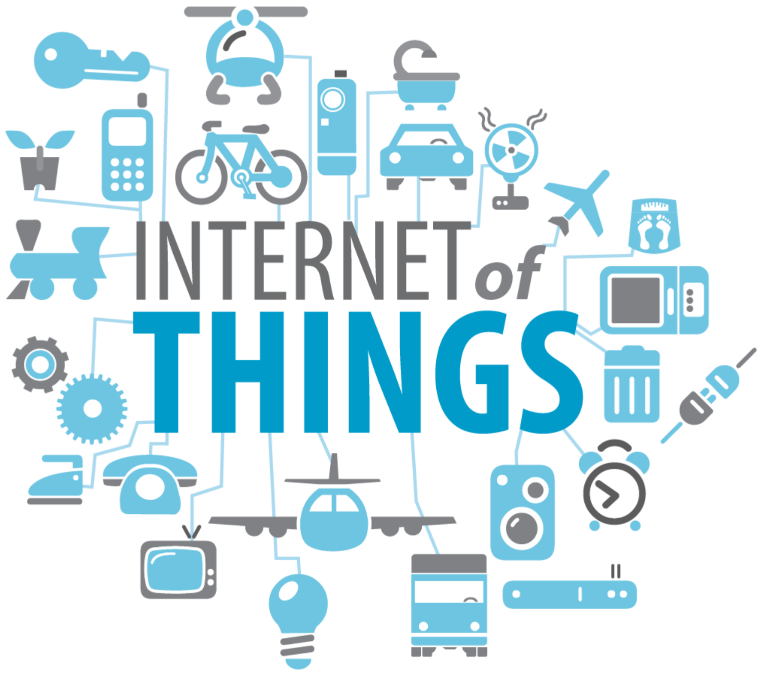
DEVELOPERS MANUAL

FRONT END PROGRAMMING OF IoT SMART OBJECTS



Submitted By:

Kaustav Ghosh

Jay Shah

Rohit Makhija

Ping He

Ketan Rajput

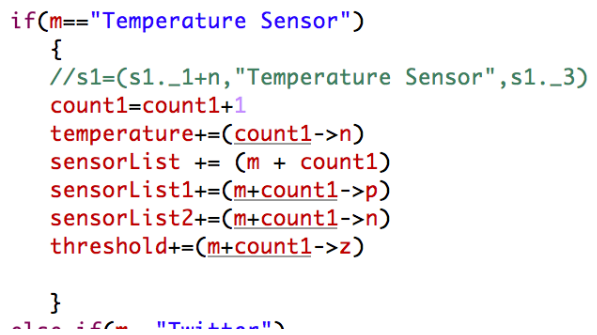
Fall 2014-COEN 296 Project 2

1.INTRODUCTION

This document gives you an overview of the internals of the application, so as to help a developer modify the code to build his/her application. So we walk you through,step by step through the modules of the application in Scala.

2.CODE WALK-THROUGH

Initially when the user opens the application, their would be a blank screen where the user would have to make his/her network. The user will start to make the network by the addition of Smart Objects to the screen using the Add Nodes option in the MenuBar.
(Note:All modules defined in the Main Object)
2.1. Addition of Nodes
We add the nodes using the Nodes option from the application in the Menubar. When a user adds a node, there is a unique name which is given to that sensor based on the number of the sensor type. So suppose the user wants to add the first Temperature Sensor, the unique name of that sensor would be stored as Temperature Sensor1. Similarly when the user adds a second instance of Temperature Sensor, the unique name for that Temperature Sensor would be Temperature Sensor2. So this ensures that each sensor has a unique name or resource ID. While adding the nodes, the user is also expected to enter a unique color for that particular node and a node corresponding to that color is drawn on the canvas.
The following diagram (Fig 1.)shows you how the temperature list stores each of the unique names of the temperature sensors using the count1 value and the type of the sensor. Each type of sensors is added similarly.



```
if(m=="Temperature Sensor")
   {
   //s1=(s1._1+n,"Temperature Sensor",s1._3)
   count1=count1+1
   temperature+=(count1->n)
   sensorList += (m + count1)
   sensorList1+=(m+count1->p)
   sensorList2+=(m+count1->n)
   threshold+=(m+count1->z)

   }
```

Fig 1.

