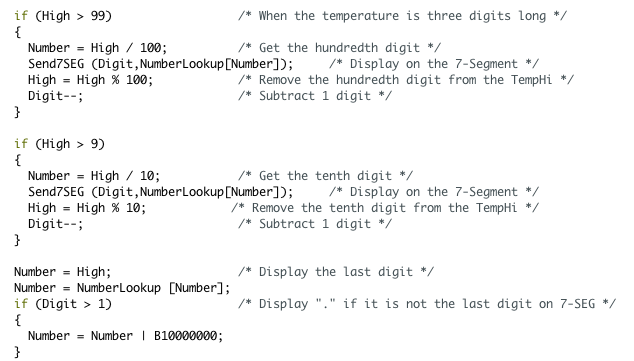
* For **another additional feature**, it will display current temperature sent from user interface. In weather mode, arduino will read in temperature sent from cpp server and simply display data in Celsius.



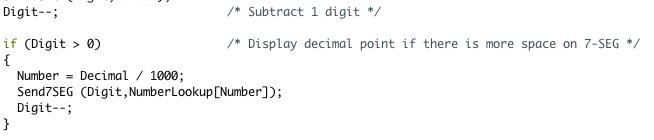
* For display part, we have a function called Dis\_7SEG which is from line 290 - line 349. It takes in the decimal part, the high and low byte of the integer part, and the bool type sign which indicates whether it is negative or positive.



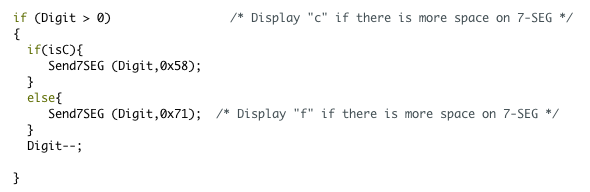
Above code deals with the negative temperature by properly setting 7-segment display bar’s high and low value.



Above code deals with each digit of the integer part in the order of high digit to low digit. If there is still enough place to hold decimal part, it will do the following and display decimal.



Besides, we also need to display right unit for the current temperature. This is done by properly enable the relative position of LED. The related code is shown below.



**Server:**

The server is written in cpp. The main function opens three threads.

The first thread is keep reading data send from arduino through the serial port, and processing the data in the right way, for example, the datas from temperature sensor are stored in a vector<double> to keep track of the history temperatures while the data from proximity sensor is stored as a string, which will be overwritten each time the new data comes in. The function first creates a file descriptor for Arduino. If the connection is successful, it starts to read bytes from Arduino and append the bytes to a string message until a ‘/n’ is read. Then it will see what current data is, if it’s from temperature sensor, then the data will be processed and stored into a vector, otherwise it will just simply saves the data from proximity sensor into a string.

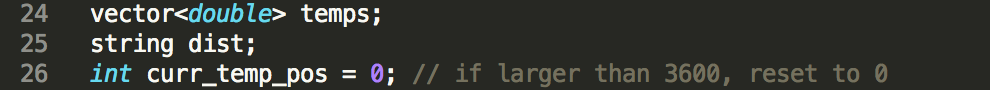
The second thread is keep server continues waiting for and handling requests from pebble. When a request arrives, the GET information is retrieved and corresponding response is sent to the client. If the request asks for temperature information, the server calls some functions to get the temperature statistics and respond with a JSON object. When the request asks to communicate with Arduino, such as change Celsius to Fahrenheit, displaying the weather of Philadelphia in it, the server sends the control or information message to Arduino by using write function.

The third thread is get user input from command line. When user enters a ‘q’, the whole connection will be closed, then the server is closed.

Detailed implementations are as following:

* Read data from Arduino:

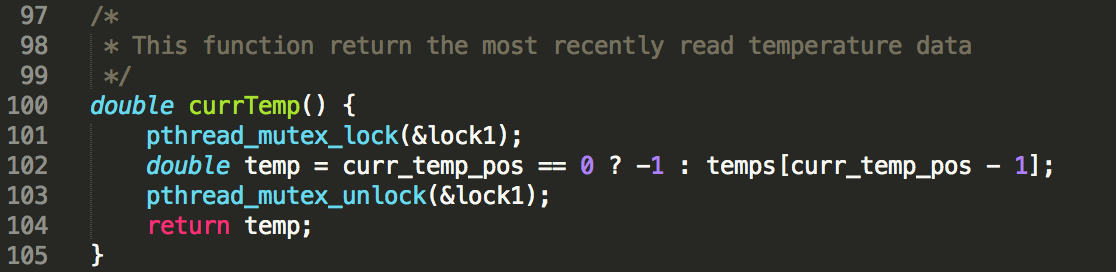
Function: void\* readData(void\* p); from line 36 -- line 95.



The data from temperature sensor are stored in a double vector, where can only contains the data from past one hour. This is implemented by an int to record the last data index read from Arduino, if the number of curr\_temp\_pos is larger than 3600 (Data is read once per second, 3600 seconds make up to 1 hour), it will be reset to 0, which makes the data that is already in the vector overwritten by a new one. Besides, the data from proximity sensor is saved into a string, which will be overwritten each time a new data comes in.

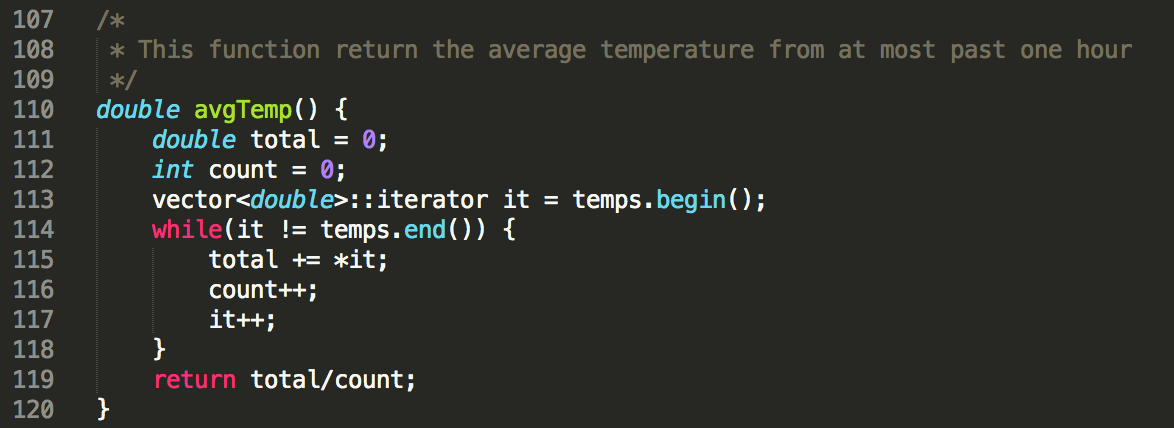
If the bytes read from the serial port becomes 0, then we will set the flag keeping track of whether the Arduino is connected successfully to be ‘not connected’.

* Keep track of temperature data history
  + Get current temperature:



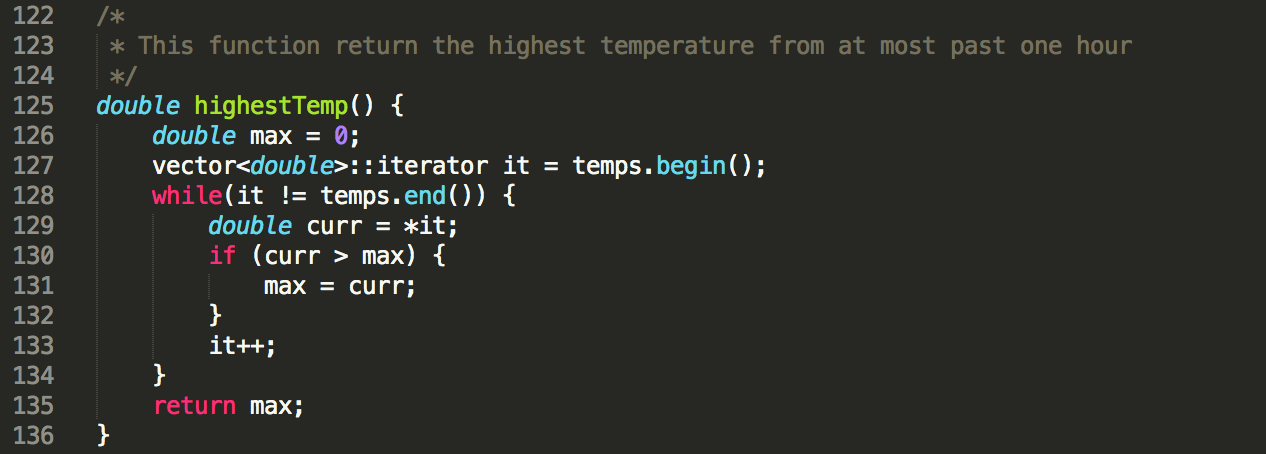
We will retrieve the current temperature by deduct 1 from the vector by curr\_temp\_pos, which record the next index position to store the data, then use the index calculated to get the most recently temperature.

