ILR 04 – Progress Review 3

Dorothy Kirlew

Team Daedalus Members: Mohak Bhardwaj, Shivam Gautam, Pranav Maheshwari, and Richa Varma

November 10, 2015

1. Individual Progress

For the progress review on November 10, I worked on two aspects of the project – Bluetooth serial communication between an Android phone and a laptop, and publishing data from Sharp IR sensors to a ROS node using an Arduino Mega.

a. Bluetooth Serial Communication

Previously, the app sent .txt files over Bluetooth to a connected device when the user pressed the "Park" or "Return" button. For this progress review, my goal was to send the same messages, but through serial communication instead of through files.

To accomplish this, I added the BluetoothChatService.java provided by Android and used this to send a string to the specified connected device. Both the name of the device and the UUID of the connected device are specified in Constants.java (section 1.f). In the future, when we have multiple mobile platforms, these constants can be used for each instance of the app to connect to a specific vehicle.

I also modified the app so the Park and Return buttons can only be pressed once before they are disabled. For example, the user can press Park once, which sends the Park message and disables the Park button (Figure 1**: Park Button Pressed**). They can then press Return. When they press Return, the Return message is sent, the Return button is disabled, and the Park button is enabled (Figure 2**: Return Button Pressed**).

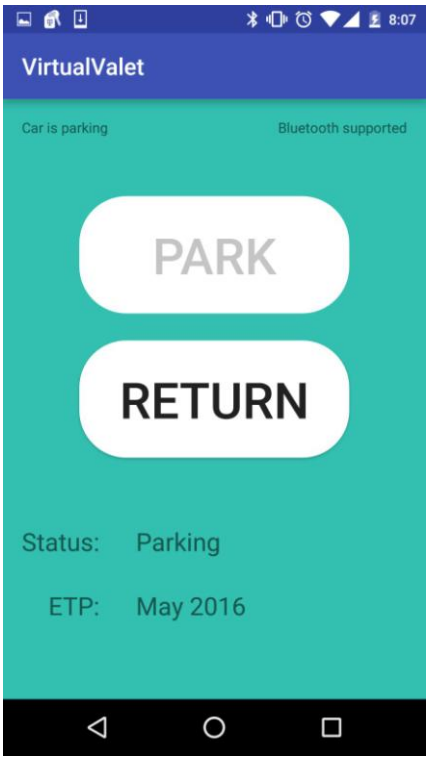

Figure 1: Park Button Pressed

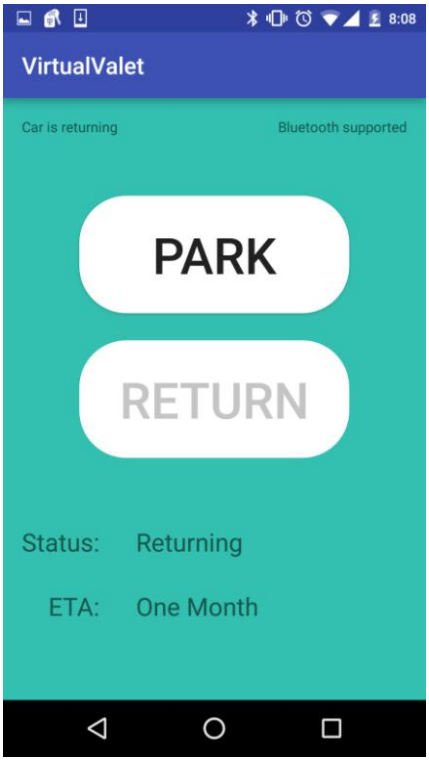

Figure 2: Return Button Pressed

When either of these buttons is pressed, a serial message is sent over Bluetooth to the connected device, which is then received and published on a ROS node, as seen in Figure 3**: Park Message Received** and Figure 4**: Return Message Received**.