Before discussing any further, we would like to show you the BorderPanel layout which we use for the overall layout of the screen. The east side of the Panel is the "Input Window" which will display :

1.Available Nodes in the network

2.All Connections presently active in the network

3.Simulator section which has the "Simulate" button to display the output of the network

The Center of the BorderPanel consists of the Canvas screen which logically shows all the sensors added to the network.

The South part of the BorderPanel consists of the Output window which displays the result of the network after we press the "Simulate" button.

The following diagram (Fig 2.) shows the code for defining the layout of the application.

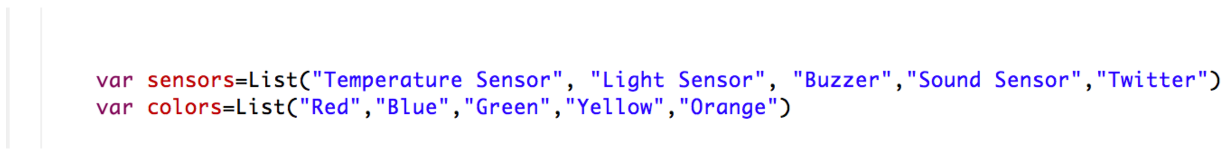
```
contents = new BorderPanel{

    layout(input2) = East

    layout(output) =South

    layout(screen) = Center


  }

}
```

Fig.2

The description of the Lists and Maps depicted in Fig.1. is as follows:
1)The temperature() list stores the names of the temperature sensors available currently
2)The sensorList() list is a list of maps containing all the available sensor nodes in the network. This list is often referred to as the “Master List”.
3)The sensorList1() list is again the list of maps containing the mapping between each available sensor in the network with its corresponding sensor value.
4)The sensorList2() list is also a list of maps containing a mapping between each of the available sensor in the networking with its corresponding unique color.
5)The threshold() list is a list of maps containing a mapping between each of the available sensors in the network and its corresponding threshold values. We’ll discuss about the threshold values in a bit more detail further in this manual.

Understanding the concepts of these above mentioned Lists is extremely crucial to understand the rest of the code.

One more List which we would like to talk about is the list of type of sensors for which we have presently made this application. Number of type of sensors can easily be increased by appending the type of sensors to this list. The definition of this list is given below.(Fig.3)



```
var sensors=List("Temperature Sensor", "Light Sensor", "Buzzer","Sound Sensor","Twitter")
var colors=List("Red","Blue","Green","Yellow","Orange")
```

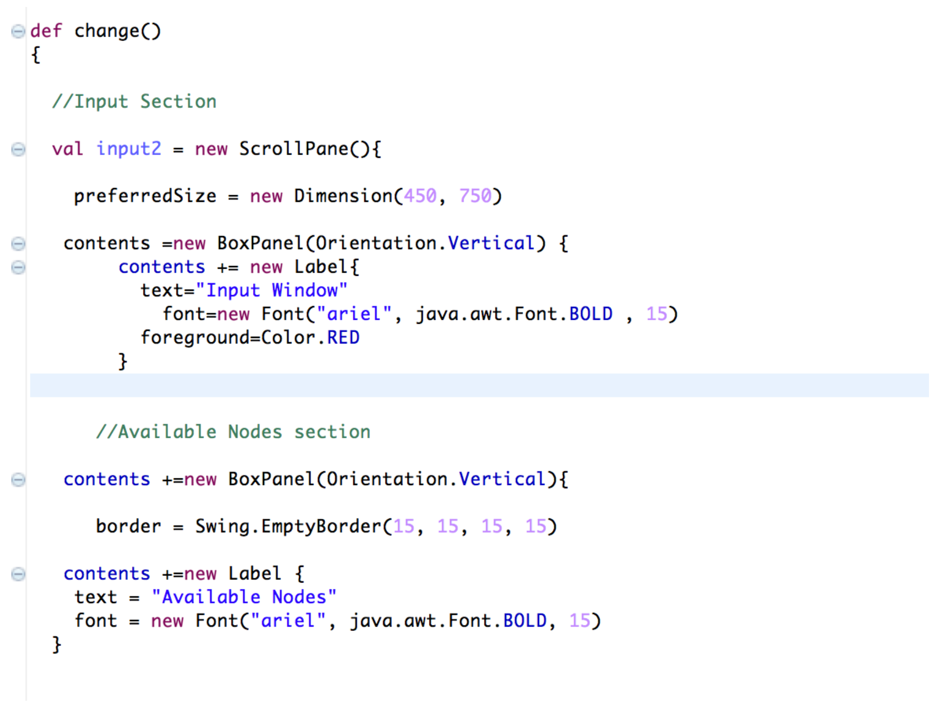
Fig.3.

