LAB 9 – Broadcast Intents and Broadcast Receivers

At the end of this lab, students should be able to:

- Send broadcast intent.
- Implement a broadcast receiver to listen for both custom and system broadcasts intents.

An Overview of Broadcast Intents

Broadcast intents are Intent objects that are broadcast via a call to the sendBroadcast(), sendStickyBroadcast() or sendOrderedBroadcast() methods of the Activity class (the latter being used when results are required from the broadcast). In addition to providing a messaging and event system between application components, broadcast intents are also used by the Android system to notify interested applications about key system events (such as the external power supply or headphones being connected or disconnected).

An Overview of Broadcast Receivers

An application listens for specific broadcast intents by registering a broadcast receiver. Broadcast receivers are implemented by extending the Android BroadcastReceiver class and overriding the onReceive() method. The broadcast receiver may then be registered, either within code (for example within an activity), or within a manifest file. Part of the registration implementation involves the creation of intent filters to indicate the specific broadcast intents the receiver is required to listen for.

System 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.

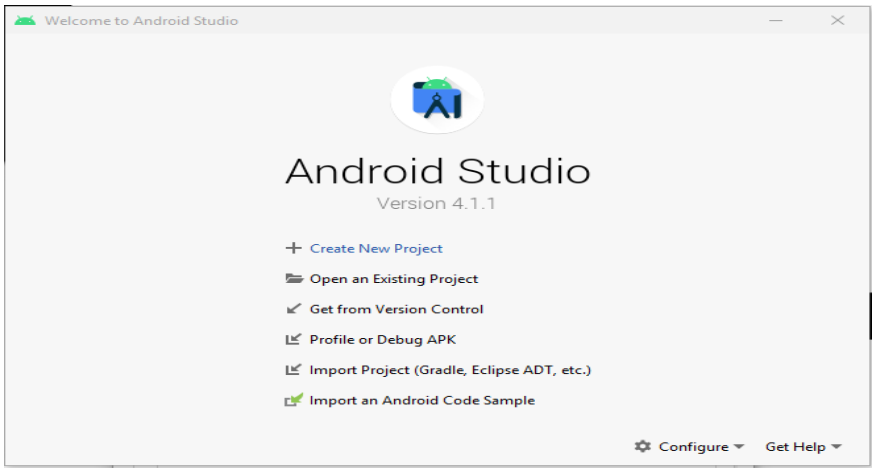
Event	Description
Intent.ACTION_BOOT_COMPLETED	Boot completed. Requires the android.permission.RECEIVE_BOOT_COMPLETED permission
Intent.ACTION_POWER_CONNECTED	Power got connected to the device.
Intent.ACTION_POWER_DISCONNECTED	Power got disconnected to the device.
Intent.ACTION_BATTERY_LOW	Triggered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

https://www.vogella.com/tutorials/AndroidBroadcastReceiver/article.html

Android Broadcast Intent and Receiver Project- An Example

a) Creating the Example Application

i. Go to File -> New Project



ii. Choose **Empty Activity** template, then click Next

iii. Enter *SendBroadcast* into the Name field and specify com.ebookfrenzy.sendbroadcast as the package name.
Change the Minimum API level setting to API26: Android 8.0(Oreo) and the Language menu to Java.
Click Finish.

b) Setting up the Layout

Locate the *activity_main.xml* file in the Project tool window and double click on it to load it into the layout editor.

i. With the Layout Editor tool in Design mode, replace the TextView object with a Button view and set the text property so that it reads “Send Broadcast”. Once the text value has been set, follow the usual steps to extract the string to a resource named *send_broadcast*.

ii. With the button still selected in the layout, locate the *onClick* property in the Attributes panel and configure it to call a method named *broadcastIntent*.

c) Creating and Sending the Broadcast Intent

Having created the framework for the SendBroadcast application, it is now time to implement the code to send the broadcast intent. This involves implementing the *broadcastIntent()* method specified previously as the *onClick* target of the Button view in the user interface.

i. Locate and double-click on the *MainActivity.java* file and modify it to add the code to create and send the broadcast intent. Once modified, the source code for this class should read as follows:

```
package com.ebookfrenzy.sendbroadcast;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.content.Intent;
import android.view.View;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_send_broadcast);
}
public void broadcastIntent(View view)
{
Intent intent = new Intent();
intent.setAction("com.ebookfrenzy.sendbroadcast");
intent.addFlags(Intent.FLAG_INCLUDE_STOPPED_PACKAGES);
sendBroadcast(intent);
}
}
```

Note that in this instance the action string for the intent is *com.ebookfrenzy.sendbroadcast*. When the broadcast receiver class is created later, it is essential that the intent filter declaration match this action string.

