Q1. What is a broadcast receiver?

**Broadcast Receivers**simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

Q2. Creating and registering a broadcast receiver?

There are following two important steps to make BroadcastReceiver works for the system broadcasted intents −

* Creating the Broadcast Receiver.
* Registering Broadcast Receiver

There is one additional steps in case you are going to implement your custom intents then you will have to create and broadcast those intents.

Creating the Broadcast Receiver

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and overriding the onReceive() method where each message is received as a **Intent** object parameter.

public class MyReceiver extends BroadcastReceiver {

@Override

public void onReceive(Context context, Intent intent) {

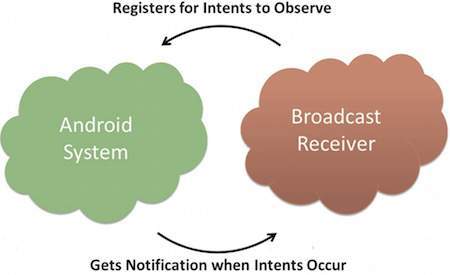
Toast.makeText(context, "Intent Detected.", Toast.LENGTH\_LONG).show();

}

}

Registering Broadcast Receiver

An application listens for specific broadcast intents by registering a broadcast receiver in *AndroidManifest.xml* file. Consider we are going to register *MyReceiver* for system generated event ACTION\_BOOT\_COMPLETED which is fired by the system once the Android system has completed the boot process.



**BROADCAST-RECEIVER**

<application

android:icon="@drawable/ic\_launcher"

android:label="@string/app\_name"

android:theme="@style/AppTheme" >

<receiver android:name="MyReceiver">

<intent-filter>

<action android:name="android.intent.action.BOOT\_COMPLETED">

</action>

</intent-filter>

</receiver>

</application>

Now whenever your Android device gets booted, it will be intercepted by BroadcastReceiver *MyReceiver* and implemented logic inside *onReceive()* will be executed.

Q3. Types of broadcast?

**Normal and Ordered Broadcasts**

**Broadly there are two major classes of broadcasts:**

* **Normal Broadcasts: These are sent with [Context.sendBroadcast()](http://developer.android.com/reference/android/content/Context.html" \l "sendBroadcast(android.content.Intent)) as you just saw sometime back. They’re completely asynchronous, i.e., the broadcasts events/intents are received by all the receivers in an asynchronous fashion. The receivers are run in an undefined order, often at the same time. It’s efficient but receivers cannot use results from other receivers or abort the entire chain of execution at a certain level.**
* **Ordered Broadcasts: These are sent with [Context.sendOrderedBroadcast()](http://developer.android.com/reference/android/content/Context.html" \l "sendOrderedBroadcast(android.content.Intent, java.lang.String)). They’re delivered to one receiver at a time. The order can be controlled with [android:priority](http://developer.android.com/reference/android/R.styleable.html" \l "AndroidManifestIntentFilter_priority) attribute of the matching intent-filter. Receivers with same priority will be executed in a random order. As each receiver executes, it can trasmit the result to the next one or even abort the entire broadcast chain so that no other receivers receive the broadcast intent and are executed.**

**We discussed normal broadcasts previously, hence in this section we’ll go through ordered broadcasts. Let’s straight away get into some code. First we’ll define two broadcast receivers:**

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | // MySecondReceiver.java    public class MySecondReceiver extends BroadcastReceiver {        private String TAG = MySecondReceiver.class.getSimpleName();        public MySecondReceiver() {      }        @Override      public void onReceive(Context context, Intent intent) {          Bundle results = getResultExtras(true);          results.putString("hierarchy", TAG);            Log.d(TAG, "MySecondReceiver");      }  } | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | // MyReceiver.java    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");      }  } |

**Pretty simple to understand, just put this also in your manifest to register the receivers:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <receiver      android:name="com.pycitup.pyc.MyReceiver"      >      <intent-filter android:priority="1">          <action android:name="com.pycitup.BroadcastReceiver" />      </intent-filter>  </receiver>  <receiver      android:name="com.pycitup.pyc.MySecondReceiver"      >      <intent-filter android:priority="2">          <action android:name="com.pycitup.BroadcastReceiver" />      </intent-filter>  </receiver> |

