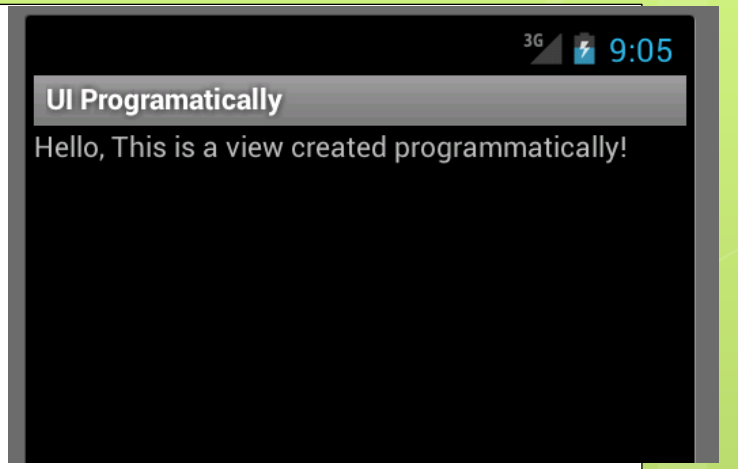
Basic UI elements: Defining Activity UI in the code

Marco Ronchetti
Università degli Studi di Trento

UI Programmatically




```
public class UIThroughCode extends Activity {
  LinearLayout lLayout;
  TextView tView;
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     lLayout = new LinearLayout(this);
     lLayout.setOrientation(LinearLayout.VERTICAL);
     lLayout.setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT,
                          LayoutParams.MATCH_PARENT));
     tView = new TextView(this);
     tView.setText("Hello, This is a view created programmatically! ")");
     tView.setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT,
                         LayoutParams.WRAP_CONTENT));
     lLayout.addView(tView);
     setContentView(lLayout);
  }
}
```

From http://saigeethamn.blogspot.it

Preferences

Marco Ronchetti
Università degli Studi di Trento

SharedPreferences

SharedPreferences allows to save and retrieve persistent key-value pairs of primitive data types. This data will persist across user sessions (even if your application is killed).

getSharedPreferences(String name, int mode)

A method of Contex

- Uses multiple preferences files identified by name, which you specify with the first parameter.

getPreferences()

A method of Activity

- Use this if you need only one preferences file for your Activity. This simply calls the underlying getSharedPreferences(String, int) method by passing in this activity's class name as the preferences name

SharedPreferences methods

boolean contains(String key)

Checks whether the preferences contains a preference.

T getT(String key, T defValue)

Value returned
If key does not exist

Retrieve a T value from the preferences where T={int, float, boolean, long, String, Set<String>}.

SharedPreferences.Editor edit()

All changes you make in an editor are batched, and not copied back to the original SharedPreferences until you call commit() or apply()

SharedPreferences.Editor methods

Void apply(), boolean commit()

Commit your preferences changes back

Editor putT(String key)

Stores a T value in the preferences where T={int, float, boolean, long, String, Set<String>}.

Editor remove(String key)

Mark in the editor that a preference value should be removed

Editor clear ()

Mark in the editor that all preference values should be removed

User Preferences

Shared preferences are not strictly for saving "user preferences," such as what ringtone a user has chosen.

For creating user preferences for your application, you should use PreferenceActivity, which provides an Activity framework for you to create user preferences, which will be automatically persisted (using shared preferences).