Fig.3. defines the List of type of sensors available to us and the List of colors which we can uniquely select for each of the nodes. We can easily extend both of these Lists to include more type of sensors and more colors.

2.2. Addition of nodes to the Input Window and Center Canvas Screen

After we add a node through the menubar, each of these sensors should come up on the Input Window as well as the Canvas. The funct() function is called when adding nodes. This function stores the sensors in their appropriate data structure.

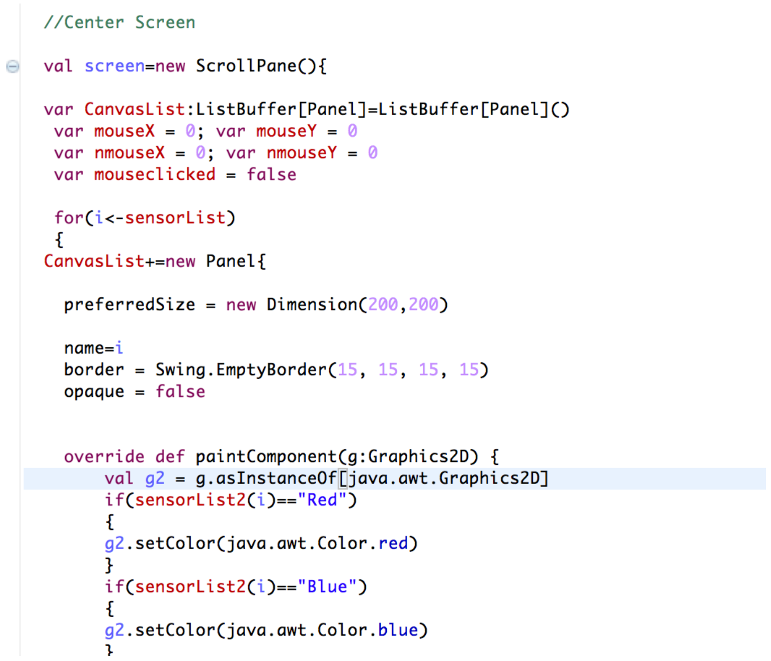
We have defined the change() function which adds the nodes to the Input as well as to the Center screen. The figure below(Fig.4.) shows the start of the change() function wherein we divide the Input Window into three parts as mentioned above and we add the nodes dynamically to the "Available sensors" region of the Input window.



```
def change()
{

  //Input Section

  val input2 = new ScrollPane(){

    preferredSize = new Dimension(450, 750)

   contents =new BoxPanel(Orientation.Vertical) {
        contents += new Label{
          text="Input Window"
            font=new Font("ariel", java.awt.Font.BOLD , 15)
          foreground=Color.RED
        }


      //Available Nodes section

   contents +=new BoxPanel(Orientation.Vertical){

      border = Swing.EmptyBorder(15, 15, 15, 15)

   contents +=new Label {
    text = "Available Nodes"
    font = new Font("ariel", java.awt.Font.BOLD, 15)
  }
```

Fig.4.

Similarly inside the change() function itself, we have the provision of adding the nodes dynamically on the Canvas screen. The snapshot of the code below(Fig.5) shows the initial code for the addition of nodes dynamically on the Canvas.



