



The Abdus Salam
International Centre
for Theoretical Physics



2434-2

**First Regional Workshop on Distributed Embedded Systems with Emphasis on
Open Source**

30 July - 17 August, 2012

Android Applications Development

Carlos Kavka
ESTECO SpA Area Science Park
Trieste
Italy



Explore new perspectives

ESTECO

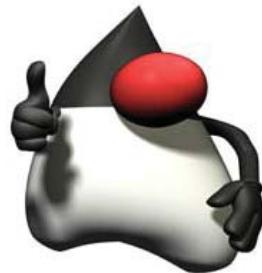
Android Applications Development

Carlos Kavka
ESTECO SpA
Area Science Park, Trieste, Italy

First Regional Workshop on Distributed Embedded Systems
Universiti Tunku Abdul Rahman (UTAR) - Kampar - Malaysia
30 July - 17 August 2012

Objectives

- ✓ provide a general **overview** of Android Applications Development



- ✓ assumes knowledge of Android SDK and Java

- ✓ not a tutorial nor a complete reference, most concepts are introduced with **examples**



Explore new perspectives

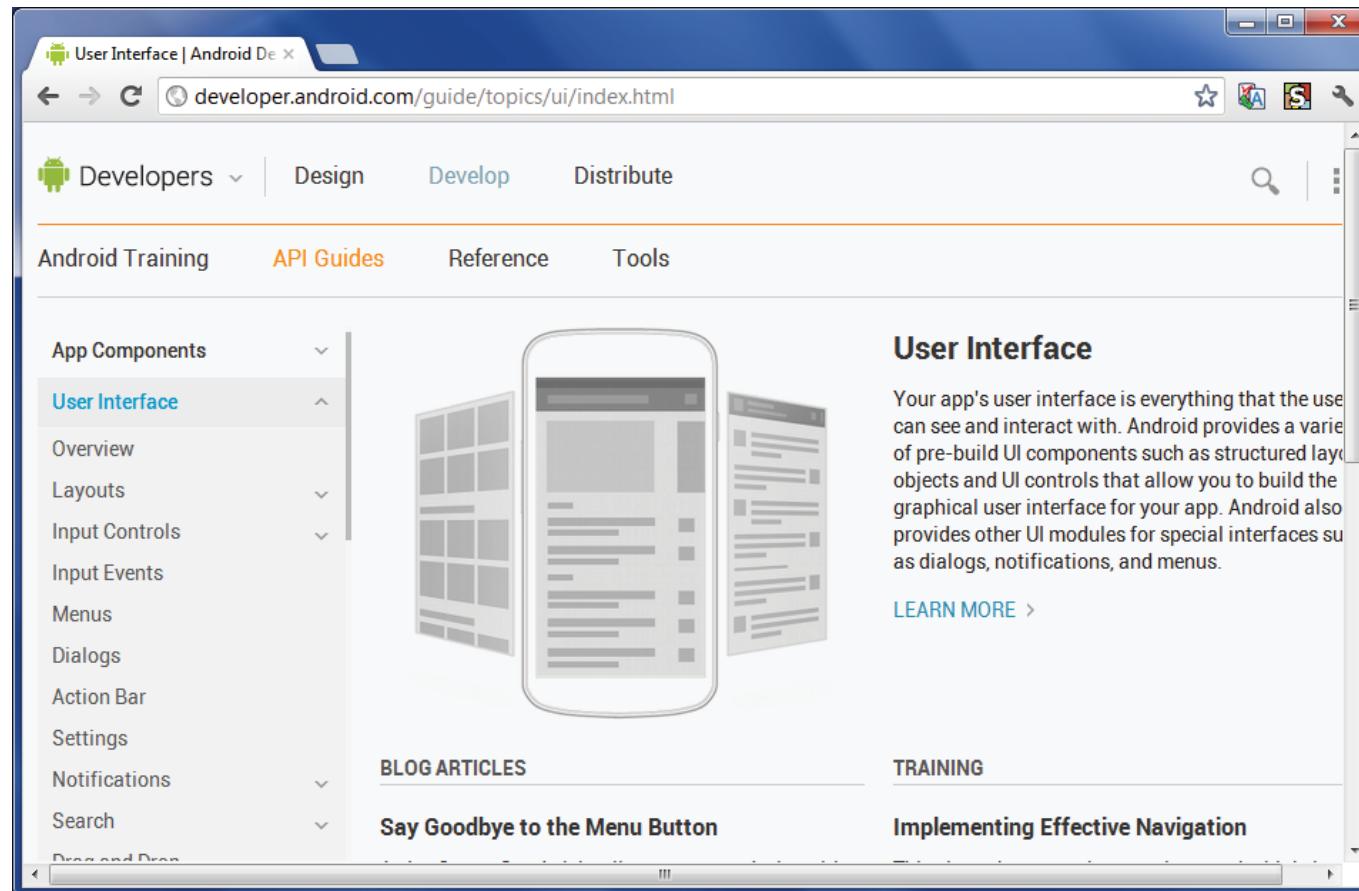
Android Applications Development

Part I: Basic concepts

ESTECO

Android developer web site

<http://developer.android.com>

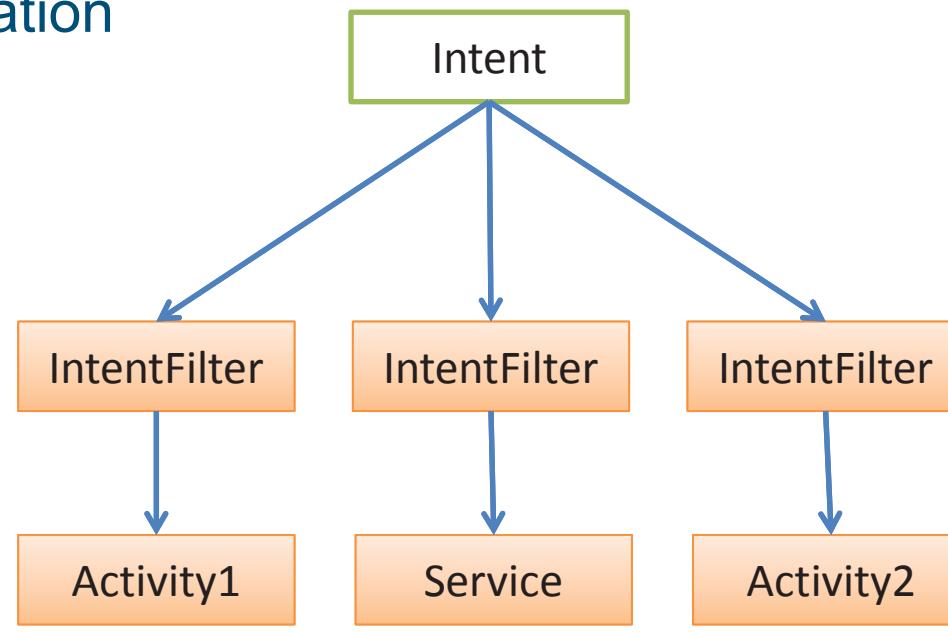


the place to go!

Application flow

There is no a **main** method like in a typical Java application

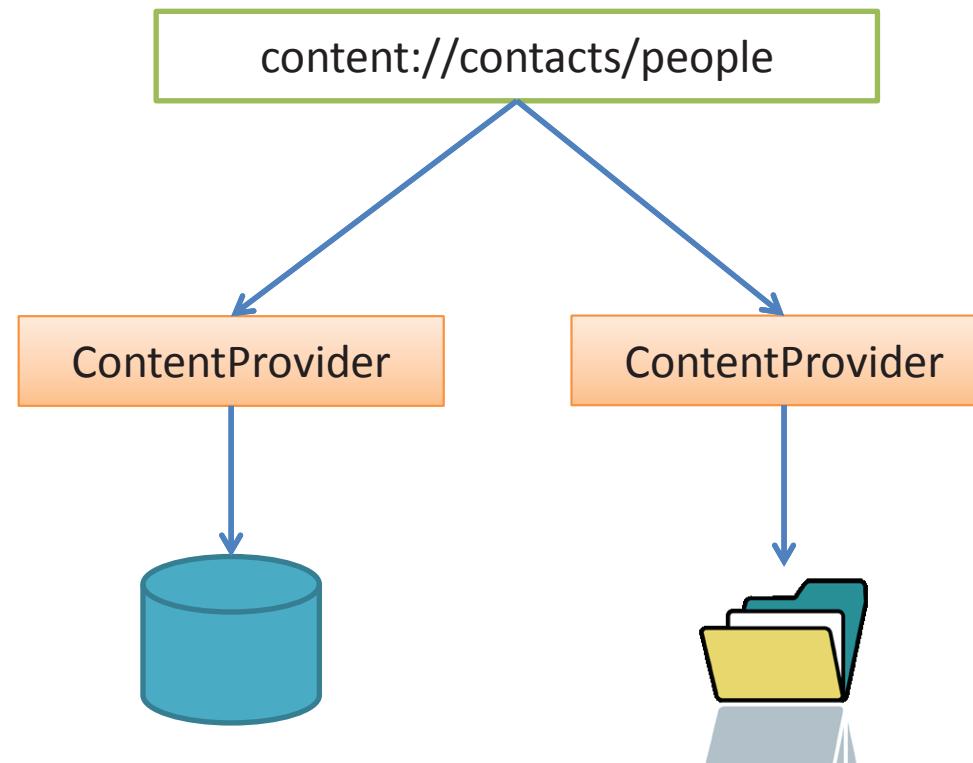
An **intent** is a declaration of need



An **intent-filter** is a declaration of capacity

Application flow

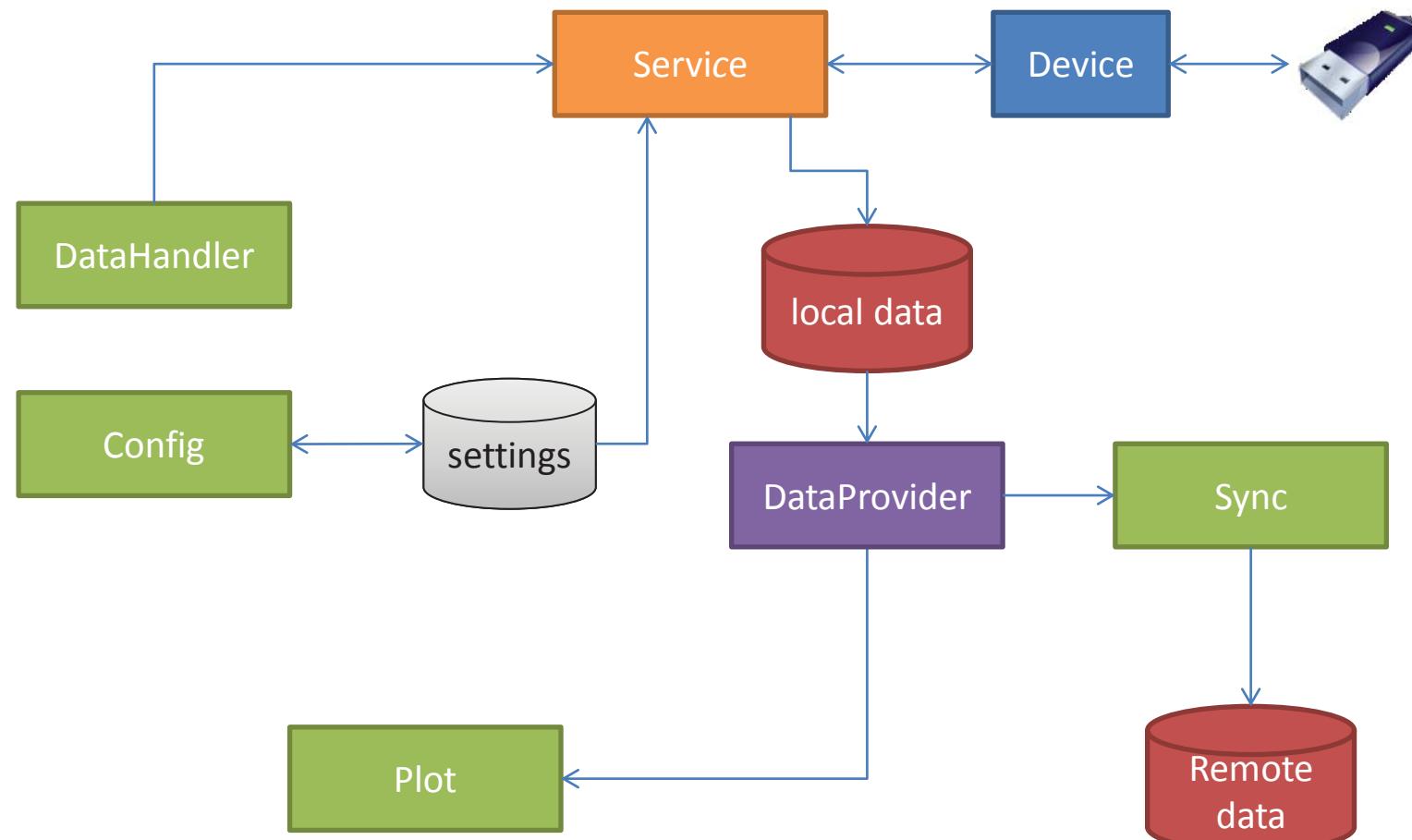
A Content Provider exposes data to other applications