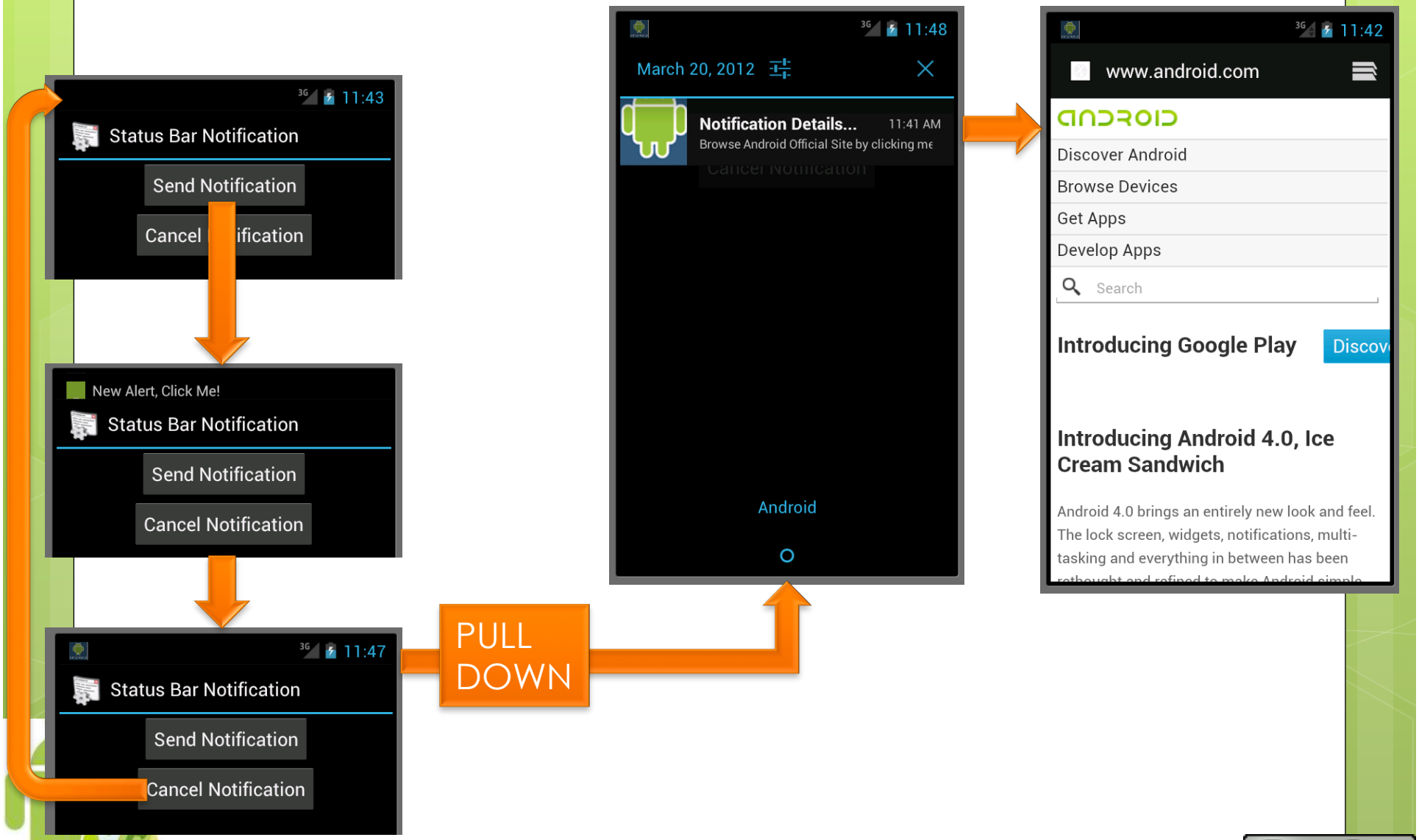
It is based on Fragments

8

Notification

Marco Ronchetti
Università degli Studi di Trento

Notification Bar



SimpleNotification

```
public class SimpleNotification extends Activity {
   private NotificationManager nm;
   private int SIMPLE_NOTIFICATION_ID;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      nm = (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
      final Notification notifyDetails = new Notification(
            R.drawable.android,"New Alert, Click Me!",
            System.currentTimeMillis());
      Button cancel = (Button)findViewById(R.id.cancelButton);
      cancel.setOnClickListener(new OnClickListener() {
         public void onClick(View v) {
            nm.cancel(SIMPLE_NOTIFICATION_ID);
         }});}
```

Adapted from http://saigeethamn.blogspot.it

SimpleNotification – part 2

```
Button start = (Button)findViewById(R.id.notifyButton);
start.setOnClickListener(new OnClickListener() {
   public void onClick(View v) {
      Context context = getApplicationContext();
      CharSequence contentTitle = "Notification Details...";
      CharSequence contentText = "Browse Android Site by clicking me";
      Intent notifyIntent = new Intent
         (android.content.Intent.ACTION_VIEW,
         Uri.parse("http://www.android.com"));
      PendingIntent intent =
      PendingIntent.getActivity(SimpleNotification.this, 0, notifyIntent,
           android.content.Intent.FLAG_ACTIVITY_NEW_TASK);
      notifyDetails.setLatestEventInfo(context, contentTitle,
          contentText, intent);
      nm.notify(SIMPLE_NOTIFICATION_ID, notifyDetails);
      }
});
}}
```

12

Sensors

Marco Ronchetti
Università degli Studi di Trento

Sensor categories

Motion sensors

- measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes.

