DESIGN DOCUMENT OF

ANDROID APPLICATION “SENSOR TAG”.

SUBMITTED BY,

ANSU MIRIAM VARGHESE

(Intern ECE department, IISc)

Date: 30-06-2014

CONTENTS

1. INTRODUCTION

1.1 Purpose

A document is made to explain the different features of an android application called sensor tag. This document is intended for proper understanding of the application and also to take forward the app development by other app developers. The document explains the architecture and flow of the android application.

1.2 Scope

The android application "Sensor Tag" is a modified form of the Sensor Tag application provided by Texas Instruments (TI). It facilitates the use of a sensor to record various human body parameters like temperature, heart rate, breathing rate, etc. In the current project, body temperature values have been recorded from different individuals. This application aims to facilitate continuous monitoring of temperature of new born babies, and display the temperature changes on mobile phone and also call for help during emergencies. Hospitals would supply monitoring devices to parents who themselves could monitor the child's condition. This application aims to make parents self-reliant and reduce neo natal child mortality rate, especially in rural areas

1.3 Overview

The document begins by explaining a few fundamentals of an Android Application so that a beginner would not feel overwhelmed with all the information provided in the document and would also be at ease when he tries to understand the background code of the application.
The document then moves on to explain what a sensor tag application is and the features provided by the app developed by Texas Instruments.
Further the improvisations made to the code are pointed out and the additional modules included in the application are explained.
For clear understanding of the application, necessary screenshots are provided.

1.4 Reference Material

A beginner can learn Android programming through the following references:

1. Java Programming for Android Developers For Dummies by Barry Burd
2. https://developer.android.com/training/basics/firstapp/index.html?hl=p
3. http://www.androidhive.info/2011/11/android-sqlite-database-tutorial/
4. http://achartengine.org/content/javadoc/
5. http://www.youtube.com/playlist?list=PL2603F3CABBF5EEB0
6. http://wptrafficanalyzer.in/blog/android-drawing-time-chart-with-timeseries-in-achartengine/
7. http://saigeethamn.blogspot.in/2009/08/android-developers-tutorial-for.html

2. Android Application Fundamentals

Android apps are written in the Java programming language using Eclipse IDE (Integrated Development Environment). The Android SDK (Software Development Kit) compiles the code along with any data and resource files into an APK (Android Package) which is an archive file with an .apk suffix. One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app.

APP COMPONENTS

There are four different types of app components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

ACTIVITIES

An activity represents a single screen with a user interface. Although the activities work together to form a cohesive use experience in an application, each one is independent of the others. An activity is always implemented as a subclass of Activity class.

SERVICES

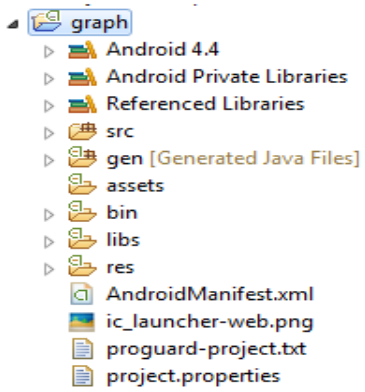
A service is a component that runs in the background to perform long –running operations or to perform work for remote processes. A service does not provide a user interface. For Example, as service might play music in the background while the user is in a different app. A service is implemented as a subclass of Service class.

CONTENT PROVIDER

A content provider manages a shared set of app data. Content providers are also useful for reading and writing data that is private to your app and not shared. For example the Note pad sample app uses a content provider to save notes. A content provider is implemented as a subclass of Content provider and must implement a standard set of APIs that enable other apps to perform transactions.

BROADCAST RECEIVERS

A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system- for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. A broadcast receiver is implemented as a subclass of "*BroadcastReceiver*" and each broadcast is delivered as an Intent object.
Every Android application has a fixed set of folders when the application is created.