Figure 3: Park Message Received, photo by Mohak Bhardwaj

```
mohak@mohak-PC:~/Documents/MRSD/MRSD Project/workspace/src/auto-park/bluetooth/s
rc$ ./bluetooth-listener.py
Waiting for connection on RFCOMM channel 1
('Accepted connection from ', ('CC:FA:00:71:69:F8', 1))
Park
Return
```

Figure 4: Return Message Received, photo by Mohak Bhardwaj

b. Sharp IR on ROS Node

The mobile platforms will have two sensing and perception systems – Kinect and IR. The Kinect has a range of 80 to 400cm. The Sharp IR sensor has a range of 10 to 80cm. Hence, the Kinect will be used to identify the surroundings of the vehicle and the Sharp IR sensors (see Figure 5: Infrared Proximity Sensor - Sharp GP2Y0A21YK) will be used as an emergency obstacle detection system. Specifically, the IR sensors are used to notify the Emergency Node of the ROS network when an obstacle is detected within the "danger zone". The danger zone is the area around the mobile platform where, if an obstacle is found within this zone, it is too late for the vehicle to route around the obstacle, and the vehicle must stop immediately to avoid collision. Currently, the danger zone is set to 20 cm, because the mobile platform, with a maximum speed of 10 cm/s, will be able to stop without collision when an item is detected within the danger zone.

Figure 5: Infrared Proximity Sensor - Sharp GP2Y0A21YK

I created an Arduino program (section 5.a) that polls the three connected Sharp IR sensors for the distance in centimeters between each sensor and the nearest obstacle. If any of the IRs detect an obstacle within the danger zone, the Arduino will publish the IR number (0, 1, or 2) and the distance of the obstacle. The Arduino prioritizes IR #0 over IR #1, which is prioritized over IR #2. This means that if more than one IR is detecting an obstacle in the danger zone, only the information from one IR will be

sent. If there are no obstacles within the danger zone, no information will be published. This is to avoid flooding the ROS node with useless information.

If an obstacle is detected within the danger zone, the Arduino will publish this information to a ROS node named “range_data”. The program currently used the following format: “#[i]-[r]”, where “#” is to distinguish the beginning of a new message, “i" is the IR number (0, 1, or 2), and “r” is the distance from the IR to the obstacle, measured in centimeters. Because the serial protocol has not been yet been defined, the message format is likely to change. However, an example of output from each sensor can be found in Figure 6**: IR Range Data**.

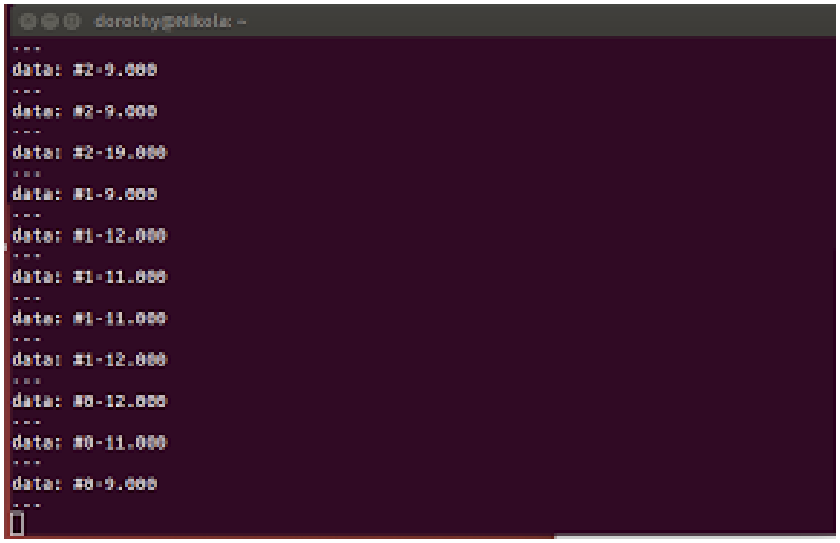

Figure 6: IR Range Data

2. Challenges

Sending a file over Bluetooth, which Richa and I did for the last Progress Review, was meant to be a stepping-stone towards sending serial messages. It turns out that sending files and sending serial messages are wildly different and using the structure for sending files to build towards sending serial messages harmed more than helped the process of sending serial messages. Almost all of that code was scrapped and it took me many hours and several useless tutorials to discover BluetoothChatService.java, which is a standard way of sending and receiving serial messages. Once I started using this, everything fell into place and it took just a few minutes to discover how to send serial messages.

Collecting and printing multiple IR readings to the serial monitor was fairly simple. My problem was creating the correct format to publish the readings to a node. The sample code for publishing IR readings was a noisy mess with entirely too much information, as seen in Figure 7**: Original IR Range Data**. I tried to create a simple format for publishing information, but it was difficult to combine the IR number and IR reading into one char array. The published array needed to be defined at a specific length and created in a specific manner. The most challenging part of this is that, regardless of syntactically correct code, ROS would throw errors with horribly vague descriptions that did not help me pinpoint the actual problem – that the char array needed to have a length of 13, not 5. Once the format was established, publishing this information went very smoothly.

