IMPLEMENTING A FALL DETECTOR AND LOCATION PRESENTER ON A SMARTPHONE

by

M. Gökberk Ergün & M. Eray Koçak

Submitted to the Department of Computer Engineering

in partial fulfillment of the requirements

for the degree of

Bachelor of Science

in

Computer Engineering

Boğaziçi University

June 2010

TABLE OF CONTENTS

İçindekiler

ABSTRACT

Project Name : Implementing a Fall Detector and Location Presenter on a SMARTPHONE

Project Team : M.Eray Koçak, M. Gökberk Ergün

Term : Spring 2010

Keywords : Android, Accelerometer, Fall Detection, GPS

Summary : First is to implement a fall detection algorithm on Android Mobile OS by analyzing the accelerometer data. Second part is to inform relatives or caregivers to take action via SMS, email or Twitter about the location of the fallen. A user friendly application and settings interface is designed.

1- Introduction

This document is written for Cmpe 492 Senior Project course given by Computer Engineering Department at Boğaziçi University. The main goal of the project is detecting a fall and informing relatives of the fallen about the probable location of the fallen. Fall detection could be achieved by analyzing the accelerometer data. Location detection could be achieved by using GPS or network support. This is not only a senior project but also this solves a real life problem. For example there are many people suffering from epilepsy fear to go outside alone in case of any epileptic seizure. Using the application Fall Detector on an Android based smartphone, these people can go outside without fear.

2- BASICS

2.1 ANDROID

We decide to implement our Fall Detector and Location Presenter on Android Platform. Android is an open source and free mobile operating system. Android SDK has a Java Framework, powerful API support (Maps, Face Recognition, Gravity Sensors, Location Providers, and Accelerometers etc.) and SQLite database management system.

2.1.1 Activity

We use activity lifecycle for designing applications user interface and getting settings from user. This application has a very simple user interface. There are four buttons on the main screen. One of them starting the service which we start fall detection and location presentation services on the background. The second button is to stop these services. The third one forwards user to the settings screen. This view has a tab-view which has 3 tabs which are SMS tab, Email tab and Twitter tab. In SMS tab, the user could easily select the numbers of contacts which receive an emergency SMS message when a fall detected. In Email Tab, the user can set the email address as he/she does in SMS tab. And in the Twitter Tab user can add, delete or update the Twitter account he/she wants to update status in an emergency situation. The fourth button, which is at the bottom of the first view, is to send an emergency message to caregivers manually. This is an option in order to inform contacts when a fall can not be detected.

2.1.2 Service

Android services are background processes. They are designed to work in long run and while running any other application can be used on smartphone. Android supports multitasking with making good use of services. In FallService we detect falls and then wait for a while for

probable false alarms. After 20 seconds WarnerService starts working to send messages, send mails and update the status of Twitter.

Service is chosen because fall detection algorithm must be awake until the user stops it. Activities on Android is awake only when they are on the screen and when user wants to execute another Activity, our activity will be pause, stop or maybe destroy. Because of that we designed these long run jobs as Android services.

2.1.3 Content Provider

We use content provider for getting contacts information from the phone's contacts activity. This provides the user to synchronize her contacts with our application's emergency contacts. This information of contacts numbers, emails or twitter account which are selected by the user are stored in SQLite Database. Hence we manipulate these data for our application via the ContentProvider.

