# BroadCast Recevier

Broadcast in android is the system-wide events that can occur when the device starts, when a message is received on the device or when incoming calls are received, or when a device goes to airplane mode, etc. Broadcast Receivers are used to respond to these system-wide events. Broadcast Receivers allow us to register for the system and application events, and when that event happens, then the register receivers get notified. There are mainly two types of Broadcast Receivers:

* **Static Broadcast Receivers:**These types of Receivers are declared in the manifest file and works even if the app is closed.
* **Dynamic Broadcast Receivers:**These types of receivers work only if the app is active or minimized.

import android.content.Intent

import android.content.IntentFilter

import android.os.Bundle

import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

// register the receiver in the main activity in order

// to receive updates of broadcasts events if they occur

lateinit var receiver: AirplaneModeChangeReceiver

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

receiver = AirplaneModeChangeReceiver()

// Intent Filter is useful to determine which apps wants to receive

// which intents,since here we want to respond to change of

// airplane mode

IntentFilter(Intent.ACTION\_AIRPLANE\_MODE\_CHANGED).also {

// registering the receiver

// it parameter which is passed in registerReceiver() function

// is the intent filter that we have just created

registerReceiver(receiver, it)

}

}

// since AirplaneModeChangeReceiver class holds a instance of Context

// and that context is actually the activity context in which

// the receiver has been created

override fun onStop() {

super.onStop()

unregisterReceiver(receiver)

}

}

import android.content.BroadcastReceiver

import android.content.Context

import android.content.Intent

import android.widget.Toast

// AirplaneModeChangeReceiver class extending BroadcastReceiver class

class AirplaneModeChangeReceiver : BroadcastReceiver() {

// this function will be executed when the user changes his

// airplane mode

override fun onReceive(context: Context?, intent: Intent?) {

// intent contains the information about the broadcast

// in our case broadcast is change of airplane mode

// if getBooleanExtra contains null value,it will directly return back

val isAirplaneModeEnabled = intent?.getBooleanExtra("state", false) ?: return

// checking whether airplane mode is enabled or not

if (isAirplaneModeEnabled) {

// showing the toast message if airplane mode is enabled

Toast.makeText(context, "Airplane Mode Enabled", Toast.LENGTH\_LONG).show()

} else {

// showing the toast message if airplane mode is disabled

Toast.makeText(context, "Airplane Mode Disabled", Toast.LENGTH\_LONG).show()

}

}

}

Since from API Level 26, most of the broadcast can only be caught by the dynamic receiver, so we have implemented dynamic receivers in our sample project given below. There are some static fields defined in the Intent class which can be used to broadcast different events. We have taken a change of airplane mode as a broadcast event, but there are many events for which broadcast register can be used. Following are some of the important system-wide generated intents:-

| **Intent** | **Description Of Event** |
| --- | --- |
| android.intent.action.BATTERY\_LOW : | Indicates low battery condition on the device. |
| android.intent.action.BOOT\_COMPLETED | This is broadcast once after the system has finished booting |
| android.intent.action.CALL | To perform a call to someone specified by the data |
| android.intent.action.DATE\_CHANGED | Indicates that the date has changed |
| android.intent.action.REBOOT | Indicates that the device has been a reboot |
| android.net.conn.CONNECTIVITY\_CHANGE | The mobile network or wifi connection is changed(or reset) |
| android.intent.ACTION\_AIRPLANE\_MODE\_CHANGED | This indicates that airplane mode has been switched on or off. |
|  |  |

Basically event triggering is intent is broadcasted . You can create broadcast first and then register either in manifest or by code. Issue in manifest, it keeps on listening things.

**Scenario**: Consider an app with a media player that needs to listen for headphone plug/unplug events. Instead of declaring a receiver in the manifest, you could dynamically register the receiver when the media player activity is active and unregister it when the activity is paused or stopped. This way, the app only listens for these events when it actually needs to respond to them.

**Use of LocalBroadcastManager**:

* To further secure your broadcasts, you can use LocalBroadcastManager, which confines the broadcast to within your app. This ensures that **only** your app can send and receive the broadcast, eliminating the risk of external interference.

java

Copy code

// Using LocalBroadcastManager

LocalBroadcastManager.getInstance(this).registerReceiver(logoutReceiver, filter);

LocalBroadcastManager.getInstance(this).sendBroadcast(new Intent("com.mybank.LOGOUT\_US

# Difference between Listener and Broadcast Receiver?

In Android development, both \*\*Listeners\*\* and \*\*Broadcast Receivers\*\* are used for responding to events, but they serve different purposes and are used in different contexts. Here’s a breakdown of the differences:

### 1. \*\*Purpose and Use Case\*\*

- \*\*Listener\*\*:

- \*\*Purpose\*\*: Listeners are used to handle specific events within a single component or a small scope, usually tied to user interactions or specific events in the lifecycle of a component.

