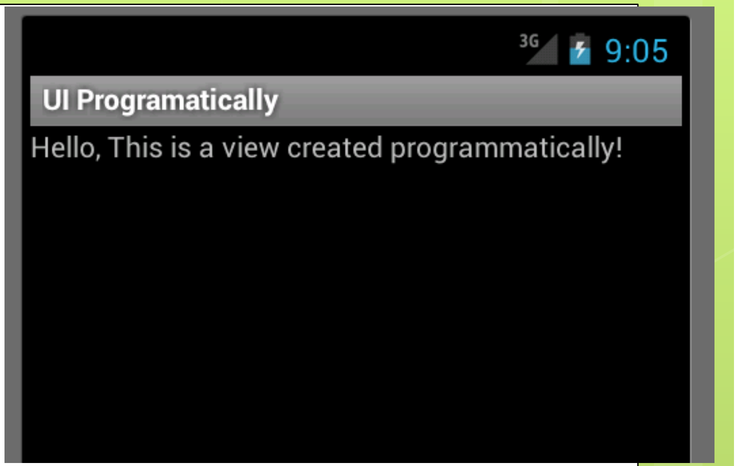
Basic UI elements: Defining Activity UI in the code

Marco Ronchetti
Università degli Studi di Trento

UI Programmatically



UI Programatically

Hello, This is a view created 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.

Value returned
If key does not exist

T getT(String key, T defValue)

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 (apply is asynchronous)

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.

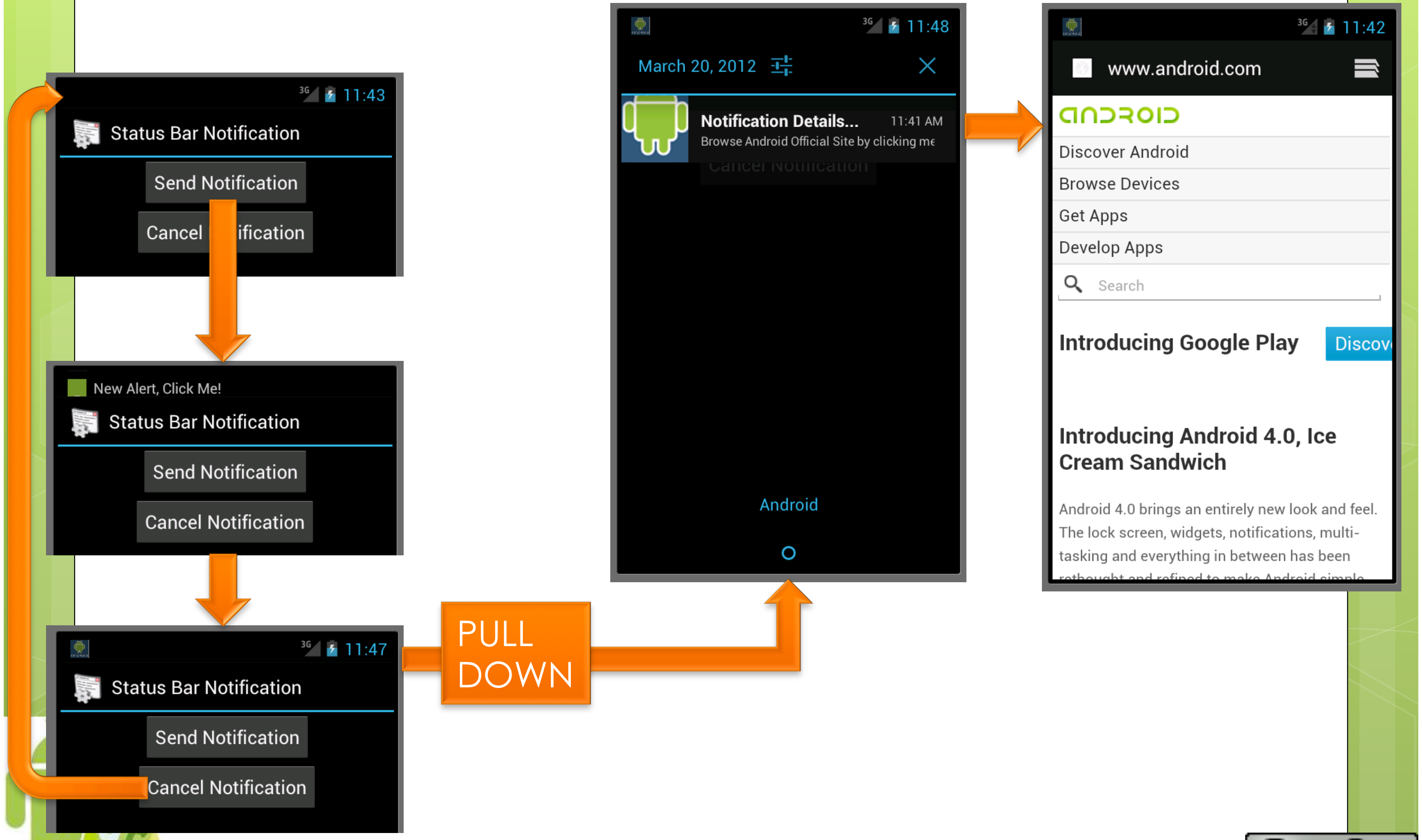
For creating user preferences for your application, you should use PreferenceActivity, which provides an Activity framework for you to create user preferences, which will be automatically persisted (using shared preferences).

