eman ta zabal zazu

BILBOKO INDUSTRIA INGENIARITZA TEKNIKOKO UNIBERTSITATE ESKOLA

INDUSTRIA ELEKTRONIKAREN ETA AUTOMATIKAREN INGENIARITZAKO GRADUA

GRADU AMAIERAKO LANA

2014 / 2015

MOBILAREKIN KOMUNIKATZEN DEN ESKUMUTURREKO ERLOJUAREN DISEINUA ETA PROTOTIPO BATEN GARAPENA

5.ERANSKINAK

IKASLEAREN DATUAK	**ZUZENDARIAREN DATUAK**
IZENA: ASIER	IZENA: OSKAR
ABIZENAK: GONZALEZ BUJEDO	ABIZENAK: CASQUERO OYARZABAL
	SAILA: AUTOMATIKA ETA SISTEMEN INGENIARITZA
SIN.:	SIN.:
DATA:	DATA:

AURKIBIDEA

```java
package com.example.smartwatch;

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;

/**
 * Created by Asier on 14/03/2015.
 */
public class ConexionBluetooth {

    private static final String TAG="Conexion Bluetooth";

    // Constantes que indican el estado de la conexion
    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

    // Tipos de Mensajes enviados de ConexionBluetooth a la 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;

    // Name for the SDP record when creating server socket
    private static final String NAME = "Conexion Bluetooth";

    //  UUID para esta aplicacion
    private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

    // Tipos de variables
    private final BluetoothAdapter mAdapter;
    private final Handler mHandler;
    private AcceptThread mAcceptThread;
```

```
48      private ConnectThread mConnectThread;
49      private ConnectedThread mConnectedThread;
50      private int mState;
51      private boolean mIsServiceStopped = false;
52
53
54
55      public ConexionBluetooth(Context context, Handler handler) {
56          mAdapter = BluetoothAdapter.getDefaultAdapter();
57          mState = STATE_NONE;
58          mHandler = handler;
59      }
60
61      private synchronized void PonerEstado(int state) {
62          Log.d(TAG, "PonerEstado() " + mState + " -> " + state);
63          mState = state;
64
65          // Darle el nuevo estado a la Handler para que la Actividad pueda actualizarlo
66          mHandler.obtainMessage(MESSAGE_STATE_CHANGE, state, -1).
    sendToTarget();
67      }
68
69
70      public synchronized int CogerEstado() {
71          return mState;
72      }
73
74
75      /**
76       * Start the chat service. Specifically start AcceptThread to begin a
77       * session in listening (server) mode.  */
78      public synchronized void start() {
79          Log.d(TAG, "Starting BluetoothManager...");
80
81          // Cancel any thread attempting to make a connection
82          if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread =
    null;}
83
84          // Cancel any thread currently running a connection
85          if (mConnectedThread != null) {mConnectedThread.cancel();
    mConnectedThread = null;}
86
87          // Start the thread to listen on a BluetoothServerSocket
88          if (mAcceptThread == null) {
89              mAcceptThread = new AcceptThread();
90              mAcceptThread.start();
91          }
92          PonerEstado(STATE_LISTEN);
93          mIsServiceStopped = false;
94      }
95
```

```
 96     /**
 97      * Start the ConnectThread to initiate a connection to a remote device.
 98      */
 99     public synchronized void connect(BluetoothDevice device) {
100         Log.d(TAG, "Connecting to: " + device);
101
102         if (mState == STATE_CONNECTED)
103             return;
104
105         // Cancel any thread attempting to make a connection
106         if (mState == STATE_CONNECTING) {
107             if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread =
    null;}
108         }
109
110         // Cancel any thread currently running a connection
111         if (mConnectedThread != null) {mConnectedThread.cancel();
    mConnectedThread = null;}
112
113         // Start the thread to connect with the given device
114         mConnectThread = new ConnectThread(device);
115         mConnectThread.start();
116         PonerEstado(STATE_CONNECTING);
117     }
118
119     /**
120      * Start the ConnectedThread to begin managing a Bluetooth connection
121      */
122     public synchronized void connected(BluetoothSocket socket, BluetoothDevice
    device) {
123         Log.d(TAG, "connected");
124
125         // Cancel the thread that completed the connection
126         if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread =
    null;}
127
128         // Cancel any thread currently running a connection
129         if (mConnectedThread != null) {mConnectedThread.cancel();
    mConnectedThread = null;}
130
131         // Cancel the accept thread because we only want to connect to one device
132         if (mAcceptThread != null) {mAcceptThread.cancel(); mAcceptThread = null;}
133
134         // Start the thread to manage the connection and perform transmissions
135         mConnectedThread = new ConnectedThread(socket);
136         mConnectedThread.start();
137
138         // Send the name of the connected device back to the UI Activity
139         Message msg = mHandler.obtainMessage(MESSAGE_DEVICE_NAME);
140         Bundle bundle = new Bundle();
141         bundle.putString(Constantes.
```

```
141  SERVICE_HANDLER_MSG_KEY_DEVICE_ADDRESS, device.getAddress());
142          bundle.putString(Constantes.
     SERVICE_HANDLER_MSG_KEY_DEVICE_NAME, device.getName());
143          msg.setData(bundle);
144          mHandler.sendMessage(msg);
145
146          PonerEstado(STATE_CONNECTED);
147      }
148
149      /**
150       * Stop all threads
151       */
152      public synchronized void stop() {
153          Log.d(TAG, "stop");
154          if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread =
     null;}
155          if (mConnectedThread != null) {mConnectedThread.cancel();
     mConnectedThread = null;}
156          if (mAcceptThread != null) {mAcceptThread.cancel(); mAcceptThread = null;}
157          PonerEstado(STATE_NONE);
158
159          mIsServiceStopped = true;
160
161      }
162
163
164      /**
165       * Write to the ConnectedThread in an unsynchronized manner
166       * @param out The bytes to write
167       * @see ConnectedThread#write(byte[])
168       */
169      public void write(byte[] out) {
170          // Create temporary object
171          ConnectedThread r;
172          // Synchronize a copy of the ConnectedThread
173          synchronized (this) {
174              if (mState != STATE_CONNECTED) return;
175              r = mConnectedThread;
176          }
177          // Perform the write unsynchronized
178          r.write(out);
179      }
180
181      /**
182       * Indicate that the connection attempt failed and notify the UI Activity.
183       */
184      private void FalloEnLaConexion() {
185          Log.d(TAG, "BluetoothManager :: connectionFailed()");
186          PonerEstado(STATE_LISTEN);
187
188          // Send a failure message back to the Activity
```

```
189         Message msg = mHandler.obtainMessage(MESSAGE_TOAST);
190         Bundle bundle = new Bundle();
191         bundle.putString(Constantes.SERVICE_HANDLER_MSG_KEY_TOAST, "
    Unable to connect device");
192         msg.setData(bundle);
193         mHandler.sendMessage(msg);
194
195
196     }
197
198     /**
199      * Indicate that the connection was lost and notify the UI Activity.
200      */
201     private void ConexionPerdida() {
202         Log.d(TAG, "BluetoothManager :: connectionLost()");
203         PonerEstado(STATE_LISTEN);
204
205         // Send a failure message back to the Activity
206         Message msg = mHandler.obtainMessage(MESSAGE_TOAST);
207         Bundle bundle = new Bundle();
208         bundle.putString(Constantes.SERVICE_HANDLER_MSG_KEY_TOAST, "
    Device connection was lost");
209         msg.setData(bundle);
210         mHandler.sendMessage(msg);
211
212
213     }
214
215
216
217     /**
218      * This thread runs while listening for incoming connections. It behaves
219      * like a server-side client. It runs until a connection is accepted
220      * (or until cancelled).
221      */
222     private class AcceptThread extends Thread {
223
224         // The local server socket
225         private final BluetoothServerSocket mmServerSocket;
226
227         public AcceptThread() {
228             BluetoothServerSocket tmp = null;
229
230             // Create a new listening server socket
231             try {
232                 tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME,
    MY_UUID);
233             } catch (IOException e) {
234                 Log.e(TAG, "listen() failed" + e.toString());
235             }
236             mmServerSocket = tmp;
```

```
237         }
238
239         public void run() {
240             Log.d(TAG, "BEGIN mAcceptThread" + this);
241             setName("AcceptThread");
242             BluetoothSocket socket = null;
243
244             // Listen to the server socket if we're not connected
245             while (mState != STATE_CONNECTED) {
246                 try {
247                     // This is a blocking call and will only return on a
248                     // successful connection or an exception
249                     if(mmServerSocket != null)
250                         socket = mmServerSocket.accept();
251                 } catch (IOException e) {
252                     Log.e(TAG, "accept() failed", e);
253                     break;
254                 }
255
256                 // If a connection was accepted
257                 if (socket != null) {
258                     synchronized (ConexionBluetooth.this) {
259                         switch (mState) {
260                             case STATE_LISTEN:
261                             case STATE_CONNECTING:
262                                 // Situation normal. Start the connected thread.
263                                 connected(socket, socket.getRemoteDevice());
264                                 break;
265                             case STATE_NONE:
266                             case STATE_CONNECTED:
267                                 // Either not ready or already connected. Terminate new socket.
268                                 try {
269                                     socket.close();
270                                 } catch (IOException e) {
271                                     Log.e(TAG, "Could not close unwanted socket", e);
272                                 }
273                                 break;
274                         }
275                     }
276                 }
277             }
278             Log.i(TAG, "END mAcceptThread");
279         }
280
281         public void cancel() {
282             Log.d(TAG, "cancel " + this);
283             try {
284                 if(mmServerSocket != null)
285                     mmServerSocket.close();
286             } catch (IOException e) {
287                 Log.e(TAG, "close() of server failed" + e.toString());
```

```
288             }
289         }
290     }  // End of class AcceptThread
291
292
293     /**
294      * This thread runs while attempting to make an outgoing connection
295      * with a device. It runs straight through; the connection either
296      * succeeds or fails.
297      */
298     private class ConnectThread extends Thread {
299         private final BluetoothSocket mmSocket;
300         private final BluetoothDevice mmDevice;
301
302         public ConnectThread(BluetoothDevice device) {
303             mmDevice = device;
304             BluetoothSocket tmp = null;
305
306             // Get a BluetoothSocket for a connection with the
307             // given BluetoothDevice
308             try {
309                 tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
310             } catch (IOException e) {
311                 Log.e(TAG, "create() failed", e);
312             }
313             mmSocket = tmp;
314         }
315
316         public void run() {
317             Log.i(TAG, "BEGIN mConnectThread");
318             setName("ConnectThread");
319
320             // Always cancel discovery because it will slow down a connection
321             mAdapter.cancelDiscovery();
322
323             // Make a connection to the BluetoothSocket
324             try {
325                 // This is a blocking call and will only return on a
326                 // successful connection or an exception
327                 mmSocket.connect();
328             } catch (IOException e) {
329                 FalloEnLaConexion();
330                 // Close the socket
331                 try {
332                     mmSocket.close();
333                 } catch (IOException e2) {
334                     Log.e(TAG, "unable to close() socket during connection failure", e2);
335                 }
336                 // Start the service over to restart listening mode
337                 ConexionBluetooth.this.start();
338                 return;
```

```
339             }
340
341             // Reset the ConnectThread because we're done
342             synchronized (ConexionBluetooth.this) {
343                 mConnectThread = null;
344             }
345
346             // Start the connected thread
347             connected(mmSocket, mmDevice);
348         }
349
350         public void cancel() {
351             try {
352                 mmSocket.close();
353             } catch (IOException e) {
354                 Log.e(TAG, "close() of connect socket failed", e);
355             }
356         }
357     }  // End of class ConnectThread
358
359     /**
360      * This thread runs during a connection with a remote device.
361      * It handles all incoming and outgoing transmissions.