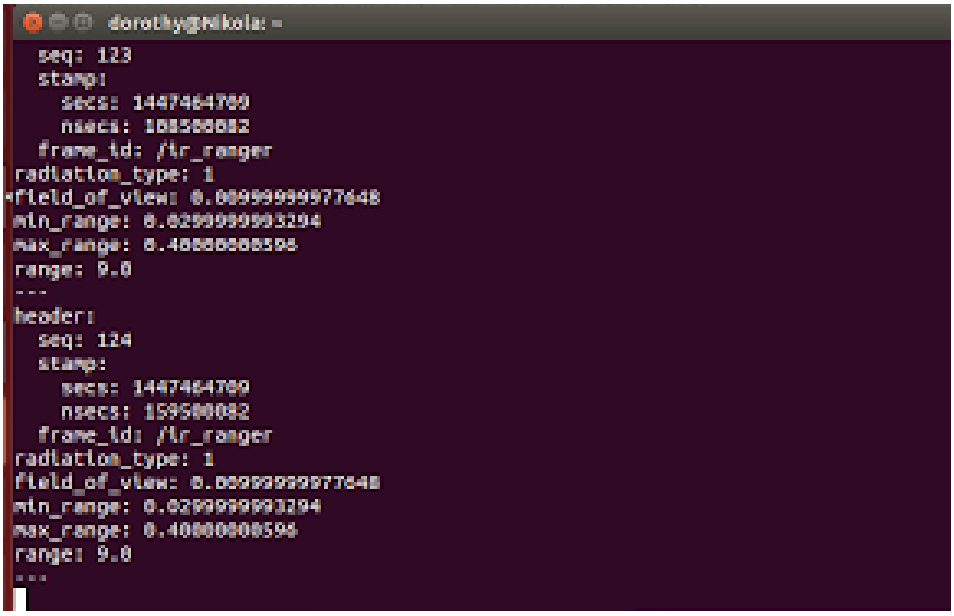

Figure 7: Original IR Range Data

There were three resource challenges I faced for this progress review. The first being an existing problem – I do not own an Android phone. Thus, I could code all I want, but I would not be able to test anything without the help of a teammate. Additionally, I needed a second laptop to connect to the Android phone. Android Studio runs on Windows and has a debugger than can be used with a connected Android phone. However, I needed a laptop with Ubuntu and ROS to connect to the phone over Bluetooth and display the serial messages published on a ROS node. Because Android Studio runs on Windows and ROS runs on Ubuntu, I was unable to debug the app and receive serial messages from the app at the same time, so I again needed to coordinate with a teammate to test the app. The final resource challenge is a personal one. I have been using my work laptop for all of my assignments and, having recently left my job, need to return this laptop. I have been trying to switch over to my personal laptop, but I had some problems getting Android Studio to work on my personal laptop.

3. Teamwork

We divided the work amongst our five team members as evenly as possible, with respect to our experience and skills. Pranav and Richa set up the XBee adapters that will be used to communicate between mobile platforms. They also tested communication between XBee adaptors and worked with Shivam to create the serial protocol definition. Shivam and Mohak worked with the Kinect to visualize objects using the sample data provided by Kinect. They began using an obstacle detection algorithm to detect cylindrical objects within our performance requirements. I worked on setting up serial communication from an Android phone to a laptop over Bluetooth. I also worked with Shivam to interface three Sharp IR sensors with an Arduino Mega that publishes emergency-stop information to a ROS node.

4. Plans

The team has five goals to complete by Progress Review 4:

1. **Mobile Platform Subsystem**:
    a. Assemble the platform and test locomotion through an SBC (Mohak and Pranav)
    b. Mount and integrate the IR sensors and Arduino Nano with platform (Shivam and Dorothy)
2. **Android App/Bluetooth Communication Subsystem**
    a. Show progress on bidirectional serial communication via Bluetooth between App and laptop (Dorothy and Mohak)
3. **Communication Subsystem**:
    a. Make two SBCs communicate and share data using DigiMesh XBee adapters (Richa and Pranav)
4. **Vision Subsystem**:
    a. Show progress on detecting cylindrical obstacles using data from Kinect (Mohak and Shivam)