- \*\*Use Case\*\*: Listeners are often used for UI events (e.g., button clicks, touch events) or other component-specific events (e.g., detecting changes in text input, responding to network requests).

- \*\*Example\*\*:

- \*\*OnClickListener\*\*: Used to handle click events on a view, such as a button.

- \*\*TextWatcher\*\*: Used to respond to changes in text within an `EditText`.

- \*\*Broadcast Receiver\*\*:

- \*\*Purpose\*\*: Broadcast Receivers are used to respond to system-wide events or application-wide events that are broadcast to multiple components, often across different processes or applications.

- \*\*Use Case\*\*: Broadcast Receivers listen for broadcasts sent by the Android system or other applications, such as when the device is booted, when the network state changes, or when a custom broadcast is sent within an app.

- \*\*Example\*\*:

- \*\*ACTION\_BOOT\_COMPLETED\*\*: To execute code when the device finishes booting.

- \*\*ConnectivityManager.CONNECTIVITY\_ACTION\*\*: To respond to changes in network connectivity.

### 2. \*\*Scope\*\*

- \*\*Listener\*\*:

- \*\*Scope\*\*: Listeners are typically tied to the lifecycle of a specific component or UI element. When that component is destroyed, the listener is also destroyed.

- \*\*Example\*\*: A `View.OnClickListener` is attached to a specific button, and when that button is no longer in use, the listener is removed.

- \*\*Broadcast Receiver\*\*:

- \*\*Scope\*\*: Broadcast Receivers can have a broader scope, listening to events that may originate from the entire system or multiple components within the app. They can be registered statically in the AndroidManifest.xml or dynamically in code.

- \*\*Example\*\*: A Broadcast Receiver registered in the manifest might listen for system events even when the app is not actively running.

### 3. \*\*Registration\*\*

- \*\*Listener\*\*:

- \*\*Registration\*\*: Listeners are usually registered in code, often in an Activity or Fragment, and are directly tied to the component that generates the event.

- \*\*Example\*\*:

```java

Button button = findViewById(R.id.my\_button);

button.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

// Handle the click event

}

});

```

- \*\*Broadcast Receiver\*\*:

- \*\*Registration\*\*: Broadcast Receivers can be registered either in the AndroidManifest.xml (static registration) or dynamically in code (dynamic registration). Static registration means the receiver is always ready to receive broadcasts, even when the app is not running.

- \*\*Example (Static)\*\*:

```xml

<receiver android:name=".MyBroadcastReceiver">

<intent-filter>

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

</intent-filter>

</receiver>

```

- \*\*Example (Dynamic)\*\*:

```java

BroadcastReceiver myReceiver = new MyBroadcastReceiver();

IntentFilter filter = new IntentFilter(Intent.ACTION\_AIRPLANE\_MODE\_CHANGED);

registerReceiver(myReceiver, filter);

```

### 4. \*\*Lifespan\*\*

- \*\*Listener\*\*:

- \*\*Lifespan\*\*: A listener’s lifespan is usually tied to the lifecycle of the component it’s attached to (e.g., Activity, Fragment). Once the component is destroyed, the listener is also cleaned up.

- \*\*Implication\*\*: This makes listeners suitable for short-lived interactions and ensures that they don't outlive the component, avoiding memory leaks.

- \*\*Broadcast Receiver\*\*:

- \*\*Lifespan\*\*: The lifespan of a Broadcast Receiver can be long-lasting if it’s registered in the manifest (static), making it active even when the app is not running. Dynamic receivers exist only while the app or specific component is running.

- \*\*Implication\*\*: Static receivers are ideal for long-lived system-wide events, while dynamic receivers are more efficient for temporary, context-specific use.

### 5. \*\*Event Source\*\*

- \*\*Listener\*\*:

- \*\*Event Source\*\*: Listeners typically respond to events generated by the user or a specific component, like a button click, a list item selection, or a sensor event.

- \*\*Broadcast Receiver\*\*:

- \*\*Event Source\*\*: Broadcast Receivers respond to broadcasts sent by the Android system or other applications. These broadcasts can be global (e.g., network change, low battery) or local within an app.

### 6. \*\*Examples in Real Life\*\*

- \*\*Listener\*\*:

- \*\*Example\*\*: A listener for a button click in a settings screen that toggles a feature on or off.

- \*\*Broadcast Receiver\*\*:

- \*\*Example\*\*: A Broadcast Receiver that listens for incoming SMS messages or monitors the device's charging state.

### Summary:

- \*\*Listeners\*\* are tied to specific components, are short-lived, and handle user interactions or component-specific events within the app.

- \*\*Broadcast Receivers\*\* handle system-wide or application-wide broadcasts, can be registered statically or dynamically, and have a broader scope, often listening for events that are relevant across different parts of the app or the entire system.