362      */
363     private class ConnectedThread extends Thread {
364         private final BluetoothSocket mmSocket;
365         private final InputStream mmInStream;
366         private final OutputStream mmOutStream;
367
368         public ConnectedThread(BluetoothSocket socket) {
369             Log.d(TAG, "create ConnectedThread");
370             mmSocket = socket;
371             InputStream tmpIn = null;
372             OutputStream tmpOut = null;
373
374             // Get the BluetoothSocket input and output streams
375             try {
376                 tmpIn = socket.getInputStream();
377                 tmpOut = socket.getOutputStream();
378             } catch (IOException e) {
379                 Log.e(TAG, "temp sockets not created", e);
380             }
381
382             mmInStream = tmpIn;
383             mmOutStream = tmpOut;
384         }
385
386         public void run() {
387             Log.i(TAG, "BEGIN mConnectedThread");
388             byte[] buffer = new byte[1024];
389             int bytes;
```

```
390
391             // Keep listening to the InputStream while connected
392             while (true) {
393                 try {
394                     // Read from the InputStream
395                     bytes = mmInStream.read(buffer);
396
397                     // Send the obtained bytes to the main thread
398                     mHandler.obtainMessage(MESSAGE_READ, bytes, -1, buffer)
399                             .sendToTarget();
400                 } catch (IOException e) {
401                     Log.e(TAG, "disconnected", e);
402                     ConexionPerdida();
403                     break;
404                 }
405             }
406         }
407
408         /**
409          * Write to the connected OutStream.
410          * @param buffer  The bytes to write
411          */
412         public void write(byte[] buffer) {
413             try {
414                 mmOutStream.write(buffer);
415
416                 // Disabled: Share the sent message back to the main thread
417                 // mHandler.obtainMessage(Constants.MESSAGE_WRITE, -1, -1, buffer)
418                 //         .sendToTarget();
419             } catch (IOException e) {
420                 Log.e(TAG, "Exception during write");
421             }
422         }
423
424         public void cancel() {
425             try {
426                 mmSocket.close();
427             } catch (IOException e) {
428                 Log.e(TAG, "close() of connect socket failed");
429             }
430         }
431
432     }  // End of class ConnectedThread
433
434
435 }
436
```

```java
package com.example.smartwatch;

import android.content.Context;
import android.content.SharedPreferences;

/**
 * Created by Asier on 14/03/2015.
 */
public class InformacionDeConexion {



    // Instance
    private static InformacionDeConexion mInstance = null;

    private Context mContext;

    // Target device's MAC address
    private String mDeviceAddress = null;
    // Name of the connected device
    private String mDeviceName = null;


    private InformacionDeConexion (Context c) {
        mContext = c;

        SharedPreferences prefs = mContext.getSharedPreferences(Constantes.
PREFERENCE_NAME, Context.MODE_PRIVATE);
        mDeviceAddress = prefs.getString(Constantes.
PREFERENCE_CONN_INFO_ADDRESS, null);
        mDeviceName = prefs.getString(Constantes.
PREFERENCE_CONN_INFO_NAME, null);
    }

    public synchronized static InformacionDeConexion getInstance(Context c) {
        if(mInstance == null) {
            if(c != null)
                mInstance = new InformacionDeConexion(c);
            else
                return null;
        }
        return mInstance;
    }

    public void resetearInformacionDeConexion() {
        mDeviceAddress = null;
        mDeviceName = null;
    }

    public String CogerNombreDispositibo() {
        return mDeviceName;
```

```
49      }
50
51      public void PonerNombreDispositibo(String name) {
52          mDeviceName = name;
53
54          // At this time, connection is established successfully.
55          // Save connection info in shared preference.
56          SharedPreferences prefs = mContext.getSharedPreferences(Constantes.
    PREFERENCE_NAME, Context.MODE_PRIVATE);
57          SharedPreferences.Editor editor = prefs.edit();
58          editor.putString(Constantes.PREFERENCE_CONN_INFO_ADDRESS,
    mDeviceAddress);
59          editor.putString(Constantes.PREFERENCE_CONN_INFO_NAME,
    mDeviceName);
60          editor.apply();
61      }
62
63      public String CogerDireccionDelDispositibo() {
64          return mDeviceAddress;
65      }
66
67      public void PonerDireccionDispositibo(String address) {
68          mDeviceAddress = address;
69      }
70
71  }
72
```

```java
package com.example.smartwatch;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.util.Log;
import android.view.View;
import android.view.Window;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;

import java.util.Set;

/**
 * Created by Asier on 14/03/2015.
 */
public class ListaDeDispositibos extends Activity {


    // Debugging
    private static final String TAG = "ListaDeDispositibos";
    private static final boolean D = true;

    // Return Intent extra
    public static String EXTRA_DEVICE_ADDRESS = "device_address";

    // Member fields
    private BluetoothAdapter mBtAdapter;
    private ArrayAdapter<String> mPairedDevicesArrayAdapter;
    private ArrayAdapter<String> mNewDevicesArrayAdapter;

    // UI stuff
    Button BuscarDispositibos = null;




    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Setup the window
```

```
52
53          setContentView(R.layout.activity_lista_de_dispositibos);
54
55          // Set result CANCELED incase the user backs out
56          setResult(Activity.RESULT_CANCELED);
57
58          // Initialize the button to perform device discovery
59          BuscarDispositibos = (Button) findViewById(R.id.button_scan);
60          BuscarDispositibos.setOnClickListener(new View.OnClickListener() {
61              public void onClick(View v) {
62                  mNewDevicesArrayAdapter.clear();
63                  doDiscovery();
64                  v.setVisibility(View.GONE);
65              }
66          });
67
68          // Initialize array adapters. One for already paired devices and
69          // one for newly discovered devices
70          mPairedDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.
    adaptador_lista_de_dispositibos);
71          mNewDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.
    adaptador_lista_de_dispositibos);
72
73          // Find and set up the ListView for paired devices
74          ListView pairedListView = (ListView) findViewById(R.id.paired_devices);
75          pairedListView.setAdapter(mPairedDevicesArrayAdapter);
76          pairedListView.setOnItemClickListener(mDeviceClickListener);
77
78          // Find and set up the ListView for newly discovered devices
79          ListView newDevicesListView = (ListView) findViewById(R.id.new_devices);
80          newDevicesListView.setAdapter(mNewDevicesArrayAdapter);
81          newDevicesListView.setOnItemClickListener(mDeviceClickListener);
82
83          // Register for broadcasts when a device is discovered
84          IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
85          this.registerReceiver(mReceiver, filter);
86
87          // Register for broadcasts when discovery has finished
88          filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED
    );
89          this.registerReceiver(mReceiver, filter);
90
91          // Get the local Bluetooth adapter
92          mBtAdapter = BluetoothAdapter.getDefaultAdapter();
93
94          // Get a set of currently paired devices
95          Set<BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();
96
97          // If there are paired devices, add each one to the ArrayAdapter
98          if (pairedDevices.size() > 0) {
99              findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);
```

```
100             for (BluetoothDevice device : pairedDevices) {
101                 mPairedDevicesArrayAdapter.add(device.getName() + "\n" + device.
    getAddress());
102             }
103         } else {
104             String noDevices = getResources().getText(R.string.none_paired).toString();
105             mPairedDevicesArrayAdapter.add(noDevices);
106         }
107     }
108
109     @Override
110     protected void onDestroy() {
111         super.onDestroy();
112
113         // Make sure we're not doing discovery anymore
114         if (mBtAdapter != null) {
115             mBtAdapter.cancelDiscovery();
116         }
117
118         // Unregister broadcast listeners
119         this.unregisterReceiver(mReceiver);
120     }
121
122     /**
123      * Start device discover with the BluetoothAdapter
124      */
125     private void doDiscovery() {
126         if (D) Log.d(TAG, "doDiscovery()");
127
128         // Indicate scanning in the title
129         setProgressBarIndeterminateVisibility(true);
130         setTitle(R.string.scanning);
131
132         // Turn on sub-title for new devices
133         findViewById(R.id.title_new_devices).setVisibility(View.VISIBLE);
134
135         // If we're already discovering, stop it
136         if (mBtAdapter.isDiscovering()) {
137             mBtAdapter.cancelDiscovery();
138         }
139
140         // Request discover from BluetoothAdapter
141         mBtAdapter.startDiscovery();
142     }
143
144     // The on-click listener for all devices in the ListViews
145     private AdapterView.OnItemClickListener mDeviceClickListener = new
    AdapterView.OnItemClickListener() {
146         public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3) {
147             // Cancel discovery because it's costly and we're about to connect
148             mBtAdapter.cancelDiscovery();
```

```
149
150             // Get the device MAC address, which is the last 17 chars in the View
151             String info = ((TextView) v).getText().toString();
152             if(info != null && info.length() > 16) {
153                 String address = info.substring(info.length() - 17);
154                 Log.d(TAG, "User selected device : " + address);
155
156                 // Create the result Intent and include the MAC address
157                 Intent intent = new Intent();
158                 intent.putExtra(EXTRA_DEVICE_ADDRESS, address);
159
160                 // Set result and finish this Activity
161                 setResult(Activity.RESULT_OK, intent);
162                 finish();
163             }
164         }
165     };
166
167     // The BroadcastReceiver that listens for discovered devices and
168     // changes the title when discovery is finished
169     private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
170         @Override
171         public void onReceive(Context context, Intent intent) {
172             String action = intent.getAction();
173
174             // When discovery finds a device
175             if (BluetoothDevice.ACTION_FOUND.equals(action)) {
176                 // Get the BluetoothDevice object from the Intent
177                 BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.
    EXTRA_DEVICE);
178                 // If it's already paired, skip it, because it's been listed already
179                 if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
180                     mNewDevicesArrayAdapter.add(device.getName() + "\n" + device.
    getAddress());
181                 }
182                 // When discovery is finished, change the Activity title
183             } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(
    action)) {
184                 setProgressBarIndeterminateVisibility(false);
185                 setTitle(R.string.select_device);
186                 if (mNewDevicesArrayAdapter.getCount() == 0) {
187                     String noDevices = getResources().getText(R.string.none_found).
    toString();
188                     mNewDevicesArrayAdapter.add(noDevices);
189                 }
190                 BuscarDispositibos.setVisibility(View.VISIBLE);
191             }
192         }
193     };
194
195 }
```

196
197
198

```
1  package com.example.smartwatch;
2
3  import android.app.Activity;
4  import android.app.AlertDialog;
5  import android.bluetooth.BluetoothAdapter;
6  import android.content.BroadcastReceiver;
7  import android.content.ComponentName;
8  import android.content.Context;
9  import android.content.DialogInterface;
10 import android.content.Intent;
11 import android.content.IntentFilter;
12 import android.content.ServiceConnection;
13 import android.graphics.Color;
14 import android.os.BatteryManager;
15 import android.os.Handler;
16 import android.os.IBinder;
17 import android.os.Message;
18 import android.support.v7.app.ActionBarActivity;
19 import android.os.Bundle;
20 import android.text.Html;
21 import android.view.Menu;
22 import android.view.MenuItem;
23 import android.widget.ImageView;
24 import android.widget.TableLayout;
25 import android.widget.TableRow;
26 import android.widget.TextView;
27 import android.util.Log;
28 import android.widget.Toast;
29
30
31
32
33 public class SmartWatchActivity extends ActionBarActivity {
34
35     // Debugging
36     private static final String TAG = "SmartWatchActivity";
37
38
39
40     // Context, System
41     private Context mContext;
42     private SmartWatchService mService;
43
44     private ActivityHandler mActivityHandler;
45
46     // Notification broadcast receiver
47     private NotificationInfo MiRecibidor;
48
49
50
51     // Global
```

```
52      private boolean mStopService = false;
53
54      private ImageView mImageBT = null;
55      private TextView mTextStatus = null;
56      TextView txtInfoCargaBateria;
57      TextView EscBat;
58      TextView TipodeCarga;
59      TextView Cargando;
60      TableLayout tab;
61
62
63
64
65      @Override
66      protected void onCreate(Bundle savedInstanceState) {
67          super.onCreate(savedInstanceState);
68
69          //----- System, Context
70          mContext = this;//.getApplicationContext();
71          mActivityHandler = new ActivityHandler();
72
73
74          setContentView(R.layout.activity_smart_watch);
75          tab=(TableLayout)findViewById(R.id.tab);
76
77
78          MiRecibidor = new NotificationInfo();
79          IntentFilter filter = new IntentFilter();
80          filter.addAction("Notificaciones");
81          registerReceiver(MiRecibidor,filter);
82
83
84          // Setup views
85          mImageBT = (ImageView) findViewById(R.id.status_title);
86          mImageBT.setImageDrawable(getResources().getDrawable(android.R.drawable.
    presence_invisible));
87          mTextStatus = (TextView) findViewById(R.id.status_text);
88          mTextStatus.setText(getResources().getString(R.string.bt_state_init));
89
90
91
92
93
94          // Do data initialization after service started and binded
95          doStartService();
96      }
97
98      @Override
99      public synchronized void onStart() {
100         super.onStart();
101
```

```
102
103     }
104
105     @Override
106     public synchronized void onPause() {
107         super.onPause();
108     }
109
110     @Override
111     public void onStop() {
112
113         super.onStop();
114     }
115
116     @Override
117     public void onDestroy() {
118         super.onDestroy();
119         finalizeActivity();
120     }
121
122     @Override
123     public void onLowMemory (){
124         super.onLowMemory();
125         // onDestroy is not always called when applications are finished by Android
    system.
