Lesson 7.3: Broadcast receivers

Introduction

Broadcasts are messages that the Android system and Android apps send when events occur that might affect the functionality of other apps or app components. For example, the Android system sends a *system broadcast* when the device boots up, or when headphones are connected or disconnected. If the wired headset is unplugged, you might like your media app to pause the music.

Your Android app can also broadcast events, for example when new data is downloaded that might interest some other app. Events that your app delivers are called *custom broadcasts*.

In general, you can use broadcasts as a messaging system across apps and outside of the normal user flow.

A broadcast is received by any app or app component that has a *broadcast receiver* registered for that action. `BroadcastReceiver` is the base class for code that receives broadcast intents. To learn more about broadcast receivers, see the Broadcasts overview and the `Intent` reference.

Note: While the `Intent` class is used to send and receive broadcasts, the `Intent` broadcast mechanism is completely separate from intents that are used to start activities.

In this practical, you create an app that responds to a change in the charging state of the device. To do this, your app receives and responds to a system broadcast, and it also sends and receives a custom broadcast.

What you should already KNOW

You should be able to:

- Identify key parts of the `AndroidManifest.xml` file.
- Create Implicit intents.

What you'll learn

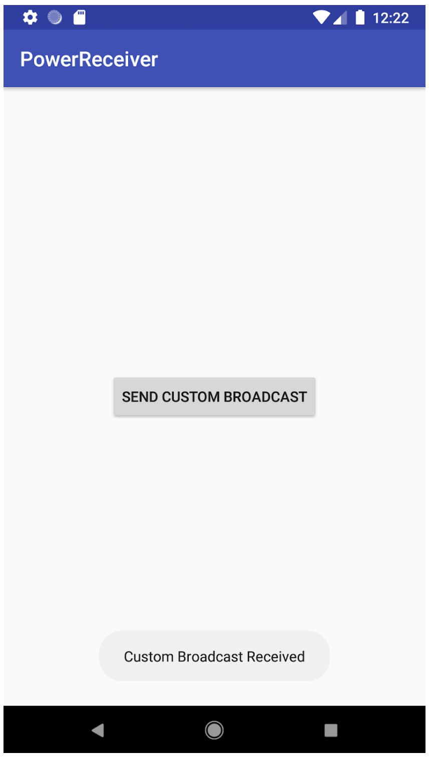
- How to subclass a `BroadcastReceiver` and implement it.
- How to register for system broadcast intents.
- How to create and send custom broadcast intents.

What you'll do

- Subclass a `BroadcastReceiver` to show a toast when a broadcast is received.
- Register your receiver to listen for system broadcasts.
- Send and receive a custom broadcast intent.

App overview

The PowerReceiver app will register a `BroadcastReceiver` that displays a toast message when the device is connected or disconnected from power. The app will also send and receive a custom broadcast to display a different toast message.



Task 1. Set up the PowerReceiver project

1.1 Create the project

1. In Android Studio, create a new Java project called **PowerReceiver**. Accept the default options and use the Empty Activity template.
2. To create a new broadcast receiver, select the package name in the Android Project View and navigate to **File > New > Other > Broadcast Receiver**.
3. Name the class **CustomReceiver**. Make sure that **Java** is selected as the source language, and that **Exported** and **Enabled** are selected. **Exported** allows your broadcast receiver to receive broadcasts from outside your app. **Enabled** allows the system to instantiate the receiver.

1.2 Register your receiver for system broadcasts

A *system broadcast* is a message that the Android system sends when a system event occurs. Each system broadcast is wrapped in an `Intent` object:

- The intent's action field contains event details such as `android.intent.action.HEADSET_PLUG`, which is sent when a wired headset is connected or disconnected.
- The intent can contain other data about the event in its extra field, for example a `boolean` extra indicating whether a headset is connected or disconnected.

Apps can register to receive specific broadcasts. When the system sends a broadcast, it routes the broadcast to apps that have registered to receive that particular type of broadcast.

A `BroadcastReceiver` is either a *static receiver* or a *dynamic receiver*, depending on how you register it:

- To register a receiver statically, use the `<receiver>` element in your `AndroidManifest.xml` file. Static receivers are also called *manifest-declared receivers*.

- To register a receiver dynamically, use the app context or activity context. The receiver receives broadcasts as long as the registering context is valid, meaning as long as the corresponding app or activity is running. Dynamic receivers are also called *context-registered receivers*.

For this app, you're interested in two system broadcasts, `ACTION_POWER_CONNECTED` and `ACTION_POWER_DISCONNECTED`. The Android system sends these broadcasts when the device's power is connected or disconnected.