It is based on Fragments

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);
              }
        });
    }}
```

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.

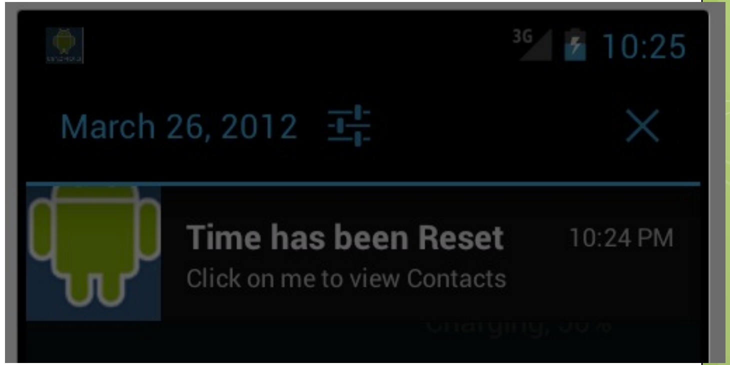
More commonly, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work e.g. it might initiate a service.

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="…l" 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
```



Time has been Reset 10:24 PM
Click on me to view Contacts

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");
    }
}
```

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.

Rooting a device

Marco Ronchetti
Università degli Studi di Trento

Rooting

The process of allowing users of Android devices to get root access. Varies widely by device, as it usually exploits a security weakness in the firmware shipped from the factory.

Goal:

- to overcome limitations imposed by that carriers and hardware manufacturers
- to alter or replace system applications and settings
- to run specialized apps that require administrator-level permissions
- to perform other operations that are otherwise inaccessible to a normal Android user.

The process of rooting

On the iphone: jailbreaking

e.g.: CyanogenMod

a replacement firmware. Offers several features, like:

- an OpenVPN client,
- a reboot menu,
- CPU overclocking and performance enhancements, app permissions management

Over 1.5 M installations

Is it legal?

On July 26, 2010, the U.S. Copyright office announced a new exemption making it officially legal to root a device and run unauthorized third-party applications, as well as the ability to unlock any cell phone for use on multiple carriers.

Industry reaction

- concern about improper functioning of devices running unofficial software and related support costs.
- offers features for which carriers would otherwise charge a premium

Technical obstacles have been introduced in many devices (e.g. locked bootloaders).

In 2011 an increasing number of devices shipped with unlocked or unlockable bootloaders.

The HTC case

“HTC is committed to listening to users and delivering customer satisfaction. We have heard your voice and starting now, we will allow our bootloader to be unlocked for 2011 models going forward.

It is our responsibility to caution you that not all claims resulting or caused by or from the unlocking of the bootloader may be covered under warranty.

We strongly suggest that you do not unlock the bootloader unless you are confident that you understand the risks involved.”

See e.g. http://htcdev.com/bootloader/