126         finalizeActivity();
127     }
128     @Override
129     public boolean onCreateOptionsMenu(Menu menu) {
130         // Inflate the menu; this adds items to the action bar if it is present.
131         getMenuInflater().inflate(R.menu.menu_smart_watch, menu);
132         return true;
133     }
134
135     @Override
136     public boolean onOptionsItemSelected(MenuItem item) {
137         // Handle action bar item clicks here. The action bar will
138         // automatically handle clicks on the Home/Up button, so long
139         // as you specify a parent activity in AndroidManifest.xml.
140         switch (item.getItemId()) {
141             case (R.id.accion_escanear):
142                 doScan();
143                 return true;
144             //noinspection SimplifiableIfStatement
145             case (R.id.battery_info):
146                mService.MandarBateria();
147                 return true;
148             case (R.id.hora_fecha):
149                 mService.MandarHorayFecha();
150
151                 return true;
```

```
152
153         /* case (R.id.action_send_notifications):
154            mService.MandarNotificaciones();
155            return true;
156          */
157
158         case R.id.quit:
159            AlertDialog.Builder dialog = new AlertDialog.Builder(this);
160
161            dialog.setMessage("¿Quieres salir de la aplicacion?");
162            dialog.setCancelable(false);
163            dialog.setPositiveButton("Si", new DialogInterface.OnClickListener() {
164
165               @Override
166               public void onClick(DialogInterface dialog, int which) {
167                  SmartWatchActivity.this.finish();
168                  mService.finalizeService();
169               }
170            });
171            dialog.setNegativeButton("No", new DialogInterface.OnClickListener() {
172
173               @Override
174               public void onClick(DialogInterface dialog, int which) {
175                  dialog.cancel();
176               }
177            });
178            dialog.show();
179
180            return true;
181
182
183      }
184      return super.onOptionsItemSelected(item);
185   }
186
187
188
189 /*********************************************************
190  *
191  * Private classes
192  *
193  *********************************************************/
194
195   /**
196    * Service connection
197    */
198   private ServiceConnection mServiceConn = new ServiceConnection() {
199
200      public void onServiceConnected(ComponentName className, IBinder binder)
   {
201         Log.d(TAG, "Activity - Service connected");
```

```
202
203             mService = ((SmartWatchService.SmartWatchServiceBinder) binder).
    getService();
204
205             // Activity couldn't work with mService until connections are made
206             // So initialize parameters and settings here, not while running onCreate()
207             initialize();
208         }
209
210         public void onServiceDisconnected(ComponentName className) {
211             mService = null;
212         }
213     };
214
215     private void doStartService() {
216         Log.d(TAG, "# Activity - doStartService()");
217         startService(new Intent(this, SmartWatchService.class));
218         bindService(new Intent(this, SmartWatchService.class), mServiceConn,
    Context.BIND_AUTO_CREATE);
219     }
220
221     private void doStopService() {
222         Log.d(TAG, "# Activity - doStopService()");
223         mService.finalizeService();
224         stopService(new Intent(this, SmartWatchService.class));
225     }
226
227     /**
228      * Initialization / Finalization
229      */
230     private void initialize() {
231         Log.d(TAG, "# Activity - initialize()");
232         mService.setupService(mActivityHandler);
233
234         // If BT is not on, request that it be enabled.
235         // RetroWatchService.setupBT() will then be called during onActivityResult
236         if(!mService.isBluetoothEnabled()) {
237             Intent enableIntent = new Intent(BluetoothAdapter.
    ACTION_REQUEST_ENABLE);
238             startActivityForResult(enableIntent, Constantes.REQUEST_ENABLE_BT);
239         }
240
241
242
243
244     }
245
246     private void finalizeActivity() {
247         Log.d(TAG, "# Activity - finalizeActivity()");
248
249         if(mStopService)
```

```
250             doStopService();
251
252         unbindService(mServiceConn);
253
254
255         System.gc();
256     }
257
258     /**
259      * Launch the DeviceListActivity to see devices and do scan
260      */
261     private void doScan() {
262         Intent intent = new Intent(this, ListaDeDispositibos.class);
263         startActivityForResult(intent, Constantes.REQUEST_CONNECT_DEVICE);
264     }
265
266     /**
267      * Launch notification settings screen
268      */
269     private void setNotificationAccess() {
270         Intent intent=new Intent("android.settings.
    ACTION_NOTIFICATION_LISTENER_SETTINGS");
271         startActivity(intent);
272     }
273
274     /**
275      * Ensure this device is discoverable by others
276      */
277     private void ensureDiscoverable() {
278         if (mService.getBluetoothScanMode() != BluetoothAdapter.
    SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
279             Intent intent = new Intent(BluetoothAdapter.
    ACTION_REQUEST_DISCOVERABLE);
280             intent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION,
    300);
281             startActivity(intent);
282         }
283     }
284
285 /*****************************************************
286  *
287  * Public classes
288  *
289  *****************************************************/
290
291     /**
292      * Receives result from external activity
293      */
294     public void onActivityResult(int requestCode, int resultCode, Intent data) {
295         Log.d(TAG, "onActivityResult " + resultCode);
296
```

```
297         switch(requestCode) {
298             case Constantes.REQUEST_CONNECT_DEVICE:
299                 // When DeviceListActivity returns with a device to connect
300                 if (resultCode == Activity.RESULT_OK) {
301                     // Get the device MAC address
302                     String address = data.getExtras().getString(ListaDeDispositibos.
    EXTRA_DEVICE_ADDRESS);
303                     // Attempt to connect to the device
304                     if(address != null && mService != null)
305                         mService.connectDevice(address);
306                 }
307                 break;
308 
309             case Constantes.REQUEST_ENABLE_BT:
310                 // When the request to enable Bluetooth returns
311                 if (resultCode == Activity.RESULT_OK) {
312                     // Bluetooth is now enabled, so set up a BT session
313                     mService.setupBT();
314                 } else {
315                     // User did not enable Bluetooth or an error occured
316                     Log.e(TAG, "BT is not enabled");
317                     Toast.makeText(this, R.string.bt_not_enabled_leaving, Toast.
    LENGTH_SHORT).show();
318                 }
319                 break;
320         }  // End of switch(requestCode)
321     }
322 
323 
324 
325 
326 
327     /**************************************************
328      *
329      * Handler, Callback, Sub-classes
330      *
331      **************************************************/
332 
333     public class ActivityHandler extends Handler {
334         @Override
335         public void handleMessage(Message msg)
336         {
337             switch(msg.what) {
338                 // BT state message
339                 case Constantes.MESSAGE_BT_STATE_INITIALIZED:
340                     mTextStatus.setText(getResources().getString(R.string.bt_title) + ": " +
341                             getResources().getString(R.string.bt_state_init));
342                     mImageBT.setImageDrawable(getResources().getDrawable(android.R.
    drawable.presence_invisible));
343                     break;
344                 case Constantes.MESSAGE_BT_STATE_LISTENING:
```

```
345                 mTextStatus.setText(getResources().getString(R.string.bt_title) + ": " +
346                         getResources().getString(R.string.bt_state_wait));
347                 mImageBT.setImageDrawable(getResources().getDrawable(android.R.
    drawable.presence_invisible));
348                 break;
349             case Constantes.MESSAGE_BT_STATE_CONNECTING:
350                 mTextStatus.setText(getResources().getString(R.string.bt_title) + ": " +
351                         getResources().getString(R.string.bt_state_connect));
352                 mImageBT.setImageDrawable(getResources().getDrawable(android.R.
    drawable.presence_away));
353                 break;
354             case Constantes.MESSAGE_BT_STATE_CONNECTED:
355                 if(mService != null) {
356                     String deviceName = mService.getDeviceName();
357                     if(deviceName != null) {
358                         mTextStatus.setText(getResources().getString(R.string.bt_title) +
    ": " +
359                                 getResources().getString(R.string.bt_state_connected) + " " +
    deviceName);
360                         mImageBT.setImageDrawable(getResources().getDrawable(android
    .R.drawable.presence_online));
361                     }
362                 }
363                 break;
364             case Constantes.MESSAGE_BT_STATE_ERROR:
365                 mTextStatus.setText(getResources().getString(R.string.bt_state_error));
366                 mImageBT.setImageDrawable(getResources().getDrawable(android.R.
    drawable.presence_busy));
367                 break;
368
369             // BT Command status
370             case Constantes.MESSAGE_CMD_ERROR_NOT_CONNECTED:
371                 mTextStatus.setText(getResources().getString(R.string.
    bt_cmd_sending_error));
372                 mImageBT.setImageDrawable(getResources().getDrawable(android.R.
    drawable.presence_busy));
373                 break;
374
375             ////////////////////////////////////////
376             // Contents changed
377             ////////////////////////////////////////
378
379             case Constantes.MESSAGE_ADD_NOTIFICATION:
380             {
381
382
383                 break;
384             }
385
386             case Constantes.MESSAGE_DELETE_NOTIFICATION:
387             {
```

```
388
389                 break;
390             }
391             default:
392                 break;
393         }
394
395         super.handleMessage(msg);
396     }
397 } // End of class ActivityHandler
398
399 class NotificationInfo extends BroadcastReceiver{
400
401     @Override
402     public void onReceive(Context context, Intent intent) {
403
404         int id = intent.getIntExtra(NotificationService.NOTIFICATION_KEY_ID, -1
    );
405         String packageName = intent.getStringExtra("paquete");
406         String textTicker = intent.getStringExtra("notificacion");
407         String title = intent.getStringExtra("titulo");
408         String text = intent.getStringExtra("texto");
409         //String bigtext=intent.getStringExtra("textogrande");
410         //String subtext=intent.getStringExtra("subtexto");
411         //String sumtext=intent.getStringExtra("resumentexto");
412         //String textlines=intent.getStringExtra("lineastexto");
413
414         switch (packageName){
415             case (Constantes.WHATSAPP_PACKAGE):
416                 mService.MandarNotificacionWhatsapp();
417                 break;
418             case (Constantes.GMAIL_PACKAGE):
419                 mService.MandarNotificacionGmail();
420
421                 break;
422             case (Constantes.FACEBOOK_PACKAGE):
423                 mService.MandarNotificacionFacebook();
424                 break;
425             case (Constantes.SMS_PACKAGE):
426                 mService.MandarNotificacionSMS();
427                 break;
428         }
429
430
431         TableRow tr = new TableRow(getApplicationContext());
432         tr.setLayoutParams(new TableRow.LayoutParams( TableRow.LayoutParams.
    MATCH_PARENT, TableRow.LayoutParams.WRAP_CONTENT));
433         TextView textview = new TextView(getApplicationContext());
434         textview.setLayoutParams(new TableRow.LayoutParams(TableRow.
    LayoutParams.WRAP_CONTENT, TableRow.LayoutParams.WRAP_CONTENT,1.