2.1.4 Sensor Manager

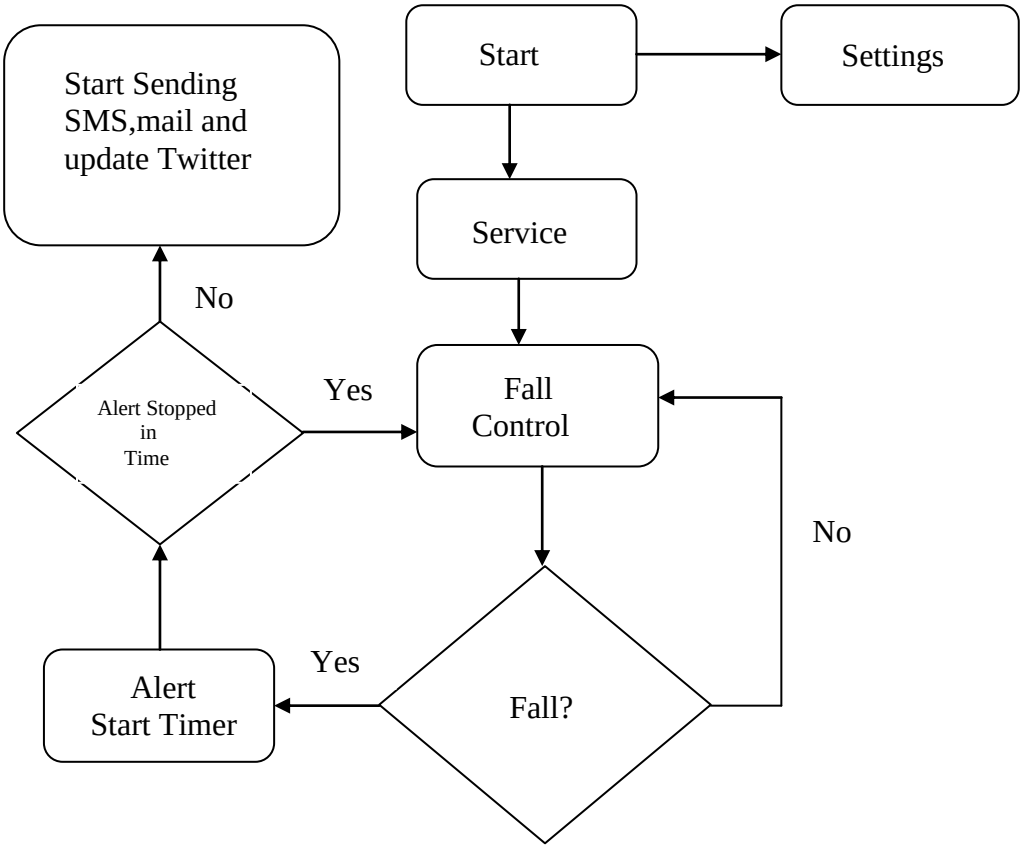
We use Android's SensorManager in order to get accelerometer values in X, Y and Z axis. We are getting an instance of this class by calling Context.getSystemService() with an argument of SENSOR_SERVICE. We implemented a SensorListener and fall detection is to be achieved in this method.

2.1.5 Location Manager

We use Android's LocationManager to get location values by using GPS or Network support. Using this manager, the system location services are accessed. These services allow us to obtain periodic updates of the device's geographical location. We retrieve this class with Context.getSystemService(Context.LOCATION_SERVICE) call. So the location is initialized in FallService and LocationListener is activated when this service starts running.

3- Work Flow

Figure 1 – Work Flow



After the application starts, service is started by activity and

4- Algorithm

Accelerometer provides acceleration values in direction of x-, y-, and z- axis.

First separate values of three axis are checked using a threshold value.

When the total amplitude of acceleration exceeds the threshold; we report that it is a fall.

$$A_T = \sqrt{A_x^2 + A_y^2 + A_z^2}$$

In the last version of our application, we made a wavelet transformation because amplitudes could be same for different physical activities. However, they could have different patterns on frequency domain. So when we apply the wavelet transformation, we could separate falls from the other physical activities those have nearly same amplitude. After the convolution we apply a new threshold value.