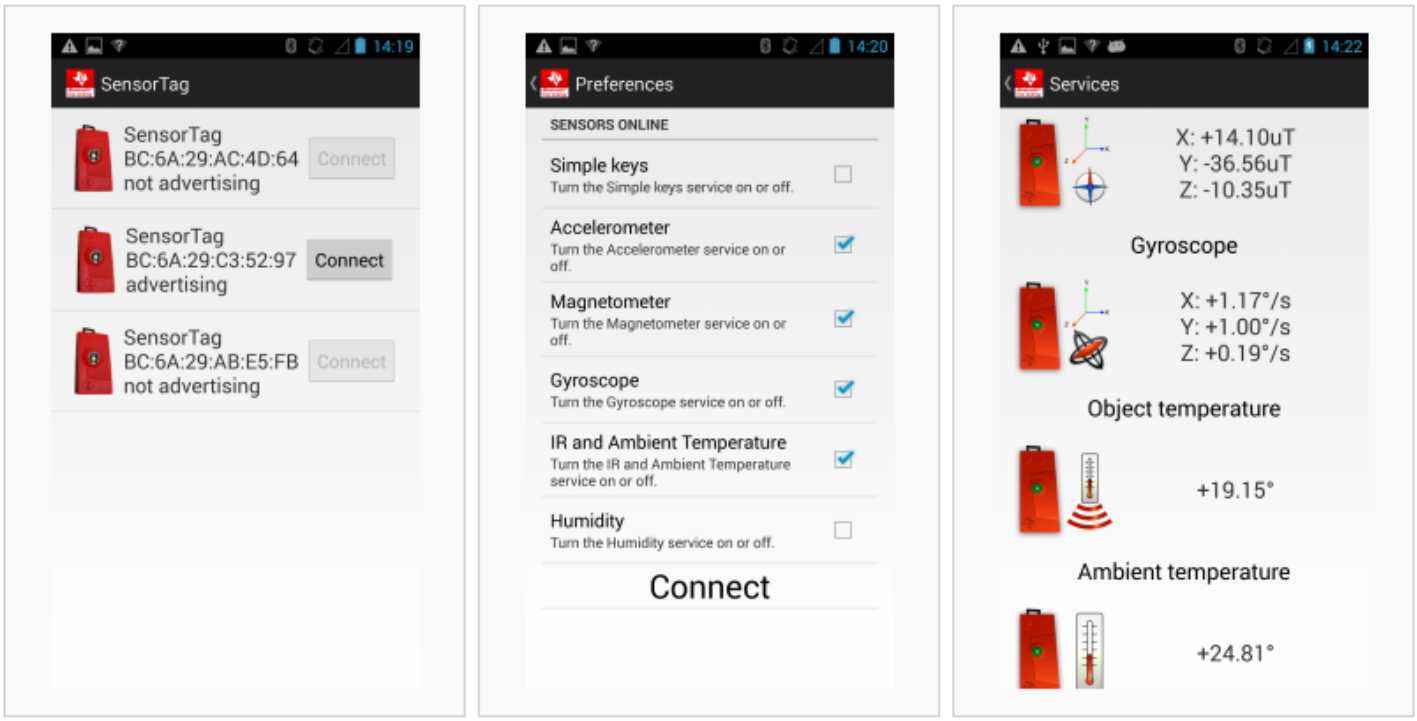


The "Src" folder contains the source code. It can contain several packages and several java files within each package.
The "Gen" folder contains automatically generated file called "R.java" which cannot be manipulated by the user. Even if the user manipulates it, the file would revert back to its original form a second later.
The "Libs" folder contains different libraries. Android 4.4 is a library that is loaded into the application by default. We can load new libraries into the application. In this project in order to support graphs into the application we load the charting library "Achartengine 1.1.0" into the "Libs" folder.
The "Res" folder contains resources. It contains the image we select for the application in several resolutions. It also contains the "Layouts" sub folder. When we create an activity for the application an xml file is created in the" Layout" folder. We can design the User Interface of the activity using the xml file of the activity. We can have several activities for an application.
The "AndroidManifest.xml" file does the following:

- It names the Java package for the application. The package name serves as a Unique identifier for the application.
- It describes the components of the application like the activities, services, broadcast receivers and content providers that the application is composed of. It names the classes that implement each of the components and publishes their capabilities.
- It declares which permissions the application must have in order to access protected parts of the API and interact with other applications.
- It declares the minimum level of the Android API that the application requires.
- It lists the libraries that the application must be linked against.

3. OVERVIEW OF SENSOR TAG APPLICATION

The Sensor Tag app can connect to devices and display live data of up to 4 sensors at a time.



1. Select device 2. Select sensors 3. View data

Stage 1.
BLE support is checked for and then scan is initiated for the advertising sensors. If a device hasn't sent an advertisement packet for the past second or so then it is considered to be not advertising. After choosing an advertising device stage 2 begins.

Stage 2.

At this stage connection is still not established, but due to the notification limit mentioned earlier, a subset of the sensors is to be chosen before connecting. Clicking connect will initiate a connection attempt, and if successful, stage 3 begins.

Stage 3.

At stage 3 there exists a connection, and if disconnected for any reason, the application returns to stage 2 which immediately starts service discovery.