Environmental sensors

- measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

Position sensors

- measure the physical position of a device. This category includes orientation sensors and magnetometers.

Basic code for managing sensors

```
public class SensorActivity extends Activity, implements SensorEventListener {
   private final SensorManager sm;
   private final Sensor sAcc;
   public SensorActivity() {
      sm= (SensorManager)getSystemService(SENSOR_SERVICE);
      sAcc= sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
   }
   protected void onPause() {
      super.onPause();
      sm.unregisterListener(this);
   }
   protected void onResume() {
      super.onResume();
      sm.registerListener(this, sAcc, SensorManager.SENSOR_DELAY_NORMAL);
   }
   public void onAccuracyChanged(Sensor sensor, int accuracy) { }
   public void onSensorChanged(SensorEvent event) { }
}
```

SensorManager

SensorManager sm=Context.getSystemService(SENSOR_SERVICE);

List<Sensor> getSensorList(int type)

- get the list of available sensors of a certain type. Sensor

Sensor getDefaultSensor(int type)

- Use this method to get the default sensor for a given type

void registerListener(SensorEventListener listener, Sensor sensor, int rate)

- Registers a SensorEventListener for the given sensor.

void unregisterListener(SensorEventListener listener, Sensor sensor)

- Unregisters a listener for the sensors with which it is registered.

void unregisterListener(SensorEventListener listener)

- Unregisters a listener for all sensors.

- Some methods for transforming data (Vector to matrix representation etc.)

Sensor types

int constants of the Sensor class describing sensor types:

TYPE_ACCELEROMETER

TYPE_ALL A constant describing all sensor types.

TYPE_AMBIENT_TEMPERATURE

TYPE_GRAVITY

TYPE_GYROSCOPE

TYPE_LIGHT

TYPE_LINEAR_ACCELERATION

TYPE_MAGNETIC_FIELD

TYPE_PRESSURE

TYPE_PROXIMITY

TYPE_RELATIVE_HUMIDITY

TYPE_ROTATION_VECTOR

Accelerometer

*"Sensor's values are in **meters/second^2** units. A sensor measures the acceleration applied to the device. For this reason, when the device is sitting on a table (and obviously not accelerating), the accelerometer reads a magnitude of **g** = 9.81 m/s^2. Similarly, when the device is in free-fall and therefore dangerously accelerating towards to ground at 9.81 m/s^2, its accelerometer reads a magnitude of 0 m/s^2."* (Android Developers – sensors)

Orientation sensor

"A compass is a navigational instrument for determining ***direction*** *relative to the Earth's* ***magnetic poles****. It consists of a magnetized pointer (usually marked on the North end) free to align itself with* ***Earth's magnetic field****."* (Compass EN Wiki)

In Android's terminology it is called **Orientation sensor**.

Gyroscope

*"A gyroscope is an instrument consisting of a rapidly **spinning wheel** so mounted as to use the tendency of such a wheel to maintain a fixed position in space, and to resist any force which tries to change it. The way it will move if a twisting force is applied depends on the extent and orientation of the force and the way the gyroscope is mounted. A free vertically spinning gyroscope remains vertical as the carrying vehicle tilts, so providing an artificial horizon. A horizontal gyroscope will maintain a certain bearing, and therefore indicate a vessel's heading as it turns. Modern gyroscopes (including those built-in in smartphones) no longer have a spinning wheel."* (Gyroscope Cambridge Encyclopedia)

*"All values are in **radians/second** and measure the rate of rotation around the **X**, **Y** and **Z** axis. The coordinate system is the same as is used for the acceleration sensor."* (Android Developers – sensors) Rotation is positive in the **counter-clockwise** direction.