    0f));
```

```
435             textview.setTextSize(20);
436             textview.setTextColor(Color.parseColor("#0B0719"));
437             textview.setText(Html.fromHtml(packageName + "<br><b>" + textTicker +
        "<br><b>" + title +":" +" <b>" + text));
438             tr.addView(textview);
439             tab.addView(tr);
440
441         }  // End of onReceive()
442
443     }  // End of class NotificationReceive
444
445 }
446
```

```
1  package com.example.smartwatch;
2
3
4  import android.content.Intent;
5  import android.service.notification.NotificationListenerService;
6  import android.service.notification.StatusBarNotification;
7  import android.util.Log;
8  import android.os.Bundle;
9
10
11 /**
12  * Created by Asier on 26/04/2015.
13  */
14 public class NotificationService extends NotificationListenerService {
15
16     private static final String TAG = "NotificationService";
17
18     // Notification broadcast intent key
19     public static final String NOTIFICATION_KEY_CMD = "
   notification_command";
20     public static final String NOTIFICATION_KEY_ID = "notification_id";
21     public static final String NOTIFICATION_KEY_PACKAGE = "
   notification_package";
22     public static final String NOTIFICATION_KEY_TICKER = "notification_ticker
   ";
23     public static final String NOTIFICATION_KEY_TITLE = "notification_title";
24     public static final String NOTIFICATION_KEY_TEXT = "notification_text";
25     public static final String NOTIFICATION_KEY_BIG_TEXT = "
   notification_big_text";
26     public static final String NOTIFICATION_KEY_SUB_TEXT = "
   notification_sub_text";
27     public static final String NOTIFICATION_KEY_SUM_TEXT = "
   notification_sum_text";
28     public static final String NOTIFICATION_KEY_TEXT_LINES = "
   notification_text_lines";
29
30
31
32
33     // Notification command type
34     public static final int NOTIFICATION_CMD_ADD = 1;
35     public static final int NOTIFICATION_CMD_REMOVE = 2;
36     public static final int NOTIFICATION_CMD_LIST = 3;
37
38
39
40
41     @Override
42     public void onCreate() {
43         super.onCreate();
44
```

```
45      }
46
47      @Override
48      public void onDestroy() {
49          super.onDestroy();
50
51      }
52
53      @Override
54      public void onNotificationPosted(StatusBarNotification sbn) {
55          Log.d(TAG,"**********  onNotificationPosted");
56          Log.d(TAG, "ID :" + sbn.getId() + "t" + sbn.getNotification().tickerText + "t"
    + sbn.getPackageName());
57
58          String ticker = sbn.getNotification().tickerText.toString();
59          Bundle extras = sbn.getNotification().extras;
60          String title = extras.getString("android.title");
61          String text = extras.getCharSequence("android.text").toString();
62          //String bigtext = extras.getCharSequence("android.bigText").toString();
63          //String subtext=extras.getCharSequence("android.subText").toString();
64          //String sumtext=extras.getCharSequence("android.summaryText").toString();
65          //String textlines=extras.getCharSequence("android.textLines").toString();
66
67          Intent i = new  Intent(Constantes.NOTIFICATION_LISTENER);
68          i.putExtra(NOTIFICATION_KEY_CMD, NOTIFICATION_CMD_ADD);
69          i.putExtra(NOTIFICATION_KEY_ID, sbn.getId());
70          i.putExtra(NOTIFICATION_KEY_PACKAGE, sbn.getPackageName());
71          i.putExtra(NOTIFICATION_KEY_TICKER, ticker);
72          i.putExtra(NOTIFICATION_KEY_TITLE, title);
73          i.putExtra(NOTIFICATION_KEY_TEXT, text);
74          // i.putExtra(NOTIFICATION_KEY_BIG_TEXT,bigtext );
75          // i.putExtra(NOTIFICATION_KEY_SUB_TEXT, subtext);
76          // i.putExtra(NOTIFICATION_KEY_SUM_TEXT, sumtext);
77          // i.putExtra(NOTIFICATION_KEY_TEXT_LINES, textlines);
78          sendBroadcast(i);
79
80      }
81
82      @Override
83      public void onNotificationRemoved(StatusBarNotification sbn) {
84          Log.d(TAG,"********** onNOtificationRemoved");
85          Log.d(TAG,"ID :" + sbn.getId() + "t" + sbn.getNotification().tickerText +"t" +
    sbn.getPackageName());
86
87          /* Intent i = new  Intent(Constantes.NOTIFICATION_LISTENER);
88          i.putExtra(NOTIFICATION_KEY_CMD, NOTIFICATION_CMD_ADD);
89          i.putExtra(NOTIFICATION_KEY_ID, sbn.getId());
90          i.putExtra(NOTIFICATION_KEY_PACKAGE, sbn.getPackageName());
91          i.putExtra(NOTIFICATION_KEY_TICKER, ticker);
92          i.putExtra(NOTIFICATION_KEY_TITLE, title);
93          i.putExtra(NOTIFICATION_KEY_TEXT, text);
```

```
94          sendBroadcast(i);*/
95      }
96
97
98
99  }
100
```

```java
package com.example.smartwatch;


import android.app.Service;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.res.Configuration;
import android.os.BatteryManager;
import android.os.Binder;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
import android.widget.Toast;
import android.util.Log;

import java.util.Timer;


/**
 * Created by Asier on 14/03/2015.
 */
public class SmartWatchService extends Service {


    private static final String TAG = "SmartWatchService";


    // Context, System
    private Context mContext = null;
    private static Handler mActivityHandler = null;
    private ServiceHandler mServiceHandler = new ServiceHandler();
    private final IBinder mBinder = new SmartWatchServiceBinder();

    // Notification broadcast receiver
    private NotificationReceiver mReceiver;

    // Bluetooth
    private BluetoothAdapter mBluetoothAdapter = null;
    private ConexionBluetooth mBtManager = null;

    private InformacionDeConexion mConnectionInfo = null;
    private ProtocoloEnvioDatos mProtocolo = null;

    // Bateria
    public int BChargingState = 0;
```

```
52      public int BChargingMethod = 0;
53      public int NivelBateria = 0;
54      public int EscalaBateria = 0;
55
56      // Notificaciones
57
58      public int id = 0;
59      public String notificacion = null;
60      public String paquete = null;
61      public String titulo = null;
62      public String texto = null;
63      public String textogrande = null;
64      public String subtexto = null;
65      public String resumentexto = null;
66      public String lineastexto = null;
67
68
69
70      public String Whatsapp;
71      public String Whatsapptext;
72      public String Facebook;
73      public String Facebooktext;
74      public String Call;
75      public String SMS;
76      public String SMStext;
77      public String Gmail;
78      public String Gmailtext;
79
80
81      // Estado del telefono
82
83      public int sonando=0;
84      public int llamadaCogida=0;
85      public String numeroDeTelefono=null;
86
87
88
89      /**
90       * ************************************************
91       * <p/>
92       * Overrided methods
93       * <p/>
94       * ************************************************
95       */
96      @Override
97      public void onCreate() {
98          Log.d(TAG, "# Service - onCreate() starts here");
99
100
101         mContext = getApplicationContext();
102         initialize();
```

```
103     }
104
105     @Override
106     public int onStartCommand(Intent intent, int flags, int startId) {
107         Log.d(TAG, "# Service - onStartCommand() starts here");
108
109         // If service returns START_STICKY, android restarts service automatically
    after forced close.
110         // At this time, onStartCommand() method in service must handle null intent.
111         return Service.START_STICKY;
112     }
113
114     @Override
115     public void onConfigurationChanged(Configuration newConfig) {
116         // This prevents reload after configuration changes
117         super.onConfigurationChanged(newConfig);
118     }
119
120     @Override
121     public IBinder onBind(Intent intent) {
122         Log.d(TAG, "# Service - onBind()");
123         return mBinder;
124     }
125
126     @Override
127     public boolean onUnbind(Intent intent) {
128         Log.d(TAG, "# Service - onUnbind()");
129         return true;
130     }
131
132     @Override
133     public void onDestroy() {
134         Log.d(TAG, "# Service - onDestroy()");
135         finalizeService();
136     }
137
138     @Override
139     public void onLowMemory() {
140         Log.d(TAG, "# Service - onLowMemory()");
141         // onDestroy is not always called when applications are finished by Android
    system.
142         finalizeService();
143     }
144
145
146     /**
147      * ************************************************
148      * <p/>
149      * Private methods
150      * <p/>
151      * ************************************************
```

```
152      */
153     private void initialize() {
154         Log.d(TAG, "# Service : initialize ---");
155 
156 
157         mConnectionInfo = InformacionDeConexion.getInstance(mContext);
158 
159 
160         IntentFilter iFilter = new IntentFilter();
161         iFilter.addAction(Intent.ACTION_BATTERY_CHANGED);
162         registerReceiver(mBatteryInfoReceiver, iFilter);
163 
164         // Set notification broadcast receiver
165         mReceiver = new NotificationReceiver();
166         IntentFilter filter = new IntentFilter();
167         filter.addAction(Constantes.NOTIFICATION_LISTENER);
168         registerReceiver(mReceiver, filter);
169 
170         // Set telephony listener
171         TelephonyStateListener telephonyListener = new TelephonyStateListener();
172         TelephonyManager TM = (TelephonyManager) mContext.getSystemService(
    Context.TELEPHONY_SERVICE);
173         TM.listen(telephonyListener, PhoneStateListener.LISTEN_CALL_STATE);
174 
175         // Get local Bluetooth adapter
176         mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
177 
178         // If the adapter is null, then Bluetooth is not supported
179         if (mBluetoothAdapter == null) {
180             Toast.makeText(this, "Bluetooth is not available", Toast.LENGTH_LONG
    ).show();
181             return;
182         }
183 
184         if (!mBluetoothAdapter.isEnabled()) {
185             // BT is not on, need to turn on manually.
186             // Activity will do this.
187         } else {
188             if (mBtManager == null) {
189                 setupBT();
190             }
191         }
192 
193 
194     }
195 
196     public void MandarHorayFecha() {
197         ProtocoloEnvioDatos.Transaccion Protocolo = mProtocolo.CrearProtocolo();
198         Protocolo.begin();
199         Protocolo.setCommand(ProtocoloEnvioDatos.Transaccion.
    COMMAND_TYPE_SET_TIME);
```

```
200         Protocolo.setDate();
201         Protocolo.settingFinished();
202         Protocolo.sendTransaction();
203
204     }
205
206     public void MandarBateria() {
207         ProtocoloEnvioDatos.Transaccion Protocolo = mProtocolo.CrearProtocolo();
208         Protocolo.begin();
209         Protocolo.setCommand(ProtocoloEnvioDatos.Transaccion.
    COMMAND_TYPE_SET_BATTERY);
210         Protocolo.setBattery(NivelBateria, EscalaBateria, BChargingMethod,
    BChargingState);
211         Protocolo.settingFinished();
212         Protocolo.sendTransaction();
213
214     }
215 /*
216     public void MandarNotificaciones() {
217         ProtocoloEnvioDatos.Transaccion Protocolo = mProtocolo.CrearProtocolo();
218         Protocolo.begin();
219         Protocolo.setCommand(ProtocoloEnvioDatos.Transaccion.
    COMMAND_TYPE_SET_NOTIFICATION_MESSAGE);
220         Protocolo.setNotificationMessage(id, notificacion);
221         Protocolo.settingFinished();
222         Protocolo.sendTransaction();
223
224     }
225 */
226 public void MandarNotificacionWhatsapp() {
227     ProtocoloEnvioDatos.Transaccion Protocolo = mProtocolo.CrearProtocolo();
228     Protocolo.begin();
229     Protocolo.setCommand(ProtocoloEnvioDatos.Transaccion.
    COMMAND_TYPE_SET_WHATSAPP_NOTIFICATION);