4. ARCHITECTURE OF ORIGINAL SENSOR TAG APPLICATION

4.1 Architectural Design

- To run the application, a check is done to see whether the android phone supports Bluetooth Low Energy. (BLE).
- If BLE is supported, check is done to verify if Bluetooth is turned on.
- The application then waits for sensor advertisements.
- Connection is enabled for the advertising sensor. If there are multiple sensors advertising at the same time, then connection is enabled for all the sensors.
- The sensor values are displayed for all the sensors.

4.2 Improvised sensor tag application

The sensor Tag application developed by Texas Instruments can be improvised according to the needs of the user and to make the applications more user friendly, according to the ideas of the app developer.

Here the application has been added with two extra modules. The first one includes the creation of a local database in the phone and storing the sensor data within the local database.
The second module includes the retrieval of data from the database in order to plot a static graph based on the sensor values. Graphical view of the sensor values gives a clear picture of the rise and fall in values provided by the sensor tag.

5. DATA DESIGN

5.1 UNDERSTANDING THE DATABASE MODULE

A local database has been created using "*SQLite*" database of android.
This module has three different java files.

The first file is a model class called "*Temperature*" class.
The variables declared are

```
String sensorId;
String temperature;
String timestamp;
```

The file includes the constructors of the "*Temperature*" class and corresponding "*getter*" and "*setter*" functions of the variables.
The second java file is usually called the database handler since it handles all the database functions. The Database Handler class called here as "*DBHandler*" extends "*SQLiteOpenHelper*" Class and hence makes use of its properties.

A database is created with the name "*SensorDB*" which has a single table in it called "*Temperature_Table*".

The schema of the table can be given as

COLLUMN NAME	**TYPE**
SENSOR ID	TEXT
TEMPERATURE	TEXT
TIMESTAMP	TEXT

SQLite allows users to insert any type of data in the form of text.

The methods defined in "*DBHandler.java*" are

1. **insertTemperature()**

This method takes an object from the Temperature class which is the model class for the database. This class includes constructors for Temperature class and getter setter functions for the variables "*Sensor_Id, Temperature and Timestamp*".
In order to insert values the database is taken into the writeable mode by the pre-defined method "*getWritableDatabase()*".
The object of the class "*ContentValues*" and a method supported by it called "*put*()" is used to insert values into the table. The put method takes only string values as parameters.
System date and time is inserted as timestamp into the table after formatting it to the form "*dd-MM-yyyy hh:mm:ss*" with the help of "*SimpleDateFormat*" class.

2. **getAllTempReadings()**

In order to read the records from the table we use the method "*getAllTempReadings()*".
This method lists out all the records of the table in the form of a "*List*". The list has all the properties of the Temperature class. In order to read the records from the table the database is taken into a readable mode with the help of the pre-defined method "*getReadableDatabase()*".

We can write queries to retrieve data from the database using the functions "*query()*" and "*rawQuery()*".
The result of query or "*rawQuery*" is given to the "*Cursor*" object.
The "*Cursor*" object helps to loop through the records obtained as a result of the query.
"*moveToFirst()*" moves the "Cursor" to the first record, "*moveToLast()*" moves to the last record and "*moveToNext()*" moves the cursor to the next record. It also supports the methods "*moveToPrevious()*" and "*moveToPosition(Position)*".

3. **deleteTempReadings()**

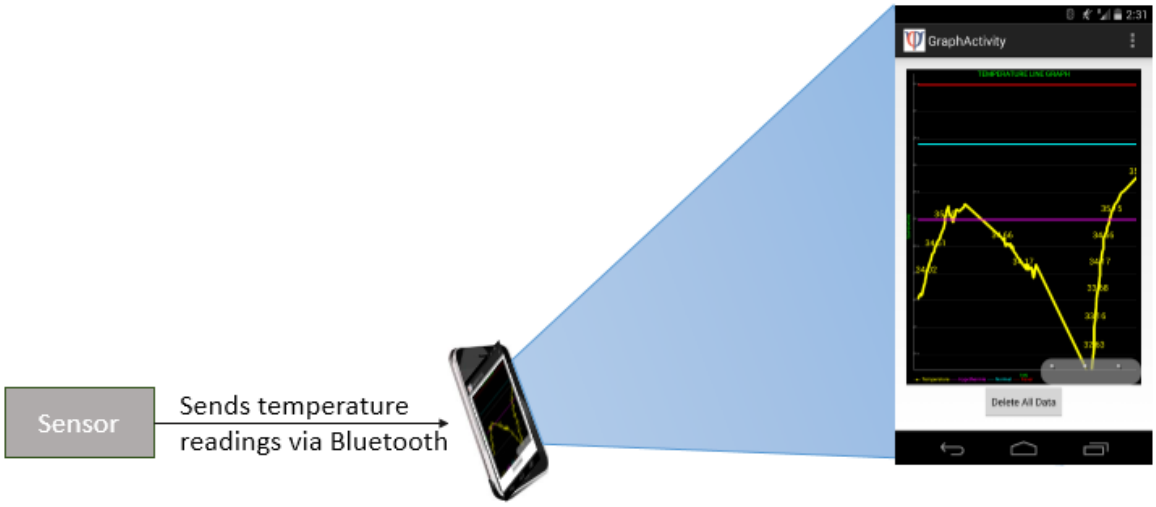
This method allows to delete all the records of the table. The query is written as a string and passed as a parameter to "*execSQL()*" which is another method to execute any query.