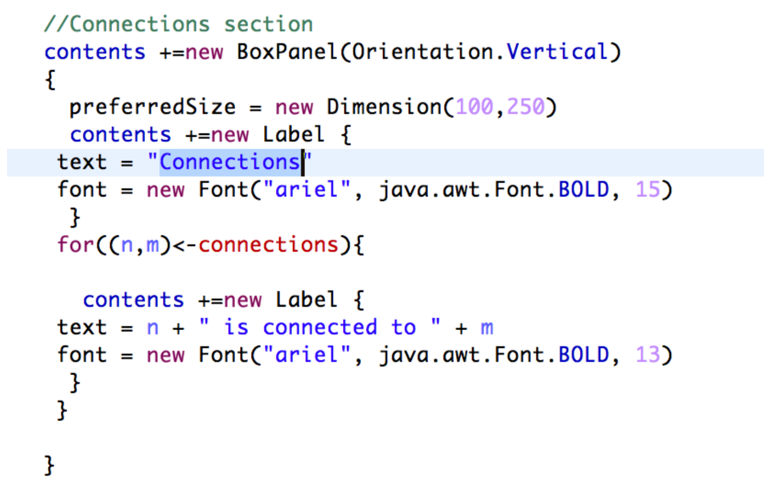
```
//Center Screen

val screen=new ScrollPane(){

var CanvasList:ListBuffer[Panel]=ListBuffer[Panel]()
 var mouseX = 0; var mouseY = 0
 var nmouseX = 0; var nmouseY = 0
 var mouseclicked = false

 for(i<-sensorList)
 {
CanvasList+=new Panel{

  preferredSize = new Dimension(200,200)

  name=i
  border = Swing.EmptyBorder(15, 15, 15, 15)
  opaque = false


  override def paintComponent(g:Graphics2D) {
      val g2 = g.asInstanceOf[java.awt.Graphics2D]
      if(sensorList2(i)=="Red")
      {
      g2.setColor(java.awt.Color.red)
      }
      if(sensorList2(i)=="Blue")
      {
      g2.setColor(java.awt.Color.blue)
      }
```

Fig.5.

2.2 Making Connections in the Network

Using the Menubar we need to add connections between the available nodes similar to the addition of the nodes. The active connections which we form, will be stored in a connection() list of maps which will map the sensors which form the active connection.The func1() is called upon when a new connection is made to store the new connection in an appropriate data structure.

After we add a connection we again call the change() function which modifies the Connection Section in the Input Window section. The following snapshot (Fig.6.) depicts the code for adding a connection in the change() function.



```
//Connections section
contents +=new BoxPanel(Orientation.Vertical)
{
  preferredSize = new Dimension(100,250)
  contents +=new Label {
 text = "Connections"
 font = new Font("ariel", java.awt.Font.BOLD, 15)
  }
 for((n,m)<-connections){

   contents +=new Label {
 text = n + " is connected to " + m
 font = new Font("ariel", java.awt.Font.BOLD, 13)
  }
 }

}
```

Fig.6.

2.3 Updating and Mutating Value of a Sensor

After addition of the nodes to the screen we get an area specific for a particular node in the UI which looks like the snapshot below(Fig.7.)

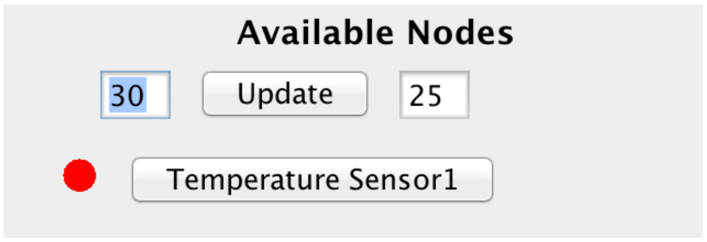
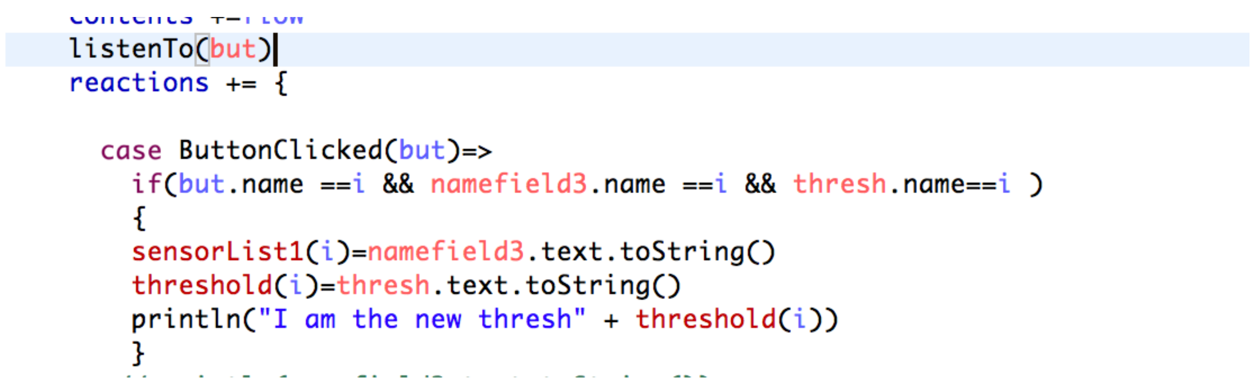


Fig.7.