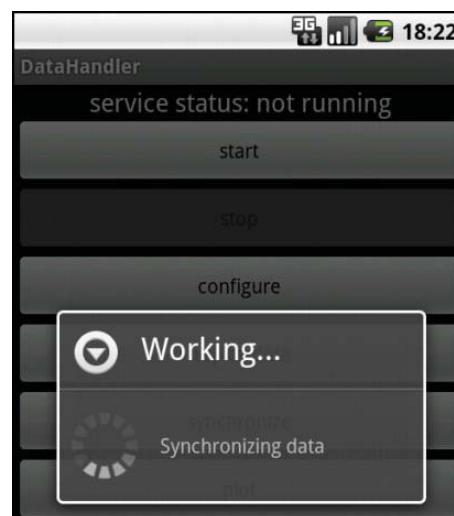
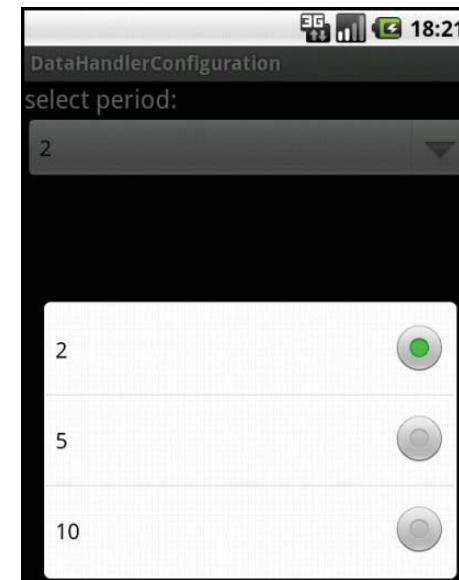
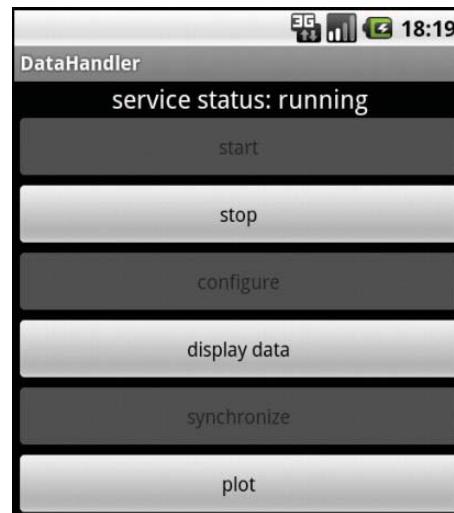
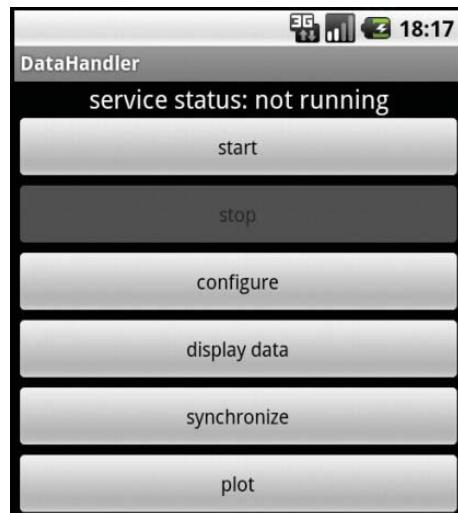
Requests are performed by providing an **URI**

Demo application

During the lectures, a demo application will be built



Demo application



Application

- ✓ include all **components**: activities, services, context providers, etc.
- ✓ If it has UI, at least an **activity** is required
- ✓ It is configured through a configuration file called **AndroidManifest.xml**

Context

- ✓ The Context defines configuration and runtime information
- ✓ It is configured through the file **AndroidManifest.xml**
- ✓ Includes the resources (graphic layouts, images, strings, arrays, etc.)

Application manifest file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="my.workshop"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="8" />

    <application android:icon="@drawable/ic_launcher"
        android:label="@string/app_name">

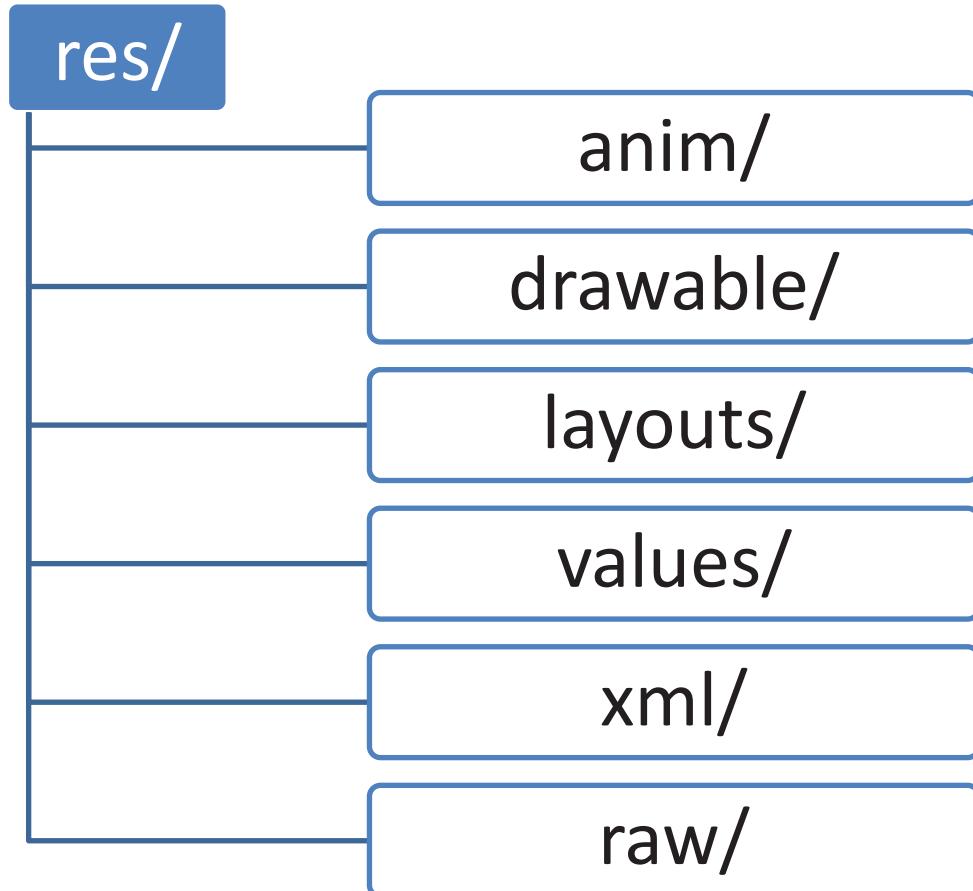
        <activity>
            ...
        </activity>
        ...
    </application>

</manifest>
```

the configuration file includes one section for each component

the notation @ is used to reference resources

Resources: directory structure



Resources: values

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">DataHandler</string>
    <string name="start_button">start</string>
    <string name="stop_button">stop</string>
    <string name="status_text">service status: not running</string>
</resources>
```

strings

arrays

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="periods_array">
        <item>2</item>
        <item>5</item>
        <item>10</item>
    </string-array>
</resources>
```

Resources: layouts

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView> ... </TextView>
    <Button> ... </Button>
    <Button> ... </Button>
</LinearLayout>
```

One entry for each UI element



Resources: a text view

```
<TextView  
    android:id="@+id/status_text"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:gravity="center"  
    android:text="@string/status_text"  
    android:textAppearance="?android:attr/textAppearanceMedium"/>
```

@+id creates and register an identifier

@string references an object from the string file



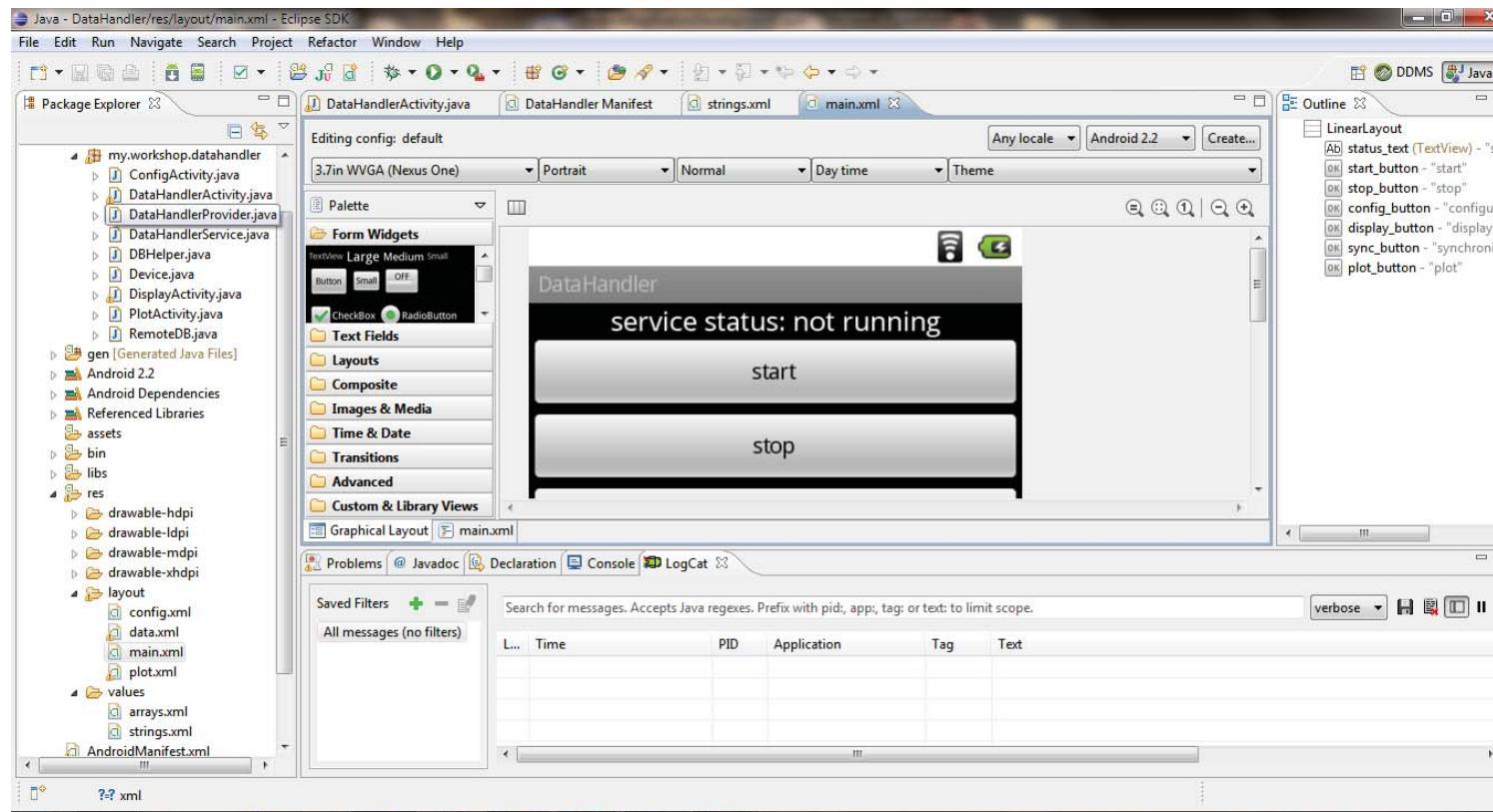
Resources: buttons

```
<Button  
    android:id="@+id/start_button"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/start_button" />  
  
<Button  
    android:id="@+id/stop_button"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/stop_button" />
```



Resources

The file is usually not written in XML, but generated **automatically** with the IDE



DEMO now

Activity

An **activity** extends the abstract Activity class

The **bundle** allows to save and restore the activity state

The **onCreate()** method is called when the activity is started

```
public class DataHandlerActivity extends Activity {  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
  
        // content view  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        ...  
    }  
}
```

Activity: accessing resources

The resources can be accessed through the R class

```
...
// get UI elements
startButton = (Button) findViewById(R.id.start_button);
stopButton = (Button) findViewById(R.id.stop_button);
status = (TextView) findViewById(R.id.status_text);

// get strings
stoppedMsg = getResources().getString(R.string.stopped_msg);
....
```

The R class is generated automatically by the Eclipse IDE

Activity: Manifest section

The application defines its
intent-filter

```
<activity
    android:name=".datahandler.DataHandlerActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

in this case, the application
can just be started from the
main launcher

Shared preferences

- ✓ Allows to save and retrieve key-value pairs of primitive data types

```
SharedPreferences preferences;
```

Private preferences:

```
preferences = getPreferences(Context.MODE_PRIVATE);
```

Preferences on file:

```
preferences = getSharedPreferences("myPreferences",  
                                Context.MODE_PRIVATE);
```

Access level: *MODE_PRIVATE, WORLD_READABLE, WORLD_WRITABLE*

Preferences

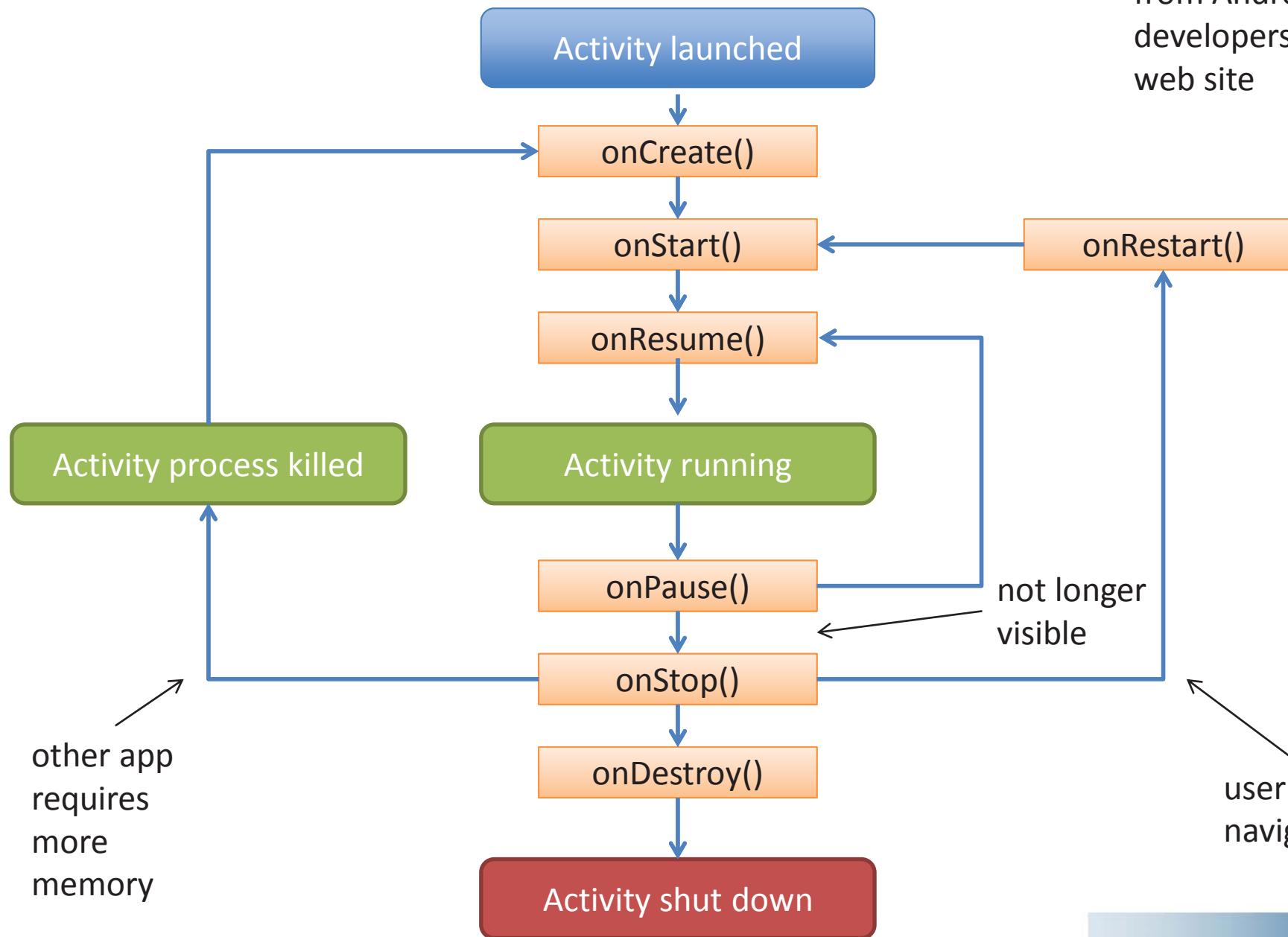
Preferences are
written using an editor:

```
int sizeValue = 25;  
voolean validStatus = true;  
  
Editor editor = preferences.edit();  
editor.putIntger("size", sizeValue);  
editor.putBoolean("valid", validStatus);  
editor.commit();
```

Preferences are read
with a specific method

```
int property = preferences.getInteger("size", 0);  
boolean validStatus = preference.getBoolean("valid", false);
```

Activity life-cycle



from Android
developers
web site

Lifecycle: saving and restoring state

```
boolean running;  
  
@Override  
protected void onPause() {  
    super.onPause();  
  
    // store preferences  
    Editor editor = preferences.edit();  
    editor.putBoolean("running", running);  
    editor.commit();  
}
```

Persistent data can be restored in onResume()

Persistent data can be saved in onPause() using a shared preferences

```
@Override  
protected void onResume() {  
    super.onResume();  
  
    // get preferences  
    running = preferences.getBoolean("running", false);  
  
    updateUI();  
}
```

Lifecycle: saving and restoring state

By using a Bundle it is possible also to save the state before an object is destroyed and restore when is being recreated

```
@Override  
protected void  
onSaveInstanceState(Bundle state) {  
    state.putString("myData", value);  
    super.onSaveInstanceState(state);  
}
```

```
@Override  
protected void  
onRestoreInstanceState(Bundle state) {  
    super.onRestoreInstanceState(state);  
    String value = state.getString("myData");  
}
```

However, persistent data must be handled with onPause() and onResume because they are part of the **lifecycle** of the Activity!



Explore new perspectives

Android Applications Development

Part II: The user interface

ESTECO

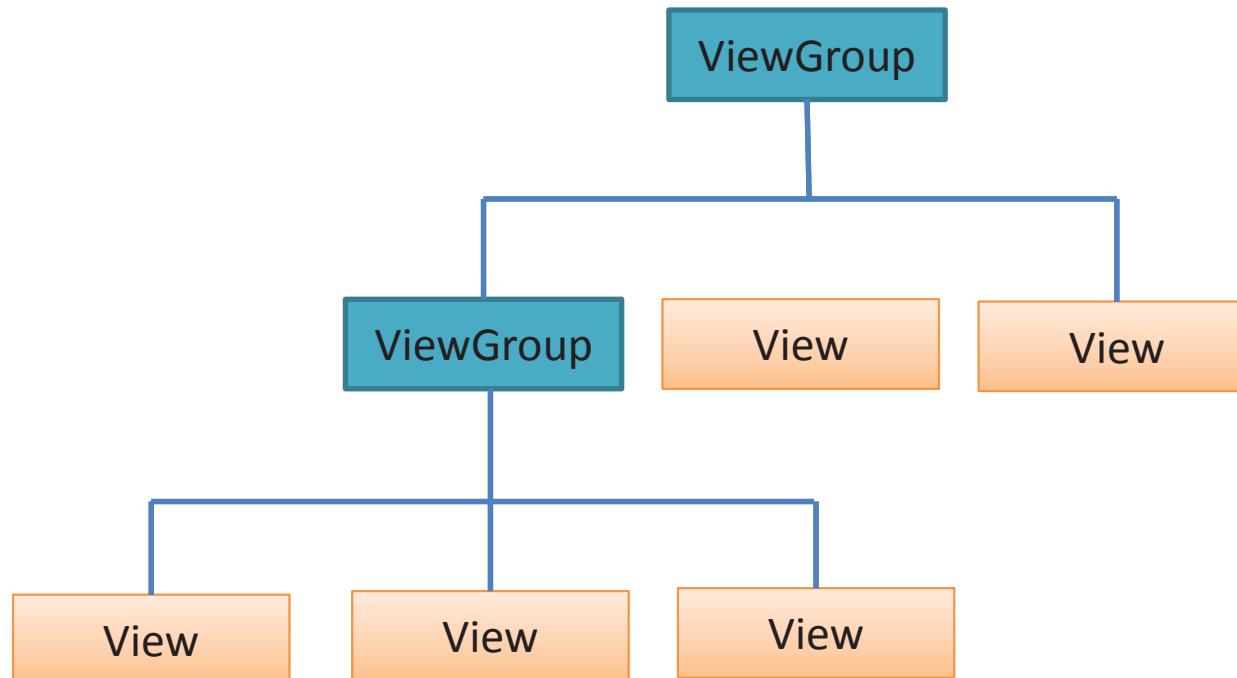
User Interface

- ✓ Android provides a **complete UI** with a large number of views
- ✓ Event driven
- ✓ It is **single threaded** to guarantee no synchronization problems

User interface

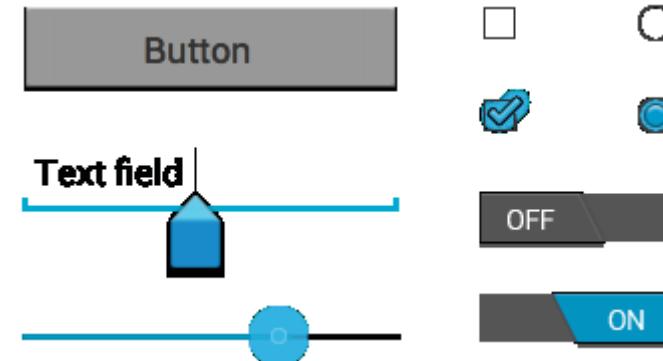
The UI is defined with a hierarchy of ViewGroup and View elements

Views are the basic UI elements (buttons, text fields, etc.)



A ViewGroup is a container that organizes Views

Some widgets



ATTENDING?

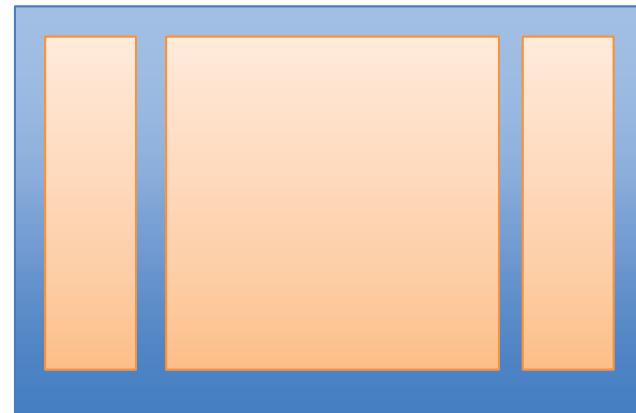
Yes Maybe No

- Sync Browser
5/31/2012 4:58 PM
- Sync Calendar
6/1/2012 11:15 AM
- Sync Contacts
6/1/2012 3:50 PM

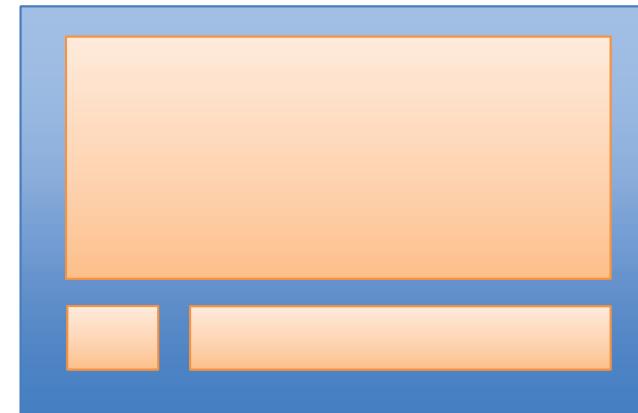
Layouts

A layout defines the **structure** and holds all the elements that appears in the screen

Linear layout



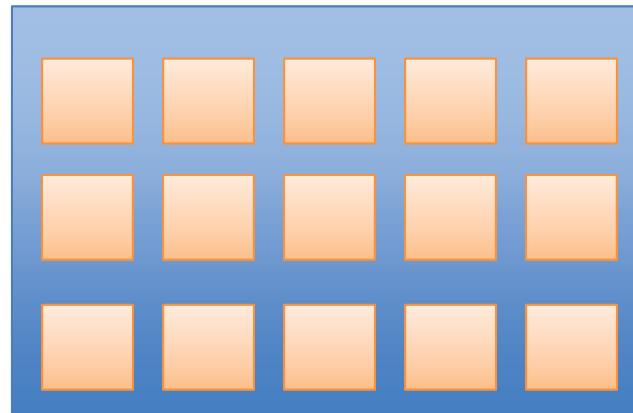
Relative layout



Layouts

Some layouts use an extra **adapter** class

Grid view



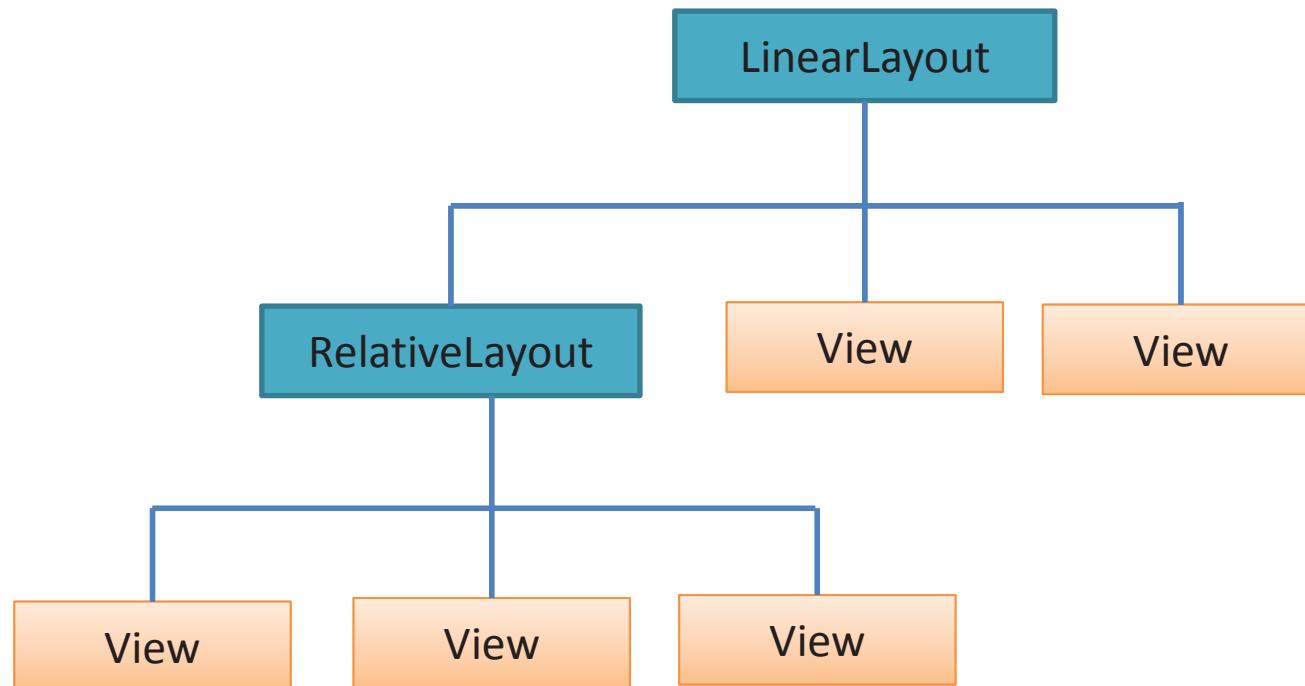
List view



An adapter **pulls** content from a source converts each item result into a view

Layouts

Can be mixed at
different levels



Linear layout

For example

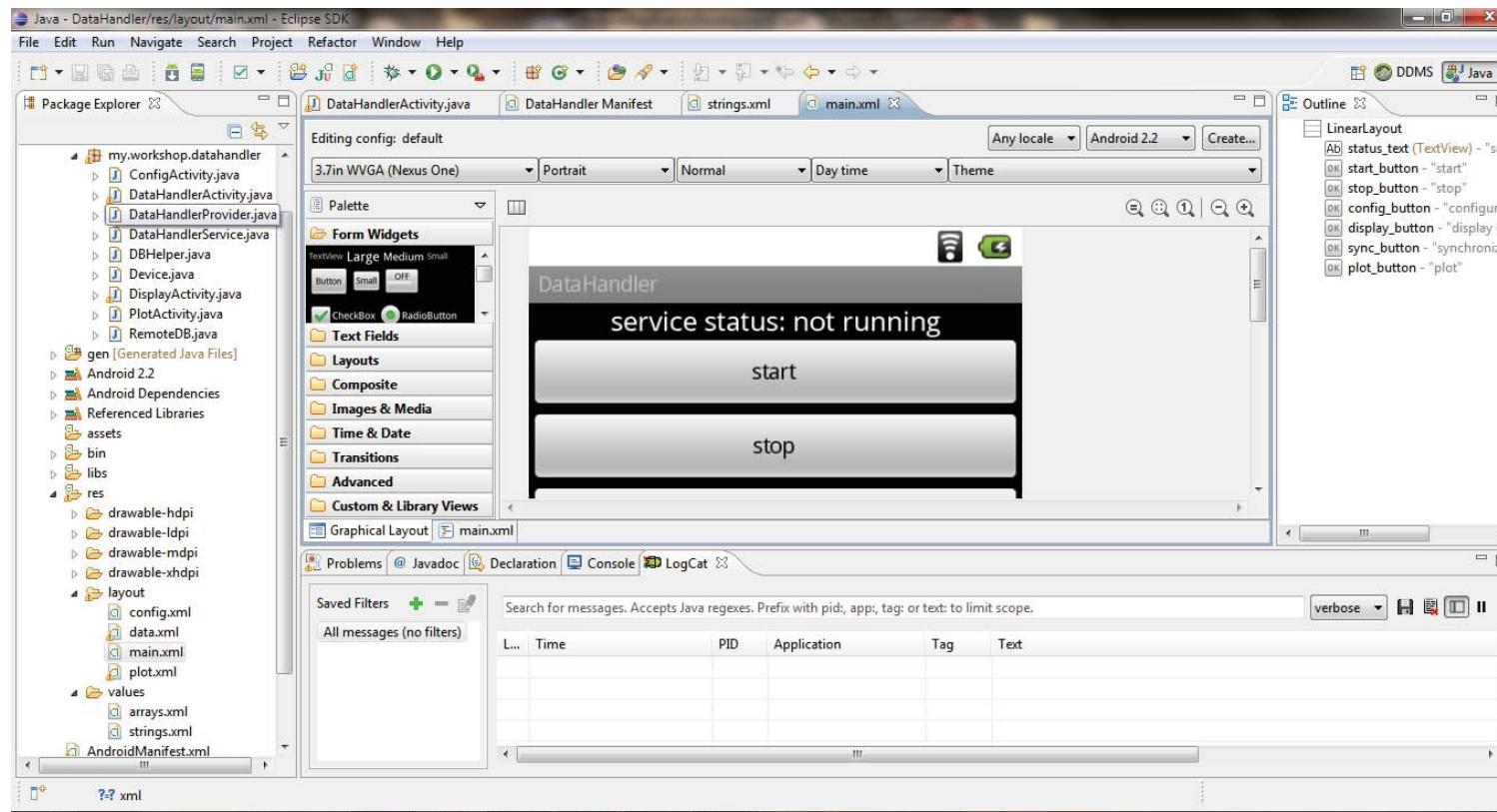
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView> ... </TextView>
    <Button> ... </Button>
    <Button> ... </Button>
</LinearLayout>
```