* + Get average temperature:



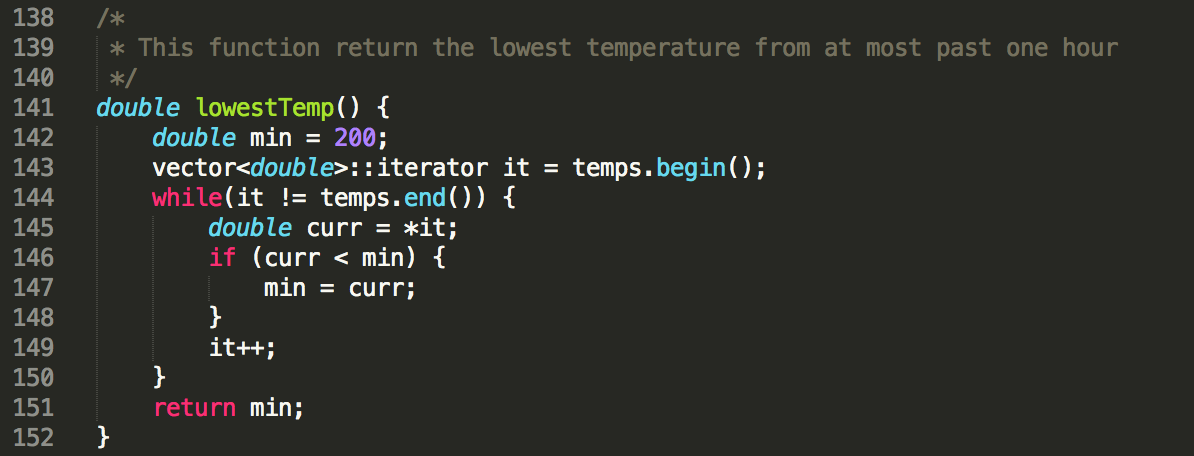
To get the average temperature, we iterate through the vector to calculate the number of temperature stored and the total amount of them. Then return the average temperature.

* + Get highest temperature:



By initialize the max temperature to be 0, then compare it with all the temperatures in the vector, then change it to be the larger one. At last, return the maximum value of temperature.

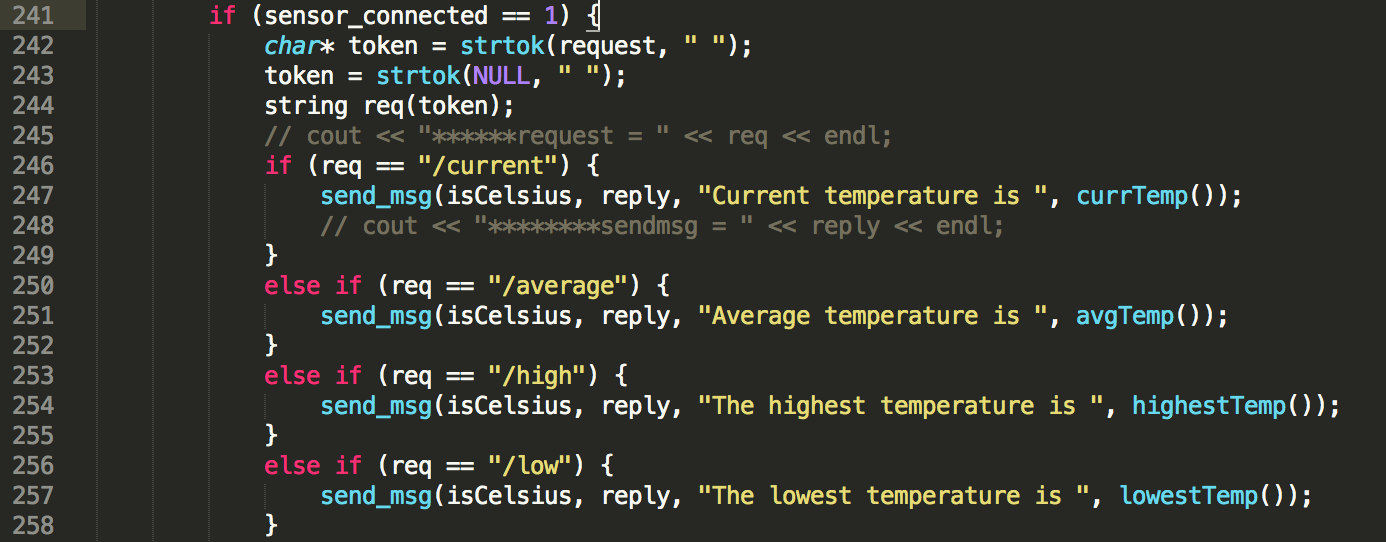
* + Get lowest temperature:

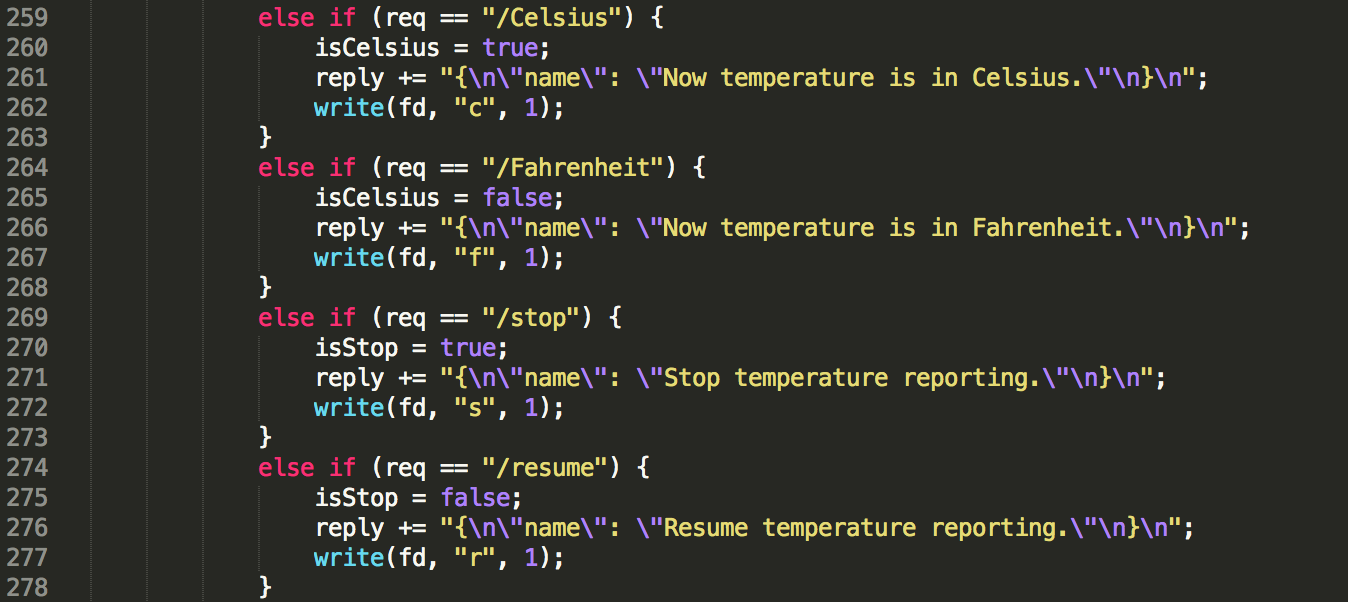


By initialize the min temperature to be a reasonable large number, here is 200, then compare it with all the temperatures in the vector, then change it to be the smaller one. At last, return the minimum value of temperature.