Sensor class

float getMaximumRange()

- maximum range of the sensor in the sensor's unit.

int getMinDelay()

- minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes

String getName()

float getPower()

- the power in mA used by this sensor while in use

float getResolution()

- resolution of the sensor in the sensor's unit.

int getType()

String getVendor()

int getVersion()

```
SensorManager sm= (SensorManager)getSystemService(SENSOR_SERVICE);
List<Sensor> sensorList = sm.getSensorList(Sensor.TYPE_ALL);
StringBuilder sensorString = new StringBuilder("Sensors:\n");
for(int i=0; i<sensorList.size(); i++) {
    sensorString.append(sensorList.get(i).getName()).append(", \n");
}
```

HTC EVO 4G
BMA150 3-axis Accelerometer
AK8973 3-axis Magnetic field sensor
AK8973 Orientation sensor
CM3602 Proximity sensor
CM3602 Light sensor

Samsung Nexus-S
KR3DM 3-axis Accelerometer
AK8973 3-axis Magnetic field sensor
AK8973 Orientation sensor
GP2A Light sensor
GP2A Proximity sensor
K3G Gyroscope sensor
Gravity Sensor
Linear Acceleration Sensor
Rotation Vector Sensor

Interface SensorEventListener

abstract void onAccuracyChanged(Sensor sensor, int accuracy)

- Called when the accuracy of a sensor has changed.

abstract void onSensorChanged(SensorEvent event)

- Called when sensor values have changed.

Code examples

- http://www.vogella.com/articles/AndroidSensor/article.html accelerometer and compass examples

- http://developer.android.com/guide/topics/sensors/sensors_overview.html and following pages

Sensors limitation - 1

From Jim Steele

Using available sensors in the Android platform: current limitations and expected improvements

Comparing the sensors on these two phones demonstrates the **sensor fragmentation** now found in Android:

1) **Non-standard sensor availability**: The Nexus-S has a gyroscope (from ST Micro), but the EVO does not. In fact, most Android devices do not have a gyroscope. There is no standard availability of sensors across devices.

2) **Non-standard sensor capability**: The BMA150 is a Bosch Sensortec 10-bit accelerometer, and the KR3DM is a ST Micro 12-bit accelerometer (using a special part number). In fact, there is no standard capability requirement for sensors across devices to ensure consistentresolution, noise floor, or update rate.

Sensors limitation - 2

3) **Sensors not fully specified**: The AK8973 is an AKM magnetometer, which is only 8-bits. Analyzing this data stream shows it is low-pass filtered. This fact is not published on the phone or even the sensor datasheet. Many sensors have characteristics not specified such as bias changes, non-uniform gain, and skew (coupling between measurement axes). **Algorithms that use sensors without knowing these extra characteristics may produce incorrect information**.

4) **Broken virtual sensors**: The AKM sensor driver abstracts out an orientation virtual sensor which is derived from the combination of two sensors: the accelerometer and magnetometer. However, support for this virtual sensor was dropped early on, so the TYPE_ORIENTATION sensor is deprecated and the method **SensorManager.getOrientation()** should be used instead. Furthermore, the new virtual sensors introduced in Android 2.3 (Gingerbread) are not supported on all devices.

The sensor differences between just these two phones is substantial. So when a developer is faced with **writing apps utilizing sensors across as many devices as possible, it is a daunting task**.

Furthermore, **the Android platform is not optimized for real-time sensor data acquisition**.

What can you do with accelerometer and gyroscope?

http://www.starlino.com/imu_guide.html

A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications.

"This guide is intended to everyone interested in inertial MEMS (Micro-Electro-Mechanical Systems) sensors, in particular Accelerometers and Gyroscopes as well as combination IMU devices (Inertial Measurement Unit)."

- what does an accelerometer measure
- what does a gyroscope (aka gyro) measure
- how to convert analog-to-digital (ADC) readings that you get from these sensor to physical units (those would be g for accelerometer, deg/s for gyroscope)
- how to combine accelerometer and gyroscope readings in order to obtain accurate information about the inclination of your device relative to the ground plane

http://www.starlino.com/dcm_tutorial.html

DCM Tutorial – An Introduction to Orientation Kinematics

Emulator limits

The emulator does not emulate sensors, so what can you do without a physical device?