**Note how we set the priority using android:priority. The second one with higher priority will get to receive the broadcast intent first and then the first one with a priority of 1 will get to receive. By default, the priority is 0 on intent filters and must be between**[**IntentFilter.SYSTEM\_LOW\_PRIORITY**](http://developer.android.com/reference/android/content/IntentFilter.html#SYSTEM_LOW_PRIORITY)**and**[**IntentFilter.SYSTEM\_HIGH\_PRIORITY**](http://developer.android.com/reference/android/content/IntentFilter.html#SYSTEM_HIGH_PRIORITY)**.**

**Finally here’s the snippet I’ll put in my MainActivity.onCreate():**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | // In MainActivity.onCreate()    IntentFilter filter = new IntentFilter("com.pycitup.BroadcastReceiver");  // filter.setPriority(10); // could do this if you want to    registerReceiver(new BroadcastReceiver() {      @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, "Anonymous class broadcast receiver");      }  }, filter);    Intent intent = new Intent("com.pycitup.BroadcastReceiver");    sendOrderedBroadcast(intent, null, new BroadcastReceiver() {      @Override      public void onReceive(Context context, Intent intent) {          Bundle results = getResultExtras(true);          String hierarchy = results.getString("hierarchy");            System.out.println(hierarchy);          Log.d(TAG, "Final Receiver");      }  }, null, Activity.RESULT\_OK, null, null); |

**Create an intent filter first and dynamically registered a new receiver with it. Then finally we created the broadcast intent and fired it via [sendOrderedBroadcast()](http://developer.android.com/reference/android/content/Context.html" \l "sendOrderedBroadcast(android.content.Intent, java.lang.String, android.content.BroadcastReceiver, android.os.Handler, int, java.lang.String, android.os.Bundle)). The third receiver argument that you see is the result receiver, so once all the receivers are executed in order, it’ll get to receive the event finally.**

**In each and every onReceive() I kept on getting the result extra data as set by the previous receiver using [getResultExtras()](http://developer.android.com/reference/android/content/BroadcastReceiver.html" \l "getResultExtras(boolean)) and then set a new string representing the trail of reception or the hierarchy. Finally the dump should seem like this:**

|  |  |
| --- | --- |
| 1  2  3  4  5 | ﹕ MySecondReceiver  ﹕ MyReceiver  ﹕ Anonymous class broadcast receiver  ﹕ MySecondReceiver->MyReceiver->MainActivity  ﹕ Final Receiver |

**You can always abort the broadcast by calling [abortBroadcast()](http://developer.android.com/reference/android/content/BroadcastReceiver.html" \l "abortBroadcast()) in any receiver’s onReceive() preventing further receivers from receiving the event. Although the final receiver’s onReceive(), set in the sendOrderedBroadcast() will still be called.**

Q4. Types of broadcast events?

There are several system generated events defined as final static fields in the **Intent** class. The following table lists a few important system events.

|  |  |
| --- | --- |
| **Sr.No** | **Event Constant & Description** |
| 1 | **android.intent.action.BATTERY\_CHANGED**  Sticky broadcast containing the charging state, level, and other information about the battery. |
| 2 | **android.intent.action.BATTERY\_LOW**  Indicates low battery condition on the device. |
| 3 | **android.intent.action.BATTERY\_OKAY**  Indicates the battery is now okay after being low. |
| 4 | **android.intent.action.BOOT\_COMPLETED**  This is broadcast once, after the system has finished booting. |
| 5 | **android.intent.action.BUG\_REPORT**  Show activity for reporting a bug. |
| 6 | **android.intent.action.CALL**  Perform a call to someone specified by the data. |
| 7 | **android.intent.action.CALL\_BUTTON**  The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call. |
| 8 | **android.intent.action.DATE\_CHANGED**  The date has changed. |
| 9 | **android.intent.action.REBOOT**  Have the device reboot. |

## Broadcasting Custom Intents

If you want your application itself should generate and send custom intents then you will have to create and send those intents by using the *sendBroadcast()* method inside your activity class. If you use the *sendStickyBroadcast(Intent)* method, the Intent is **sticky**, meaning the *Intent* you are sending stays around after the broadcast is complete.

public void broadcastIntent(View view) {

Intent intent = new Intent();

intent.setAction("com.tutorialspoint.CUSTOM\_INTENT");

sendBroadcast(intent);

}

This intent *com.tutorialspoint.CUSTOM\_INTENT* can also be registered in similar way as we have regsitered system generated intent.