So the textfield on the left side is for setting/updating value of that particular sensor and the textfield on the right is for setting the threshold value for that particular sensor.

We use the update button in between to update both the sensor value as well as threshold value of the sensor. We define threshold value as the value such that if the sensor value is greater then that then a connected sensor like a buzzer will beep and will update its value to 1(on state).

When we click the Update button the code in the snapshot (Fig.8.) below in the change() function is called upon and the threshold() and sensorList1() lists are correspondingly updated.



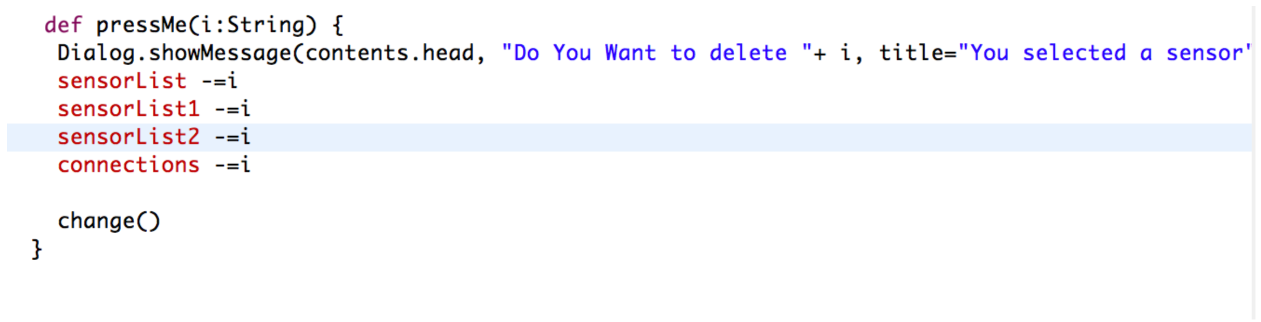
```
  contents +=flow
  listenTo(but)
  reactions += {

    case ButtonClicked(but)=>
      if(but.name ==i && namefield3.name ==i && thresh.name==i )
      {
      sensorList1(i)=namefield3.text.toString()
      threshold(i)=thresh.text.toString()
      println("I am the new thresh" + threshold(i))
      }
```

Fig.8.

2.4. Deletion of the Nodes and their corresponding connections

We can click on the sensor button in the Input Window which we want to delete and then confirm that we want to delete the node and then the node as well as all the connections associated with that node is deleted. The function pressMe() is called upon when deleting a node from the input window.

The following snapshot (Fig.9.) shows the code for the pressMe() function.