BUT…

There is an app that emulates many sensors, and that you can use as data provider!

SensorSimulator

OpenIntents SensorSimulator lets you simulate sensor events from accelerometer, compass, orientation, temperature, light, proximity, pressure, linear acceleration, gravity, gyroscope and rotation vector sensors.

Moreover, you can simulate your battery level and your gps position too, using a telnet connection.

It transmits the simulated sensor data to an Android emulator.

Also, it can record sensor data from an real Android device

See https://github.com/openintents/sensorsimulator

Broadcast receivers

Marco Ronchetti
Università degli Studi di Trento

Bradcast receiver

a component that responds to system-wide broadcast announcements.

Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.

Applications can initiate broadcasts—e.g. to let other applications know that some data has been downloaded to the device and is available for them to use.

Broadcast receivers don't display a user interface, but they can crate a status bar notification.

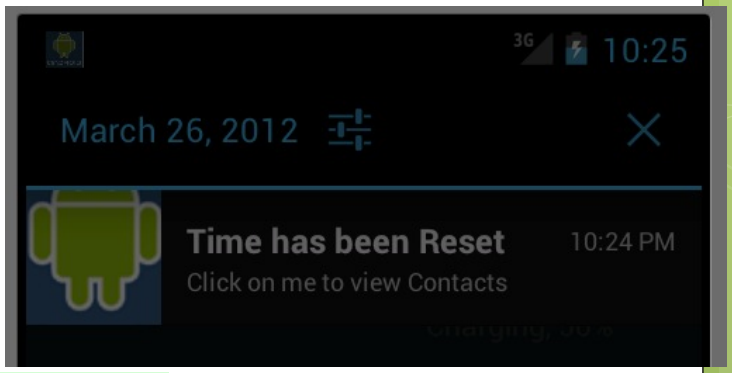
More commonly, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work e.g. it might initiate a service.

Broadcast receiver

```
public class MyBroadcastReceiver extends BroadcastReceiver {
    …
    public void onReceive(Context context, Intent intent) {
            …
    }
}
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="…I" android:versionCode="1"  android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <receiver android:name=".MyBroadcastReceiver">
            <intent-filter>
            <action android:name="android.intent.action.TIME_SET"/>
            </intent-filter>
        </receiver>
    </application>
    <uses-sdk android:minSdkVersion="13" />
</manifest>
```

```
>adb shell
# date +%s
1332793443
# date -s +%s 1332793443
time 1332793443 -> 1332793443.0
settimeofday failed Invalid argument
```



Adapted from saigeethamn.blogspot.it

Broadcast receiver

```
public class MyBroadcastReceiver extends BroadcastReceiver {
   private NotificationManager nm;
   private int SIMPLE_NOTFICATION_ID;

   @Override
   public void onReceive(Context context, Intent intent) {
    nm = (NotificationManager) context.getSystemService
                        (Context.NOTIFICATION_SERVICE);
    Notification n= new Notification(R.drawable.android,"Time Reset!",
                        System.currentTimeMillis());
    PendingIntent myIntent = PendingIntent.getActivity(context, 0,
            new Intent(Intent.ACTION_VIEW, People.CONTENT_URI), 0);
    n.setLatestEventInfo(context, "Time has been Reset",
                        "Click on me to view Contacts", myIntent);
     n|= Notification.FLAG_AUTO_CANCEL;
     n|= Notification.DEFAULT_SOUND;
     nm.notify(SIMPLE_NOTFICATION_ID, n);
     Log.i(getClass().getSimpleName(),"Sucessfully Changed Time");
   }
}
```

Adapted from saigeethamn.blogspot.it

Sending broadcast events

(in Context)

sendBroadcast (Intent intent, String receiverPermission)

Broadcast the given intent to all interested BroadcastReceivers, allowing an optional required permission to be enforced.

This call is asynchronous; it returns immediately, and you will continue executing while the receivers are run.

No results are propagated from receivers and receivers can not abort the broadcast.

Sending ordered broadcast events

(in Context)

sendOrderedBroadcast (Intent intent, String receiverPermission)

Broadcast the given intent to all interested BroadcastReceivers, delivering them one at a time to allow more preferred receivers to consume the broadcast before it is delivered to less preferred receivers.