5. Code

a. ros_arduino_ir_5.ino

```
// Dorothy Kirlew
// Using rosserial_arduino_demos

#include<stdlib.h>
#include <ros.h>
#include <ros/time.h>
#include <std_msgs/String.h>

ros::NodeHandle  nh;

std_msgs::String ir_msg;
ros::Publisher ir_range("range_data", &ir_msg);
char irVAL[13]; // this HAS to be a big number here, regardless of the
actual size of the value
char result[15];

const int analog_pins[] = {0, 2, 4};
float readings[3];
char irName[2];
char final[20];
char temp[10];

const int danger_zone = 20;

boolean tooClose;

unsigned long range_timer;

// converts IR output to cm
float getRange(int pin_num)
```

```
{
    int sample;
    sample = analogRead(pin_num)/4;

    // if reading is too low, then we're super far away
    if(sample < 10)
    {
      // max range
      // TODO: not sure about 254 being the max range... should be 10-80 cm
      // actually doesn't matter.  as long as it's > danger_zone,
ultimately ignored
        return 254;
    }
    // Magic numbers to get cm
    sample= 1309/(sample-3);
    //return (sample - 1)/100; //convert to meters
    return sample;
}


void setup()
{
  // initialize node
  // publish data on ir_range
  nh.initNode();
  nh.advertise(ir_range);
}

void loop()
{
  // reset tooClose
  tooClose = false;
  result[0]='\0';

  // publish the range value every 50 milliseconds
  //   since it takes that long for the sensor to stabilize
  if ( (millis()-range_timer) > 50)
  {
    int i = 0;

    // loop through three IRs
    while(i < 3 && !tooClose)
    {
      readings[i]=getRange(analog_pins[i]);
      // if IR is reading value within danger_zone, publish data
      if(readings[i] <= danger_zone)
      {

        dtostrf(readings[i], 2, 3, temp);

        sprintf(final,"#%i-%s", i, temp);

        ir_msg.data = final;

        ir_range.publish(&ir_msg);
        // tooClose!!!
        tooClose = true;
      }
      i++;
    }
```

```
    range_timer =  millis();
  }
  nh.spinOnce();
}
```

b. activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
    android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity"
    android:background="#35c0b2">

    <Button
        android:layout_width="230dp"
        android:layout_height="100dp"
        android:textSize="50sp"
        android:background = "@drawable/unpressed_button"
        android:text="Park"
        android:id="@+id/parkButton"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="67dp"
        android:clickable="true"
        android:onClick="parkCar"
        android:enabled="true" />

    <Button
        android:layout_width="230dp"
        android:layout_height="100dp"
        android:textSize="50sp"
        android:background = "@drawable/pressed_button"
        android:text="Return"
        android:id="@+id/returnButton"
        android:enabled="true"
        android:layout_centerVertical="true"
        android:layout_alignStart="@+id/parkButton"
        android:clickable="true"
        android:onClick="returnCar"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="25sp"
        android:text="Status:"
        android:id="@+id/justStatusLabel"
        android:layout_marginTop="57dp"
        android:layout_below="@+id/returnButton"
        android:layout_alignParentStart="true" />

    <TextView
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:textSize="25sp"
        android:text="Returning"
        android:id="@+id/statusText"
        android:layout_marginStart="31dp"
        android:layout_alignTop="@+id/justStatusLabel"
        android:layout_toEndOf="@+id/justStatusLabel" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="25sp"
        android:text="ETA:"
        android:id="@+id/justETALabel"
        android:layout_below="@+id/justStatusLabel"
        android:layout_toStartOf="@+id/statusText"
        android:layout_marginTop="26dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="25sp"
        android:text="01:30"
        android:id="@+id/ETText"
        android:layout_alignTop="@+id/justETALabel"
        android:layout_alignStart="@+id/statusText" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bluetooth not connected"
        android:id="@+id/bluetoothStatusText"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="No message sent"
        android:id="@+id/messageTextView"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true" />

</RelativeLayout>
```

c. AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="teami.virtualvalet" >
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
```

```
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

d. MainActivity.java

```
package teami.virtualvalet;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import java.util.Set;

public class MainActivity extends AppCompatActivity
{
    String message = "";
    TextView bluetoothStatus;
    TextView messageToUser;