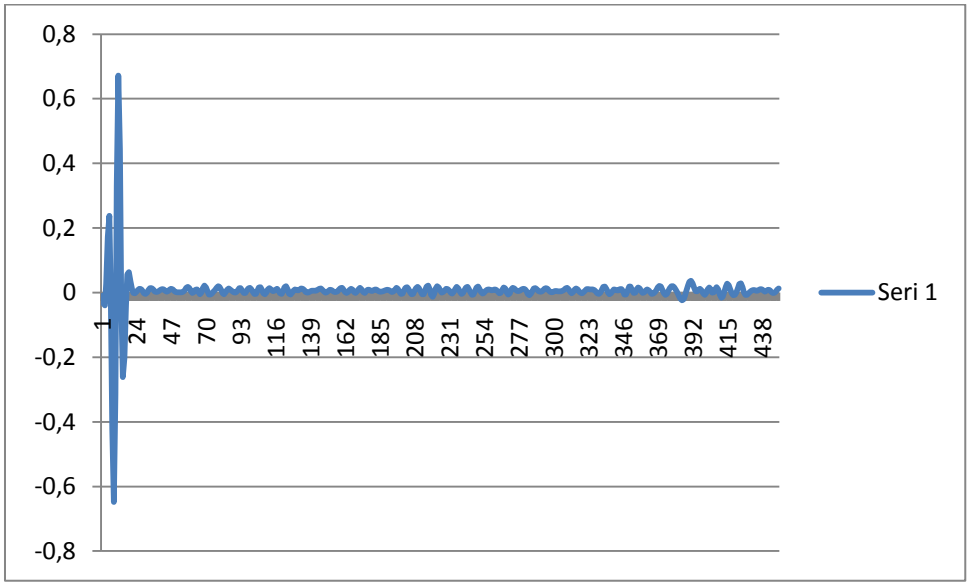
These are the some graphs, we generate by gathering data from our experiment with two wavelet function with different sample sizes. One of them has 40 sample point, the other has 21 sample point. But we consider that our experiment results are quite erroneous.

For wavelet sample:

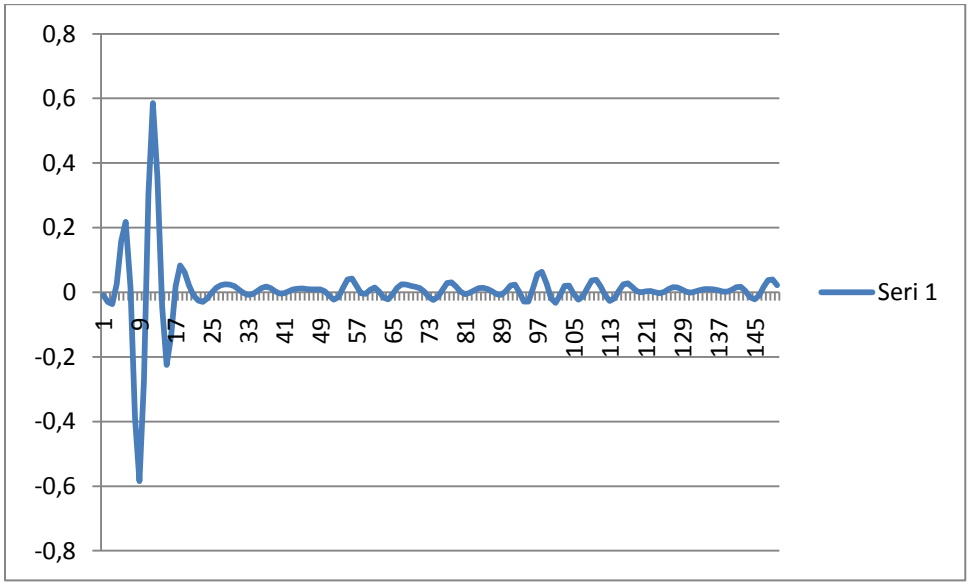
```
private double w4[] = {
      -0.000978365614096,     // 1
      -0.00227167643509,      // 2
      -0.000716059883604,     // 3
      0.0067604869749,  // 4
      0.0144825264796,  // 5
      0.00664334926724, // 6
      -0.0219418217014, // 7
      -0.0439709665794, // 8
      -0.0225756156944, // 9
      0.0330479986849,  // 10
      0.0636619772368,  // 11
      0.0330479986849,  // 12
      -0.0225756156944, // 13
      -0.0439709665794, // 14
      -0.0219418217014, // 15
      0.00664334926724, // 16
      0.0144825264796,  // 17
      0.0067604869749,  // 18
      -0.000716059883604,     // 19
```

```
-0.00227167643509,      // 20
-0.000978365614096      // 21
};
```