This concludes the creation of the application to send the broadcast intent. All that remains is to build a matching broadcast receiver.

d) Creating the Broadcast Receiver

In order to create the broadcast receiver, a new class needs to be created which subclasses the BroadcastReceiver superclass.

i. Within the Project tool window, navigate to *app -> java* and right-click on the package name. From the resulting menu, select the *New -> Other -> Broadcast Receiver* menu option, name the class *MyReceiver* and make sure the *Exported* and *Enabled* options are selected. These settings allow the Android system to launch the receiver when needed and ensure that the class can receive messages sent by other applications on the device. With the class configured, click on Finish.

ii. Once created, Android Studio will automatically load the new *MyReceiver.java* class file into the editor where it should read as follows:

```
package com.ebookfrenzy.sendbroadcast;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
public class MyReceiver extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
// TODO: This method is called when the BroadcastReceiver is receiving
```

```
                    // an Intent broadcast.
throw new UnsupportedOperationException("Not yet
implemented");
}
}
```

As can be seen in the code, Android Studio has generated a template for the new class and generated a stub for the onReceive() method. A number of changes now need to be made to the class to implement the required behavior. Remaining in the MyReceiver.java file, therefore, modify the code so that it reads as follows:

```
package com.ebookfrenzy.sendbroadcast;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;
public class MyReceiver extends BroadcastReceiver {
public MyReceiver() {
}
@Override
public void onReceive(Context context, Intent intent) {
// TODO: This method is called when the BroadcastReceiver
is receiving
// an Intent broadcast.
throw new UnsupportedOperationException("Not yet
implemented");
Toast.makeText(context, "Broadcast Intent Detected.",
Toast.LENGTH_LONG).show();
}
}
```

The code for the broadcast receiver is now complete.

e) Registering the Broadcast Receiver

The project needs to publicize the presence of the broadcast receiver and must include an intent filter to specify the broadcast intents in which the receiver is interested. When the BroadcastReceiver class was created in the previous section, Android Studio automatically added a <receiver> element to the manifest file.

i. All that remains, therefore, is to add code within the *MainActivity.java* file to create an intent filter and to register the receiver:

```
package com.ebookfrenzy.sendbroadcast;
.
.
import android.content.BroadcastReceiver;
import android.content.IntentFilter;
.
.
```

```
public class MainActivity extends AppCompatActivity {
BroadcastReceiver receiver;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_send_broadcast);
configureReceiver();
}
private void configureReceiver() {
IntentFilter filter = new IntentFilter();
filter.addAction("com.ebookfrenzy.sendbroadcast");
receiver = new MyReceiver();
registerReceiver(receiver, filter);
}
.
.
}
```

ii. It is also important to unregister the broadcast receiver when it is no longer needed:

```
@Override
protected void onDestroy() {
super.onDestroy();
unregisterReceiver(receiver);
}
```

f) Testing the Broadcast Example

In order to test the broadcast sender and receiver, run the SendBroadcast app on a device or AVD and wait for it to appear on the display. Once running, touch the button, at which point the toast message reading "Broadcast Intent Detected." should pop up for a few seconds before fading away.

g) Listening for System Broadcasts

The final stage is to modify the intent filter for the broadcast receiver to listen also for the system intent that is broadcast when external power is disconnected from the device.
That action is *android.intent.action.ACTION_POWER_DISCONNECTED.*

i. Modify the *onCreate()* method in the *MainActivity.java* file to add this additional filter:

```
private void configureReceiver() {
IntentFilter filter = new IntentFilter();
filter.addAction("com.ebookfrenzy.sendbroadcast");
filter.addAction(
"android.intent.action.ACTION_POWER_DISCONNECTED");
receiver = new MyReceiver();
registerReceiver(receiver, filter);
}
```

ii. Since the *onReceive()* method is now going to be listening for two types of broadcast intent, it is worthwhile to modify the code so that the action string of the current intent is also displayed in the toast message. This string can be obtained via a call to the *getAction()* method of the intent object passed as an argument to the *onReceive()* method:

```
public void onReceive(Context context, Intent intent) {
String message = "Broadcast intent detected "+ intent.getAction();
Toast.makeText(context, message,
Toast.LENGTH_LONG).show();
}
```

Test the receiver by re-installing the modified *BroadcastReceiver* package. Touching the button in the *SendBroadcast* application should now result in a new message containing the custom action string:

Broadcast intent detected com.ebookfrenzy.sendbroadcast

Next, remove the USB connector that is currently supplying power to the Android device, at which point the receiver should report the following in the toast message.
If the app is running on an emulator, display the extended controls, select the Battery option and change the Charger connection setting to None.

Broadcast intent detected
android.intent.action.ACTION_POWER_DISCONNECTED

To avoid this message appearing every time the device is disconnected from a power supply launch the Settings app on the device and select the Apps & notifications option.
Select the BroadcastReceiver app from the resulting list and tap the Uninstall button.