* Start server:

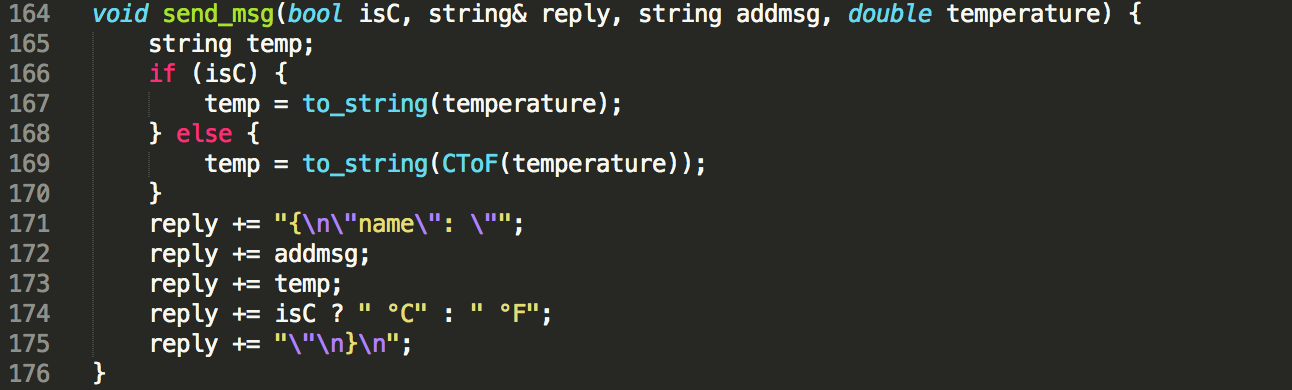
Function void\* start\_server(void\* p); from line 178 -- line 323





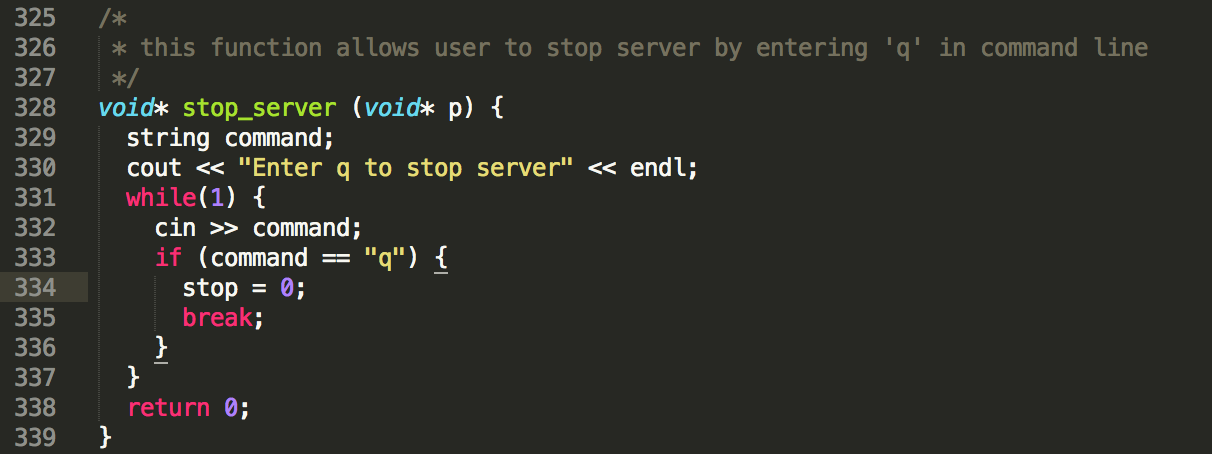
If the sensor is connected, we will keep waiting for the request from pebble, after the request is received, according to different request, reply message is attached to ‘reply’ and message will be send to Arduino if needed.

Here, we use a helper function called send\_meg.



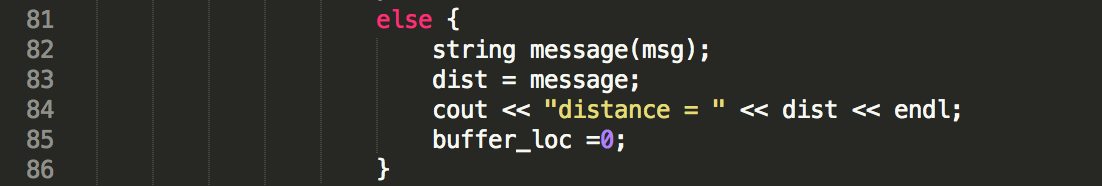
This helper function will get the temperature we need, and attach the right message to ‘reply’, which is in JSON format to be send through server.

* Stop server:

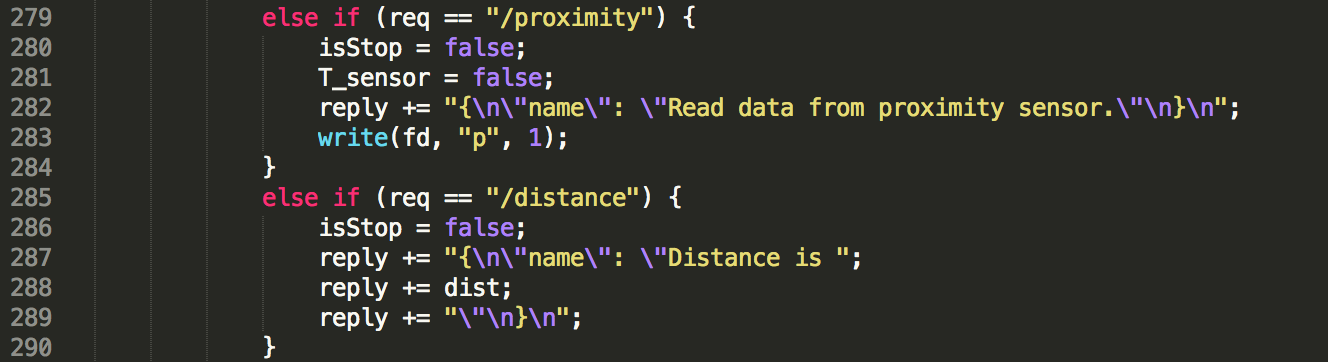


There is a global variable, which keep track if the user wants to stop the server. If the user enters ‘q’, then both readData and startServer will stop looping and all the threads in main function will join. Finally, the server close its connection in this way.

* Additional features:
  + Process data read from proximity sensor

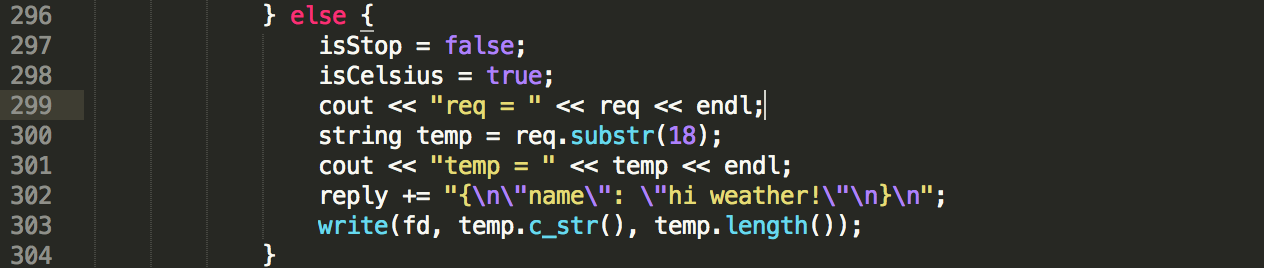


In readData function, read proximity sensor data and saved into a string.