    BluetoothAdapter mBluetoothAdapter;
    BluetoothDevice mBluetoothDevice;
    private BluetoothChatService mChatService = null;

    Handler mHandler = new Handler();

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        messageToUser = (TextView) findViewById(R.id.messageTextView);
        bluetoothStatus = (TextView)
findViewById(R.id.bluetoothStatusText);

        // create chat service and adapter
        mChatService = new BluetoothChatService(this, mHandler);
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

        // update text on display to indicate if bluetooth is available
        if(mBluetoothAdapter == null)
        {
            bluetoothStatus.setText("Bluetooth not supported");
        }
        else
```

```
        {
            bluetoothStatus.setText("Bluetooth supported");
        }

        // enable bluetooth adapter
        if(!mBluetoothAdapter.isEnabled())
        {
            Intent enableBluetooth = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableBluetooth, 1);
        }

        // Find and save device that matches DEVICE_NAME, saved in
Constants.Java
        // This is currently the name of the laptop, will be the name of
the mobile platform
        Set <BluetoothDevice> pairedDevices =
mBluetoothAdapter.getBondedDevices();
        if(pairedDevices.size() > 0)
        {
            for(BluetoothDevice device : pairedDevices)
            {
                if (device.getName().equals(Constants.DEVICE_NAME))
                {
                    mBluetoothDevice = device;
                }
            }
        }
        // start chat service and connect to bluetooth
        mChatService.start();
        Toast.makeText(this, "start mChatService",
Toast.LENGTH_SHORT).show();
        mChatService.connect(mBluetoothDevice, false);
        Toast.makeText(this, "connect mChatService",
Toast.LENGTH_SHORT).show();
    }

    public void parkCar(View view)
    {
        TextView status = (TextView) findViewById(R.id.statusText);
        TextView ETA = (TextView) findViewById(R.id.ETText);
        TextView ETALabel = (TextView) findViewById(R.id.justETALabel);
        Button parkButton = (Button) findViewById(R.id.parkButton);
        Button returnButton = (Button) findViewById(R.id.returnButton);

        // update message to user, debugging only
        // todo: remove this once app is functional
        messageToUser.setText("Car is parking");

        // update status and ETA.  enable Return and disable Park buttons
        status.setText("Parking");
        ETA.setText("May 2016");
        ETALabel.setText("ETP");
        returnButton.setEnabled(true);
        parkButton.setEnabled(false);
        message = "Park";

        // send park message to chat service
        mChatService.write(message.getBytes());
    }
```

```
    public void returnCar(View view)
    {
        TextView status = (TextView) findViewById(R.id.statusText);
        TextView ETA = (TextView) findViewById(R.id.ETText);
        TextView ETALabel = (TextView) findViewById(R.id.justETALabel);
        Button parkButton = (Button) findViewById(R.id.parkButton);
        Button returnButton = (Button) findViewById(R.id.returnButton);

        // update message to user, debugging only
        // todo: remove this once app is functional
        messageToUser.setText("Car is returning");

        // update status and ETA.  enable Park and disable Return buttons
        status.setText("Returning");
        ETA.setText("One Month");
        ETALabel.setText("ETA");
        parkButton.setEnabled(true);
        returnButton.setEnabled(false);
        message = "Return";

        // send return message to chat service
        mChatService.write(message.getBytes());
    }
}
```

e. BluetoothChatService.java

```
package teami.virtualvalet;

/**
 * Created by dorothy.kirlew on 11/9/2015.
 */
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;


import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;



import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;

/**
 * This class does all the work for setting up and managing Bluetooth
 * connections with other devices. It has a thread that listens for
 * incoming connections, a thread for connecting with a device, and a
```

```
 * thread for performing data transmissions when connected.
 */
public class BluetoothChatService {
    // Debugging
    private static final String TAG = "BluetoothChatService";

    // Name for the SDP record when creating server socket
    private static final String NAME_SECURE = "BluetoothChatSecure";
    private static final String NAME_INSECURE = "BluetoothChatInsecure";


    // Member fields
    private final BluetoothAdapter mAdapter;
    private final Handler mHandler;
    private AcceptThread mSecureAcceptThread;
    private AcceptThread mInsecureAcceptThread;
    private ConnectThread mConnectThread;
    private ConnectedThread mConnectedThread;
    private int mState;