230     Protocolo.setWhatsappNotification(Whatsapp,Whatsapptext);
231     Protocolo.settingFinished();
232     Protocolo.sendTransaction();
233 }
234     public void MandarNotificacionGmail() {
235         ProtocoloEnvioDatos.Transaccion Protocolo = mProtocolo.CrearProtocolo();
236         Protocolo.begin();
237         Protocolo.setCommand(ProtocoloEnvioDatos.Transaccion.
    COMMAND_TYPE_SET_GMAIL_NOTIFICATION);
238         Protocolo.setGmailNotification(Gmail,Gmailtext);
239         Protocolo.settingFinished();
240         Protocolo.sendTransaction();
241     }
242
243     public void MandarNotificacionLlamada() {
244         ProtocoloEnvioDatos.Transaccion Protocolo = mProtocolo.CrearProtocolo();
245         Protocolo.begin();
```

```
246         Protocolo.setCommand(ProtocoloEnvioDatos.Transaccion.
    COMMAND_TYPE_SET_CALL_NOTIFICATION);
247         Protocolo.setCallNotification(Call);
248         Protocolo.settingFinished();
249         Protocolo.sendTransaction();
250     }
251
252     public void MandarNotificacionFacebook() {
253         ProtocoloEnvioDatos.Transaccion Protocolo = mProtocolo.CrearProtocolo();
254         Protocolo.begin();
255         Protocolo.setCommand(ProtocoloEnvioDatos.Transaccion.
    COMMAND_TYPE_SET_FACEBOOK_NOTIFICATION);
256         Protocolo.setFacebookNotification(Facebook,Facebooktext);
257         Protocolo.settingFinished();
258         Protocolo.sendTransaction();
259     }
260     public void MandarNotificacionSMS() {
261         ProtocoloEnvioDatos.Transaccion Protocolo = mProtocolo.CrearProtocolo();
262         Protocolo.begin();
263         Protocolo.setCommand(ProtocoloEnvioDatos.Transaccion.
    COMMAND_TYPE_SET_SMS_NOTIFICATION);
264         Protocolo.setSMSNotification(SMS);
265         Protocolo.settingFinished();
266         Protocolo.sendTransaction();
267     }
268     public void MandarNotificaciones(){
269         MandarNotificacionWhatsapp();
270         MandarNotificacionGmail();
271         MandarNotificacionFacebook();
272         MandarNotificacionLlamada();
273     }
274     public void MandarEstadoDelTelefono() {
275         ProtocoloEnvioDatos.Transaccion Protocolo = mProtocolo.CrearProtocolo();
276         Protocolo.begin();
277         Protocolo.setCommand(ProtocoloEnvioDatos.Transaccion.
    COMMAND_TYPE_SET_TELEPHONY_CALL);
278         Protocolo.setTelephoneNumber(numeroDeTelefono);
279         //Protocolo.callState(sonando,llamadaCogida);
280         Protocolo.settingFinished();
281         Protocolo.sendTransaction();
282
283     }
284
285     /**
286      * ************************************************
287      * <p/>
288      * Public methods
289      * <p/>
290      * ************************************************
291      */
292     public void finalizeService() {
```

```
293         Log.d(TAG, "# Service : finalize ---");
294
295         mBluetoothAdapter = null;
296         // Stop the bluetooth session
297         if (mBtManager != null)
298             mBtManager.stop();
299         mBtManager = null;
300
301         // Unregister broadcast receiver
302
303         if (mBatteryInfoReceiver != null)
304             unregisterReceiver(mBatteryInfoReceiver);
305         mBatteryInfoReceiver = null;
306
307         if(mReceiver != null)
308             unregisterReceiver(mReceiver);
309         mReceiver = null;
310
311
312     }
313
314     public void setupService(Handler h) {
315         mActivityHandler = h;
316
317
318         if (mBtManager == null)
319             setupBT();
320
321         // Inicializar Protocolo
322         if (mProtocolo == null)
323             mProtocolo = new ProtocoloEnvioDatos(mBtManager, mActivityHandler);
324
325
326         // If ConnectionInfo holds previous connection info,
327         // try to connect using it.
328         if (mConnectionInfo.CogerDireccionDelDispositibo() != null &&
    mConnectionInfo.CogerNombreDispositibo() != null) {
329             connectDevice(mConnectionInfo.CogerDireccionDelDispositibo());
330         }
331         // or wait in listening mode
332         else {
333             if (mBtManager.CogerEstado() == ConexionBluetooth.STATE_NONE) {
334                 // Start the bluetooth services
335                 mBtManager.start();
336             }
337         }
338     }
339
340     /**
341      * Setup and initialize BT manager
342      */
```

```
343     public void setupBT() {
344         Log.d(TAG, "Service - setupBT()");
345
346         // Initialize the BluetoothManager to perform bluetooth connections
347         if (mBtManager == null)
348             mBtManager = new ConexionBluetooth(this, mServiceHandler);
349     }
350
351     /**
352      * Check bluetooth is enabled or not.
353      */
354     public boolean isBluetoothEnabled() {
355         if (mBluetoothAdapter == null) {
356             Log.e(TAG, "# Service - cannot find bluetooth adapter. Restart app.");
357             return false;
358         }
359         return mBluetoothAdapter.isEnabled();
360     }
361
362     /**
363      * Get scan mode
364      */
365     public int getBluetoothScanMode() {
366         int scanMode = -1;
367         if (mBluetoothAdapter != null)
368             scanMode = mBluetoothAdapter.getScanMode();
369
370         return scanMode;
371     }
372
373     /**
374      * Initiate a connection to a remote device.
375      */
376     public void connectDevice(String address) {
377         Log.d(TAG, "Service - connect to " + address);
378
379         // Get the BluetoothDevice object
380         if (mBluetoothAdapter != null) {
381             BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
382
383             if (device != null && mBtManager != null) {
384                 mBtManager.connect(device);
385             }
386         }
387     }
388
389     /**
390      * Connect to a remote device.
391      */
392     public void connectDevice(BluetoothDevice device) {
393         if (device != null && mBtManager != null) {
```

```
394             mBtManager.connect(device);
395         }
396     }
397
398     /**
399      * Get connected device name
400      */
401     public String getDeviceName() {
402         return mConnectionInfo.CogerNombreDispositibo();
403     }
404
405     /**
406      * ************************************************
407      * <p/>
408      * Handler, Listener, Timer, Sub classes
409      * <p/>
410      * ************************************************
411      */
412     public class SmartWatchServiceBinder extends Binder {
413         public SmartWatchService getService() {
414             return SmartWatchService.this;
415         }
416     }
417
418     class ServiceHandler extends Handler {
419         @Override
420         public void handleMessage(Message msg) {
421
422             switch (msg.what) {
423                 case ConexionBluetooth.MESSAGE_STATE_CHANGE:
424                     // Bluetooth state Changed
425                     Log.d(TAG, "Service - MESSAGE_STATE_CHANGE: " + msg.arg1
    );
426
427                     switch (msg.arg1) {
428                         case ConexionBluetooth.STATE_NONE:
429                             mActivityHandler.obtainMessage(Constantes.
    MESSAGE_BT_STATE_INITIALIZED).sendToTarget();
430
431                             break;
432
433                         case ConexionBluetooth.STATE_LISTEN:
434                             mActivityHandler.obtainMessage(Constantes.
    MESSAGE_BT_STATE_LISTENING).sendToTarget();
435                             break;
436
437                         case ConexionBluetooth.STATE_CONNECTING:
438                             mActivityHandler.obtainMessage(Constantes.
    MESSAGE_BT_STATE_CONNECTING).sendToTarget();
439                             break;
440
```

```
441                     case ConexionBluetooth.STATE_CONNECTED:
442                         mActivityHandler.obtainMessage(Constantes.
    MESSAGE_BT_STATE_CONNECTED).sendToTarget();
443
444
445                         break;
446                 }
447                 break;
448
449             case ConexionBluetooth.MESSAGE_WRITE:
450                 Log.d(TAG, "Service - MESSAGE_WRITE: ");
451                 break;
452
453             case ConexionBluetooth.MESSAGE_READ:
454                 Log.d(TAG, "Service - MESSAGE_READ: ");
455
456                 byte[] readBuf = (byte[]) msg.obj;
457                 // construct commands from the valid bytes in the buffer
458
459                 break;
460
461             case ConexionBluetooth.MESSAGE_DEVICE_NAME:
462                 Log.d(TAG, "Service - MESSAGE_DEVICE_NAME: ");
463
464                 // save connected device's name and notify using toast
465                 String deviceAddress = msg.getData().getString(Constantes.
    SERVICE_HANDLER_MSG_KEY_DEVICE_ADDRESS);
466                 String deviceName = msg.getData().getString(Constantes.
    SERVICE_HANDLER_MSG_KEY_DEVICE_NAME);
467
468                 if (deviceName != null && deviceAddress != null) {
469                     // Remember device's address and name
470                     mConnectionInfo.PonerDireccionDispositibo(deviceAddress);
471                     mConnectionInfo.PonerNombreDispositibo(deviceName);
472
473                     Toast.makeText(getApplicationContext(),
474                             "Connected to " + deviceName, Toast.LENGTH_SHORT).
    show();
475                 }
476                 break;
477
478             case ConexionBluetooth.MESSAGE_TOAST:
479                 Log.d(TAG, "Service - MESSAGE_TOAST: ");
480
481 //              Toast.makeText(getApplicationContext(),
482 //                  msg.getData().getString(Constants.
    SERVICE_HANDLER_MSG_KEY_TOAST),
483 //                  Toast.LENGTH_SHORT).show();
484                 break;
485
486         }   // End of switch(msg.what)
```

```
487
488             super.handleMessage(msg);
489         }
490     }    // End of class MainHandler
491
492     private BroadcastReceiver mBatteryInfoReceiver = new BroadcastReceiver() {
493         @Override
494         public void onReceive(Context context, Intent intent) {
495             String action = intent.getAction();
496             if (Intent.ACTION_BATTERY_CHANGED.equals(action)) {
497                 // Are we charging / charged?
498                 int status = intent.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
499                 boolean isCharging = status == BatteryManager.
    BATTERY_STATUS_CHARGING ||
500                         status == BatteryManager.BATTERY_STATUS_FULL;
501
502                 // How are we charging?
503                 int chargePlug = intent.getIntExtra(BatteryManager.EXTRA_PLUGGED, -
    1);
504                 boolean usbCharge = chargePlug == BatteryManager.
    BATTERY_PLUGGED_USB;
505                 boolean acCharge = chargePlug == BatteryManager.
    BATTERY_PLUGGED_AC;
506
507
508                 if (isCharging) {
509                     BChargingState = 1;
510                 }//Esta Cargando
511                 else {
512                     BChargingState = 0;
513                 } //  Esta Descargando
514
515                 if (usbCharge) {
516                     BChargingMethod = 2;
517                 }//Cargando en USB
518                 else if (acCharge) {
519                     BChargingMethod = 1;
520                 }// Cargando con la corriente
521                 else {
522                     BChargingMethod = 0;
523                 }// Esta descargandose
524
525
526             }
527             int level = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
528             int scale = intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
529             NivelBateria = level;
530             EscalaBateria = scale;
531
532         }
533
```

```
534     };
535
536     /**
537      * Broadcast receiver class. Receives notification data
538      */
539     class NotificationReceiver extends BroadcastReceiver {
540
541         @Override
542         public void onReceive(Context context, Intent intent) {
543             int cmd = intent.getIntExtra(NotificationService.
    NOTIFICATION_KEY_CMD, 0);
544             int noti_id = intent.getIntExtra(NotificationService.
    NOTIFICATION_KEY_ID, -1);
545             String packageName = intent.getStringExtra(NotificationService.
    NOTIFICATION_KEY_PACKAGE);
546             String textTicker = intent.getStringExtra(NotificationService.
    NOTIFICATION_KEY_TICKER);
547             String title=intent.getStringExtra(NotificationService.
    NOTIFICATION_KEY_TITLE);
548             String text=intent.getStringExtra(NotificationService.
    NOTIFICATION_KEY_TEXT);
549             // String bigtext=intent.getStringExtra(NotificationService.
    NOTIFICATION_KEY_BIG_TEXT);
550             // String subtext=intent.getStringExtra(NotificationService.
    NOTIFICATION_KEY_SUB_TEXT);
551             // String sumtext=intent.getStringExtra(NotificationService.