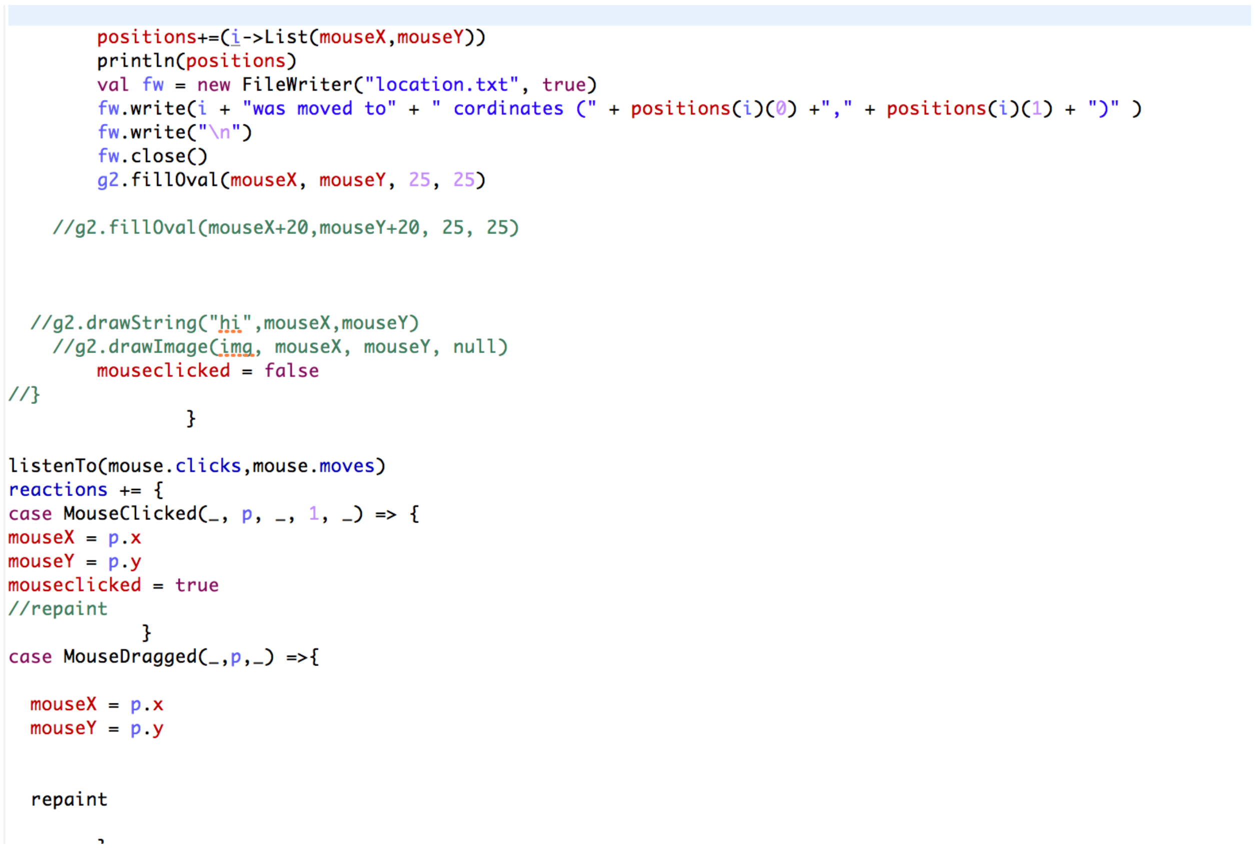
```
 def pressMe(i:String) {
  Dialog.showMessage(contents.head, "Do You Want to delete "+ i, title="You selected a sensor'
  sensorList -=i
  sensorList1 -=i
  sensorList2 -=i
  connections -=i

  change()
}
```

Fig.9.

2.5.Location Sensing of the Nodes

One of the main purposes of mapping the sensors on the canvas is for location sensing. We presently haven't implemented any location sensing based connections but developers can surely use this feature for developing location based sensing applications.

The sensors on the screen are draggable and the location of each of the sensors when moved is stored in a "location.txt" file. The implementation for this feature is done in the Center screen code of the change() function, the snapshot (Fig.10.) of which is given below. We have used the Even Handler classes of the mouse to locate the position of a particular sensor in the canvas.



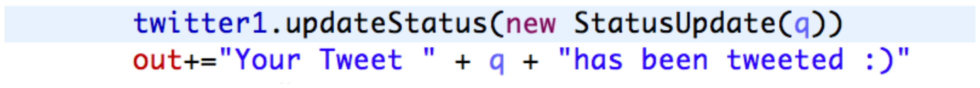
```
        positions+=(i->List(mouseX,mouseY))
        println(positions)
        val fw = new FileWriter("location.txt", true)
        fw.write(i + "was moved to" + " cordinates (" + positions(i)(0) +"," + positions(i)(1) + ")" )
        fw.write("\n")
        fw.close()
        g2.fillOval(mouseX, mouseY, 25, 25)

    //g2.fillOval(mouseX+20,mouseY+20, 25, 25)



  //g2.drawString("hi",mouseX,mouseY)
    //g2.drawImage(img, mouseX, mouseY, null)
        mouseclicked = false
//}
                }

listenTo(mouse.clicks,mouse.moves)
reactions += {
case MouseClicked(_, p, _, 1, _) => {
mouseX = p.x
mouseY = p.y
mouseclicked = true
//repaint
            }
case MouseDragged(_,p,_) =>{

  mouseX = p.x
  mouseY = p.y


  repaint
```

Fig.10.

2.6.Addition of the Twitter API

We have included a node which is named a Twitter. So if it is supposed connected to a Temperature Sensor and if the sensor value of the Temperature Sensor is greater than it's defined threshold value, the the value we set for the Twitter Node in the textfield at that moment is posted on the Timeline of the user whose authentication credentials are provided to the application.

We have defined the twitter connectivity in the perform() function. The following snapshot (Fig.11.) shows the method function to update a status using a twitter object-twitter1 which we have defined while the initialization of the Twitter Node.