This call is asynchronous; it returns immediately, and you will continue executing while the receivers are run.

Sending ordered broadcast events

(in Context)

sendOrderedBroadcast (…)

Version of sendBroadcast(Intent) that allows you to receive data back from the broadcast.

You supply your own BroadcastReceiver when calling: its onReceive(Context, Intent) method will be called with the result values collected from the other receivers.

The broadcast will be serialized in the same way as calling sendOrderedBroadcast(Intent, String).

LocalBroadcastManager

Helper to register for and send broadcasts of Intents to local objects within your process.

Advantages of Local vs Global B.M.:

- the data you are broadcasting will not leave your app
  - (you don't need to worry about leaking private data).
- it is not possible for other applications to send these broadcasts to your app
  - (you don't need to worry about having security holes)
- it is more efficient than sending a global broadcast through the system.

Fragments

Fragments

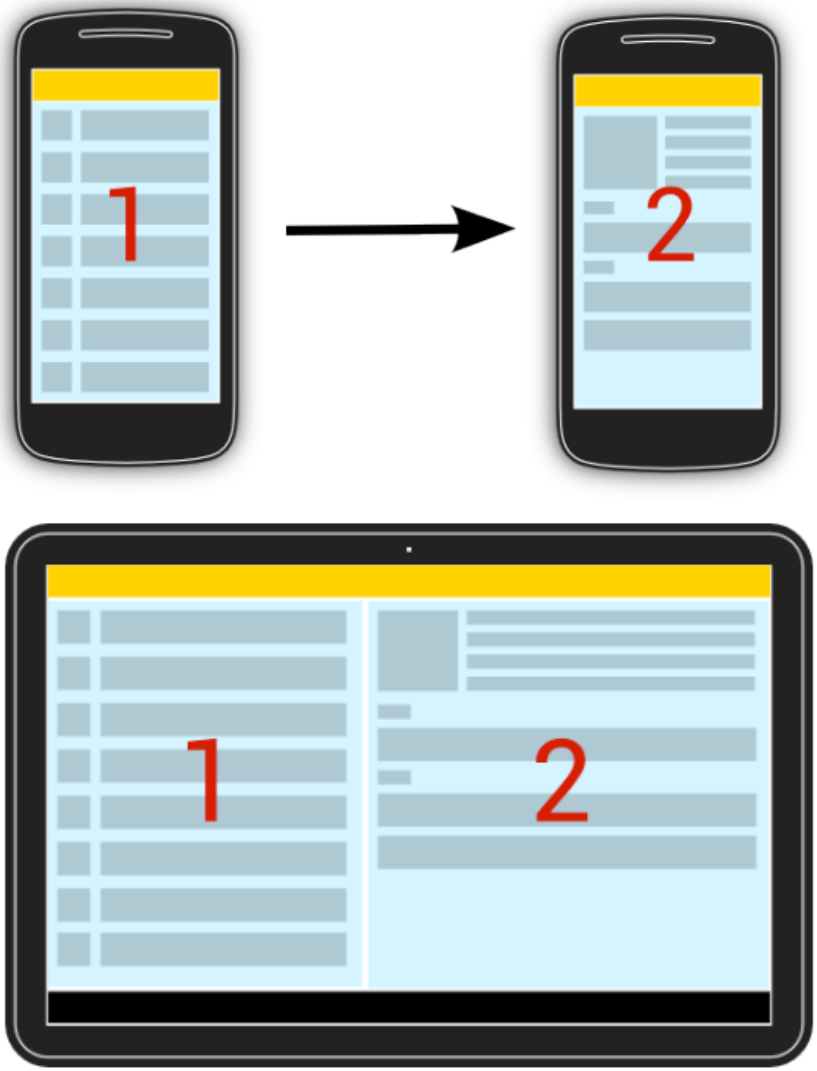
A fragment is a self-contained, modular section of an application's user interface and corresponding behavior that can be embedded within an activity.

Fragments can be assembled to create an activity during the application design phase, and added to, or removed from an activity during application runtime to create a dynamically changing user interface.

Fragments may only be used as part of an activity and cannot be instantiated as standalone application elements.

A fragment can be thought of as a functional "sub-activity" with its own lifecycle similar to that of a full activity.