    // Constants that indicate the current connection state
    public static final int STATE_NONE = 0;       // we're doing nothing
    public static final int STATE_LISTEN = 1;     // now listening for
incoming connections
    public static final int STATE_CONNECTING = 2; // now initiating an
outgoing connection
    public static final int STATE_CONNECTED = 3;  // now connected to a
remote device

    /**
     * Constructor. Prepares a new BluetoothChat session.
     *
     * @param context The UI Activity Context
     * @param handler A Handler to send messages back to the UI Activity
     */
    public BluetoothChatService(Context context, Handler handler) {
        mAdapter = BluetoothAdapter.getDefaultAdapter();
        mState = STATE_NONE;
        mHandler = handler;
    }

    /**
     * Set the current state of the chat connection
     *
     * @param state An integer defining the current connection state
     */
    private synchronized void setState(int state) {
        Log.d(TAG, "setState() " + mState + " -> " + state);
        mState = state;

        // Give the new state to the Handler so the UI Activity can update
        mHandler.obtainMessage(Constants.MESSAGE_STATE_CHANGE, state, -
1).sendToTarget();

    }

    /**
     * Return the current connection state.
     */
    public synchronized int getState() {
```

```
        return mState;
    }

    /**
     * Start the chat service. Specifically start AcceptThread to begin a
     * session in listening (server) mode. Called by the Activity
onResume()
     */
    public synchronized void start() {
        Log.d(TAG, "start");

        // Cancel any thread attempting to make a connection
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }

        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }

        setState(STATE_LISTEN);

        // Start the thread to listen on a BluetoothServerSocket
        if (mSecureAcceptThread == null) {
            mSecureAcceptThread = new AcceptThread(true);
            mSecureAcceptThread.start();
        }
        if (mInsecureAcceptThread == null) {
            mInsecureAcceptThread = new AcceptThread(false);
            mInsecureAcceptThread.start();
        }
    }

    /**
     * Start the ConnectThread to initiate a connection to a remote device.
     *
     * @param device The BluetoothDevice to connect
     * @param secure Socket Security type - Secure (true) , Insecure
(false)
     */
    public synchronized void connect(BluetoothDevice device, boolean
secure) {
        Log.d(TAG, "connect to: " + device);

        // Cancel any thread attempting to make a connection
        if (mState == STATE_CONNECTING) {
            if (mConnectThread != null) {
                mConnectThread.cancel();
                mConnectThread = null;
            }
        }

        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }
```

```
        // Start the thread to connect with the given device
        mConnectThread = new ConnectThread(device, secure);
        mConnectThread.start();
        setState(STATE_CONNECTING);
    }

    /**
     * Start the ConnectedThread to begin managing a Bluetooth connection
     *
     * @param socket The BluetoothSocket on which the connection was made
     * @param device The BluetoothDevice that has been connected
     */
    public synchronized void connected(BluetoothSocket socket,
BluetoothDevice
            device, final String socketType) {
        Log.d(TAG, "connected, Socket Type:" + socketType);

        // Cancel the thread that completed the connection
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }

        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }

        // Cancel the accept thread because we only want to connect to one
device
        if (mSecureAcceptThread != null) {
            mSecureAcceptThread.cancel();
            mSecureAcceptThread = null;
        }
        if (mInsecureAcceptThread != null) {
            mInsecureAcceptThread.cancel();
            mInsecureAcceptThread = null;
        }

        // Start the thread to manage the connection and perform
transmissions
        mConnectedThread = new ConnectedThread(socket, socketType);
        mConnectedThread.start();

        // Send the name of the connected device back to the UI Activity
        Message msg =
mHandler.obtainMessage(Constants.MESSAGE_DEVICE_NAME);
        Bundle bundle = new Bundle();
        bundle.putString(Constants.DEVICE_NAME, device.getName());
        msg.setData(bundle);
        mHandler.sendMessage(msg);

        setState(STATE_CONNECTED);
    }

    /**
     * Stop all threads
     */
```

```
    public synchronized void stop() {
        Log.d(TAG, "stop");

        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }

        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }

        if (mSecureAcceptThread != null) {
            mSecureAcceptThread.cancel();
            mSecureAcceptThread = null;
        }

        if (mInsecureAcceptThread != null) {
            mInsecureAcceptThread.cancel();
            mInsecureAcceptThread = null;
        }
        setState(STATE_NONE);
    }