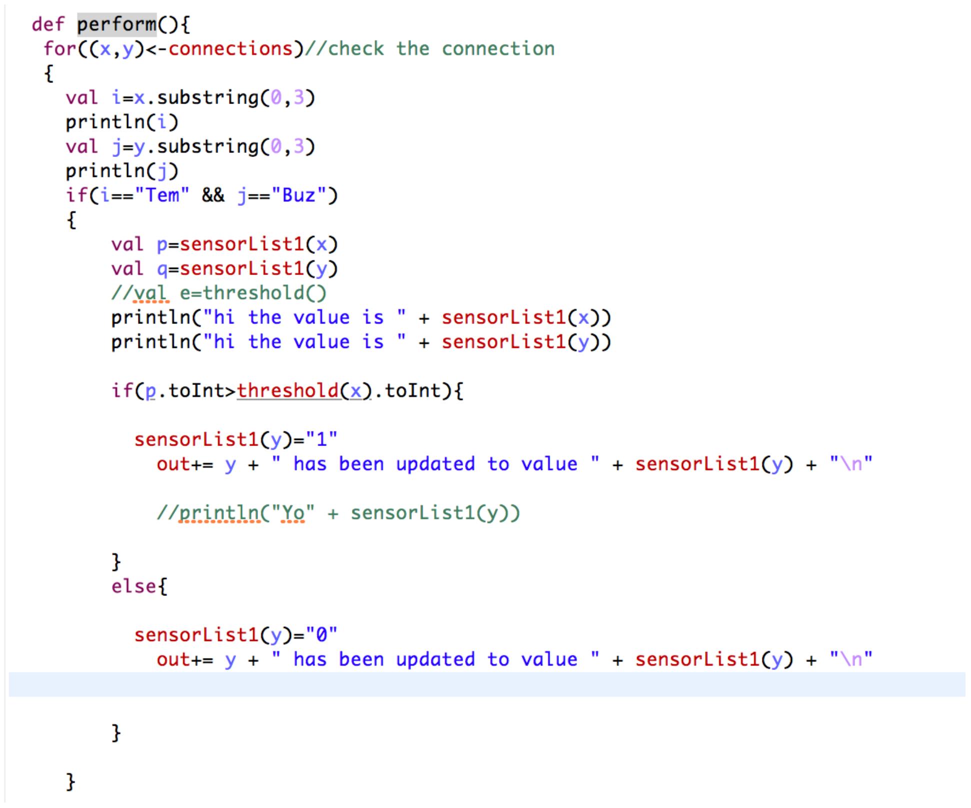
```
twitter1.updateStatus(new StatusUpdate(q))
out+="Your Tweet " + q + "has been tweeted :)"
```

Fig.11.

2.7.Simulating the Network

For simulating the entire network and get the desired output of the network we can press the "Simulate" button in the Simulation section of the Input Window.

The perform() function generates the desired output with respect to the connections made by the user and the thresholds values defined by the user. The snapshot (Fig. 12.) gives us a picture about how the Output gets generated depending upon the User's input of connections.