The graph of convolution amplitude to sample count :



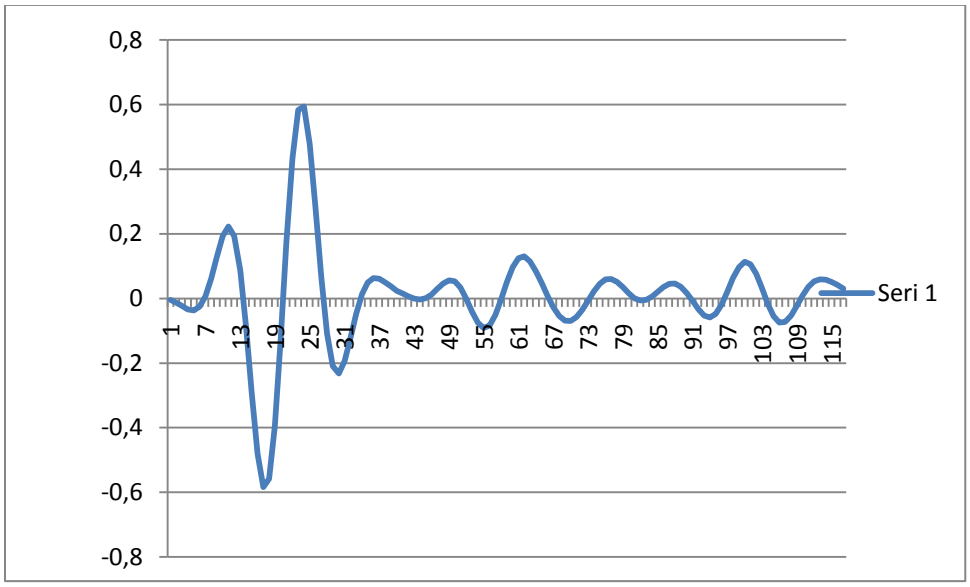
Another graph for w4:




```
private double w3[] = {
      -0.000489182807048,     // 1
      -0.000858652010111,     // 2
      -0.00113583821754,      // 3
      -0.00106498662267,      // 4
      -0.000358029941802,     // 5
      0.00116294815188, // 6
      0.00338024348745, // 7
      0.00573592771904, // 8
      0.00724126323981, // 9
      0.00672663269903, // 10
      0.00332167463362, // 11
```

```
-0.00298492586564,      // 12
-0.0109709108507, // 13
-0.0182613771714, // 14
-0.0219854832897, // 15
-0.0198603493183, // 16
-0.0112878078472, // 17
0.00205783949989, // 18
0.0165239993424,  // 19
0.0276563694057,  // 20
0.0318309886184,  // 21
0.0276563694057,  // 22
0.0165239993424,  // 23
0.00205783949989, // 24
-0.0112878078472, // 25
-0.0198603493183, // 26
-0.0219854832897, // 27
-0.0182613771714, // 28
-0.0109709108507, // 29
-0.00298492586564,      // 30
0.00332167463362, // 31
0.00672663269903, // 32
0.00724126323981, // 33
0.00573592771904, // 34
0.00338024348745, // 35
0.00116294815188, // 36
-0.000358029941802,     // 37
-0.00106498662267,      // 38
-0.00113583821754,      // 39
-0.000858652010111      // 40
};
```

And a graph for w3 with 40 sample point.



5- Service Implementations

5.1. Fall Detection

Fall detection is achieved in FallService class using algorithms explained above.