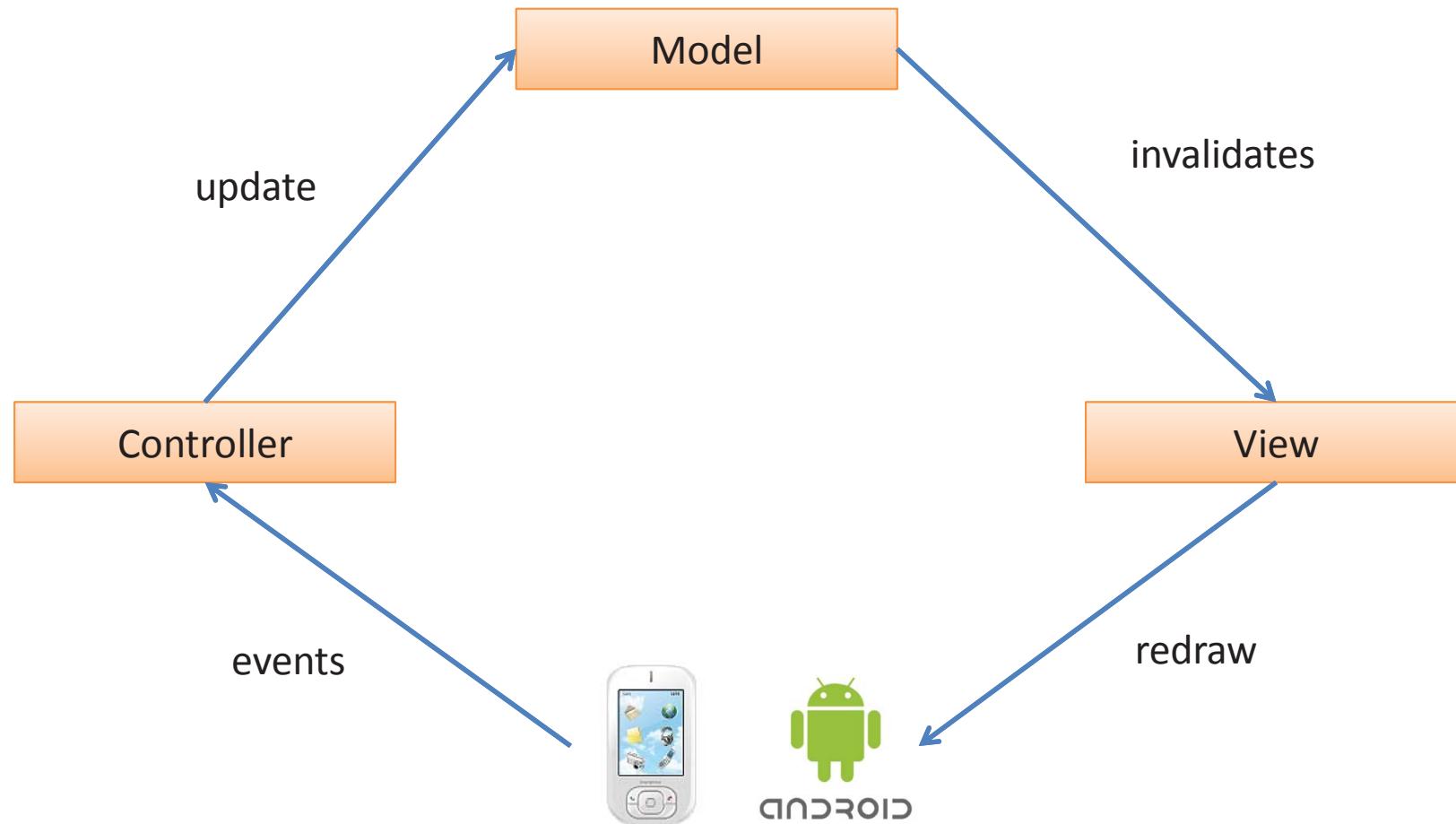
Combined layouts

The XML layout can be generated automatically with the IDE



DEMO now

Controller View Model



Controller View Model

- ✓ Event driven
 - ✓ There is a **single** event queue
 - ✓ Single threaded to guarantee no synchronization problems
 - ✓ UI must be **responsive**, long running tasks will be interrupted by Android

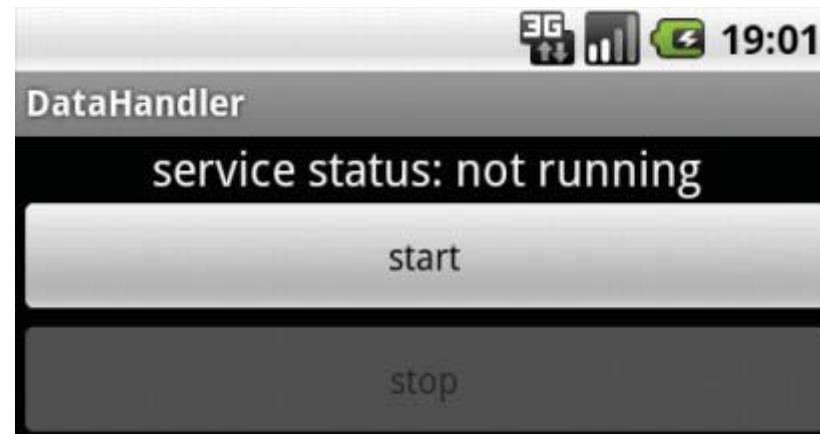
Handling events

```
startButton = (Button) findViewById(R.id.start_button);

// start button action
startButton.setOnClickListener(new Button.OnClickListener() {

    @Override
    public void onClick(View v) {
        ...
    });
});
```

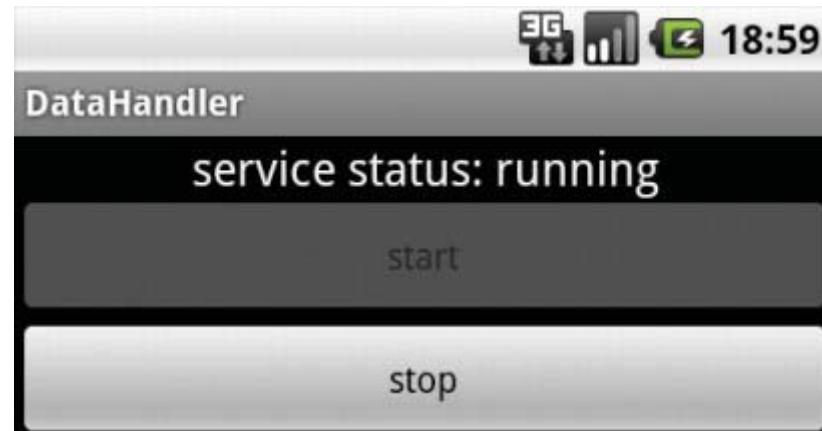
When the button is clicked, its `onClick()` method is executed