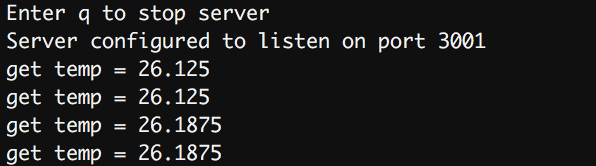
In startServer function, once we receive the request that client wants to get information from proximity sensor, the server will told Arduino to send data from proximity sensor. After that, if the client want to retrieved the data, the server will send the message to pebble.

* + Get updated philadelphia’s temperature and display it in Arduino:

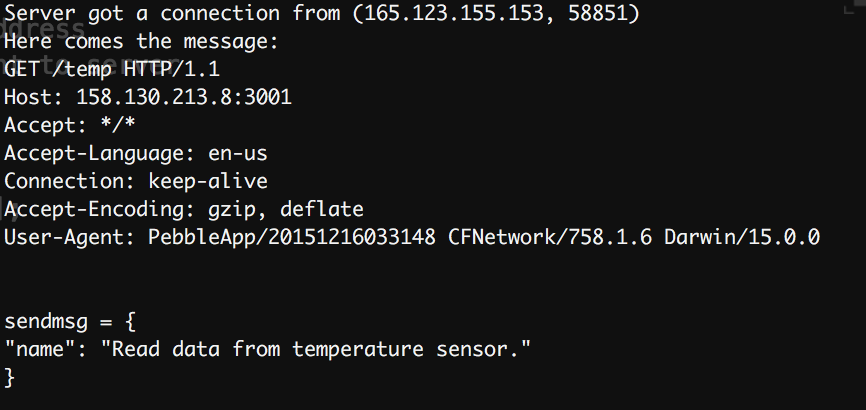


In startServer function, if the client wants to send current weather to Arduino, server will receive the number of temperature and then send it to Arduino for displaying.

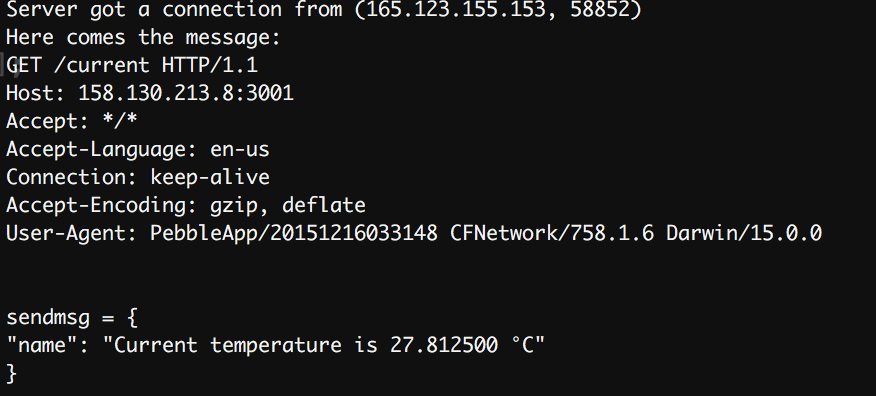
* + (Server part don’t contains code for the third additional feature)
* Messages printed in terminal:
  + Initial information displaying when starts the server.



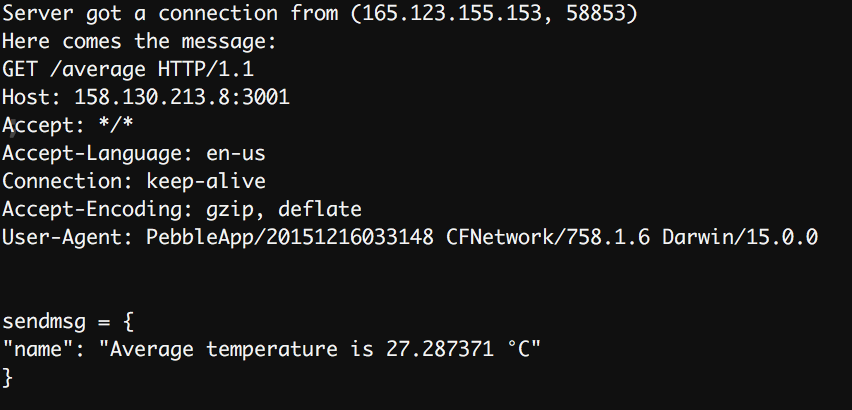
* + When client request /temp, then server send message back.



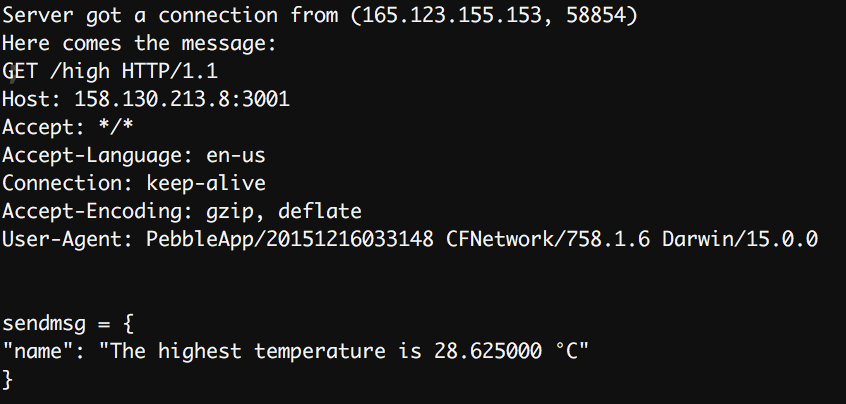
* + When client request /current, then server send message that contains the current temperature back.



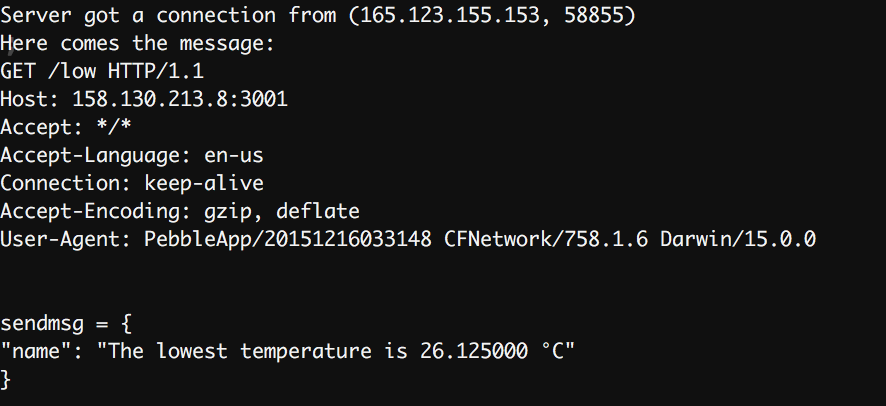
* + When client request /average, then server send message that contains the average temperature back.



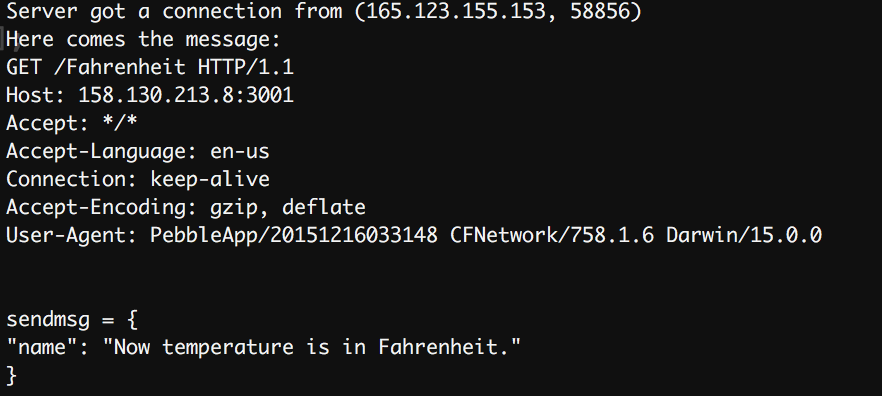
* + When client request /high, then server send message that contains the highest temperature back.