Code pieces that are responsible for this job can be seen below:

```
//*********************************************************************
            //The second version of fall detection algorithm is implemented
here.
            //The third version which is implemented in processSample()
method is used now with commenting out below.


      /*
            x2 =  values[0]/SensorManager.GRAVITY_EARTH;
            y2 =  values[1]/SensorManager.GRAVITY_EARTH;
            z2 =  values[2]/SensorManager.GRAVITY_EARTH;
            t = Math.sqrt(Math.abs(x2)*Math.abs(x2) +
Math.abs(y2)*Math.abs(y2) + Math.abs(z2)*Math.abs(z2));
            if( t > 2.5){

                  stopFallService();

                  showStatusbarNotification("Fall Detector !!!", "Are you
OK?", 1);
                  FallService.timer = new Timer();
              FallService.timer.schedule(new MyTimerTask(), 20*1000);

            }
            */


 //*********************************************************************


//*********************************************************************
        //The third version of the algorithm. Sample wavelet array and the
data buffer convolution is used with a threshold.

        ++sampleCtr;
            if( sampleCtr > 0xFFFFFFF0L )
                  sampleCtr = 0L;

        double ampl = Math.sqrt( (double)values[0]*values[0]+

                                          (double)values[1]*values[1]+

      (double)values[2]*values[2]);

            sampleBuffer[sampleBufferPtr++] = ampl;
            sampleBufferPtr = sampleBufferPtr % MAX_BUFFER_LEN;


            double wout = convolution( w );
```

```
            if( wout > 1.8){

                  stopFallService();

                  showStatusbarNotification("Fall Detector !!!", "Are you
OK?", 1);
                  FallService.timer = new Timer();
               FallService.timer.schedule(new MyTimerTask(), 20*1000);

            }


      //****************************************************************
****
```

5.2. Location Provider

Location is gathered using either GPS or NETWORK within the creation of the FallService. Code piece that are responsible for this job can be seen below:

```
//LocationManager is used to get the location.

            LocationManager locationManager;
            locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);


            //Our criterias about location retrieval. Accuracy is
important.

            Criteria criteria = new Criteria();
            criteria.setAccuracy(Criteria.ACCURACY_FINE);
            criteria.setAltitudeRequired(false);
            criteria.setBearingRequired(true);

            //Best provider (GPS or NETWORK) is used.
            String provider = locationManager.getBestProvider(criteria,
true);

            //Last provided location with the best provider is gathered.
            Location location =
locationManager.getLastKnownLocation(provider);


            MyLocation.setLatitude(location.getLatitude());
            MyLocation.setLongitude(location.getLongitude());

            //Location updates are requested in all possible conditions.
            locationManager.requestLocationUpdates(provider, 0, 0, new
MyLocationListener());
```

6. Classes

6.1. SettingsTab Class Reference

import android.app.TabActivity;

import android.content.Intent;

import android.content.res.Resources;

import android.os.Bundle;

import android.widget.TabHost;

6.1.1. Public Member Functions

void onCreate(Bundle savedInstanceState)

6.1.2. Detailed Description

SettingsTab class: Main Activity of Settings Part. It is used for creating tabs of settings screen and giving their names and determining their images. After that point, activity of each tab is assigned to corresponding tab with a distinct action. This enables same activity to make different operations when performing different actions. Also SettingsTab changes the default tab according to the action.

6.1.3. Member Function Documentation

void onCreate(Bundle savedInstanceState) ：is called when the activity is created. Tabs are created and their activities are assigned onto them in this part. Also layout of the activity is assigned in this function by calling **setContentView()** function.

References setContentView(int layout)

References SettingList

Referenced by CreateEvent

6.2. SettingsList Class Reference

import android.app.ListActivity;

import android.content.ContentResolver;

import android.content.Intent;

import android.database.Cursor;

import android.net.Uri;

import android.os.Bundle;

import android.provider.Contacts;

import android.provider.Contacts.People;

6.2.1. Public Member Functions

void onCreate(Bundle savedInstanceState)

6.2.2. Private Member Functions

void setDataList()

Uri getUri(String action)

String[] getProjection(String action)

6.2.3. Detailed Description

SettingsList class: All contacts information is listed in this class. ContentView comprises two nearly same layouts. Differences come from the current action of the activity. User can set the contact information as emergency contact information by simply clicking the button on the same line.