Handling events

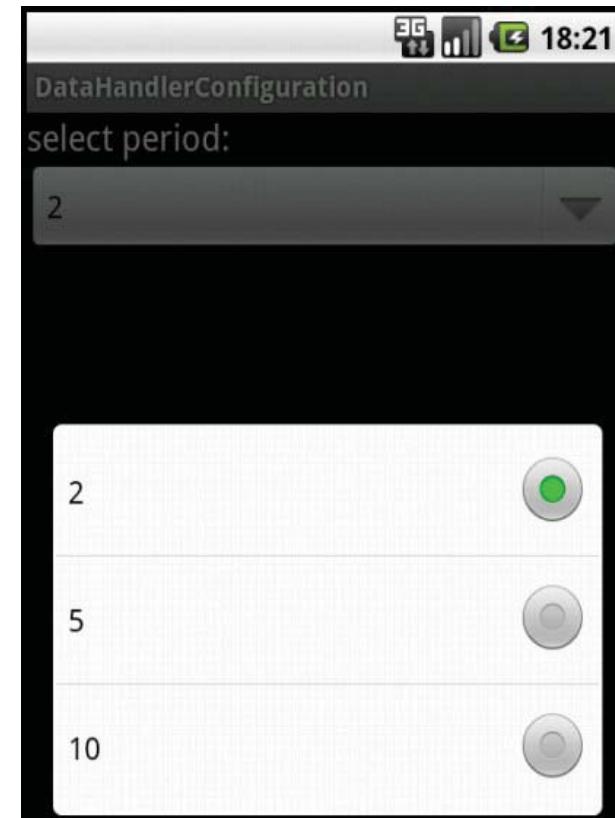
```
status = (TextView) findViewById(R.id.status_text);  
...  
  
@Override  
public void onClick(View v) {  
    startButton.setEnabled(false);  
    status.setText("service status: running");  
});
```

For example, it can update the **textview** and **disable itself**



Spinner

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="periods_array">
        <item >2</item>
        <item >5</item>
        <item >10</item>
    </string-array>
</resources>
```



Spinner

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="@string/period_text"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <Spinner
        android:id="@+id/period_spinner"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

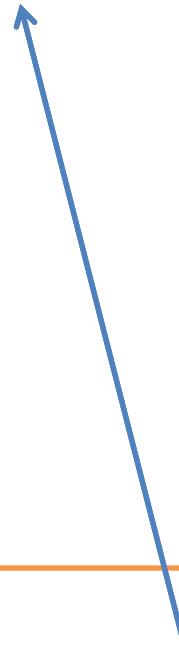
Spinner

```
public class ConfigActivity extends Activity
    implements OnItemSelectedListener {

    // tag for logging
    private String tag = "ConfigActivity";

    // service preferences
    private SharedPreferences preferences;
    private int period = 5000;
    private int item = 1;

    // UI
    private Spinner periodSpinner;
```



Note that the activity **implements** the
OnItemSelectedListener interface

Spinner

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.config);  
  
    // get preferences  
    preferences = getSharedPreferences("preferences", Context.MODE_PRIVATE);  
    period = preferences.getInt("period", period);  
    item = preferences.getInt("item" item);  
  
    // set UI elements  
    periodSpinner = (Spinner) findViewById(R.id.period_spinner);  
    ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,  
        R.array.periods_array, android.R.layout.simple_spinner_item);  
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);  
    periodSpinner.setAdapter(adapter);  
    periodSpinner.setOnItemSelectedListener(this);  
}
```

Spinner

```
@Override  
protected void onPause() {  
    super.onPause();  
    updatePreferences();  
}  
  
@Override  
protected void onResume() {  
    super.onResume();  
    periodSpinner.setSelection(item);  
}
```

The status has to be
saved and restored!

```
private void updatePreferences() {  
    Editor editor = preferences.edit();  
    editor.putInt("period", period);  
    editor.putInt("item", item);  
    editor.commit();  
}
```

Spinner

```
@Override  
public void onItemSelected(AdapterView<?> parent, View view,  
    int pos, long id) {  
  
    Object value = parent.getItemAtPosition(pos);  
    period = Integer.parseInt((String)value) * 1000;  
    item = pos;  
    updatePreferences();  
}  
  
@Override  
public void onNothingSelected(AdapterView<?> arg0) {  
}  
}
```

this is the
action to be
executed
when an item
is selected

Long tasks

✓ How to execute **long** tasks?

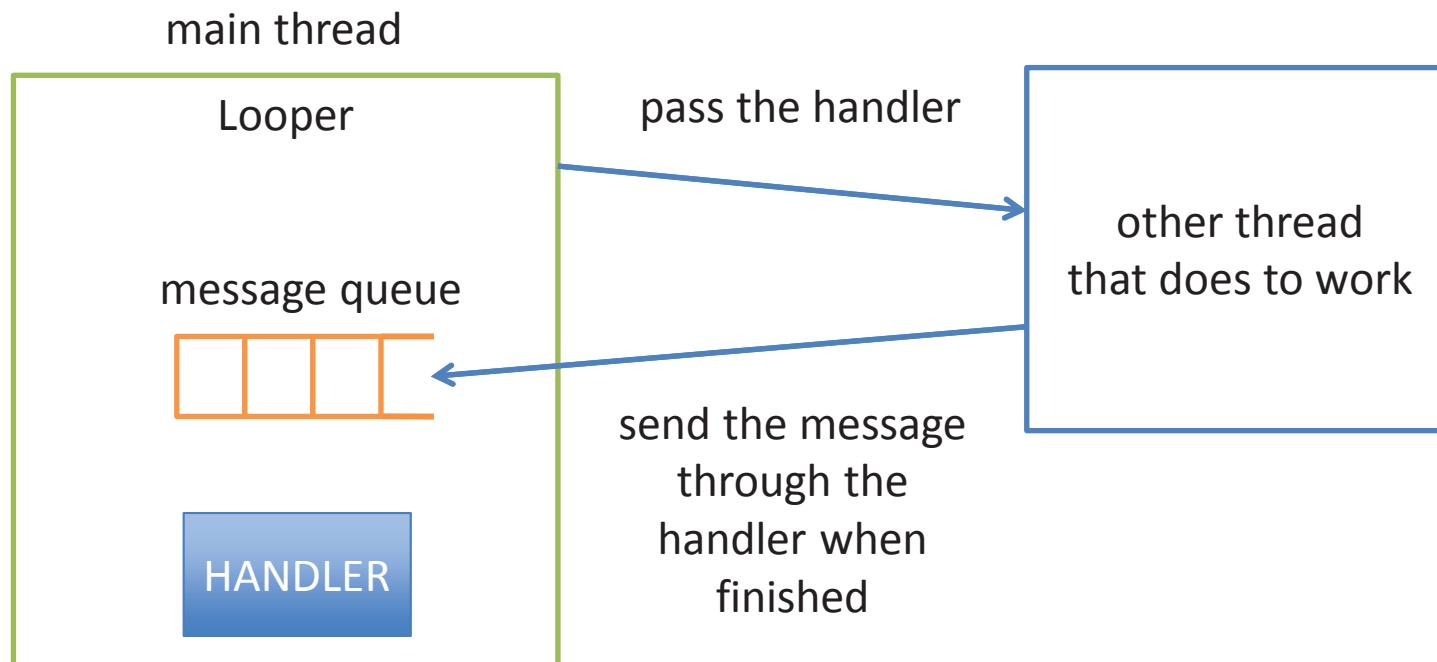
Handling events

- ✓ The handler **should not execute long tasks**
- ✓ However, it can start a **thread!**
- ✓ The thread idea can be **combined** with Handler and Loopers!

```
@Override  
public void onClick(View v) {  
    new Thread() {  
        public void run() {  
            ...  
            }.start();  
    });
```

Handler and loopers

- ✓ handler and Loopers allow to perform work in a **different thread** and inform the main UI thread when finished



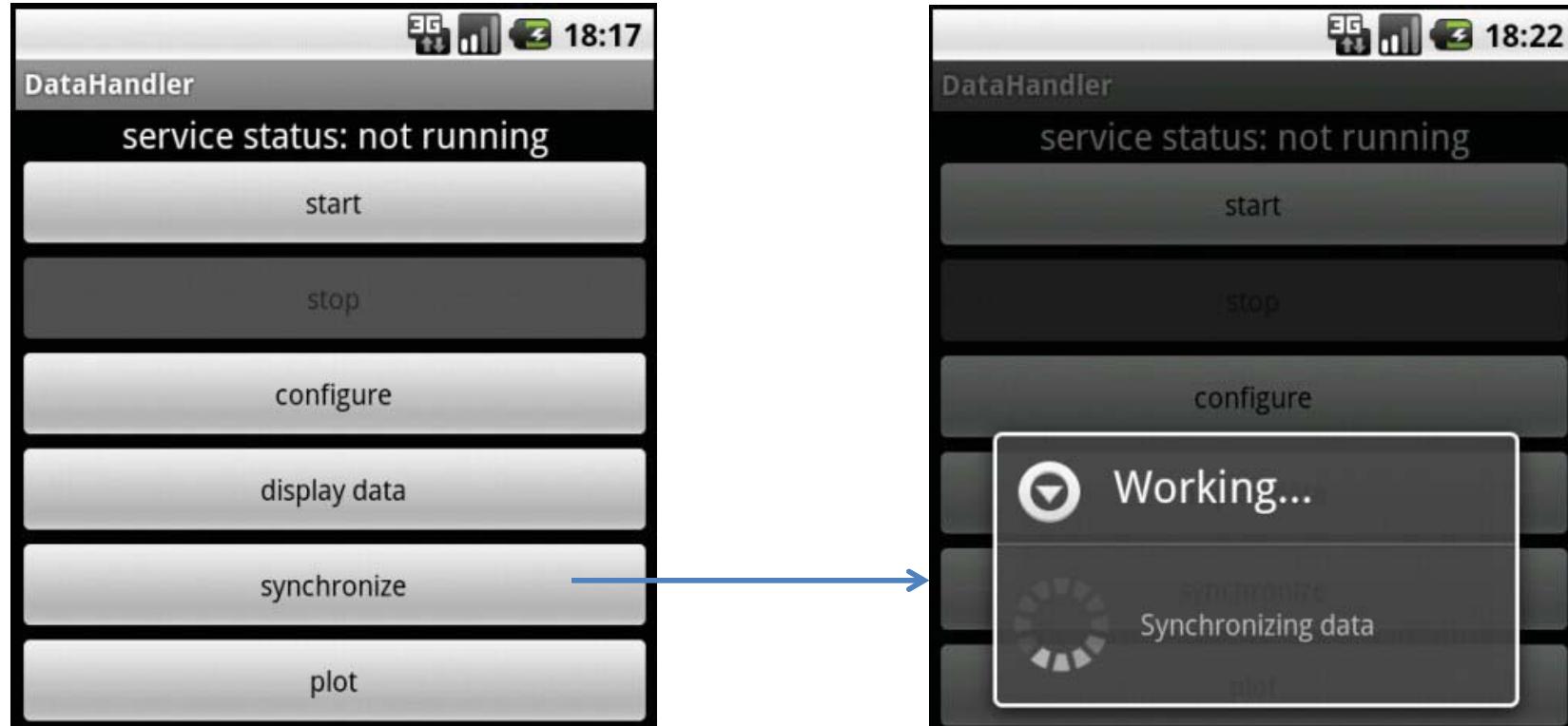
Handler and Loopers

```
Handler myHandler = new Handler() {  
    public void handleMessage (Message m) {  
        ...  
    }  
};  
  
new Thread() {  
    public void run() {  
  
        Message msg = myHandler.obtainMessage();  
        Bundle bundle = new Bundle();  
        bundle.putString("key", "value");  
        msg.setData(bundle);  
        myHandler.sendMessage(msg);  
    }  
}.start();
```

the main UI
thread **handles**
the message

the thread
sends the
message

An example



An example

```
syncButton.setOnClickListener(new Button.OnClickListener() {
    @Override
    public void onClick(View v) {
        progressDialog = ProgressDialog.show(v.getContext(),
                " Working...", " Synchronizing data", true, false);
        new Thread() {
            public void run() {
                String result = null;
                try {
                    sleep(1000); result = "OK";
                } catch (InterruptedException e) { e.printStackTrace(); }
                Message message = handler.obtainMessage();
                Bundle bundle = new Bundle();
                bundle.putString("result", result);
                message.setData(bundle);
                handler.sendMessage(message);
            }
        }.start();
    }
});
```

a progress dialog is created

a message is sent

An example

```
private ProgressDialog progressDialog;

public void handleMessage(final Message msg) {
    progressDialog.dismiss();

    String bundleResult = msg.getData().getString("result");
    if (bundleResult == null || !bundleResult.equals("OK")) {
        Log.e("ExampleTask", "Error on synchronize action");
        return;
    }
    Log.e("ExampleTask", "Synchronize action OK");
};
```

the progress dialog is closed

the data sent in the message is evaluated



Explore new perspectives

Android Applications Development

Part III: Data handling

ESTECO

Data handling

- ✓ in a mobile application,
data must be saved, not kept
in memory!
- ✓ Android supports database,
filesystem and preferences
- ✓ Data is usually shared through
a content provider