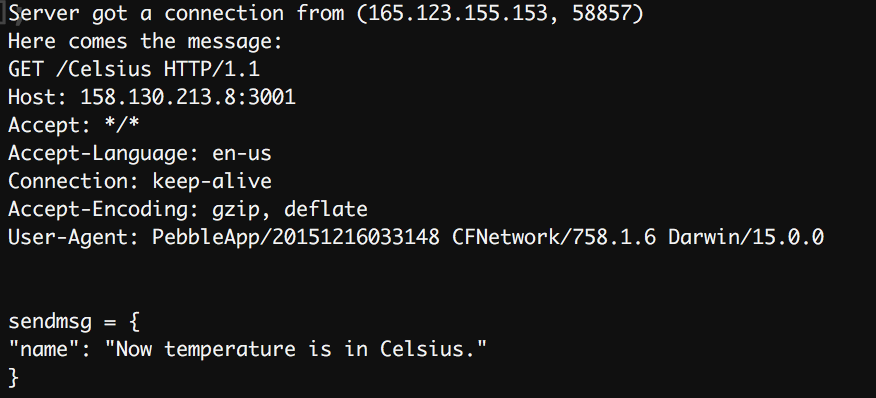
* + When client request /low, then server send message that contains the lowest temperature back.



* + When client request /Fahrenheit, then server send message back.



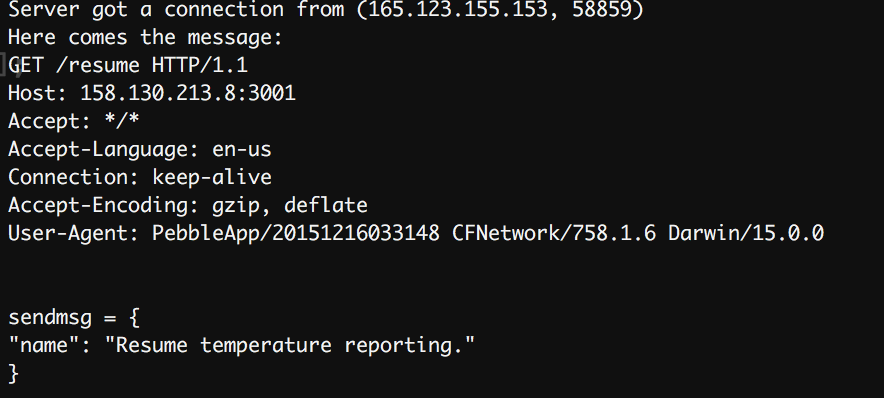
* + When client request /Celsius, then server send message back.



* + When client request /stop, then server send message back.



* + When client request /resume, then server send message back.



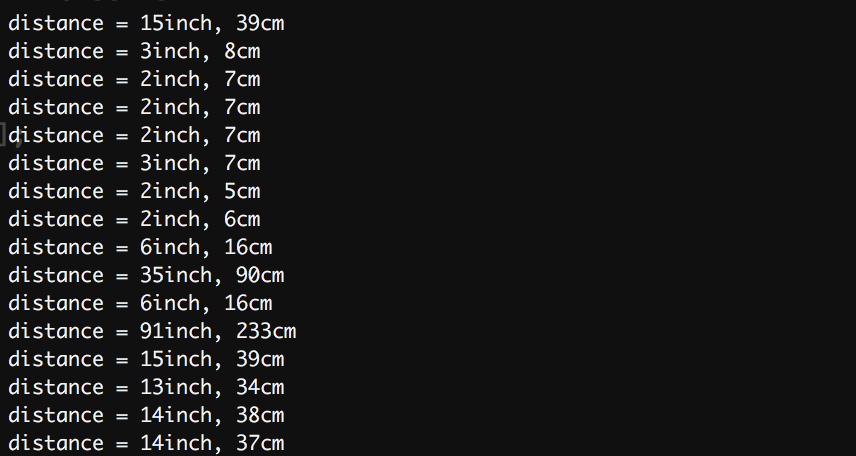
* + When client request /proximity, then server send message back.



* + When client request /distance, then server will send message that contains the distance data from proximity sensor back.



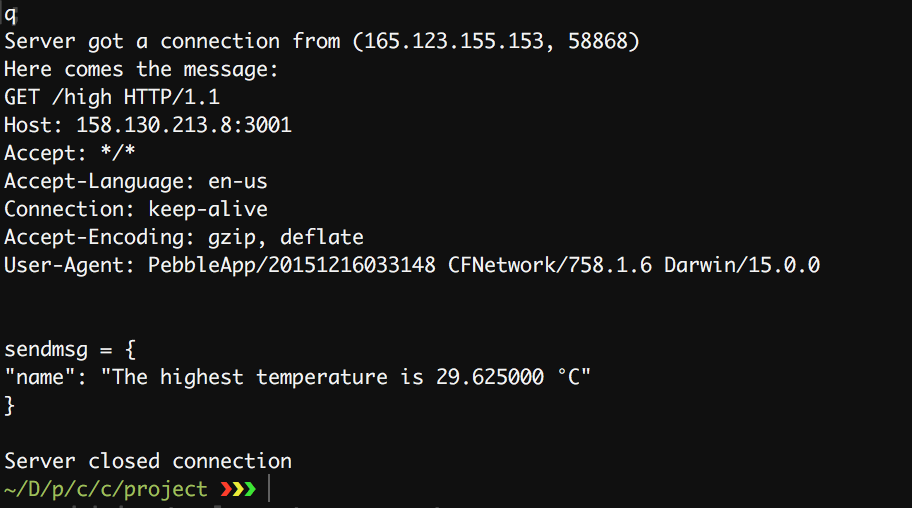
* + This the display of the data read from proximity sensor.



* + When client request /wether\_sendback, where the temperature of Philadelphia is attached as /5.9 in Celsius, then server will send message back.



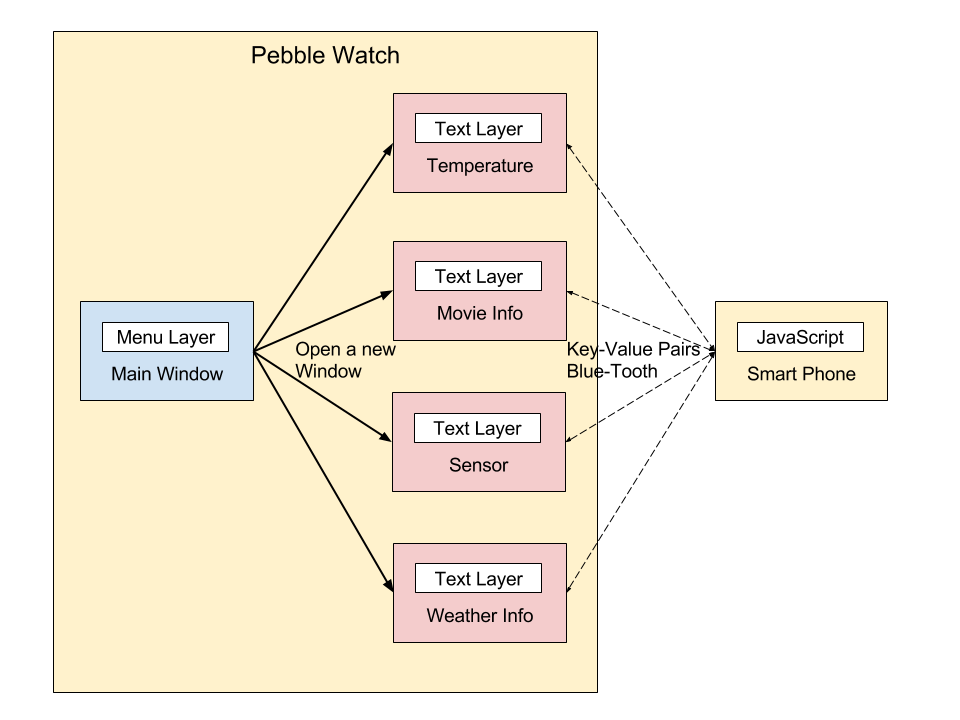
* + When we type in a ‘q’ in command line, after one more request, the server will close the connection.