    NOTIFICATION_KEY_SUM_TEXT);
552             // String textlines=intent.getStringExtra(NotificationService.
    NOTIFICATION_KEY_TEXT_LINES);
553             switch (cmd) {
554
555                 case NotificationService.NOTIFICATION_CMD_ADD:
556                     if (packageName != null) {
557                         Log.d(TAG, "** Service - Add noti=" + noti_id + ", package=" +
    packageName);
558                         switch (packageName){
559                             case (Constantes.WHATSAPP_PACKAGE):
560                                 Whatsapp=title;
561                                 Whatsapptext=text;
562
563                                 break;
564                             case (Constantes.GMAIL_PACKAGE):
565                                 Gmail=title;
566                                 Gmailtext=text;
567
568                                 break;
569                             case Constantes.FACEBOOK_PACKAGE:
570                                 Facebook=title;
571                                 Facebooktext=text;
572                                 break;
573                             /*case Constantes.FACEBOOK_CHAT_PACKAGE:
```

```
574                         Facebook=textTicker;
575                         break;*/
576                     case Constantes.SMS_PACKAGE:
577                         SMS=textTicker;
578                         //SMStext=text;
579                         break;
580                 }
581
582
583
584                 id = noti_id;
585                 notificacion = textTicker;
586                 paquete = packageName;
587                 titulo=title;
588                 texto=text;
589                 //textogrande=bigtext;
590                 //subtexto=subtext;
591                 //resumentexto=sumtext;
592                 // lineastexto=textlines;
593
594                 Intent i = new Intent("Notificaciones");
595                 i.putExtra("id", id);
596                 i.putExtra("paquete", paquete);
597                 i.putExtra("notificacion", notificacion);
598                 i.putExtra("titulo", titulo);
599                 i.putExtra("texto", texto);
600                 // i.putExtra("textogrande", textogrande);
601                 // i.putExtra("subtexto", subtexto);
602                 // i.putExtra("resumentexto", resumentexto);
603                 // i.putExtra("lineastexto", lineastexto);
604                 sendBroadcast(i);
605
606                 // notify to activity
607                 mActivityHandler.obtainMessage(Constantes.
    MESSAGE_ADD_NOTIFICATION, noti_id, 0).sendToTarget();
608
609             }
610
611             break;
612
613         case NotificationService.NOTIFICATION_CMD_REMOVE:
614             Log.d(TAG, "** Service - Delete noti=" + noti_id + ", package=" +
    packageName);
615
616             id = 0;
617             notificacion = null;
618             paquete = null;
619             titulo=null;
620             texto=null;
621             // textogrande=null;
622             // subtexto=null;
```

```
623                 // resumentexto=null;
624                 // lineastexto=null;
625
626                 Intent i = new Intent("Notificaciones");
627                 i.putExtra("id", id);
628                 i.putExtra("paquete", paquete);
629                 i.putExtra("notificacion", notificacion);
630                 i.putExtra("titulo", titulo);
631                 i.putExtra("texto", texto);
632                 //i.putExtra("textogrande", textogrande);
633                 // i.putExtra("subtexto", subtexto);
634                 // i.putExtra("resumentexto", resumentexto);
635                 // i.putExtra("lineastexto", lineastexto);
636                 sendBroadcast(i);
637                 // notify to activity
638                 mActivityHandler.obtainMessage(Constantes.
    MESSAGE_DELETE_NOTIFICATION, noti_id, 0).sendToTarget();
639
640
641                 break;
642             }   // End of switch(cmd)
643         }   // End of onReceive()
644
645     }   // End of class NotificationReceive
646
647
648     public class TelephonyStateListener extends PhoneStateListener {
649         @Override
650         public void onCallStateChanged(int state, String incomingNumber) {
651             Log.d(TAG, "PhoneStateListener - onCallStateChanged();");
652             switch (state) {
653                 case TelephonyManager.CALL_STATE_IDLE:
654                     //sonando=0;
655                     //llamadaCogida=0;
656                     // Toast.makeText(SmartWatchService.this, "SIN LLAMADAS", Toast.
    LENGTH_SHORT).show();
657                     break;
658
659                 case TelephonyManager.CALL_STATE_RINGING:
660                     //sonando=1;
661                     // llamadaCogida=0;
662                     numeroDeTelefono=incomingNumber;
663                     // Toast.makeText(SmartWatchService.this, "LLAMANDO", Toast.
    LENGTH_LONG).show();
664
665
666                     // send to device
667
668                     MandarEstadoDelTelefono();
669
670                     break;
```

```
671                 case TelephonyManager.CALL_STATE_OFFHOOK:
672                     //sonando=0;
673                     //llamadaCogida=1;
674                     numeroDeTelefono=incomingNumber;
675                     //Toast.makeText(SmartWatchService.this, "HABLANDO", Toast.
    LENGTH_LONG).show();
676
677                     MandarEstadoDelTelefono();
678
679                     break;
680
681                 default:
682                     Log.d(TAG, "PhoneStateListener - Default state=" + state + ",
    Number=" + incomingNumber);
683                     break;
684             }
685         }
686
687
688     }
689 }
```

```
1  package com.example.smartwatch;
2
3
4  import android.os.Handler;
5  import java.util.Calendar;
6  import android.util.Log;
7
8  /**
9   * Created by Asier on 15/03/2015.
10  */
11 public class ProtocoloEnvioDatos {
12
13    private static final String TAG = "Protocolo Envio Datos";
14
15    private ConexionBluetooth mBTManager = null;
16    private Handler mHandler = null;
17
18
19
20
21
22    public ProtocoloEnvioDatos(ConexionBluetooth cb, Handler errorHandler) {
23       mBTManager = cb;
24       mHandler = errorHandler;
25       // Poner un Broadcast receiver para recoger el estado de la bateria
26
27    }
28
29    public Transaccion CrearProtocolo() {
30       return new Transaccion();
31    }
32
33    public class Transaccion {
34
35       public static final int MAX_MESSAGE_LENGTH = 10000;
36
37       // Command types
38       public static final int COMMAND_TYPE_NONE = 0x00;
39
40       public static final int COMMAND_TYPE_SET_TIME = 0x02;
41       public static final int COMMAND_TYPE_SET_BATTERY = 0x03;
42       public static final int COMMAND_TYPE_SET_TELEPHONY_CALL= 0x04;
43      // public static final int COMMAND_TYPE_SET_NOTIFICATION_MESSAGE =
    0x05;
44       public static final int
    COMMAND_TYPE_SET_WHATSAPP_NOTIFICATION = 0x05;
45       public static final int COMMAND_TYPE_SET_GMAIL_NOTIFICATION =
    0x06;
46       public static final int
    COMMAND_TYPE_SET_FACEBOOK_NOTIFICATION = 0x07;
47       public static final int COMMAND_TYPE_SET_CALL_NOTIFICATION =
```

```
47  0x08;
48          public static final int COMMAND_TYPE_SET_SMS_NOTIFICATION = 0x09
    ;
49
50
51
52
53
54
55
56          // byte definitions for buffer setting
57          private static final byte TRANSACTION_START_BYTE = (byte)0xfc;
58          private static final byte TRANSACTION_END_BYTE = (byte)0xfd;
59
60
61          // Transaction instance status
62          private static final int STATE_NONE = 0;     // Instance created
63          private static final int STATE_BEGIN = 1;    // Initialize transaction
64          private static final int STATE_SETTING_FINISHED = 2;    // End of setting
    parameters
65          private static final int STATE_TRANSFERED = 3;  // End of sending
    transaction data
66          private static final int STATE_ERROR = -1;     // Error occurred
67
68          // Transaction parameters
69          private int mState = STATE_NONE;
70
71          private byte[] mBuffer = null;
72
73          private int mCommandType = COMMAND_TYPE_NONE;
74          private int mId=0x00;
75          private byte mDateMonth = 0x00;
76          private byte mDateDay = 0x00;
77          private byte mDateWeek = 0x00;
78          private byte mDateNoon = 0x00;
79          private byte mDateHour = 0x00;
80          private byte mDateMinute = 0x00;
81          public byte MiNivelBateria=0x00;
82          public byte MiEscalaBateria=0x00;
83          public byte MiMetodoCargaBateria=0x00;
84          public byte MiEstadoCargaBateria=0x00;
85         // private String mNotification = null;
86          private String WhatsappNotification=null;
87          private String WhatsappNotificationInfo=null;
88          private String FacebookNotification=null;
89          private String FacebookNotificationInfo=null;
90          private String GmailNotification=null;
91          private String GmailNotificationInfo=null;
92          private String CallNotification=null;
93          private String SMSNotification=null;
94          private String SMSNotificationInfo=null;
```

```
95      private String mPackage=null;
96      private String TelephoneNumber=null;
97      private String dospuntos=":";
98      byte[]TwoPoint=dospuntos.getBytes();
99      //private byte Ringing=0x00;
100     //private byte Dialing=0x01;
101
102
103     public void begin() {
104        mState = STATE_BEGIN;
105
106        mCommandType = COMMAND_TYPE_NONE;
107
108
109        //Fecha y Hora
110        mDateMonth = 0x00;
111        mDateDay = 0x00;
112        mDateWeek = 0x00;
113        mDateNoon = 0x00;
114        mDateHour = 0x00;
115        mDateMinute = 0x00;
116
117        // Mensaje de Notificaciones
118
119        //mNotification = null;
120        WhatsappNotification=null;
121        FacebookNotification=null;
122        GmailNotification=null;
123        CallNotification=null;
124        SMSNotification=null;
125        WhatsappNotificationInfo=null;
126        FacebookNotificationInfo=null;
127        GmailNotificationInfo=null;
128        SMSNotificationInfo=null;
129
130        //Bateria
131        MiNivelBateria=0x00;
132        MiEscalaBateria=0x00;
133        MiMetodoCargaBateria=0x00;
134        MiEstadoCargaBateria=0x00;
135
136        // Numero de telefono
137        TelephoneNumber=null;
138        //Ringing=0x00;
139        //Dialing=0x01;
140
141        mBuffer = null;
142     }
143
144     public void setCommand(int cmd) {
145        switch(cmd) {
```

```
146
147
148             case COMMAND_TYPE_SET_TIME:
149             case COMMAND_TYPE_SET_BATTERY:
150             case COMMAND_TYPE_SET_TELEPHONY_CALL:
151             //case COMMAND_TYPE_SET_NOTIFICATION_MESSAGE:
152             case COMMAND_TYPE_SET_WHATSAPP_NOTIFICATION:
153             case COMMAND_TYPE_SET_GMAIL_NOTIFICATION:
154             case COMMAND_TYPE_SET_FACEBOOK_NOTIFICATION:
155             case COMMAND_TYPE_SET_CALL_NOTIFICATION:
156             case COMMAND_TYPE_SET_SMS_NOTIFICATION:
157                 mCommandType = cmd;
158                 break;
159             default:
160                 mCommandType = COMMAND_TYPE_NONE;
161                 break;
162         }
163     }
164
165     public void setDate(int month, int day, int week, int noon, int hour, int minute
    ) {
166         mDateMonth = (byte)month;
167         mDateDay = (byte)day;
168         mDateWeek = (byte)week;
169         mDateNoon = (byte)noon;
170         mDateHour = (byte)hour;
171         mDateMinute = (byte)minute;
172     }
173
174     public void setDate() {
175         Calendar c = Calendar.getInstance();
176         int month = c.get(Calendar.MONTH);
177         int day = c.get(Calendar.DAY_OF_MONTH);
178         int week = c.get(Calendar.DAY_OF_WEEK);
179         int noon = c.get(Calendar.AM_PM);
180         int hour = c.get(Calendar.HOUR_OF_DAY);
181         // int hour = c.get(Calendar.HOUR);
182         int minute = c.get(Calendar.MINUTE);
183
184         mDateMonth = (byte)month;
185         mDateDay = (byte)day;
186         mDateWeek = (byte)week;
187         mDateNoon = (byte)noon;
188         mDateHour = (byte)hour;
189         mDateMinute = (byte)minute;
190     }
191     public void setBattery (int mnb,int meb, int mmcb, int mecb){
192
193         MiNivelBateria= (byte)mnb;
194         MiEscalaBateria= (byte) meb;
195         MiMetodoCargaBateria=(byte) mmcb;
```