System services and Broadcast receiver- An Example

https://www.vogella.com/tutorials/AndroidBroadcastReceiver/article.html

In this example we will schedule a receiver via the Android alert manager system service. Once called, it uses the Android vibrator manager and a popup message (Toast) to notify the user.

a) Creating project and Layout

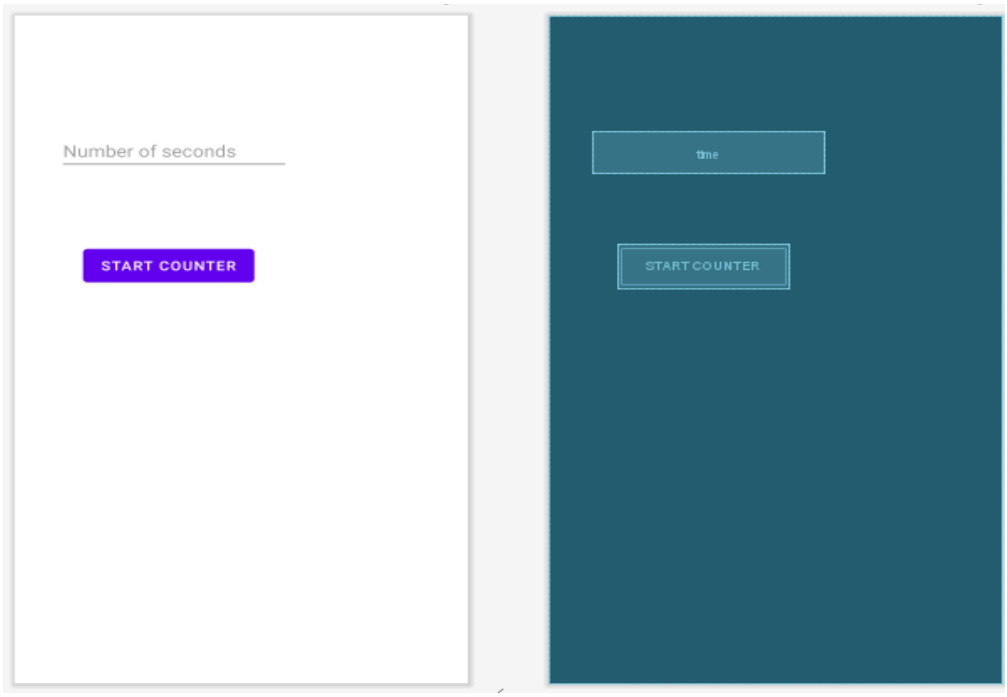
i. Go to File -> New Project
ii. Choose **Empty Activity** template, then click Next
iii. Enter *AlarmaApp* into the Name field and specify com.ebookfrenzy.alarmapp
Change the Minimum API level setting to API26: Android 8.0(Oreo) and the Language menu to Java.
Click Finish.

iv. Create the following layout:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="62dp"
        android:layout_marginTop="250dp"
        android:onClick="startAlert"
        android:text="@string/start_counter"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/time"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="40dp"
        android:layout_marginTop="126dp"
        android:layout_marginBottom="127dp"
        android:ems="10"
        android:hint="Number of seconds"
        android:inputType="numberDecimal"
        app:layout_constraintBottom_toBottomOf="@+id/ok"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```



b) Creating broadcast receiver

i. Navigate to *app -> java* and right-click on the package name. From the resulting menu, select the *New -> Other -> Broadcast Receiver* menu option, name the class *MyReceiver* and make sure the *Exported* and *Enabled* options are selected. Modify the broadcast receiver class and set the vibrator service as follow:

```
package com.ebookfrenzy.alarmapp;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Vibrator;
import android.widget.Toast;

public class MyReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO: This method is called when the BroadcastReceiver is receiving
        // an Intent broadcast.
        //throw new UnsupportedOperationException("Not yet implemented");
        Toast.makeText(context, "Don't panic but your time is up!!!!.",
                Toast.LENGTH_LONG).show();
        // Vibrate the mobile phone
        Vibrator vibrator = (Vibrator)
context.getSystemService(Context.VIBRATOR_SERVICE);
        vibrator.vibrate(2000);
    }
}
```

ii. Register permission to vibrate phone in the AndroidManifest.xml

```
<uses-permission android:name="android.permission.VIBRATE" />
```

c) Modify the MainActivity.java

Modify the file as follow:

```
package com.ebookfrenzy.alarmapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
    }
    public void startAlert(View view) {
        EditText text = (EditText) findViewById(R.id.time);
        int i = Integer.parseInt(text.getText().toString());
        Intent intent = new Intent(this, MyReceiver.class);
        PendingIntent pendingIntent = PendingIntent.getBroadcast(
                this.getApplicationContext(), 234324243, intent, 0);
        AlarmManager alarmManager = (AlarmManager)
getSystemService(ALARM_SERVICE);
        alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis()
                + (i * 1000), pendingIntent);
        Toast.makeText(this, "Alarm set in " + i + " seconds",
                Toast.LENGTH_LONG).show();

    }

}
```

d) Testing the app

Run your application on the device. Set your time and start the alarm. After the defined number of seconds a *Toast* should be displayed. Keep in mind that the vibration alarm does not work on the Android emulator.