**Pebble App :**

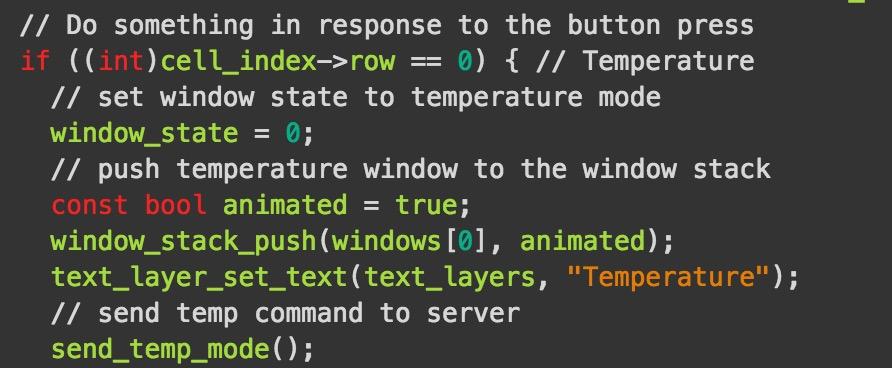


The Pebble App runs in C. It has four different tasks: temperature data retrieving, movie information gathering, weather broadcast and sensor data display. To combine all the tasks in one programme, multiple-window-layer technology was applied. The main window contains a menu layer, which shows all the four features you can use. After one of them is chosen, a new window will open, send messages to a smart-phone, and show the results on the watch screen.

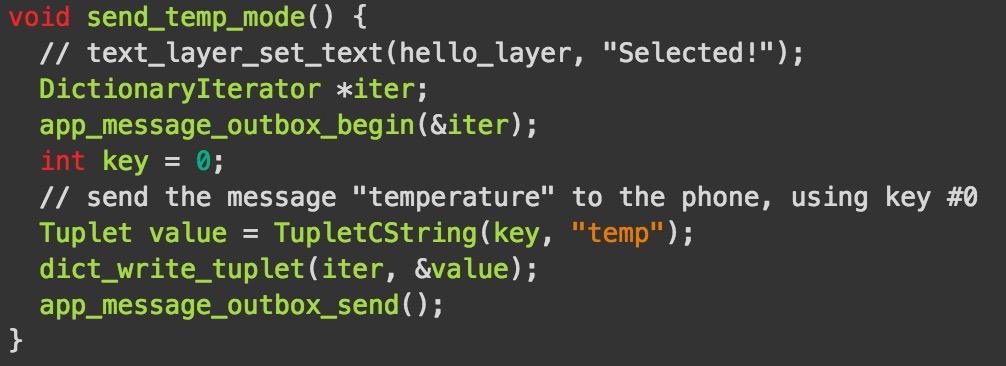


Generally, when a child window is opened, the watch will send a key-value pair to the smart-phone after SELECT button is pressed, and then wait for the send back result. But each task has slightly different features. More specific details are as followed.

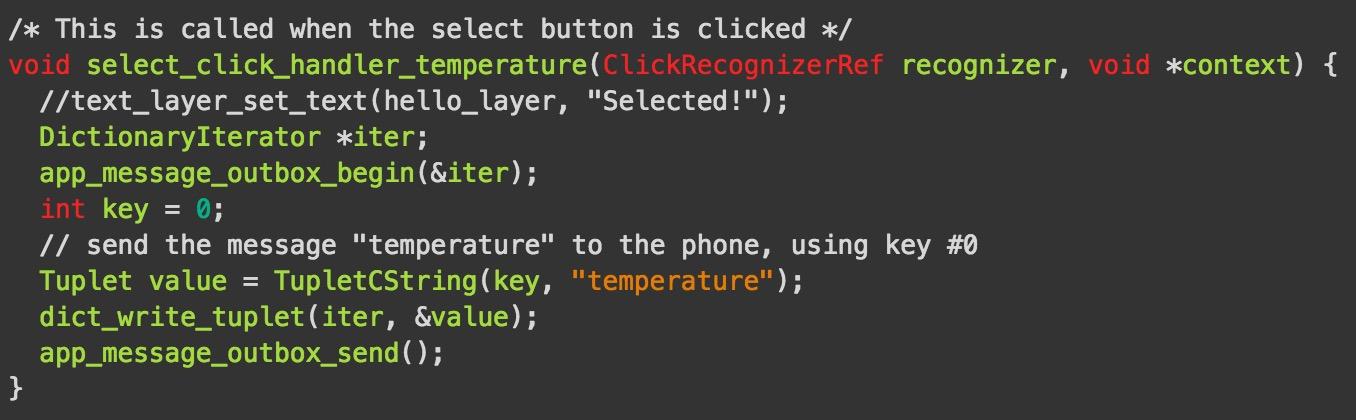
*Temperature:*



1. After the temperature window is open, it will automatically send a key-value pair: 0-temp to the cell phone through wifi. After the cell phone receive this message, it will tell the server be ready to transfer temperature data. Then, the pebble watch will wait for the message send back from the cell phone.



1. If you click SELECT button in temperature mode, it will send “0-temperature” to the cell phone. After receiving that message, the cell phone will send “current”, “highest”, “lowest”, “average” requests to the server. Then the watch will wait for the data send back from the cell phone.



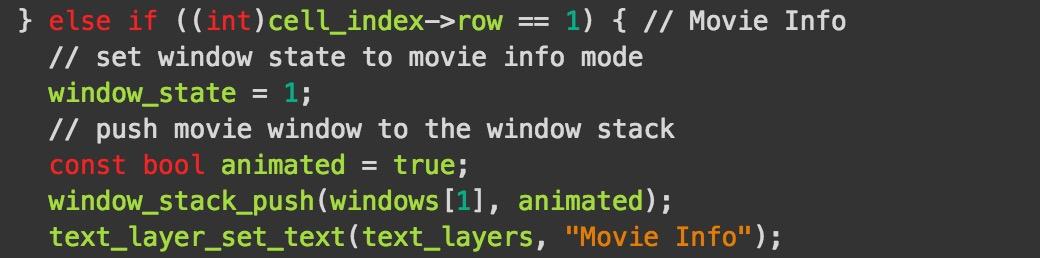
1. If you click UP button in temperature mode, it will send “0-unit” to the cell phone. Then the cell phone will send a unit-change request to the server. And the watch will display the message sent back.



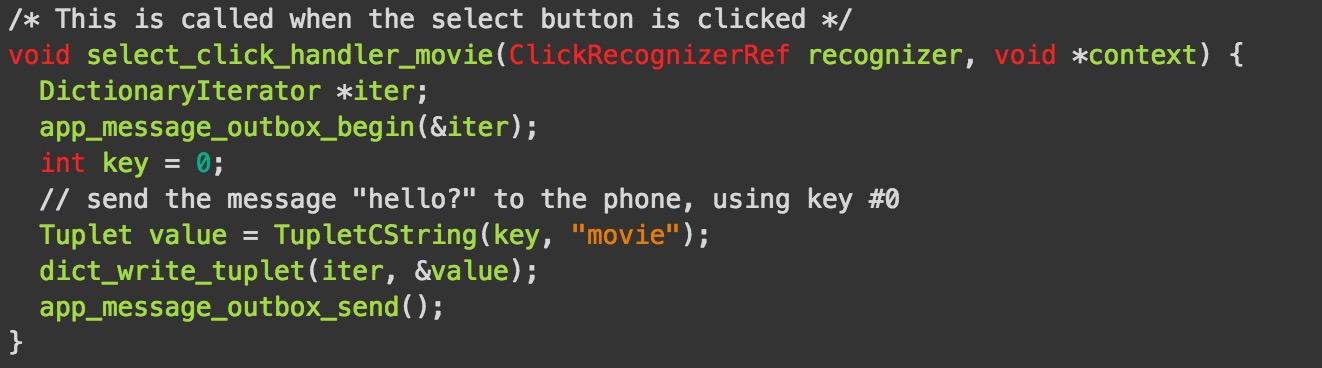
1. If you click DOWN button in temperature mode, it will send “0-standby” to the cell phone. Then the cell phone will send a “resume” or “stop” request to the server. The watch will display the message as well.