The third java file is called the activity file called here as "*Graph Activity*". Here both the Temperature class as well as the database handler files are imported and their functionalities are utilized. The "*Graph activity*" has both a java file and also an xml file. The "*xml*" file gives the layout of the activity. It includes a vertical Linear Layout along with a button at the bottom. The graph is visible within the linear layout, and the button at the bottom is used to delete the data within the database.

5.2 UNDERSTANDING THE GRAPH MODULE

In the graph Module, a graph is created which represents the sensor values .The values displayed on the screen get stored in the database. When the graph is displayed the data from the database are retrieved and displayed as a graph.

"*Achartengine*" is a charting library used for android applications. Version 1.1.0 of "*achartengine*" is used in this application. It is not available by default when an android application is created. It has to be downloaded separately and loaded into the library folder of the android application. It supports different types of graphs like line graph, bar graph, scatter graph etc. Here sensor values are displayed with the help of a line graph with timestamp as X axis and temperature values as Y axis.



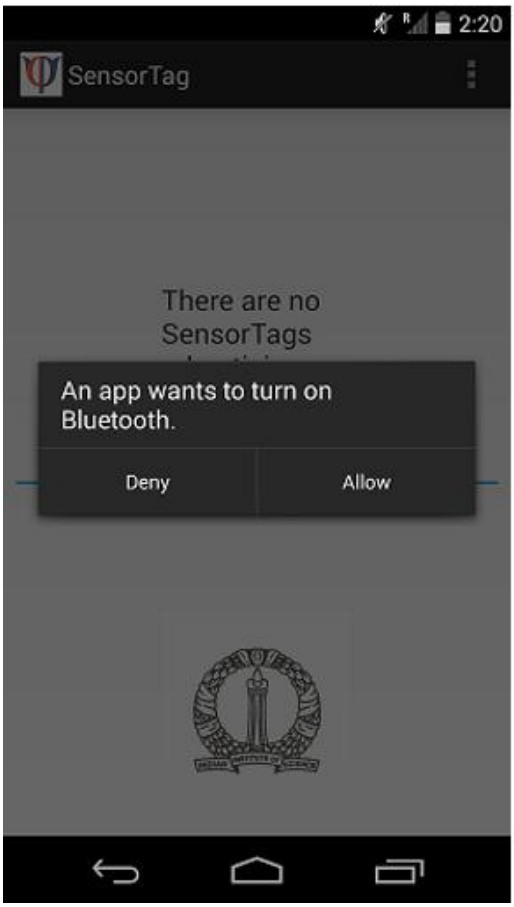
The graph activity's java file contains the java code in order to retrieve data from the database and display the graph. In this file the "*DBHandler*" class is imported and its properties are utilized. The sensor values are retrieved using the "*getAllTempReadings()*" explained earlier. The values obtained include the temperature values which is stored as "*TArray[ ]*" and timestamp values is stored as "*TSArray[ ]*". The "*array*" size is the same as the "*List*" size. "*Achartengine*" allows to plot graph when the X and Y axis values are provided as an "*array*". A line graph can be plotted with both values of the type double on the X and Y axis or values of type "*Date*" on X axis and values of type "*Double*" on Y axis.
Hence we convert "*timestamp*" values from "*String*" to "*Date*" and take the "*Date*" values on "*TSArray[ ]*" on X axis. The "*String*" values of "*temperature*" is converted to "*Double*" using "*Double.parseDouble()*". The "*double*" values are then stored as "*TArray[ ]*" and used on the Y axis.