- ✓ self contained transactional SQL database for Android
- ✓ It will be covered in database lectures later on
- ✓ We will just have a look at some basic concepts in order to be able to define a Content Provider

SQL

Information is stored
in **tables**

there is usually a **key** which
identifies univocally each row

measurements

ID	TEMPERATURE	HUMIDITY	TIMESTAMP
1	13.34	75.02	7/8/12, 6:18:52 PM
2	12.01	76.75	7/8/12, 6:18:54 PM
3	12.92	76.29	7/8/12, 6:18:56 PM
4	14.78	72.84	7/8/12, 6:18:58 PM

in our example we are
going to use a single table

The table is created
with a CREATE
command

```
CREATE TABLE measurements (
    _ID INTEGER PRIMARY KEY,
    humidity FLOAT,
    temperature FLOAT,
    timestamp STRING);
```

measurements

ID	TEMPERATURE	HUMIDITY	TIMESTAMP
1	13.34	75.02	7/8/12, 6:18:52 PM
2	12.01	76.75	7/8/12, 6:18:54 PM
3	12.92	76.29	7/8/12, 6:18:56 PM
4	14.78	72.84	7/8/12, 6:18:58 PM

The table is removed
with the DROP TABLE
command

DROP TABLE measurements;

DELETE FROM measurements;

Rows are removed with
the DELETE command

measurements

ID	TEMPERATURE	HUMIDITY	TIMESTAMP
1	13.34	75.02	7/8/12, 6:18:52 PM
2	12.01	76.75	7/8/12, 6:18:54 PM
3	12.92	76.29	7/8/12, 6:18:56 PM
4	14.78	72.84	7/8/12, 6:18:58 PM

Data can be inserted with the command **INSERT**

```
INSERT INTO measurements (temperature, humidity, timestamp)  
values (15.74, 69.94, "7/8/12, 6:19:00 PM");
```

measurements

ID	TEMPERATURE	HUMIDITY	TIMESTAMP
1	13.34	75.02	7/8/12, 6:18:52 PM
2	12.01	76.75	7/8/12, 6:18:54 PM
3	12.92	76.29	7/8/12, 6:18:56 PM
4	14.78	72.84	7/8/12, 6:18:58 PM
5	15.74	69.94	7/8/12, 6:19:00 PM

SQLiteOpenHelper class

- ✓ Android provides a convenience class to **extend** in order to provide easy access to a database
- ✓ only a few methods have to be redefined

SQLiteOpenHelper class

```
public class DBHelper extends SQLiteOpenHelper {  
  
    private static final String DATABASE_NAME = "measures";  
    private static final int DATABASE_VERSION = 2;  
    private static final String TABLE_NAME = "measures";  
    private static final String TABLE_CREATE = "CREATE TABLE " + TABLE_NAME +  
    " (_ID INTEGER PRIMARY KEY, timestamp STRING, temperature FLOAT, humidity  
    FLOAT);"  
  
    public DBHelper(Context context) {  
        super(context, DATABASE_NAME, null, DATABASE_VERSION);  
    }  
  
    @Override  
    public void onCreate(SQLiteDatabase db) {  
        db.execSQL(TABLE_CREATE);  
    }  
}
```

SQLiteOpenHelper class

```
@Override  
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);  
    onCreate(db);  
}  
  
public void clean(SQLiteDatabase db) {  
    db.execSQL("DELETE FROM measures");  
}  
  
public SQLiteDatabase getDatabase() {  
    return this.getReadableDatabase();  
}  
}
```

This is a simple example, a better solution will be presented during database lectures!

Cursor

```
String[] projection = new String[] { "_ID", "timestamp",
                                    "temperature", "humidity" };
Cursor c = db.getReadableDatabase().query("measures", projection,
                                           null, null, null, null, null);
```

```
int numRows = c.getCount();

c.moveToFirst();

for (int i = 0; i < numRows; i++) {

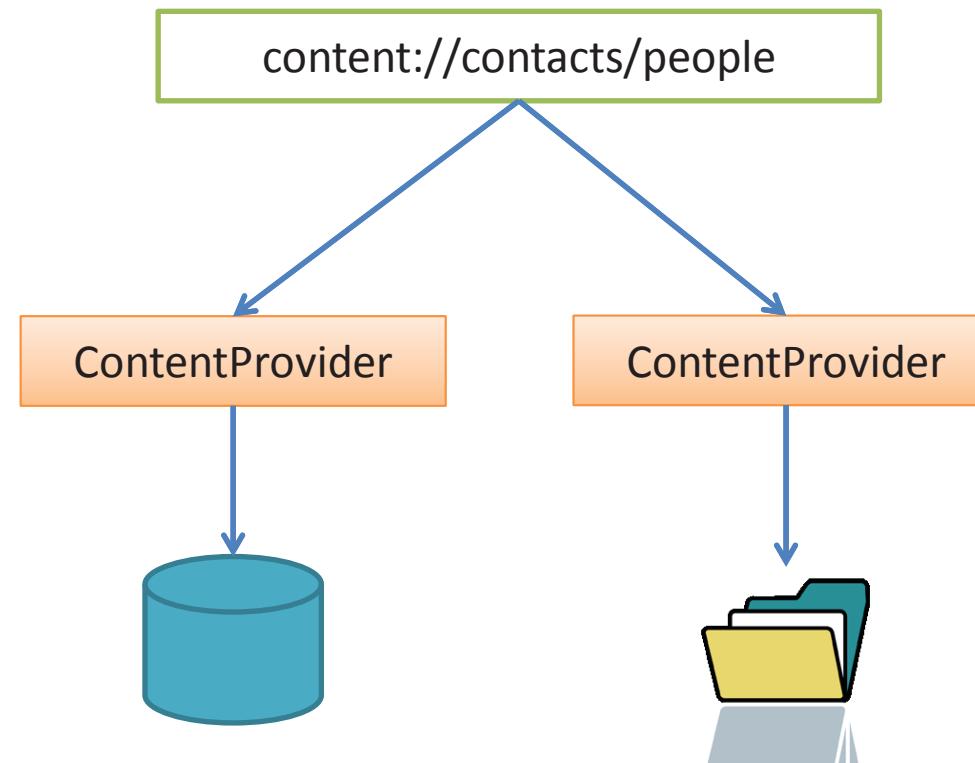
    int id = c.getInt(0);
    String timeStamp = c.getString(1);
    float temperature = c.getFloat(2);
    float humidity = c.getFloat(3);

    c.moveToNext();
}
```

A query returns a cursor object, which can be explored to get the rows

Content Provider

A Content Provider exposes data to other applications



Requests are performed by providing an **URI**

Content URI

The content provider is access through a content URI

`content://<authority>/<path>`

Examples:

`content://contacts/people/1`

`content://contacts/people/1/phone/3`

`content://my.workshop.datahandler/measurements`

`content://my.workshop.datahandler/measurements/3`

Content provider definition

In order to define a content provider, it is necessary to define:

1. A content **URI**
2. The **column names**
3. The **MIME types**
4. Methods for insert, query, delete, etc.
5. potentially notify Observers

Content provider example

```
public class DataHandlerProvider extends ContentProvider {  
  
    public static final String MIME_DIR_PREFIX = "vnd.android.cursor.dir";  
    public static final String MIME_ITEM_PREFIX = "vnd.android.cursor.item";  
    public static final String MIME_ITEM = "vnd.my.workshop";  
  
    public static final String MIME_TYPE_SINGLE = MIME_ITEM_PREFIX + "/" + MIME_ITEM;  
    public static final String MIME_TYPE_MULTIPLE = MIME_DIR_PREFIX + "/" + MIME_ITEM;  
  
    public static final String AUTHORITY = "my.workshop.datahandler";  
    public static final String PATH_SINGLE = "measures/#";  
    public static final String PATH_MULTIPLE = "measures";  
  
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY  
    + "/" + PATH_MULTIPLE);  
  
    ...  
}
```

Content provider example

```
private static final UriMatcher sURIMatcher = new
                                         UriMatcher(UriMatcher.NO_MATCH);

static {
    sURIMatcher.addURI(AUTHORITY, PATH_MULTIPLE, 1);
    sURIMatcher.addURI(AUTHORITY, PATH_SINGLE, 2);
}

@Override
public String getType(Uri uri) {
    switch (sURIMatcher.match(uri)) {
        case 1:
            return MIME_TYPE_MULTIPLE;
        case 2:
            return MIME_TYPE_SINGLE;
        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }
}
```

Arbitrary values defined to identify the query

Content provider example

```
private DBHelper db;  
  
@Override  
public boolean onCreate() {  
    db = new DBHelper(getContext());  
    return true;  
}
```

We are using the
DBHelper class
defined before

```
@Override  
public Cursor query(Uri uri, String[] projection, String selection,  
String[] selectionArgs, String sortOrder) {  
  
    Cursor c = db.getReadableDatabase().query("measures", projection, null,  
                                              null, null, null, null);  
  
    return c;  
}
```

Content provider example

```
@Override  
public int delete(Uri uri, String selection, String[] selectionArgs) {  
    throw new UnsupportedOperationException(); }  
  
@Override  
public Uri insert(Uri uri, ContentValues values) {  
    throw new UnsupportedOperationException(); }  
  
@Override  
public int update(Uri uri, ContentValues values, String selection,  
String[] selectionArgs) {  
    throw new UnsupportedOperationException(); }
```

We are currently
implementing **only** the query
operation

Accessing the Content Provider

A query to the Content Provider
returns a **cursor**

```
Uri providerUri =  
    Uri.parse("content://my.workshop.datahandler/measures");  
  
String[] projection = new String[] { "_ID", "timestamp", "temperature",  
                                    "humidity" };  
  
// perform query  
Cursor c = managedQuery(providerUri, projection, null, null, null);
```



managedQuery() is **deprecated**. Only used
here due to old Ronzi hardware!

Accessing the Content Provider

```
int numRows = c.getCount();

c.moveToFirst();

for (int i = 0; i < numRows; i++) {

    int id = c.getInt(0);
    String timeStamp = c.getString(1);
    float temperature = c.getFloat(2);
    float humidity = c.getFloat(3);

    c.moveToNext();
}
```

This code is used in the DataHandlerDisplay activity





Explore new perspectives

Android Applications Development

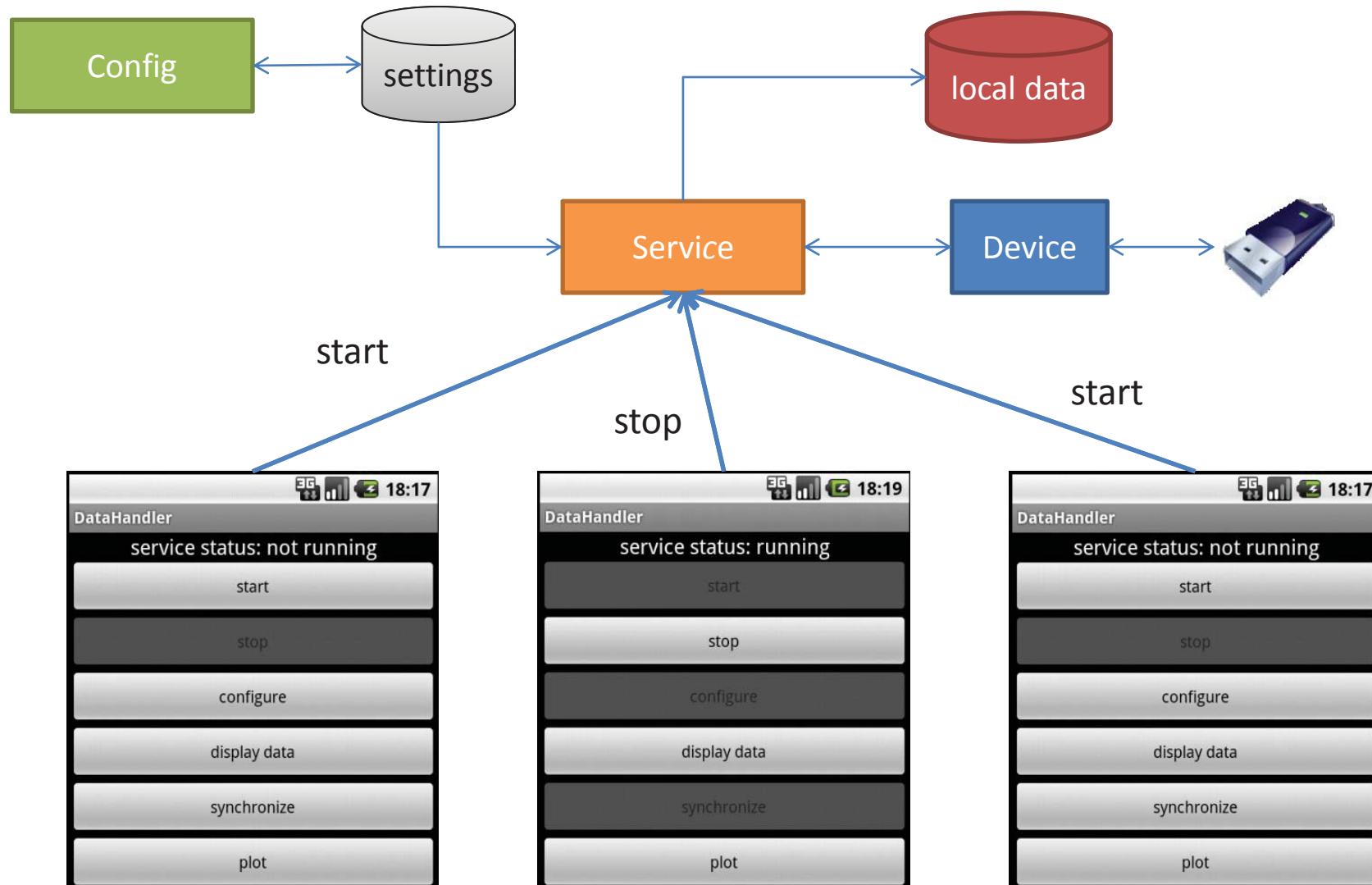
Part IV: Services

ESTECO

Services

- ✓ A service runs always in background as a not visible task
- ✓ Android can request it to be stopped if resources are required
- ✓ intended to implement long term operations