```
196             MiEstadoCargaBateria=(byte) mecb;
197         }
198
199         public void setTelephoneNumber (String number){
200             TelephoneNumber= number;
201         }
202
203         //public void callState (byte ring, byte speak){
204           // Ringing= ring;
205           // Dialing= speak;
206         // }
207
208         public void setId(int id) {
209             mId = id;
210         }
211
212         /**
213          * Set Notification message to send
214          * @param id  Identifier - WARNING: use lower 1 byte only
215          * @param mensaje    String to send
216          */
217         public void setNotificationMessage(int id, String mensaje ) {
218             mId = id;
219             //mNotification = mensaje;
220
221         }
222
223         public void setWhatsappNotification (String W, String X){
224             WhatsappNotification=W;
225             WhatsappNotificationInfo=X;
226
227         }
228         public void setFacebookNotification (String F, String H ){
229             FacebookNotification=F;
230             FacebookNotificationInfo=H;
231
232         }
233         public void setGmailNotification (String G, String E){
234             GmailNotification=G;
235             GmailNotificationInfo=E;
236
237         }
238         public void setCallNotification (String C){
239             CallNotification=C;
240
241         }
242         public void setSMSNotification (String A){
243             SMSNotification=A;
244           // SMSNotificationInfo=B;
245
246         }
```

```
247
248
249
250         public void settingFinished() {
251             mState = STATE_SETTING_FINISHED;
252
253             switch(mCommandType) {
254
255
256                 // Command byte + Date bytes
257                 case COMMAND_TYPE_SET_TIME:
258                     // Make buffer
259                     // [Transaction start signal : 1byte : 0xfd]
260                     // [command type : 1byte]
261                     // [data packet : month(1byte), day(1byte), week(1byte), noon(1byte),
    hour(1byte), min(1byte) ]
262                     // [Transaction end signal : 1byte : 0xfe]
263                     mBuffer = new byte[9];
264
265                     mBuffer[0] = TRANSACTION_START_BYTE; // Transaction start
    signal
266                     mBuffer[1] = (byte)mCommandType;   // Command
267                     mBuffer[2] = mDateMonth;
268                     mBuffer[3] = mDateDay;
269                     mBuffer[4] = mDateWeek;
270                     mBuffer[5] = mDateNoon;
271                     mBuffer[6] = mDateHour;
272                     mBuffer[7] = mDateMinute;
273                     mBuffer[8] = TRANSACTION_END_BYTE;
274
275                     break;
276                 case COMMAND_TYPE_SET_BATTERY:
277                     mBuffer = new byte[7];
278
279                     mBuffer[0] = TRANSACTION_START_BYTE; // Transaction start
    signal
280                     mBuffer[1] = (byte)mCommandType;   // Command
281                     mBuffer[2] = MiNivelBateria ;
282                     mBuffer[3] = MiEscalaBateria;
283                     mBuffer[4] = MiEstadoCargaBateria;
284                     mBuffer[5] = MiMetodoCargaBateria;
285                     mBuffer[6] = TRANSACTION_END_BYTE;
286
287                     break;
288
289                 /*
290                 case COMMAND_TYPE_SET_NOTIFICATION_MESSAGE:
291                     if (mNotification==null||mNotification.length()<1){
292                     mState=STATE_ERROR;
293                         break;
294                     }
```

```
295
296                 byte[] notificationBuffer=mNotification.getBytes();
297                 // Make buffer
298                 // [Transaction start signal : 1byte : 0xfd]
299                 // [command type : 1byte]
300                 // [Notification ID : 1byte]
301                 // [data packet : various size: Currently max 1000 byte ]
302                 // [Transaction end signal : 1byte : 0xfe]
303                 mBuffer=new byte[3+notificationBuffer.length];
304                 mBuffer[0] = TRANSACTION_START_BYTE; // Transaction start
    signal
305                 mBuffer[1] = (byte)mCommandType;       // Command
306                // mBuffer[2] = (byte)mId;// Message ID
307
308                 System.arraycopy(notificationBuffer, 0, mBuffer, 2,
309                         (notificationBuffer.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : notificationBuffer.length);
310
311                 mBuffer[mBuffer.length-1] = TRANSACTION_END_BYTE;
312
313
314                 break;
315                 */
316
317             case COMMAND_TYPE_SET_WHATSAPP_NOTIFICATION:
318                 if (WhatsappNotification==null||WhatsappNotification.length()<1){
319                     mState=STATE_ERROR;
320                     break;
321                 }
322
323                 byte[] WhatsappBuffer=WhatsappNotification.getBytes();
324                 byte[] WhatsappTextBuffer=WhatsappNotificationInfo.getBytes();
325                 // Make buffer
326                 // [Transaction start signal : 1byte : 0xfd]
327                 // [command type : 1byte]
328                 // [data packet : various size: Currently max 1000 byte ]
329                 // [Transaction end signal : 1byte : 0xfe]
330                 mBuffer=new byte[3+WhatsappBuffer.length+WhatsappTextBuffer.
    length+TwoPoint.length];
331                 mBuffer[0] = TRANSACTION_START_BYTE; // Transaction start
    signal
332                 mBuffer[1] = (byte)mCommandType;       // Command
333                 // mBuffer[2] = (byte)mId;// Message ID
334
335                 System.arraycopy(WhatsappBuffer, 0, mBuffer, 2,
336                         (WhatsappBuffer.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : WhatsappBuffer.length);
337
338                 System.arraycopy(TwoPoint, 0, mBuffer,WhatsappBuffer.length+2 ,
339                         (TwoPoint.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : TwoPoint.length);
```

```
340
341                 System.arraycopy(WhatsappTextBuffer,0 , mBuffer,WhatsappBuffer.
    length+TwoPoint.length+2,
342                         (WhatsappTextBuffer.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : WhatsappTextBuffer.length);
343
344
345                 mBuffer[mBuffer.length-1] = TRANSACTION_END_BYTE;
346
347                 break;
348             case COMMAND_TYPE_SET_GMAIL_NOTIFICATION:
349                 if (GmailNotification==null||GmailNotification.length()<1){
350                     mState=STATE_ERROR;
351                     break;
352                 }
353
354                 byte[] GmailBuffer=GmailNotification.getBytes();
355                 byte[] GmailTextBuffer=GmailNotificationInfo.getBytes();
356                 // Make buffer
357                 // [Transaction start signal : 1byte : 0xfd]
358                 // [command type : 1byte]
359                 // [data packet : various size: Currently max 1000 byte ]
360                 // [Transaction end signal : 1byte : 0xfe]
361                 mBuffer=new byte[3+GmailBuffer.length+TwoPoint.length+
    GmailTextBuffer.length];
362                 mBuffer[0] = TRANSACTION_START_BYTE; // Transaction start
    signal
363                 mBuffer[1] = (byte)mCommandType;      // Command
364                 // mBuffer[2]  = (byte)mId;// Message ID
365
366                 System.arraycopy(GmailBuffer, 0, mBuffer, 2,
367                         (GmailBuffer.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : GmailBuffer.length);
368
369                 System.arraycopy(TwoPoint, 0, mBuffer,GmailBuffer.length+2 ,
370                         (TwoPoint.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : TwoPoint.length);
371
372                 System.arraycopy(GmailTextBuffer,0 , mBuffer,GmailBuffer.length+
    TwoPoint.length+2,
373                         (GmailTextBuffer.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : GmailTextBuffer.length);
374
375                 mBuffer[mBuffer.length-1] = TRANSACTION_END_BYTE;
376
377                 break;
378
379             case COMMAND_TYPE_SET_FACEBOOK_NOTIFICATION:
380                 if (FacebookNotification==null||FacebookNotification.length()<1){
381                     mState=STATE_ERROR;
382                     break;
```

```
383                 }
384
385                 byte[] FacebookBuffer=FacebookNotification.getBytes();
386                 byte[] FacebookTextBuffer=FacebookNotificationInfo.getBytes();
387                 // Make buffer
388                 // [Transaction start signal : 1byte : 0xfd]
389                 // [command type : 1byte]
390                 // [data packet : various size: Currently max 1000 byte ]
391                 // [Transaction end signal : 1byte : 0xfe]
392                 mBuffer=new byte[3+FacebookBuffer.length+TwoPoint.length+
    FacebookTextBuffer.length];
393                 mBuffer[0] = TRANSACTION_START_BYTE; // Transaction start
    signal
394                 mBuffer[1] = (byte)mCommandType;      // Command
395                 // mBuffer[2]  = (byte)mId;// Message ID
396
397                 System.arraycopy(FacebookBuffer, 0, mBuffer, 2,
398                         (FacebookBuffer.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : FacebookBuffer.length);
399
400                 System.arraycopy(TwoPoint, 0, mBuffer,FacebookBuffer.length+2 ,
401                         (TwoPoint.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : TwoPoint.length);
402
403                 System.arraycopy(FacebookTextBuffer,0 , mBuffer,FacebookBuffer.
    length+TwoPoint.length+2,
404                         (FacebookTextBuffer.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : FacebookTextBuffer.length);
405
406                 mBuffer[mBuffer.length-1] = TRANSACTION_END_BYTE;
407
408                 break;
409
410             case COMMAND_TYPE_SET_CALL_NOTIFICATION:
411                 if (CallNotification==null||CallNotification.length()<1){
412                     mState=STATE_ERROR;
413                     break;
414                 }
415
416                 byte[] CallBuffer=CallNotification.getBytes();
417                 // Make buffer
418                 // [Transaction start signal : 1byte : 0xfd]
419                 // [command type : 1byte]
420                 // [data packet : various size: Currently max 1000 byte ]
421                 // [Transaction end signal : 1byte : 0xfe]
422                 mBuffer=new byte[3+CallBuffer.length];
423                 mBuffer[0] = TRANSACTION_START_BYTE; // Transaction start
    signal
424                 mBuffer[1] = (byte)mCommandType;      // Command
425                 // mBuffer[2]  = (byte)mId;// Message ID
426
```

```
427                 System.arraycopy(CallBuffer, 0, mBuffer, 2,
428                         (CallBuffer.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : CallBuffer.length);
429
430                 mBuffer[mBuffer.length-1] = TRANSACTION_END_BYTE;
431
432                 break;
433             case COMMAND_TYPE_SET_SMS_NOTIFICATION:
434                 if (SMSNotification==null||SMSNotification.length()<1){
435                     mState=STATE_ERROR;
436                     break;
437                 }
438
439                 byte[] SMSBuffer=SMSNotification.getBytes();
440                     //byte[] SMSTextBuffer=SMSNotificationInfo.getBytes();
441                 // Make buffer
442                 // [Transaction start signal : 1byte : 0xfd]
443                 // [command type : 1byte]
444                 // [data packet : various size: Currently max 1000 byte ]
445                 // [Transaction end signal : 1byte : 0xfe]
446                 mBuffer=new byte[3+SMSBuffer.length];
447                 mBuffer[0] = TRANSACTION_START_BYTE; // Transaction start
    signal
448                 mBuffer[1] = (byte)mCommandType;      // Command
449                 // mBuffer[2]  = (byte)mId;// Message ID
450
451                 System.arraycopy(SMSBuffer, 0, mBuffer, 2,
452                         (SMSBuffer.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : SMSBuffer.length);
453
454             /* System.arraycopy(TwoPoint, 0, mBuffer,SMSBuffer.length+2 ,
455                         (TwoPoint.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : TwoPoint.length);
456
457                 System.arraycopy(SMSTextBuffer,0 , mBuffer,SMSBuffer.length+
    TwoPoint.length+2,
458                         (SMSTextBuffer.length > MAX_MESSAGE_LENGTH) ?