<application

android:icon="@drawable/ic\_launcher"

android:label="@string/app\_name"

android:theme="@style/AppTheme" >

<receiver android:name="MyReceiver">

<intent-filter>

<action android:name="com.tutorialspoint.CUSTOM\_INTENT">

</action>

</intent-filter>

</receiver>

</application>

## Example

This example will explain you how to create *BroadcastReceiver* to intercept custom intent. Once you are familiar with custom intent, then you can program your application to intercept system generated intents. So let's follow the following steps to modify the Android application we created in *Hello World Example* chapter −

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use Android studio to create an Android application and name it as *My Application* under a package *com.example.tutorialspoint7.myapplication* as explained in the *Hello World Example* chapter. |
| 2 | Modify main activity file *MainActivity.java* to add *broadcastIntent()* method. |
| 3 | Create a new java file called *MyReceiver.java* under the package *com.example.tutorialspoint7.myapplication* to define a BroadcastReceiver. |
| 4 | An application can handle one or more custom and system intents without any restrictions. Every intent you want to intercept must be registered in your *AndroidManifest.xml* file using <receiver.../> tag |
| 5 | Modify the default content of *res/layout/activity\_main.xml* file to include a button to broadcast intent. |
| 6 | No need to modify the string file, Android studio take care of string.xml file. |
| 7 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

Following is the content of the modified main activity file **MainActivity.java**. This file can include each of the fundamental life cycle methods. We have added *broadcastIntent()* method to broadcast a custom intent.

package com.example.tutorialspoint7.myapplication;

import android.app.Activity;

import android.content.Intent;

import android.os.Bundle;

import android.view.View;

public class MainActivity extends Activity {

/\*\* Called when the activity is first created. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

// broadcast a custom intent.

public void broadcastIntent(View view){

Intent intent = new Intent();

intent.setAction("com.tutorialspoint.CUSTOM\_INTENT"); sendBroadcast(intent);

}

}

Following is the content of **MyReceiver.java**:

package com.example.tutorialspoint7.myapplication;

import android.content.BroadcastReceiver;

import android.content.Context;

import android.content.Intent;

import android.widget.Toast;

/\*\*

\* Created by TutorialsPoint7 on 8/23/2016.

\*/

public class MyReceiver extends BroadcastReceiver{

@Override

public void onReceive(Context context, Intent intent) {

Toast.makeText(context, "Intent Detected.", Toast.LENGTH\_LONG).show();

}

}

Following will the modified content of *AndroidManifest.xml* file. Here we have added <receiver.../> tag to include our service:

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.tutorialspoint7.myapplication">

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".MainActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

<receiver android:name="MyReceiver">

<intent-filter>

<action android:name="com.tutorialspoint.CUSTOM\_INTENT">

</action>

</intent-filter>

</receiver>

</application>

</manifest>

Following will be the content of **res/layout/activity\_main.xml** file to include a button to broadcast our custom intent −

<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"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

android:paddingBottom="@dimen/activity\_vertical\_margin"

tools:context=".MainActivity">

<TextView

android:id="@+id/textView1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Example of Broadcast"

android:layout\_alignParentTop="true"

android:layout\_centerHorizontal="true"

android:textSize="30dp" />

<TextView

android:id="@+id/textView2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Tutorials point "

android:textColor="#ff87ff09"

android:textSize="30dp"

android:layout\_above="@+id/imageButton"

android:layout\_centerHorizontal="true"

android:layout\_marginBottom="40dp" />

<ImageButton

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/imageButton"

android:src="@drawable/abc"

android:layout\_centerVertical="true"

android:layout\_centerHorizontal="true" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/button2"

android:text="Broadcast Intent"

android:onClick="broadcastIntent"

android:layout\_below="@+id/imageButton"

android:layout\_centerHorizontal="true" />

</RelativeLayout>

Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment set-up. To run the app from Android studio, open one of your project's activity files and click Run Eclipse Run Icon icon from the tool bar. Android Studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window −



Now to broadcast our custom intent, let's click on **Broadcast Intent** button, this will broadcast our custom intent *"com.tutorialspoint.CUSTOM\_INTENT"* which will be intercepted by our registered BroadcastReceiver i.e. MyReceiver and as per our implemented logic a toast will appear on the bottom of the the simulator as follows −



You can try implementing other BroadcastReceiver to intercept system generated intents like system boot up, date changed, low battery etc.