DataHandlerService



The DataHandlerService class

```
public class DataHandlerService extends Service
    implements Runnable {
    private Thread serviceThread;
    private Device device;

    // database helper
    private DBHelper db;

    // service preferences
    private SharedPreferences preferences;
    private String period_tag;
    private final int defaultPeriod = 5000;
    private int period = defaultPeriod;

    ...
}
```

Can be
executed as a
thread

Uses the DBHelper
class to access the
database

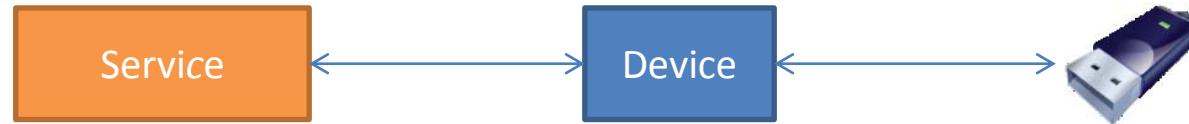
The onCreate() method

```
@Override  
public void onCreate() {  
    super.onCreate();  
  
    // create device instance  
    device = new Device();  
  
    // get preferences  
    preferences = getSharedPreferences("dataHandlerPreferences",  
                                         Context.MODE_PRIVATE);  
    period_tag = getResources().getString(R.string.period_tag);  
    period = preferences.getInt(period_tag, period);  
  
    // open database  
    db = new DBHelper(getApplicationContext());  
  
    // start service thread  
    serviceThread = new Thread(this);  
    serviceThread.start();  
}
```

Get the configuration parameters

The service thread will run the core of the service

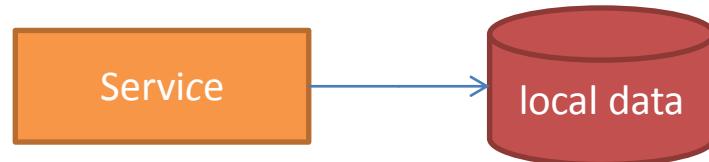
The service thread



```
@Override  
public void run() {  
  
    device.open();  
  
    while (true) {  
  
        // read data from device  
        device.read();  
  
        float temperature = device.getTemperature();  
        float humidity = device.getHumidity();  
        String timeStamp = device.getTimeStamp();  
  
        ....  
    }  
}
```

The service thread
queries the device
for data

The service thread



Data is written into
the database

```
....  
// store into database  
ContentValues values = new ContentValues();  
values.put("timestamp", timeStamp);  
values.put("temperature", temperature);  
values.put("humidity", humidity);  
db.getReadableDatabase().insert("measures", null, values);  
  
try {  
    Thread.sleep(period);  
} catch (Exception e) {  
    return;  
}  
}
```

Other methods

```
@Override  
public void onDestroy() {  
  
    // stop service thread  
    serviceThread.interrupt();  
    db.close();  
    super.onDestroy();  
}
```

```
@Override  
public IBinder onBind(Intent intent) {  
    return null;  
}
```

stops the thread
when service is
destroyed

The **binder** is used for inter-process
communication, we are not using it here



Explore new perspectives

Android Applications Development

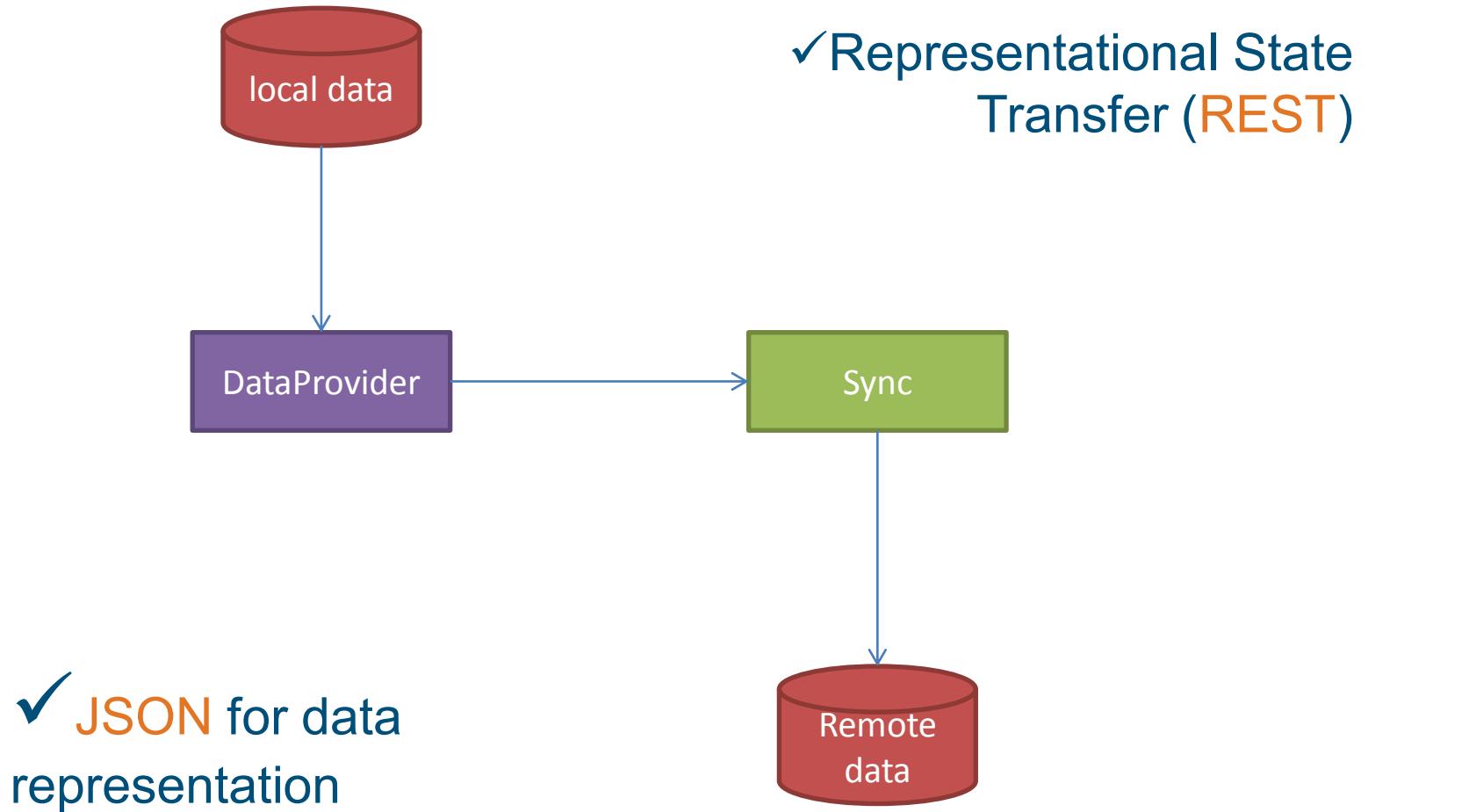
Part V: Web services

ESTECO

Web services

- ✓ Web services allows to expose an API through the network
- ✓ Many different implementations (SOAP, POX, REST, ...)
- ✓ More details on networking lectures later on!

Web services for our application

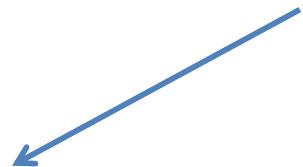


Synchronizer UI

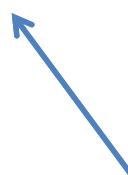


The Synchronizer thread

Synchronization **starts** when the user clicks on the corresponding button



```
syncButton.setOnClickListener(new Button.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
  
        // create progress dialog  
        progressDialog = ProgressDialog.show(v.getContext(),  
            " Working...", " Synchronizing data", true, false);  
    }  
}
```



A **progress dialog** is started

The Synchronizer thread

```
new Thread() {
    public void run() {
        String result = null;
        try {
            RemoteDB remoteDB =
                new RemoteDB(getApplicationContext());
            result = remoteDB.updateDB();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        Message message = handler.obtainMessage();
        Bundle bundle = new Bundle();
        bundle.putString("result", result);
        message.setData(bundle);
        handler.sendMessage(message);
    }
}.start();
});
```

Send data to
remote DB

Send the message
to the main UI
thread

The Synchronizer thread

```
// synchronize end event
public void handleMessage(final Message msg) {
    progressDialog.dismiss();                                ← Close the progress dialog

    String bundleResult = msg.getData().getString("result");
    Log.i(tag, "BUNDLE: " + bundleResult);
    if (bundleResult == null || !bundleResult.equals("OK")) {
        Log.e(tag, "Error on synchronize action");
        return;                                         ← Check message
    }

    DBHelper db = new DBHelper(getApplicationContext());
    db.clean(db.getReadableDatabase());
    db.close();
    Log.i(tag, "Synchronize finished!");
}
```

Close the progress dialog

Check message

Clear local DB

JSON

- ✓ Lightweight data interchange format
- ✓ Human readable
- ✓ Easy to parse for computing systems
- ✓ More information on <http://www.json.org>

JSON: an example

JSON object creation

```
JSONObject x = new JSONObject();

x.put("timestamp", " 8/6/12 6:37:42 AM ");
x.put("humidity", "49.49");
x.put("temperature", "15.15");
```

Internal representation

```
{
    "timestamp" : "8/6/12 6:37:42 AM",
    "humidity" : "49.29",
    "temperature" : "15.15"
}
```

Creation of the JSON array

```
public JSONArray getData() {  
  
    JSONArray a = new JSONArray();  
  
    Uri providerUri =  
        Uri.parse("content://my.workshop.datahandler/measures");  
    String[] projection = new String[] { "_ID", "timestamp",  
                                         "temperature", "humidity" };  
  
    Cursor c = context.getContentResolver().query(providerUri,  
                                                projection, null, null, null);  
  
    // process results  
    int numRows = c.getCount();  
    c.moveToFirst();  
  
    ....  
}
```

create
JSON
array

get data
through
content
provider

Creation of the JSON array

```
...
for (int i = 0; i < numRows; i++) {

    JSONObject json = new JSONObject();

    try {

        json.put("timestamp", c.getString(1));
        json.put("temperature", c.getString(2));
        json.put("humidity", c.getString(3));

    } catch (JSONException e) {
        e.printStackTrace(); return null;
    }

    a.put(json);
}
c.close();
}
```