6.2.4. Member Function Documentation

void onCreate(Bundle savedInstanceState) : Called when the SettingsList Activity is created.

Referenced by SettingsList

References setDataList();

void setDataList(): is used for getting contact information from the Contacts Book of Smartphone by using contentProvider put this information to the listview of settingList by using ContactDataAdapter.

Referenced by onCreate

References getProjection(String action)

References getUri(String action)

String[] getProjection(String action): Returns columns of the table according to the current action.

Referenced by setDataList

Uri getUri(String action): Returns the Uri according to the current action.

Referenced by setDataList

6.3. CustomDataAdapter class reference

```
import android.content.ContentValues;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.View.OnClickListener;
import android.widget.BaseAdapter;
import android.widget.Button;
```

import android.widget.ImageView;

import android.widget.TextView;

import android.widget.Toast;

6.3.1 Public Member Functions

int getCount()

Object getItem(int)

long getItemId(int)

View getView(int, View, ViewGroup)

6.3.2 Private Member Functions

void makeToast(String)

Uri returnUri(String[])

int getType()

6.3.3. Detailed Description

CustomDataAdapter class: is responsible for getting contact information from Contacts Book and then control each information whether that is also emergency contact or not. And then set them to the ListView.

6.3.4. Member Function Documentation

int getCount() : Returns the number of rows of the Cursor which is full with the contacts information. Number of list items cannot exceed the number of rows of cursor.

Object getItem(int positon) : Returns the position of the wanted item.

long getItemId(int position) : Returns the position of the item.

View getView(int position, View ConvertView, ViewGroup parent) : is the function which makes the most of the work. For each item of list, it gets the row from the cursor in that

position and sets the features of View elements for that list item. And then it adds a button on that list item. If that list item's information is also in the emergency contact database, a "remove button" is added. Otherwise, an "add button" is added. And after an inserting or a deleting operation, a toast is shown about the success of the operation. And the SettingsTab Activity is called with the same action.

void makeToast(String msg) : puts the string on a Toast and the toast is shown.

Uri returnUri(String[]) : puts the type of the data according to the action into the values which will be inserted.

int getType() : Returns the static integer which indicates type of the data according to the action.

6.4. CustomDataProvider class reference

```
import java.util.HashMap;
import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;
import android.util.Log;
```

import com.falldetector.settings.ContactData.Datas;

6.4.1 Public Member Functions

void onCreate()

int delete(Uri uri, String where, String[] whereArgs)

Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)

String getType(Uri uri)

Uri insert(Uri uri, ContentValues initalValues)

int update(Uri uri, ContentValues values, String where, String[] whereArgs)

6.4.2. Detailed Description

ContactDataProvider class : extends the ContentProvider class. It is providing the data from the emergency contact database to all applications. Any application could reach the data by only using the **uri** of the Content Provider. Android OS search for ContentProviders which can be created for the first time, when creating the ContentProvider an instance of SQLiteOpenHelper is also created for creating the table of the emergency contact data. And then insert, update, delete and query operations is also defined.

6.4.3 Member Function Documentation

void onCreate() : is called when the ContactDataProvider is created. It instantiates a DatabaseHelper object which extends SQLiteDatabaseHelper class. DatabaseHelper creates the database table if it does not exist. It upgrades the table, if any change is occurred on database.

int delete(Uri uri, String where, String[] whereArgs) : when a delete operation is performed on table, delete function is called with parameters; uri of the ContentProvider, selection condition , and the values of the variables those are in selection condition.

Returns the number of rows effected from delete operation.

Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) : Returns the cursor which is filled with the columns of data resulted from the selection operation on the projection of columns in a sorted order according to **sortOrder.** In ContentProvider, a default sort order is also defined, when the **sortOrder** is null then the data is sorted according to default value. And there is no need to selection condition, if all data is wanted.