Starting from Android 8.0 (API level 26 and higher), you can't use static receivers to receive most Android system broadcasts, with some exceptions. So for this task, you use dynamic receivers:

1. (Optional) Navigate to your `AndroidManifest.xml` file. Android Studio has generated a `<receiver>` element, but you don't need it, because you can't use a static receiver to listen for power-connection system broadcasts. Delete the entire `<receiver>` element.
2. In `MainActivity.java`, create a `CustomReceiver` object as a member variable and initialize it.

```
private CustomReceiver mReceiver = new CustomReceiver();
```

Create an intent filter with Intent actions

Intent filters specify the types of intents a component can receive. They are used in filtering out the intents based on `Intent` values like action and category.

1. In `MainActivity.java`, at the end of the `onCreate()` method, create an [`IntentFilter`]() object.

```
IntentFilter filter = new IntentFilter();
```

   When the system receives an `Intent` as a broadcast, it searches the broadcast receivers based on the action value specified in the `IntentFilter` object.

2. In `MainActivity.java`, at the end of `onCreate()`, add the actions `ACTION_POWER_CONNECTED` and `ACTION_POWER_DISCONNECTED` to the `IntentFilter` object.

```
filter.addAction(Intent.ACTION_POWER_DISCONNECTED);
filter.addAction(Intent.ACTION_POWER_CONNECTED);
```

Register and unregister the receiver

1. In `MainActivity.java`, at the end of `onCreate()`, register your receiver using the `MainActivity` context. Your receiver is active and able to receive broadcasts as long as your `MainActivity` is running.

```
// Register the receiver using the activity context.
this.registerReceiver(mReceiver, filter);
```

2. In `MainActivity.java`, override the `onDestroy()` method and unregister your receiver. To save system resources and avoid leaks, dynamic receivers must be unregistered when they are no longer needed or before the corresponding activity or app is destroyed, depending on the context used.

```
@Override
   protected void onDestroy() {
       //Unregister the receiver
       this.unregisterReceiver(mReceiver);
       super.onDestroy();
   }
```

1.3 Implement onReceive() in your BroadcastReceiver

When a broadcast receiver intercepts a broadcast that it's registered for, the `Intent` is delivered to the receiver's `onReceive()` method.

In `CustomReceiver.java`, inside the `onReceive()` method, implement the following steps:

1. Delete the entire `onReceive()` method implementation, including the `UnsupportedOperationException` code.
2. Get the `Intent` action from the `intent` parameter and store it in a `String` variable called `intentAction`.

```
@Override
public void onReceive(Context context, Intent intent) {
   String intentAction = intent.getAction();
}
```

3. Create a `switch` statement with the `intentAction` string. (Before using `intentAction`, do a `null` check on it.) Display a different toast message for each action your receiver is registered for.

```
if (intentAction != null) {
   String toastMessage = "unknown intent action";
   switch (intentAction){
       case Intent.ACTION_POWER_CONNECTED:
           toastMessage = "Power connected!";
           break;
       case Intent.ACTION_POWER_DISCONNECTED:
           toastMessage = "Power disconnected!";
           break;
   }
```

```
  //Display the toast.
}
```

4. After the `switch` statement, add code to display a toast message for a short time:

```
Toast.makeText(context, toastMessage, Toast.LENGTH_SHORT).show();
```

5. Run your app. After the app is running, connect or disconnect your device's power supply. A `Toast` is displayed each time you connect or disconnect the power supply, as long as your `Activity` is running.

Note: If you're using an emulator, toggle the power connection state by selecting the ellipses icon for the menu. Select **Battery** in the left bar, then use the **Charger connection** setting.

Task 2. Send and receive a custom broadcast

In addition to responding to system broadcasts, your app can send and receive custom broadcasts. Use a custom broadcast when you want your app to take an action without launching an activity, for example when you want to let other apps know that data has been downloaded to the device.

Android provides three ways for your app to send custom broadcasts:

- *Normal broadcasts* are asynchronous. Receivers of normal broadcasts run in an undefined order, often at the same time. To send a normal broadcast, create a broadcast intent and pass it to `sendBroadcast(Intent)`.
- *Local broadcasts* are sent to receivers that are in the same app as the sender. To send a local broadcast, create a broadcast intent and pass it to `LocalBroadcastManager.sendBroadcast`.
- *Ordered broadcasts* are delivered to one receiver at a time. As each receiver executes, it can propagate a result to the next receiver, or it can cancel the broadcast so that the broadcast is

not passed to other receivers. To send an ordered broadcast, create a broadcast intent and pass it to `sendOrderedBroadcast(Intent, String)`.

This practical doesn't cover ordered broadcasts, but for more information about them, see Sending broadcasts.

The broadcast message is wrapped in an `Intent` object. The `Intent` action string must provide the app's Java package name syntax and uniquely identify the broadcast event.