*Movie:*



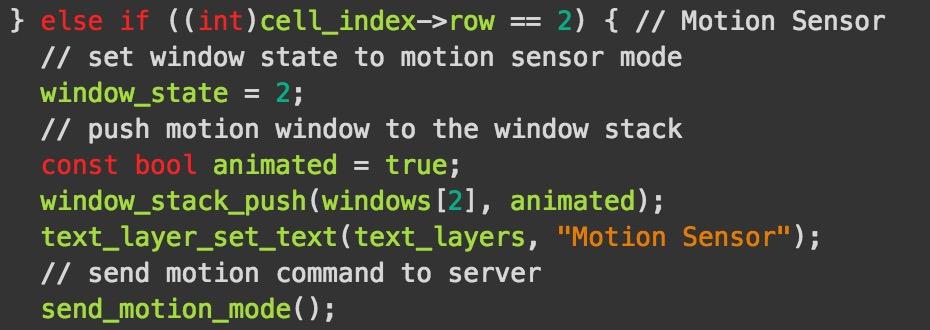
1. In the movie window, only SELECT button is implemented. After you click the button, 0-movie message will be sent to the cell phone. After receiving that message, the cell phone will send a http request to a movie web site and retrieve the content of the response. When the data processing is finished, it will send today’s movies back to the watch.



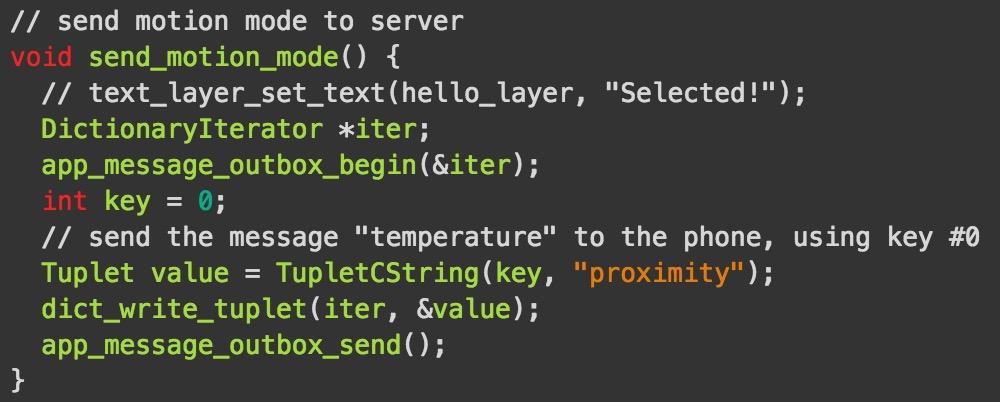
1. However, if the value in key-value pair is too long, it will cause an error. Therefore, the cell phone will send the movie name one by one. After receive one, the watch will inform the cell phone SUCCESS and then the cell phone will send next name. When the pebble get all the data, it will display them on the screen.



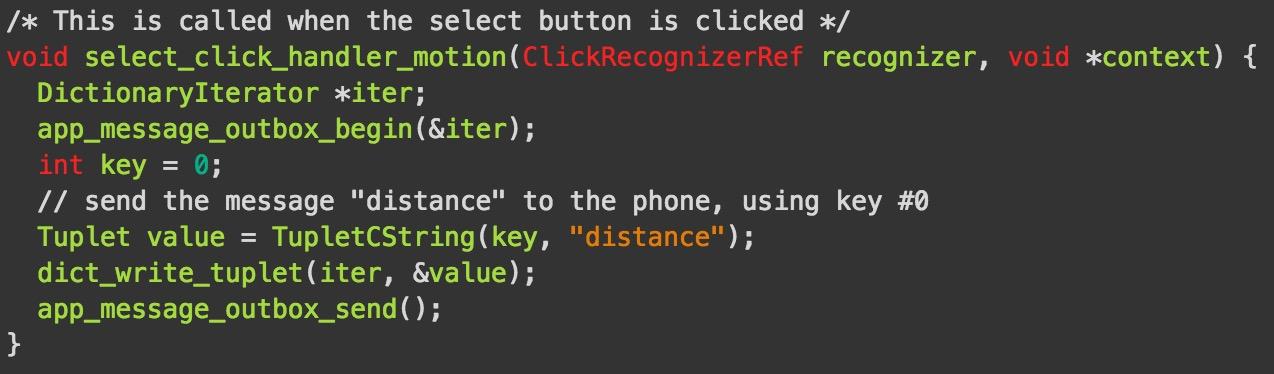
*Proximity Sensor:*



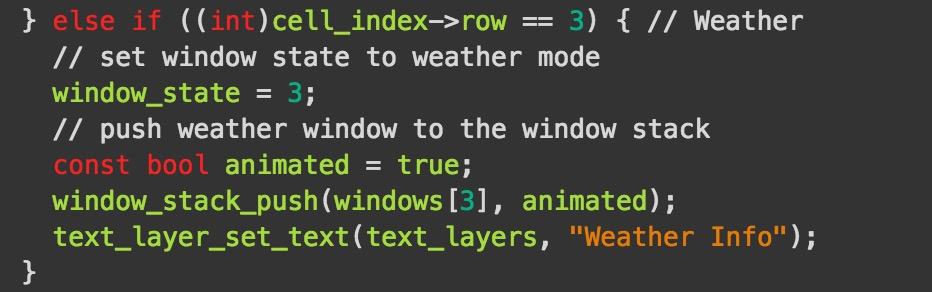
1. When the sensor window is opened, it will send a message “0-proximity” to the cell phone automatically, then the cell phone will inform the server to switch to sensor mode. Then the pebble watch will display the response.



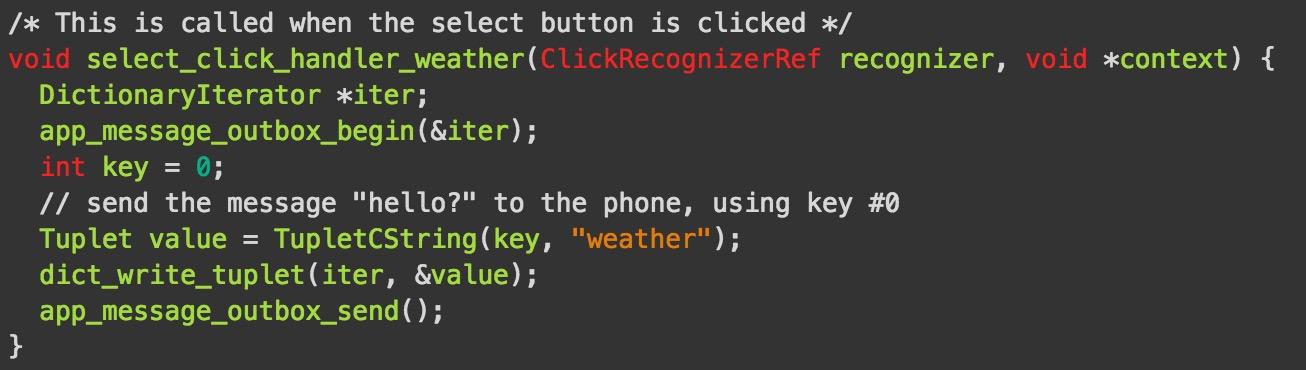
1. After you click the button, it will send “0-distance” to the cell phone. Then the cell phone will send a sensor data request to the server. And the watch will display the message sent back.



