Broadcast Receivers

- What is a Broadcast Receiver?
- Creating a Broadcast Receiver
- Registering → statically/dynamically
- System wide broadcasts
- Creating and sending the Intent
- Ordered Broadcasts
- Local Broadcasts
- Pending Intents
- Manipulate Statically registered receivers
- Security

Broadcast Receivers

A BroadcastReceiver is an Android app **component** that responds to system-wide **broadcast** announcements.

Unlike Activities broadcast receivers **do not have any user interface** but may create a status bar notification, or start a service.

It receives an **Intent** object

«broadcast receivers are like dormant app components that can register for various system or application events (intents). Once any of those events occur the system notifies all the registered broadcast receivers and brings them up into action»

Creating a Broadcast Receiver

Extend the **abstract** class BroadcastReceiver → implement onReceive()

```
public class MyReceiver extends BroadcastReceiver {
   public MyReceiver() {
   }

   @Override
   public void onReceive(Context context, Intent intent) {
      // This method is called when this BroadcastReceiver receives an
      //Intent broadcast.
      Toast.makeText(context, "Action: " + intent.getAction(),
            Toast.LENGTH_SHORT).show();
   }
}
```

- The Intent may have extras and other info
- The context may be used to start other tasks (i.e. a service)

Registering the broadcast receiver

For the Broadcast receiving Intents, it **must be registered**

There are two ways of registering a broadcast:

- Statically → in the AndroidManifest.xml
- Dynamically → Through Java code

Registering the broadcast receiver

Statically

Adding a **<receiver>** tag in the AndroidManifest.xml

```
<receiver
   android:name="com.pycitup.pyc.MyReceiver"
   android:enabled="true"
   android:exported="true" >
   <intent-filter>
      <action android:name="es.pue.BroadcastReceiver" />
   </intent-filter>
</receiver>
```

It is matched by using an **IntentFilter**

Registering the broadcast receiver

Programatically

- Creating an IntentFilter object
- Instantiate the receiver
- Call Context.registerReceiver

```
IntentFilter filter = new IntentFilter("es.pue.BroadcastReceiver");

MyReceiver myReceiver = new MyReceiver();
registerReceiver(myReceiver, filter);
```

The receiver lasts as long as the **caller component is not destroyed**. Once the component that made the **registerReceiver** is destroyed, the sendBroadcast() will no have effect.

```
@Override
protected void onPause() {
   unregisterReceiver(mReceiver);
   super.onPause();
}
```

With statically registered, this problem does not happen

Registering the broadcast receiver

Statically

For system wide Broadcasts

Show up some notification

For boradcasts sent by other apps

Dynamically

For custom broadcasts that change the screen the user is using

For putting in the background some work

Some broadcasts may only be registered dynamically
To avoid wasting battery or resources:
Intent.ACTION_TIME_TICK

System wide broadcasts

Several system events are defined as **final static** fields in the **Intent** class. Other Android system classes also define events, e.g., the **TelephonyManager** defines events for the change of the phone state.

The following table lists a few important system events.

Event	Description
Intent.ACTION_BOOT_COMPLETED	Boot completed. Requires the `android.permission.RECEIVE_BOOT_COMPLETED` permission.
Intent.ACTION_POWER_CONNECTED	Power got connected to the device.
Intent.ACTION_POWER_DISCONNECTED	Power got disconnected to the device.
Intent.ACTION_BATTERY_LOW	Triggered on low battery. Typically used to reduce activities in your app which consume power.
Intent.ACTION_BATTERY_OKAY	Battery status good again.

Creating and sending the broadcast (Implicit)

- Create an **Intent**
- Set the **action** in the Intent as the one our Reciever listens to
- If needed, add extras
- Call sendBroadcast()

```
Intent intent = new Intent();
intent.addFlags(Intent.FLAG_INCLUDE_STOPPED_PACKAGES);
intent.setAction("com.pycitup.BroadcastReceiver");
intent.putExtra("Foo", "Bar");
sendBroadcast(intent);
```

The flag indicates to add stopped packages (Default is the opposite)

The packages must have
been started once by the user
And not sopped in the app manager

Creating and sending the broadcast (Explicit)

- The BroadCast may also target an **explicit reciever class**

```
Intent intent = new Intent(this, MyReceiver.class);
sendBroadcast(intent);
```

The reciever must be registered!

Ordered Broadcasts

Normal Broadcasts

- What we have seen until now
- Asynchronous → are received by the receivers in an unpredictable order and even in parallel
- Efficent, but does not allow abort or change the broadcast in a chain

Ordered Broadcasts

- Sent by Context.sendOrderedBroadcast()
- They are delivered to one receiver at a time, in a queue
- The order may be controlled with **android:priority** in the receiver tag
- You can receive and pass data in the chain by using **getResultExtras**
- You can abort the chain by using **abortBroadcast()**

Ordered Broadcasts

```
public class MyReceiver extends BroadcastReceiver {

   private String TAG = MyReceiver.class.getSimpleName();

   public MyReceiver() {
   }

   @Override
   public void onReceive(Context context, Intent intent) {
      Bundle results = getResultExtras(true);
      String hierarchy = results.getString("hierarchy");

      results.putString("hierarchy", hierarchy + "->" + TAG);

      Log.d(TAG, "MyReceiver");
   }
}
```