For a custom broadcast, you define your own `Intent` action (a unique string). You can create `Intent` objects with custom actions and broadcast them yourself from your app using one of the methods above. The broadcasts are received by apps that have a `BroadcastReceiver` registered for that action.

In this task, you add a button to your activity that sends a local broadcast intent. Your receiver registers the broadcast intent and displays the result in a toast message.

2.1 Define your custom broadcast action string

Both the sender and receiver of a custom broadcast must agree on a unique action string for the `Intent` being broadcast. It's a common practice to create a unique action string by prepending your action name with your app's package name.

One of the simplest ways to get your app's package name is to use `BuildConfig.APPLICATION_ID`, which returns the `applicationId` property's value from your module-level `build.gradle` file.

1. Create a constant member variable in both your `MainActivity` and your `CustomReceiver` class. You'll use this variable as the broadcast `Intent` action.

```
private static final String ACTION_CUSTOM_BROADCAST =
BuildConfig.APPLICATION_ID + ".ACTION_CUSTOM_BROADCAST";
```

Important: Although intents are used both for sending broadcasts and starting activities with `startActivity(Intent)`, these actions are completely unrelated. Broadcast receivers can't see or capture an `Intent` that's used to start an activity. Likewise, when you broadcast an `Intent`, you can't use that `Intent` to find or start an activity.

2.2 Add a "Send Custom Broadcast" button

1. In your `activity_main.xml` layout file, replace the Hello World `Textview` with a `Button` that has the following attributes:

```
<Button
   android:id = "@+id/sendBroadcast"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Send Custom Broadcast"
   android:onClick="sendCustomBroadcast"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintLeft_toLeftOf="parent"
   app:layout_constraintRight_toRightOf="parent"
   app:layout_constraintTop_toTopOf="parent" />
```

2. Extract the string resource.

The `sendCustomBroadcast()` method will be the click-event handler for the button. To create a stub for `sendCustomBroadcast()` in Android Studio:

1. Click the yellow highlighted `sendCustomBroadcast` method name. A red light bulb appears on the left.
2. Click the red light bulb and select **Create 'sendCustomBroadcast(View)' in 'MainActivity'**.

2.3 Implement sendCustomBroadcast()

Because this broadcast is meant to be used solely by your app, use `LocalBroadcastManager` to manage the broadcast. `LocalBroadcastManager` is a class that allows you to register for and send broadcasts that are of interest to components within your app.

By keeping broadcasts local, you ensure that your app data isn't shared with other Android apps. Local broadcasts keep your information more secure and maintain system efficiency.

In `MainActivity.java`, inside the `sendCustomBroadcast()` method, implement the following steps:

1. Create a new `Intent`, with your custom action string as the argument.

```
Intent customBroadcastIntent = new Intent(ACTION_CUSTOM_BROADCAST);
```

2. After the custom `Intent` declaration, send the broadcast using the `LocalBroadcastManager` class:

```

LocalBroadcastManager.getInstance(this).sendBroadcast(customBroadcastIntent)
;
```

2.4 Register and unregister your custom broadcast

Registering for a local broadcast is similar to registering for a system broadcast, which you do using a dynamic receiver. For broadcasts sent using `LocalBroadcastManager`, static registration in the manifest is not allowed.

If you register a broadcast receiver dynamically, you must unregister the receiver when it is no longer needed. In your app, the receiver only needs to respond to the custom broadcast when the app is running, so you can register the action in `onCreate()` and unregister it in `onDestroy()`.

1. In `MainActivity.java`, inside `onCreate()` method, get an instance of `LocalBroadcastManager` and register your receiver with the custom `Intent` action:

```
LocalBroadcastManager.getInstance(this)
             .registerReceiver(mReceiver,
                          new IntentFilter(ACTION_CUSTOM_BROADCAST));
```

2. In `MainActivity.java`, inside the `onDestroy()` method, unregister your receiver from the `LocalBroadcastManager`:

```
LocalBroadcastManager.getInstance(this)
          .unregisterReceiver(mReceiver);
```

2.5 Respond to the custom broadcast

1. In `CustomReceiver.java`, inside the `onReceive()` method, add another `case` statement in the `switch` block for the custom `Intent` action. Use `"Custom Broadcast Received"` as the text for the toast message.

```
case ACTION_CUSTOM_BROADCAST:
   toastMessage = "Custom Broadcast Received";
   break;
```

2. Extract the string resource.
3. Run your app and tap the **Send Custom Broadcast** button to send a custom broadcast. Your receiver (`CustomReceiver`) displays a toast message.

That's it! Your app delivers a custom broadcast and is able to receive both system and custom broadcasts.