Uri insert(Uri uri, ContentValues initalValues) : gets the values will be inserted as a parameter but checks them not to cause an undesired condition. If there is a null column, that column is filled with a dummy data. And returns the uri of the inserted row (content uri + id).

int update(Uri uri, ContentValues values, String where, String[] whereArgs) :
gets the values will be updated as a parameter. And update condition and condition arguments are also parameters. This updates the rows/row which satisfy/satisfies the update condition with the values.

String getType(Uri uri) : ContactDataProvider extends the ContentProvider , so getType function must be overridden. But Indeed, it does not do anything.

6.5. CustomData class reference

import android.net.Uri;

import android.provider.BaseColumns;

6.5.1. Detailed Description

CustomData class : It has no member function. It is not initiated. It comprises a static class Datas which extends BaseColumns. **Datas** indicates the column names of the table and also indicates the uri and authority of the ContentProvider.

6.6. TwitterAdd class reference

import android.app.Activity;

import android.content.ContentValues;

import android.content.Intent;

import android.database.Cursor;

import android.net.Uri;

import android.os.Bundle;

import android.util.Log;

import android.view.Gravity;

import android.view.KeyEvent;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;

import android.widget.EditText;

import android.widget.ImageView;

import android.widget.TextView;

import android.widget.Toast;

6.6.1. Public Member Functions

void onCreate(Bundle savedInstanceState)

boolean onKeyDown(int keyCode,KeyEvent event)

6.6.2. Private Member Functions

Cursor getData()

void setTexts(Cursor cur)

ContentValues setValues()

void makeToast(String uri)

6.6.3. Detailed Descriptions

TwitterAdd class : is for getting the account information from the user and insert these information to emergency contact database or update the current account with these values.

6.6.4. Member Function Documentation

void onCreate(Bundle savedInstanceState) : like all the other **activity.onCreate** functions , it is called when the activity is created. It sets the content view of the activity and define the onclick event of the button.

References getData()

boolean onKeyDown(int keyCode,KeyEvent event) : it is an overridden function for overriding the functionality of the back button.

Cursor getData() : OnClick function of the save button differs according to whether there is a current account or not. getData shows whether there is a current account or not.

void setTexts(Cursor cur) : sets the account name and password of the current account to EditTexts.

ContentValues setValues() : set the values those will be inserted or updated.

6.7. TwitterData class reference

```
import android.app.Activity;
import android.content.Intent;
import android.database.Cursor;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;
```

6.7.1. Public Member Functions

void onCreate(Bundle savedInstanceState)

boolean onCreateOptionsMenu(Menu menu)

boolean onOptionsItemSelected(MenuItem item)

void getData()

6.7.2. Detailed Descriptions

TwitterData class : is for getting and showing the current account in database there is no button in the layout but an option menu is defined on the activity, User could select an option to perform from that menu.

6.6.4. Member Function Documentation

void onCreate(Bundle savedInstanceState) : like all the other **activity.onCreate** functions , it is called when the activity is created.

Refernces getData().

boolean onCreateOptionsMenu(Menu menu) : is called when the Option Menu is created. There are three different options but at most two of them could be stayed on the menu at the same time. These options are “Insert”, “Update”, and “Delete”. Last two of three are stayed on the menu, when there is an account in the database. Otherwise, There is only “Insert” option.

boolean onOptionsItemSelected(MenuItem item) : determines the operation performed for each option, when the any of them is selected.

void getData() : Get data returns the account information, if it is found in the database.

7. Account Information

The application uses a Gmail account in order to send emails.

```
//Gmail account info that is used for mailing.
static final String GMAIL_ID = "falldetector";
static final String GMAIL_PASSWORD = "wedetectfalls";
```

If the user has not specify a Twitter account info, account given below is used as default.

```
//Twitter account info if user did not specified hers.
static final String TWITTER_ID = "falldetector";
static final String TWITTER_PASSWORD = "wedetectfalls";
```