Ordered Broadcasts

```
<receiver
   android:name="es.pue.MyReceiver"
   >
   <intent-filter android:priority="1">
      <action android:name="es.pue.BroadcastReceiver" />
   </intent-filter>
</receiver>
<receiver
   android:name="es.pue.MySecondReceiver"
   >
   <intent-filter android:priority="2">
      <action android:name="es.pue.BroadcastReceiver" />
   </intent-filter>
</receiver>
```


Local Broadcasts

Is a helper class to simplify broadcasts within an app
→ More secure
→ More efficient
It must be used when registering and when sending:

Don't forget unregistering it!

```
LocalBroadcastManager.getInstance(this).registerReceiver(new B    adcastReceiver() {
   @Override
   public void onReceive(Context context, Intent intent) {
      String messag
      Log.d("LocalB
   }
}, new IntentFilter
```

```
@Override
protected void onPause() {
   LocalBroadcastManager.getInstance(this).
            unregisterReceiver(mReceiver);
   super.onPause();
}
```

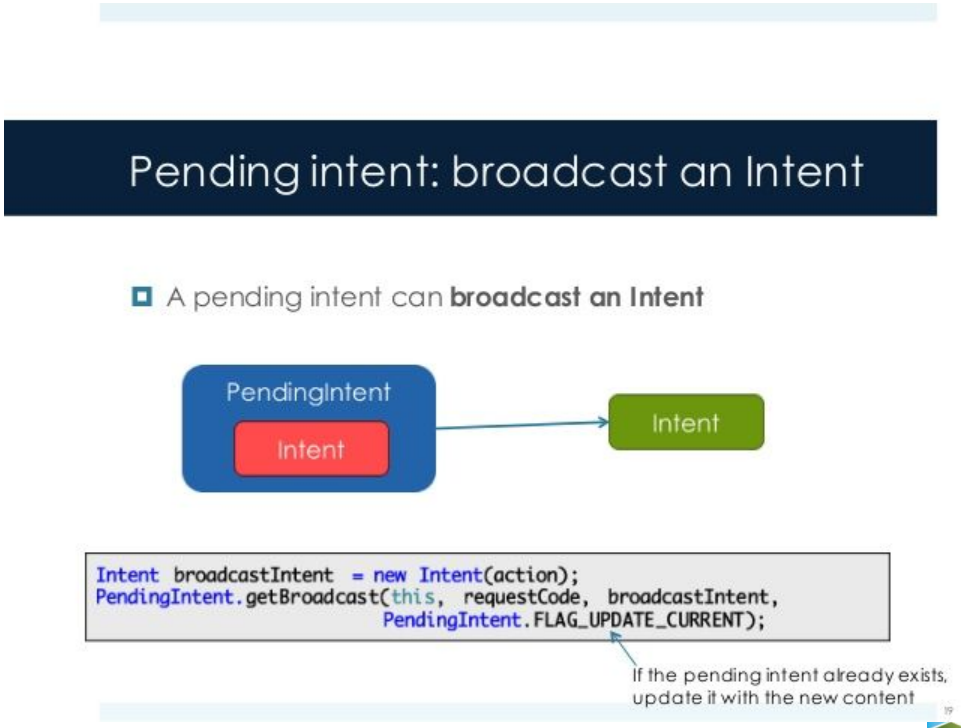
```
// Send

Intent intent = new Intent("my-custom-event");
intent.putExtra("foo", "bar");
LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
```

Pending Intents

Intent that his execution is **delayed,** a wrapper for the Intent

The Intent may be executed by other app but with the permissions of yours

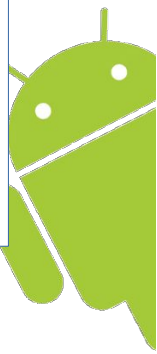
With **PendingIntent.getBroadcast()** we can get a Pending intent that will in the future, send the Broadcast as if it was our app



Pending Intents

Example:

```
// In the MainActivity.onCreate()

int seconds = 3;
// Create an intent that will be wrapped in PendingIntent
Intent intent = new Intent(this, MyReceiver.class);

// Create the pending intent and wrap our intent
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 1, intent, 0);

// Get the alarm manager service and schedule it to go off after 3s
AlarmManager alarmManager =
            (AlarmManager) getSystemService(ALARM_SERVICE);
alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis()
+ (seconds * 1000), pendingIntent);

Toast.makeText(this, "Alarm set in " + seconds + " seconds",
      Toast.LENGTH_LONG).show();
```



Unregister statically registered receivers

You can always disable them (or enable again when required) using the **PackageManager** saving your battery from draining:

```
PackageManager pm = getPackageManager();
ComponentName compName =
      new ComponentName(getApplicationContext(), YourReceiver.class);

pm.setComponentEnabledSetting(compName,
      PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
      PackageManager.DONT_KILL_APP);
```

Supported states are:

COMPONENT_ENABLED_STATE_DEFAULT – As specified in the manifest file.
COMPONENT_ENABLED_STATE_DISABLED
COMPONENT_ENABLED_STATE_ENABLED

DONT_KILL_APP prevents from killing the app → a component's state change can make the app's behaviour unpredictable.

Permissions and security

Listening for some broadcasts may require specific permissions

For example Intent.ACTION_BOOT_COMPLETED requires the RECEIVE_BOOT_COMPLETED permission.

→ **Other apps can send events to your receivers!**

You should set **custom permissions** with a combination of **uses-permission** tag, **permission** tag, sendBroadcast() with the **permission** string and also the **android:permission** attribute on the receiver tag.