    /**
     * Write to the ConnectedThread in an unsynchronized manner
     *
     * @param out The bytes to write
     * @see ConnectedThread#write(byte[])
     */
    public void write(byte[] out) {
        // Create temporary object
        ConnectedThread r;
        // Synchronize a copy of the ConnectedThread
        synchronized (this) {
            if (mState != STATE_CONNECTED) return;
            r = mConnectedThread;
        }
        // Perform the write unsynchronized
        r.write(out);
    }

    /**
     * Indicate that the connection attempt failed and notify the UI
Activity.
     */
    private void connectionFailed() {
        // Send a failure message back to the Activity
        Message msg = mHandler.obtainMessage(Constants.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(Constants.TOAST, "Unable to connect device");
        msg.setData(bundle);
        mHandler.sendMessage(msg);

        // Start the service over to restart listening mode
        BluetoothChatService.this.start();
    }

    /**
```

```
     * Indicate that the connection was lost and notify the UI Activity.
     */
    private void connectionLost() {
        // Send a failure message back to the Activity
        Message msg = mHandler.obtainMessage(Constants.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(Constants.TOAST, "Device connection was lost");
        msg.setData(bundle);
        mHandler.sendMessage(msg);

        // Start the service over to restart listening mode
        BluetoothChatService.this.start();
    }

    /**
     * This thread runs while listening for incoming connections. It
behaves
     * like a server-side client. It runs until a connection is accepted
     * (or until cancelled).
     */
    private class AcceptThread extends Thread {
        // The local server socket
        private final BluetoothServerSocket mmServerSocket;
        private String mSocketType;

        public AcceptThread(boolean secure) {
            BluetoothServerSocket tmp = null;
            mSocketType = secure ? "Secure" : "Insecure";

            // Create a new listening server socket
            try {
                if (secure) {
                    tmp =
mAdapter.listenUsingRfcommWithServiceRecord(NAME_SECURE,
                            Constants.MY_UUID_SECURE);
                } else {
                    tmp =
mAdapter.listenUsingInsecureRfcommWithServiceRecord(
                            NAME_INSECURE, Constants.MY_UUID_INSECURE);
                }
            } catch (IOException e) {
                Log.e(TAG, "Socket Type: " + mSocketType + "listen()
failed", e);
            }
            mmServerSocket = tmp;
        }

        public void run() {
            Log.d(TAG, "Socket Type: " + mSocketType +
                    "BEGIN mAcceptThread" + this);
            setName("AcceptThread" + mSocketType);

            BluetoothSocket socket = null;

            // Listen to the server socket if we're not connected
            while (mState != STATE_CONNECTED) {
                try {
                    // This is a blocking call and will only return on a
                    // successful connection or an exception
                    socket = mmServerSocket.accept();
```

```
                } catch (IOException e) {
                    Log.e(TAG, "Socket Type: " + mSocketType + "accept()
failed", e);
                    break;
                }

                // If a connection was accepted
                if (socket != null) {
                    synchronized (BluetoothChatService.this) {
                        switch (mState) {
                            case STATE_LISTEN:
                            case STATE_CONNECTING:
                                // Situation normal. Start the connected
thread.
                                connected(socket, socket.getRemoteDevice(),
                                        mSocketType);
                                break;
                            case STATE_NONE:
                            case STATE_CONNECTED:
                                // Either not ready or already connected.
Terminate new socket.
                                try {
                                    socket.close();
                                } catch (IOException e) {
                                    Log.e(TAG, "Could not close unwanted
socket", e);
                                }
                                break;
                        }
                    }
                }
            }
            Log.i(TAG, "END mAcceptThread, socket Type: " + mSocketType);

        }

        public void cancel() {
            Log.d(TAG, "Socket Type" + mSocketType + "cancel " + this);
            try {
                mmServerSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "Socket Type" + mSocketType + "close() of server
failed", e);
            }
        }
    }


    /**
     * This thread runs while attempting to make an outgoing connection
     * with a device. It runs straight through; the connection either
     * succeeds or fails.
     */
    private class ConnectThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final BluetoothDevice mmDevice;
        private String mSocketType;

        public ConnectThread(BluetoothDevice device, boolean secure) {
            mmDevice = device;
```

```
            BluetoothSocket tmp = null;
            mSocketType = secure ? "Secure" : "Insecure";