A "*Series*" can be created using the class "*TimeSeries*". Every point on the line has to be added to the "*TimeSeries*" using the method "*add*".
Multiple series can be displayed on a single graph but each series has to be added together with a "*dataset*" with the help of "*XYMultipleSeriesDataset*" class.
Further, properties can be added to each series using the class "*XYSeriesRenderer*".
Every renderer, then has to be added together by "*XYMultipleSeriesRenderer*" class with the method "*addSeries()*".
The object of the class "*XYMultipleSeriesRenderer*" allows the user to set the properties of the whole graph.
"*getLineChartView()*" method allows to draw line graph with "*double*" on X and Y axis.
"*getTimeChartView()*" method allows to draw line graph with "*Date*" on X axis and "*double*" values on Y axis.

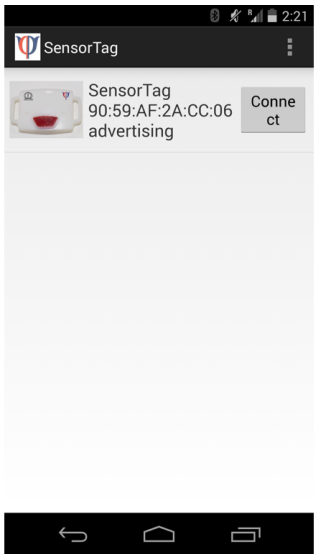
6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

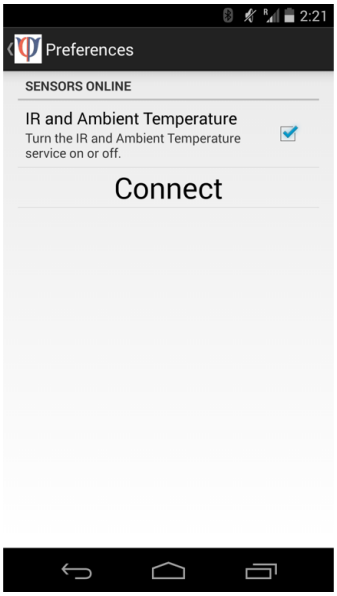
The User Interface (UI) of the Sensor Tag application includes five activity screens. The first screen requests the user to turn on the Bluetooth of the android phone and then waits for sensor advertisements.



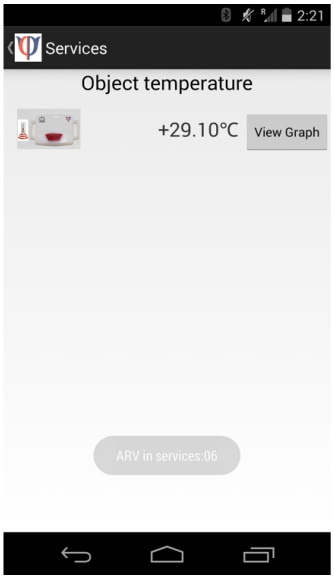
The second activity displays the advertising sensor with its unique ID and waits for the connection. Multiple sensors can advertise at the same time, but in this application only one sensor is used.



The third activity called "*Preferences*" displays the different services provided by the sensor. A sensor might provide multiple services. In this application only one service is provided.



The fourth activity called "*Services*" displays the sensor values read by each service. Multiple services can be displayed at the same time, but in this application only a single service is displayed.



The displayed sensor values are recorded into a SQLite database and also posted to a Google app engine at "*https://rnicu-cloud.appspot.com*".
The services activity's User Interface is provided with a button "*View Graph*" which when clicked takes the user to the fifth activity. The movement from one screen to another is done with the help of "*Intent*". The "*Intent*" is within a thread in order to speed up the process of moving from the services activity to graph activity, since the entire work is done by the main thread and hence causes delay. The above procedure is implemented by the following code.

```
Button btn = (Button)findViewById(R.id.btnGraph);
btn.setOnClickListener(new View.OnClickListener()
   {
     @Override
      public void onClick(View v) {
            new Thread(new Runnable(){
                  @Override
                   public void run() {
Intent tIntent=new Intent(getApplicationContext(),GraphActivity.class); /*moving from services activity to graph activity*/
startActivity(tIntent);
                                          }
                                          }).start();
                                                }
                                                });
```

By clicking the button "*View Graph*" the user is taken to the fifth activity screen called Graph Activity. Here sensor readings (temperature readings) are plotted against timestamp on the X axis using a line Graph. The bottom portion of the activity provides a button "*Delete All Data*" also allows the user to delete the values from the table in the database