Using fragments



Fragments lifecycle

Method	Description
onAttach()	The fragment instance is associated with an activity instance.The activity is not yet fully initialized
onCreate()	Fragment is created
onCreateView()	The fragment instance creates its view hierarchy. The inflated views become part of the view hierarchy of its containing activity.
onActivityCreated()	Activity and fragment instance have been created as well as thier view hierarchy. At this point, view can be accessed with the `findViewById()` method. example.
onResume()	Fragment becomes visible and active.
onPause()	Fragment is visibile but becomes not active anymore, e.g., if another activity is animating on top of the activity which contains the fragment.
onStop()	Fragment becomes not visible.

Defining a new fragment (from code)

To define a new fragment you either extend the android.app.Fragment class or one of its subclasses, for example, ListFragment, DialogFragment, PreferenceFragment or WebViewFragment.

Defining a new fragment (from code)

```
public class DetailFragment extends Fragment {
 @Override
 public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
    View view=inflater.inflate(
        R.layout.fragment_rssitem_detail,
        container, false);
   return view;
 }
 public void setText(String item) {
    TextView view = (TextView)
                getView().findViewById(R.id.detailsText);
    view.setText(item);
 }
}
```

XML-based fragments

```
<RelativeLayout xmlns:android="http://schemas.android.com/
apk/res/android" xmlns:tools="http://schemas.android.com/
tools" android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".FragmentDemoActivity" >
<fragment android:id="@+id/fragment_one"
android:name="com.example.myfragmentdemo.FragmentOne"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_alignParentLeft="true"
android:layout_centerVertical="true" tools:layout="@layout/
fragment_one_layout" />
</RelativeLayout>
```

Adding-removing fragments at runtime

The FragmentManager class and the FragmentTransaction class allow you to add, remove and replace fragments in the layout of your *activity*.

Fragments can be dynamically modified via transactions. To dynamically add fragments to an existing layout you typically define a container in the XML layout file in which you add a *Fragment*.

```
FragmentTransaction ft =
getFragmentManager().beginTransaction();
ft.replace(R.id.your_placehodler, new
YourFragment());
ft.commit();
```

A new *Fragment* will replace an existing *Fragment* that was previously added to the container.

Finding if a fragment is already part of your Activity

```
DetailFragment fragment = (DetailFragment)
        getFragmentManager().
        findFragmentById(R.id.detail_frag);


if (fragment==null) {
                // start new Activity
} else {
                fragment.update(...);
}
```

Communication: activity -> fragment

In order for an activity to communicate with a fragment, the activity must identify the fragment object via the ID assigned to it using the findViewById() method. Once this reference has been obtained, the activity can simply call the public methods of the fragment object.

Communication: fragment-> activity

Communicating in the other direction (from fragment to activity) is a little more complicated.

A) the fragment must define a listener interface, which is then implemented within the activity class.

```
public class MyFragment extends Fragment {
  AListener activityCallback;
  public interface AListener {
            public void someMethod(int par1, String par2);
  }
  …
```

Communication: fragment-> activity

B. the onAttach() method of the fragment class needs to be overridden and implemented. The method is passed a reference to the activity in which the fragment is contained. The method must store a local reference to this activity and verify that it implements the interface.

```
public void onAttach(Activity activity) {
  super.onAttach(activity);
  try { activityCallback = (AListener) activity;
  } catch (ClassCastException e) {
      throw new ClassCastException(
            activity.toString()
            + " must implement ToolbarListener");
} }
```

Communication: fragment-> activity

C. The next step is to call the callback method of the activity from within the fragment. When and how this happens is entirely dependent on the circumstances under which the activity needs to be contacted by the fragment. For the sake of an example, the following code calls the callback method on the activity when a button is clicked:

```
public void buttonClicked(View view) {
    activityCallback.someMethod(arg1, arg2);
}
```

Communication: fragment-> activity

All that remains is to modify the activity class so that it implements the ToolbarListener interface.

```
public class MyActivity extends
FragmentActivity implements
MyFragment.AListener {
 public void someMethod(String arg1, int arg2)
      {
      // Implement code for callback method
      }
.
.
}
```

Esempio

vedi
http://www.vogella.com/tutorials/AndroidFragments/article.html

sez. 10