            // Get a BluetoothSocket for a connection with the
            // given BluetoothDevice
            try {
                if (secure) {
                    tmp =
device.createRfcommSocketToServiceRecord(Constants.MY_UUID_SECURE);
                } else {
                    tmp =
device.createInsecureRfcommSocketToServiceRecord(Constants.MY_UUID_INSECURE
);
                }
            } catch (IOException e) {
                Log.e(TAG, "Socket Type: " + mSocketType + "create()
failed", e);
            }
            mmSocket = tmp;
        }

        public void run() {
            Log.i(TAG, "BEGIN mConnectThread SocketType:" + mSocketType);
            setName("ConnectThread" + mSocketType);

            // Always cancel discovery because it will slow down a
connection
            mAdapter.cancelDiscovery();

            // Make a connection to the BluetoothSocket
            try {
                // This is a blocking call and will only return on a
                // successful connection or an exception
                mmSocket.connect();
            } catch (IOException e) {
                // Close the socket
                try {
                    mmSocket.close();
                } catch (IOException e2) {
                    Log.e(TAG, "unable to close() " + mSocketType +
                            " socket during connection failure", e2);
                }
                connectionFailed();
                return;
            }

            // Reset the ConnectThread because we're done
            synchronized (BluetoothChatService.this) {
                mConnectThread = null;
            }

            // Start the connected thread
            connected(mmSocket, mmDevice, mSocketType);
        }

        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() of connect " + mSocketType + " socket
```

```
failed", e);
            }
        }
    }

    /**
     * This thread runs during a connection with a remote device.
     * It handles all incoming and outgoing transmissions.
     */
    private class ConnectedThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        public ConnectedThread(BluetoothSocket socket, String socketType) {
            Log.d(TAG, "create ConnectedThread: " + socketType);
            mmSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            // Get the BluetoothSocket input and output streams
            try {
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) {
                Log.e(TAG, "temp sockets not created", e);
            }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        public void run() {
            Log.i(TAG, "BEGIN mConnectedThread");
            byte[] buffer = new byte[1024];
            int bytes;

            // Keep listening to the InputStream while connected
            while (true) {
                try {
                    // Read from the InputStream
                    bytes = mmInStream.read(buffer);

                    // Send the obtained bytes to the UI Activity
                    mHandler.obtainMessage(Constants.MESSAGE_READ, bytes, -
1, buffer)
                            .sendToTarget();
                } catch (IOException e) {
                    Log.e(TAG, "disconnected", e);
                    connectionLost();
                    // Start the service over to restart listening mode
                    BluetoothChatService.this.start();
                    break;
                }
            }
        }

        /**
         * Write to the connected OutStream.
         *
```

```
         * @param buffer The bytes to write
         */
        public void write(byte[] buffer) {
            try {
                mmOutStream.write(buffer);

                // Share the sent message back to the UI Activity
                mHandler.obtainMessage(Constants.MESSAGE_WRITE, -1, -1,
buffer)
                        .sendToTarget();
            } catch (IOException e) {
                Log.e(TAG, "Exception during write", e);
            }
        }

        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() of connect socket failed", e);
            }
        }
    }
}
```

f. Constants.java

```
package teami.virtualvalet;

import java.util.UUID;

/**
 * Created by dorothy.kirlew on 11/9/2015.
 */
public interface Constants
{
    // Message types sent from the BluetoothChatService Handler
    public static final int MESSAGE_STATE_CHANGE = 1;
    public static final int MESSAGE_READ = 2;
    public static final int MESSAGE_WRITE = 3;
    public static final int MESSAGE_DEVICE_NAME = 4;
    public static final int MESSAGE_TOAST = 5;

    // Unique UUID for this application
    public static final UUID MY_UUID_SECURE = UUID.fromString("94f39d29-
7d6d-437d-973b-fba39e49d4ee");
    public static final UUID MY_UUID_INSECURE = UUID.fromString("94f39d29-
7d6d-437d-973b-fba39e49d4ee");


    // Key names received from the BluetoothChatService Handler
    public static final String DEVICE_NAME = "ubuntu-0";
    public static final String TOAST = "toast";
}
```

g. pressed_button.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:shape="rectangle">
    <corners android:radius="40dp"/>
    <gradient android:startColor="#FFFFFF" android:endColor="#FFFFFF"  />
</shape>
```

h. unpressed_button.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <corners android:radius="40dp"/>
    <gradient android:startColor="#FFFFFF" android:endColor="#FFFFFF"  />
</shape>
```