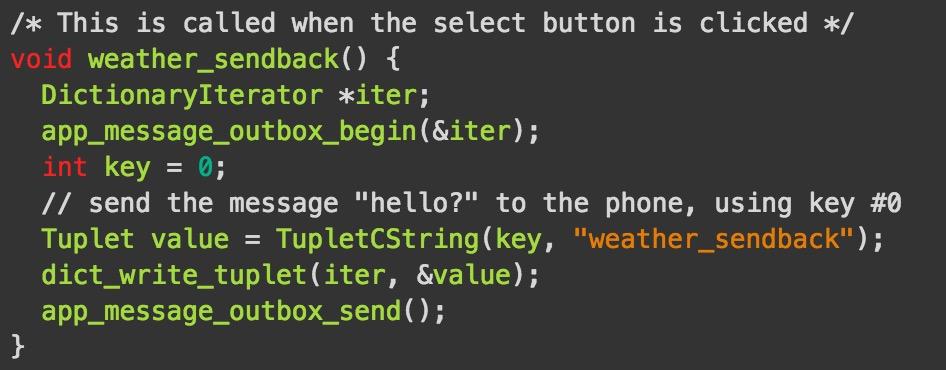
*Weather:*



1. Similar as the movie mode. After you click the SELECT button, message “0-weather” will be sent to the cell phone. Then the cell phone will send a request to a weather and retrieve the responseand send back to the pebble watch.



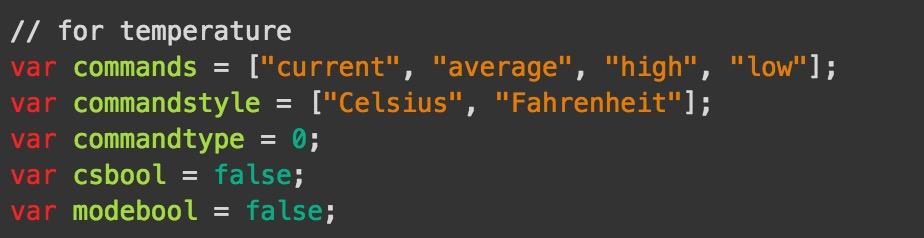
1. The response is too long for one key-value pair. Therefore, the cell phone will send weather information one by one. When receive one, the watch will inform the cell phone SUCCESS and then the cell phone will send next name. When the pebble get all the data, it will display them on the screen.
2. After the pebble watch display the weather information, it will try to send a new message to the cell phone. Then the cell phone will send the weather information to the server for Arduino to use.

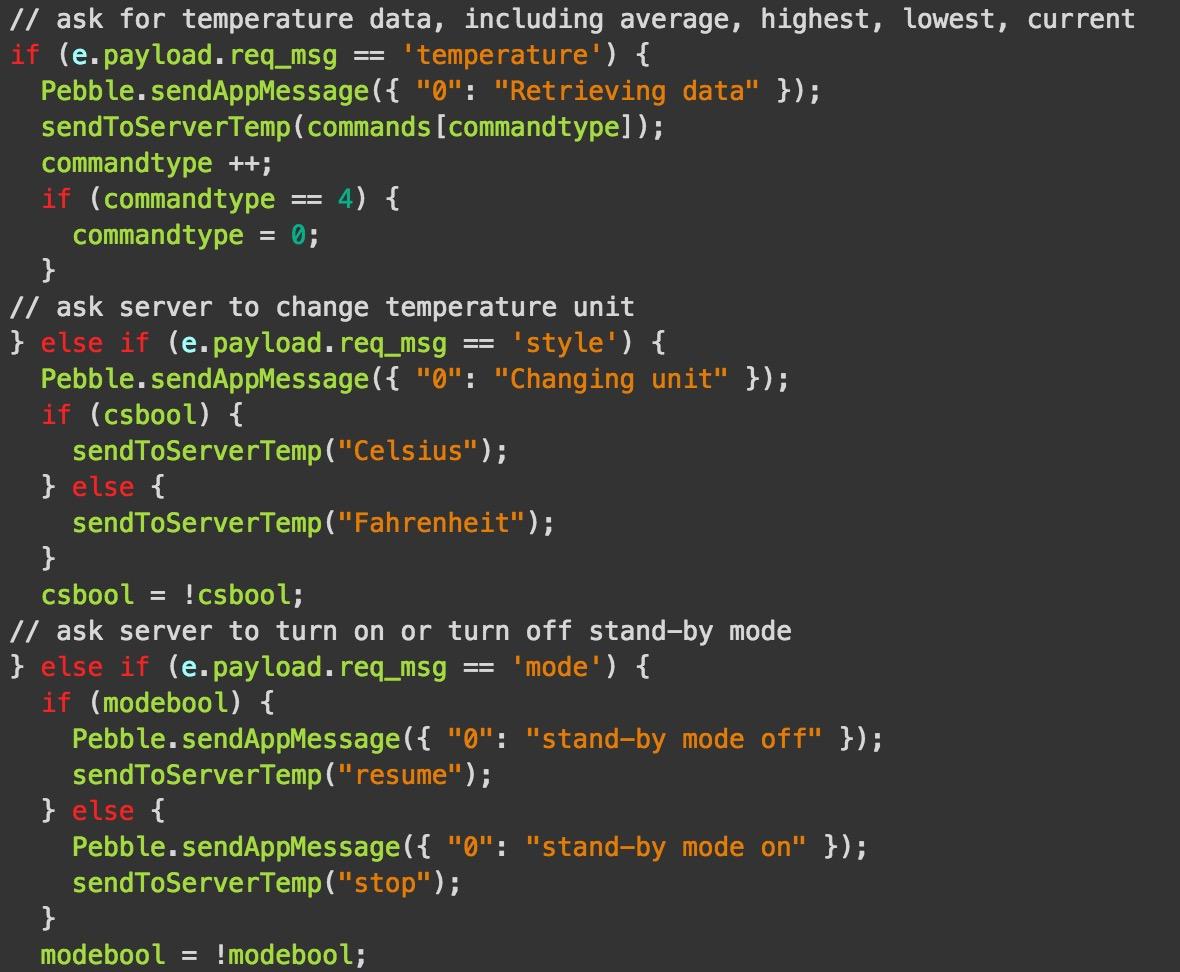


**JavaScript:**

The javaScript code runs on the cell phone. What it does is receiving messages from the watch, processing messages, sending requests to the server or a web sites, retrieving response, processing response, sending back messages to the watch. More details are as followed.

*Temperature:*





If the JavaScript receives message “0-temp”, it will send a request like “GET /temp HTTP/1.1” and something else. If receives message “0-style”, it will send a request like “GET /Celsius HTTP/1.1” or “GET /Fahrenheit HTTP/1.1”.When receiving message “0-mode”, it will send a request like “GET /resume HTTP/1.1” or “GET /stop HTTP/1.1”. After the cell phone receive response, it will send back whatever it got from the server to the watch. There are three flags to flag stand-by mode, unit state and temperature commands, so you can turn on and off of stand-by mode, switch the temperature unit and switch the temperature commands.

*Movie:*



When receiving message “0-movie” from the watch, the JavaScript will send a GET request to the website “http://www.cinemark.com/mobiletheatreshowtimes.aspx?node\_id =83876”. It is a mobile website, whose content is simpler than full sites. After get the website content in the response, it will split the content with keyword “title=\>” and then get the movies’ names. After that, all the names will be pushed in an array and send back the messages to the watch with *sendList(movie)* function.



The *sendList(movie)* is a template function in the developer guide in pebble website. Every time it will send one piece of message to the watch, if it get the SUCCESS callback, JavaScript will send next message, until all the messages are sent.

*Sensor:*



After “0-proximity” is received, JavaScript will send “GET /proximity HTTP/1.1” to the server, which will tell the server to switch to sensor mode. If “0-distance” is received, the JavaScript will send “GET /distance HTTP/1.1”. And then, the JavaScript send back the response back to the watch.

*Weather:*

Similar as the movie mode, JavaScript will send a GET request to <http://api.wunderground.com/api/94a5d903355cd511/conditions/q/PA/Philadelphia.json> after it receive message “0-weather”. It is a weather website. The response of the website is in Json format.



The JavaScript will parse the Json and send the message back to the watch. Also, JavaScript will send the messages one by one by using *sendList(weather)*. After the watch receive all the messages, it will send a new message “0-weather\_sendback” to JavaScript, and then, the cell phone will send the weather information to the server.

**Division of Labor:**

Middleware (server design) : Jue Liu

Sensor and Display (Arduino programming) : Ao Sun

Pebble App & JavaScript: Xiaojun Sun