    MAX_MESSAGE_LENGTH : SMSTextBuffer.length);*/
459
460                 mBuffer[mBuffer.length-1] = TRANSACTION_END_BYTE;
461
462
463                 break;
464             case COMMAND_TYPE_SET_TELEPHONY_CALL:
465                 if (TelephoneNumber==null||TelephoneNumber.length()<1){
466                     mState=STATE_ERROR;
467                     break;
468                 }
469
470                 byte[]TelephoneBuffer=TelephoneNumber.getBytes();
471                 // Make buffer
```

```
472                 // [Transaction start signal : 1byte : 0xfd]
473                 // [command type : 1byte]
474                 // [telephone number: size 9 ]
475                 // [Transaction end signal : 1byte : 0xfe]
476                 mBuffer=new byte[3+TelephoneBuffer.length];
477                 mBuffer[0] = TRANSACTION_START_BYTE; // Transaction start
    signal
478                 mBuffer[1] = (byte)mCommandType;      // Command
479                 //mBuffer[2] = (byte)Ringing;          // =1 phone ringing ( someone
    calling)
480                 //mBuffer[3] = (byte)Dialing;         //=1 call taken
481                 System.arraycopy(TelephoneBuffer, 0, mBuffer, 2,TelephoneBuffer.
    length);
482                 mBuffer[mBuffer.length-1] = TRANSACTION_END_BYTE;
483                 break;
484
485             default:
486                 mState = STATE_ERROR;
487                 break;
488         }
489     }
490
491     public byte[] getPacket() {
492         if(mState == STATE_SETTING_FINISHED) {
493             return mBuffer;
494         }
495         return null;
496     }
497
498
499
500     public boolean sendTransaction() {
501         if(mBuffer == null) {
502             Log.e(TAG, "##### Ooooooops!! No sending buffer!! Check command
    !!");
503             return false;
504         }
505
506         // For debug
507         if(mBuffer.length > 0) {
508             StringBuilder sb = new StringBuilder();
509
510             switch(mBuffer[1]) {
511
512
513                 case COMMAND_TYPE_SET_TIME:
514                     sb.append("COMMAND_TYPE_SET_TIME : ");
515                     break;
516                 case COMMAND_TYPE_SET_BATTERY:
517                     sb.append("COMMAND_TYPE_SET_BATTERY : ");
518                     break;
```

```
519                 /* case COMMAND_TYPE_SET_NOTIFICATION_MESSAGE:
520                    sb.append("COMMAND_TYPE_SET_NOTIFICATION_MESSAGE
    : ");
521                     break;*/
522                 case COMMAND_TYPE_SET_TELEPHONY_CALL:
523                    sb.append("COMMAND_TYPE_SET_TELEPHONY_CALL : ");
524                    break;
525                 default:
526                    break;
527              }
528
529              for(int i=0; i<mBuffer.length; i++) {
530                 sb.append(String.format("%02X, ", mBuffer[i]));
531              }
532
533
534              Log.d(TAG, sb.toString());
535           }
536
537           if(mState == STATE_SETTING_FINISHED) {
538              if(mBTManager != null) {
539                 // Check that we're actually connected before trying anything
540                 if (mBTManager.CogerEstado() == ConexionBluetooth.
    STATE_CONNECTED) {
541                    // Check that there's actually something to send
542                    if (mBuffer.length > 0) {
543                       // Get the message bytes and tell the BluetoothChatService to write
544                       mBTManager.write(mBuffer);
545
546                       mState = STATE_TRANSFERED;
547                       return true;
548                    }
549                    mState = STATE_ERROR;
550                 }
551                 mHandler.obtainMessage(Constantes.
    MESSAGE_CMD_ERROR_NOT_CONNECTED).sendToTarget();
552              }
553           }
554           return false;
555        }
556     }  // End of class Transaccion
557  // Battery BroadcastReceiver
558
559
560  }
561
562
563
```

```java
package com.example.smartwatch;

/**
 * Created by Asier on 14/03/2015.
 */
public class Constantes {


    // Notification intent action string
    public static final String NOTIFICATION_LISTENER_SERVICE = "com.example.smartwatch.NOTIFICATION_LISTENER_SERVICE";
    public static final String NOTIFICATION_LISTENER = "com.example.smartwatch.NOTIFICATION_LISTENER";

    //Notification packages
    public static final String WHATSAPP_PACKAGE =    "com.whatsapp";
    public static final String CALL_PACKAGE = "com.android.phone";
    public static final String FACEBOOK_PACKAGE = "com.facebook.katana";
    public static final String FACEBOOK_CHAT_PACKAGE = "com.facebook.orca";
    public static final String GMAIL_PACKAGE = "com.google.android.gm";
    public static final String SMS_PACKAGE = "com.android.mms";



    // Service handler message key
    public static final String SERVICE_HANDLER_MSG_KEY_DEVICE_NAME = "device_name";
    public static final String SERVICE_HANDLER_MSG_KEY_DEVICE_ADDRESS = "device_address";
    public static final String SERVICE_HANDLER_MSG_KEY_TOAST = "toast";

    // Intent request codes
    public static final int REQUEST_CONNECT_DEVICE = 1;
    public static final int REQUEST_ENABLE_BT = 2;

    // Message types sent from Service to Activity
    public static final int MESSAGE_CMD_ERROR_NOT_CONNECTED = -50;

    public static final int MESSAGE_BT_STATE_INITIALIZED = 1;
    public static final int MESSAGE_BT_STATE_LISTENING = 2;
    public static final int MESSAGE_BT_STATE_CONNECTING = 3;
    public static final int MESSAGE_BT_STATE_CONNECTED = 4;
    public static final int MESSAGE_BT_STATE_ERROR = 10;

    public static final int MESSAGE_ADD_NOTIFICATION = 101;
    public static final int MESSAGE_DELETE_NOTIFICATION = 105;


    // Preference
    public static final String PREFERENCE_NAME = "SmartWatchPref";
```

```
47
48      public static final String PREFERENCE_CONN_INFO_ADDRESS = "
    device_address";
49      public static final String PREFERENCE_CONN_INFO_NAME = "device_name
    ";
50
51
52
53  }
54
```

File - C:\Users\Asier\Documents\AndroidStudioProjects\SmartWatch\app\src\main\AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.example.smartwatch" >
4      <uses-permission android:name="android.permission.BLUETOOTH_ADMIN
   " />
5      <uses-permission android:name="android.permission.BLUETOOTH" />
6      <uses-permission android:name="android.permission.
   READ_PHONE_STATE" />
7      <application
8          android:allowBackup="true"
9          android:icon="@drawable/ic_launcher"
10         android:label="@string/app_name"
11         android:theme="@style/AppTheme" >
12         <activity
13             android:name=".SmartWatchActivity"
14             android:label="@string/app_name" >
15             <intent-filter>
16                 <action android:name="android.intent.action.MAIN" />
17
18                 <category android:name="android.intent.category.LAUNCHER" />
19             </intent-filter>
20         </activity>
21         <activity android:name=".ListaDeDispositibos"
22             android:label="@string/select_device"
23             android:theme="@android:style/Theme.Dialog"
24             android:configChanges="keyboardHidden|orientation|screenSize" >
25         </activity>
26         <!-- Service -->
27         <service
28             android:name= "com.example.smartwatch.SmartWatchService"
29             android:icon="@drawable/ic_launcher"
30             android:label= "@string/service_name"
31             android:configChanges="keyboardHidden|orientation|screenSize" >
32         </service>
33
34         <service android:name=".NotificationService"
35             android:label="@string/noti_receiver_name"
36             android:permission="android.permission.
   BIND_NOTIFICATION_LISTENER_SERVICE">
37             <intent-filter>
38                 <action android:name="android.service.notification.
   NotificationListenerService" />
39             </intent-filter>
40         </service>
41
42     </application>
43
44 </manifest>
45
```

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:tools="http://schemas.android.com/tools" android:layout_width="
   match_parent"
3     android:layout_height="match_parent" android:paddingLeft="@dimen/
   activity_horizontal_margin"
4     android:paddingRight="@dimen/activity_horizontal_margin"
5     android:paddingTop="@dimen/activity_vertical_margin"
6     android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".
   SmartWatchActivity">
7
8
9     <ImageView
10       android:id="@+id/status_title"
11       android:layout_width="wrap_content"
12       android:layout_height="20dip"
13       android:layout_margin="3dip"
14
15       android:layout_alignParentEnd="false"
16       android:layout_alignParentStart="false" />
17    <TextView
18       android:id="@+id/status_text"
19       android:layout_width="match_parent"
20       android:layout_height="wrap_content"
21
22       android:layout_alignParentStart="false"
23       android:layout_alignParentEnd="false"
24       android:layout_toEndOf="@id/status_title"
25       android:layout_marginLeft="3dip"
26       android:layout_marginRight="3dip"
27       android:layout_marginBottom="3dip"
28       android:layout_alignTop="@+id/status_title" />
29
30
31    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
32       android:layout_width="fill_parent"
33       android:layout_height="fill_parent"
34       android:orientation="vertical"
35       android:layout_alignParentBottom="true"
36       android:layout_alignParentStart="true"
37       android:weightSum="1"
38       android:layout_alignParentEnd="false"
39       android:layout_below="@+id/status_text">
40
41       <TextView
42          android:layout_width="fill_parent"
43          android:layout_height="wrap_content" />
44
45
46
47
48
```

```
49        <ScrollView
50           android:layout_width="match_parent"
51           android:layout_height="match_parent">
52           <TableLayout
53              android:layout_width="match_parent"
54              android:layout_height="wrap_content"
55              android:id="@+id/tab" />
56        </ScrollView>
57     </LinearLayout>
58
59  </RelativeLayout>
60
```

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
4      android:orientation="vertical"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      >
8      <TextView android:id="@+id/title_paired_devices"
9          android:layout_width="match_parent"
10         android:layout_height="wrap_content"
11         android:text="@string/title_paired_devices"
12         android:visibility="gone"
13         android:background="#666"
14         android:textColor="#fff"
15         android:paddingLeft="5dp"
16         />
17     <ListView android:id="@+id/paired_devices"
18         android:layout_width="match_parent"
19         android:layout_height="wrap_content"
20         android:stackFromBottom="true"
21         android:layout_weight="1"
22         />
23     <TextView android:id="@+id/title_new_devices"
24         android:layout_width="match_parent"
25         android:layout_height="wrap_content"
26         android:text="@string/title_other_devices"
27         android:visibility="gone"
28         android:background="#666"
29         android:textColor="#fff"
30         android:paddingLeft="5dp"
31         />
32     <ListView android:id="@+id/new_devices"
33         android:layout_width="match_parent"
34         android:layout_height="wrap_content"
35         android:stackFromBottom="true"
36         android:layout_weight="2"
37         />
38     <Button android:id="@+id/button_scan"
39         android:layout_width="match_parent"
40         android:layout_height="wrap_content"
41         android:text="@string/button_scan"
42         />
43 </LinearLayout>
```

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3
4     <TextView xmlns:android="http://schemas.android.com/apk/res/android"
5        android:layout_width="match_parent"
6        android:layout_height="wrap_content"
7        android:textSize="18sp"
8        android:padding="5dp" >
9
10       </TextView>
11
12
```

```
1  <menu xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:app="http://schemas.android.com/apk/res-auto"
3     xmlns:tools="http://schemas.android.com/tools" tools:context=".
   SmartWatchActivity">
4
5
6     <item
7        android:id="@+id/accion_escanear"
8        android:icon="@drawable/ic_action_action_bluetooth"
9        android:orderInCategory="10"
10       app:showAsAction="always"
11       android:title="@string/connect"/>
12
13
14
15    <item
16       android:id="@+id/hora_fecha"
17       android:icon="@drawable/ic_time"
18       android:orderInCategory="20"
19       app:showAsAction="always"
20       android:title="@string/send_contents"/>
21
22
23    <item android:id="@+id/battery_info"
24       android:icon="@drawable/ic_action_device_battery"
25       android:title="@string/battery_info"
26       android:orderInCategory="30"
27       app:showAsAction="always" />
28
29    <item android:id="@+id/quit"
30       android:icon="@drawable/ic_action_quit"
31       android:orderInCategory="40"
32       android:title="Quit "
33       app:showAsAction="always"/>
34
35
36
37 </menu>
38
```