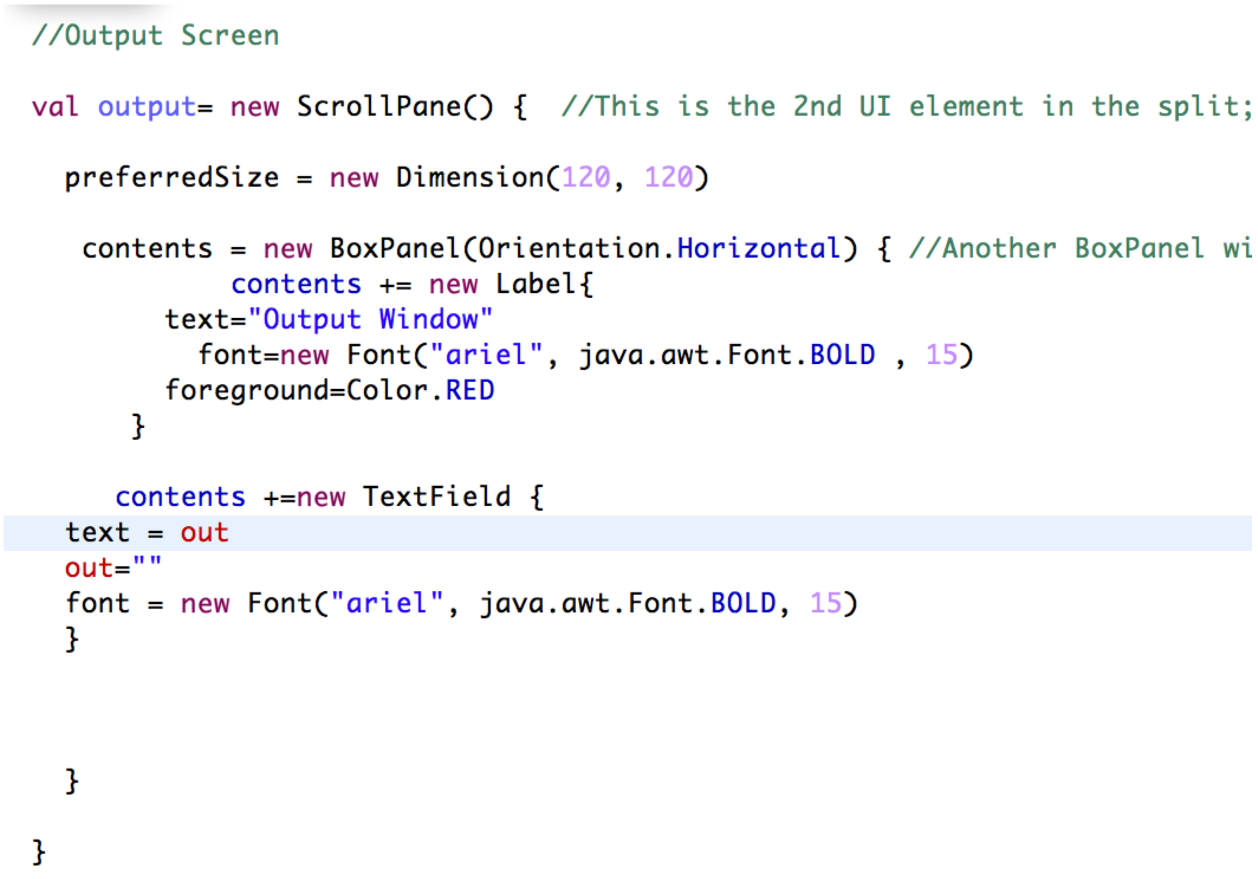
```
def perform(){
 for((x,y)<-connections)//check the connection
 {
   val i=x.substring(0,3)
   println(i)
   val j=y.substring(0,3)
   println(j)
   if(i=="Tem" && j=="Buz")
   {
       val p=sensorList1(x)
       val q=sensorList1(y)
       //val e=threshold()
       println("hi the value is " + sensorList1(x))
       println("hi the value is " + sensorList1(y))

       if(p.toInt>threshold(x).toInt){

         sensorList1(y)="1"
           out+= y + " has been updated to value " + sensorList1(y) + "\n"

           //println("Yo" + sensorList1(y))

       }
       else{

         sensorList1(y)="0"
           out+= y + " has been updated to value " + sensorList1(y) + "\n"


       }

   }
```

Fig.12.

2.8.Output

Once the perform() function computes the logic and the output of the network, it calls the change() function which changes the Output Window located at the bottom side of the application window.

We can have a look at the Output window section of the code in the change() function which displays the results of the network logic.(Fig.13.)



```
//Output Screen

val output= new ScrollPane() {  //This is the 2nd UI element in the split;

  preferredSize = new Dimension(120, 120)

   contents = new BoxPanel(Orientation.Horizontal) { //Another BoxPanel wi
            contents += new Label{
        text="Output Window"
          font=new Font("ariel", java.awt.Font.BOLD , 15)
        foreground=Color.RED
      }

     contents +=new TextField {
  text = out
  out=""
  font = new Font("ariel", java.awt.Font.BOLD, 15)
  }



  }

}
```

Fig.13.

3 .CONCLUSION

So our present development is open to extension, as in addition of different types of sensors with various logic conditions can be defined in the perform() function. Also we can add as many nodes we want to this application and have as many connections as we need, thus our application is scalable.We were able to implement functionalities like connect() for connecting objects, access() for reading the values of the sensors, mutate() for changing values of sensor nodes and also close() for deleting the nodes from the network. We were also able to successfully use the Twitter API for connecting the sensor nodes to Twitter.