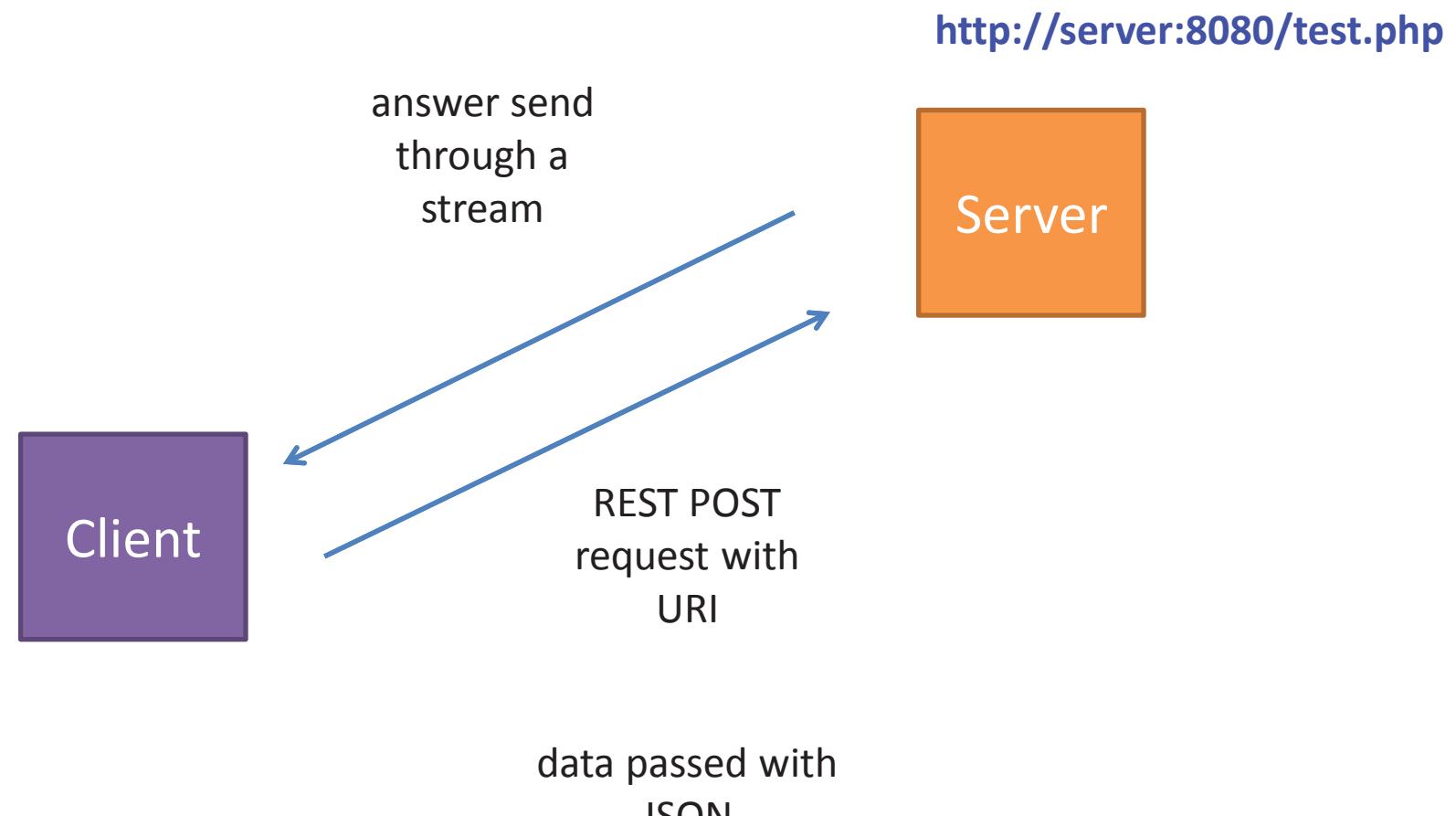
create JSON object

insert data from DB

insert into JSON array

The diagram illustrates the process of creating a JSON array from database data. It shows a code snippet within a blue-bordered box. Three arrows point from text annotations to specific parts of the code: 1) An arrow from 'create JSON object' points to the line 'JSONObject json = new JSONObject();'. 2) An arrow from 'insert data from DB' points to the three 'json.put()' statements. 3) An arrow from 'insert into JSON array' points to the line 'a.put(json);'.

REST



REST uses HTTP

REST web service request

```
public String updateDB() {  
  
    JSONArray a = getData();  
  
    String result;  
  
    try {  
  
        ...  
  
        result = ....  
  
    } catch (Exception e) {  
        Log.e("log_tag", "Error in http connection");  
        return "ERROR: " + e.toString();  
    }  
    return result;  
}
```

get JSON array

call the web
service and pass
the data to it
(on next slide)

handle result

REST web service request

```
HttpClient httpclient = new DefaultHttpClient();
HttpPost request = new HttpPost("http://server:8080/test.php");
request.setHeader("Content-type", "application/json");
request.setEntity(new StringEntity(a.toString()));
HttpResponse response = httpclient.execute(request);
HttpEntity entity = response.getEntity();
result = streamToString(new InputStreamReader(entity.getContent()));
```

create a client

Pass the JSON string

Execute the request

get the result through a stream

The PHP script on the server

```
<?php  
$json = file_get_contents('php://input');  
$obj = json_decode($json, true);  
  
if (count($obj) == 0) {  
    echo "OK";  
    return;  
}  
  
mysql_connect("localhost","user")  
or die( "Cannot connect");  
mysql_select_db("data")  
or die( "Unable to select database");  
  
....
```

access to JSON objects

connect to DB

The PHP script on the server

```
$query = 'insert into measures (timestamp, temperature, humidity) values ';
```

```
foreach ($obj as $value) {  
    $ts = $value['timestamp'];  
    $temp = $value['temperature'];  
    $h = $value['humidity'];  
    $query = $query.' "'.$ts.'",'.$temp.','.$h.')',';  
}  
$query = substr($query,0,-1);
```

```
$sql=mysql_query($query) or die( "Cannot execute query");
```

```
if(!$sql)  
    echo "\nError in query: ".mysql_error();  
else  
    echo "\nOK\n";
```

```
mysql_close();
```

```
?>
```

← prepare query

↑ execute query

↑ return exit status



Explore new perspectives

Android Applications Development

Part VI: Discussions on other topics

ESTECO

Intents

Explicit intents

```
Intent configIntent = new Intent(v.getContext(), ConfigActivity.class);  
startActivity(configIntent);
```

Implicit intents

```
Uri uri = Uri.parse("http://www.esteco.com");  
Intent intent = new Intent(Intent.ACTION_VIEW, uri);  
startActivity(intent);
```

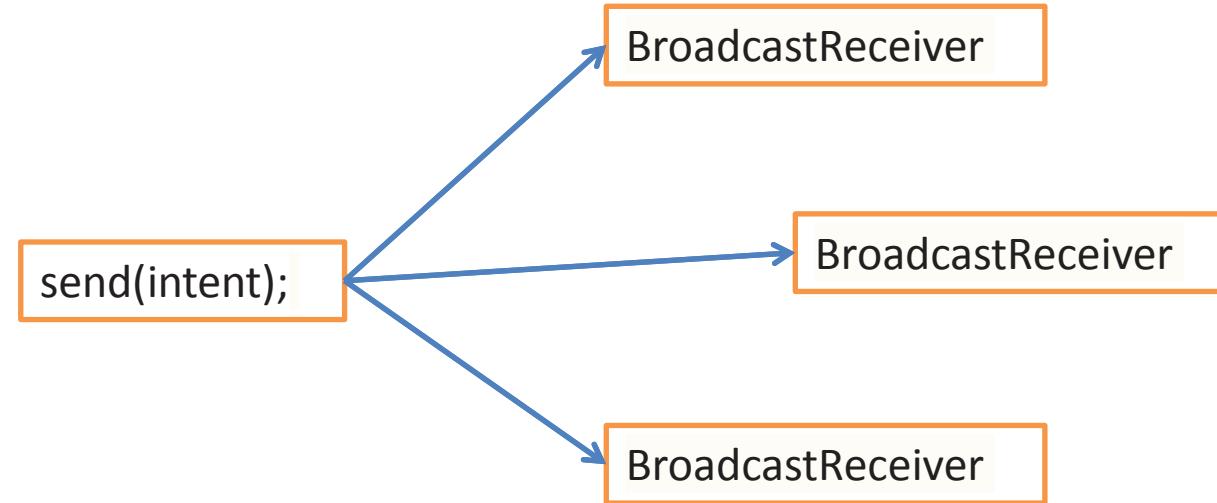
Permissions

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    ...
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

Requested permission for the application can be specified in the AndroidManifest.xml file

The user must accept them when installing the application

Broadcast receivers



Many predefined
intents that are
broadcasted by
Android

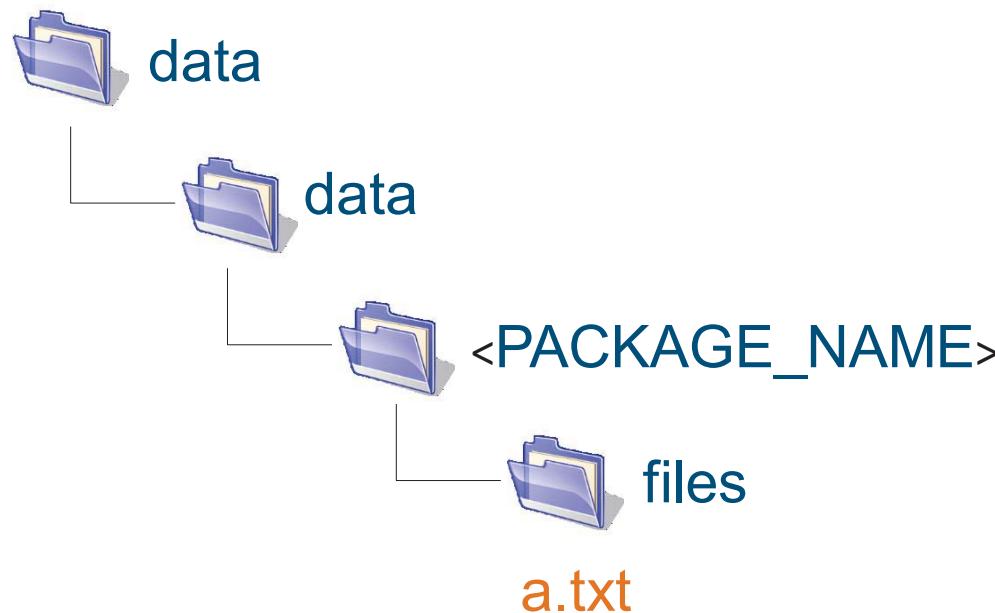
ACTION_TIME_TICK
ACTION_TIME_CHANGED
ACTION_TIMEZONE_CHANGED
ACTION_BOOT_COMPLETED
...

An intent is
broadcasted to
interested
receivers

File handling

Provides a stream to a file

```
FileOutputStream f;  
f = openFileOutput("a.txt", Context.MODE_PRIVATE);
```



Android market

Android Apps on Google Play <https://play.google.com/store>

Top Paid

1 Beautiful Widgets LEVELUP STUDIO	2 Where's My Water? DISNEY	3 Smart Tools SMART TOOLS CO.	4 Angry Birds Space Premium ROVIO MOBILE LTD.	5 Root Explorer (File Manager) SPEED SOFTWARE
---------------------------------------	-------------------------------	----------------------------------	--------------------------------------------------	--------------------------------------------------

All Top Paid Apps ›

Top Free

1 WhatsApp Messenger WHATSAPP INC.

FxR
MOONRABBIT
★★★★★ (2,430)

FANTASYxRUNNERS
MOONRABBIT
★★★★★ (54,169)

Calorie Counter by F...
FATSECRET
★★★★★ (153)

Zen Bound 2
SECRET EXIT
★★★★★ (9,755)

Radiant Defense
HEXAGE LTD
★★★★★ (9,755)

Fancy
THINGD
★★★★★ (3,426)

Bloons TD 4
DIGITAL GOLDFISH LTD
★★★★★ (3,008)

MUTANT ROADKILL
GLU MOBILE
★★★★★ (21,044)

Scan
SCAN, INC.
★★★★★ (6,191)

Babel Rising 3D
AMALTD
★★★★★ (107)

Slices for Twitter
ONELOUDER APPS
★★★★★ (2,656)

Puddle THD
NEKO ENTERTAINMENT
★★★★★ (79)

Google Currents
GOOGLE INC.
★★★★★ (16,544)

A more complete example

The screenshot shows a web browser displaying the Google Play Store page for the modeFRONTIER UM2012 app. The URL in the address bar is https://play.google.com/store/apps/details?id=com.esteco&feature=search_result#t=W251bGwsMSwyLDEsIm. The page includes a navigation bar with links like +You, Search, Images, Mail, Drive, Calendar, Sites, Groups, More, and a user account link kavka@esteco.com. A banner at the top says "Introducing Google Play" with a "LEARN MORE" link. The main content area features the Google Play logo and a search bar. On the left, there's a sidebar for "SHOP ANDROID APPS" and a "MY ANDROID" section. The main content area displays the app's title "modeFRONTIER UM2012" by Esteco SpA, its logo (yellow "um" and blue "12"), a 5-star rating with "(3)" reviews, and a large "INSTALL" button. A green callout box below the logo states "This app is compatible with your Wind Samsung GT-I9300." To the right, there are tabs for "OVERVIEW", "USER REVIEWS", "WHAT'S NEW", and "PERMISSIONS". The "OVERVIEW" tab is selected, showing a "Description" section with text about the modeFRONTIER 2012 Users' Meeting. Below the description are links to "Visit Developer's Website" and "Email Developer". Further down are sections for "App Screenshots" and "Users who viewed this also viewed". On the far right, there's a sidebar with social sharing options ("+1" and "Tweet"), developer information ("ABOUT THIS", "RATING", "UPDATED", "CURRENT VI", "REQUIRES A"), and system requirements ("2.1 and up").



Thank you!

