Broadcast Receivers

- Broadcast Receivers are one of four components of an App (other three are Activity, Services and Content Providers).
- Respond to system wide announcements. Announcements can be generated by the Android system or other apps.
- Examples of system generated announcements: Screen turned off, battery is low, picture captured etc.
- Apps can also generate custom broadcasts by calling Context.sendBroadcast() method and passing an intent as a parameter.
- To listen to broadcasts we need to register a receiver. This can be done:
  - statically by declaring in the manifest file.
  - dynamically using method Context.registerReceiver() and passing an instance of class that extends BroadcastReceiver class.
- BroadcastReceiver.onReceive() method is invoked whenever a broadcast with intent matches our receiver’s filter.

- The app instantiates a class which extends BroadcastReciever class

```
private final BroadcastReceiver broadcastReceiver = new BroadcastReceiver() {
   @Override
   public void onReceive(Context context, Intent intent) {
      //Code to execute when broadcast is received.
   }
};
```

- Register receiver in onResume() callback method.

```
IntentFilter intentFilterWiFi = new IntentFilter();                    // Instantiate new IntentFilter Class
intentFilterWiFi.addAction(WifiManager.WIFI_STATE_CHANGED_ACTION);     // Specify the type of broadcast to listen
registerReceiver(broadcastReceiverWiFi, intentFilterWiFi);             // Register the receiver.
```

- Unregister the receiver in onPause() callback method to conserve resources.

```
unregisterReceiver(broadcastReceiverWiFi);
```

- Send custom broadcast in our app.

```
Intent intent = new Intent();                                          //Instantiate Intent Class
intent.setAction(getPackageName() + ".uniqueIntentCustom");            //Specify the intent action that other apps
                                                                       //should use to receive this broadcast
sendBroadcast(intent);                                                 //Broadcast using sendBroadcast() method
```

- The app also registers Power connected/disconnected receiver statically in the AndroidManifest.xml file. This allows the onReceive() method to be invoked automatically when ever power is connected or removed even when the app is not running.

```
<receiver android:name=".PowerConnBroadcastReceiver">
    <intent-filter>
        <action android:name="android.intent.action.ACTION_POWER_DISCONNECTED"/>
        <action android:name="android.intent.action.ACTION_POWER_CONNECTED"/>
    </intent-filter>
</receiver>
```

- The onReceive() method of the receiver invokes the main activity. This has the effect that when ever power is connected or disconnected, the MainActivity.class is invoked.

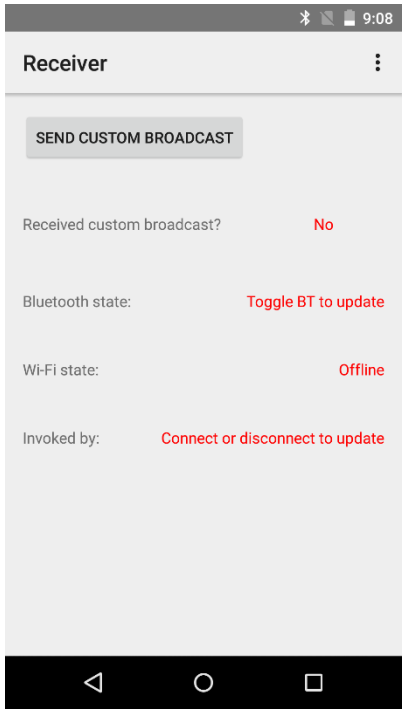
```
public void onReceive(Context context, Intent intent) {
    Intent intentNewActivity = new Intent(context, MainActivity.class);
    intentNewActivity.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
        .addFlags(intent.FLAG_ACTIVITY_SINGLE_TOP);
    context.startActivity(intentNewActivity);
}
```

- Flags used (in onReceive() callback method):
  - FLAG_ACTIVITY_NEW_TASK:
    - Start the activity in a new task. If task is already running with the activity, invoke this instance and pass intent to onNewIntent() callback method.
    - Same as setting launchMode attribute with value of singleTask in AndroidManifest.xml.
  - FLAG_ACTIVITY_SINGLE_TOP:
    - If activity is currently on top of back stack, it is invoked instead of creating new instances of activity. onNewIntent() callback method is invoked.
    - Same as setting launchMode attribute with value of singleTop in AndroidManifest.xml.
- onNewIntent() callback method in invoked when the parent activity is relaunched if the "FLAG_ACTIVITY_SINGLE_TOP“ flag is set on the intent used to start new activity.

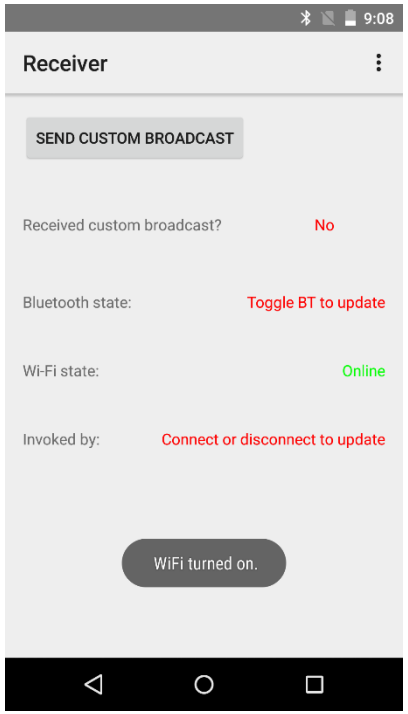
```
protected void onNewIntent(Intent intent) {
    super.onNewIntent(intent);
    // Code to execute once activity is re-launched
}
```

- The app sends and receives one custom broadcast (click button) and receives two system broadcasts(BT and Wi-Fi).

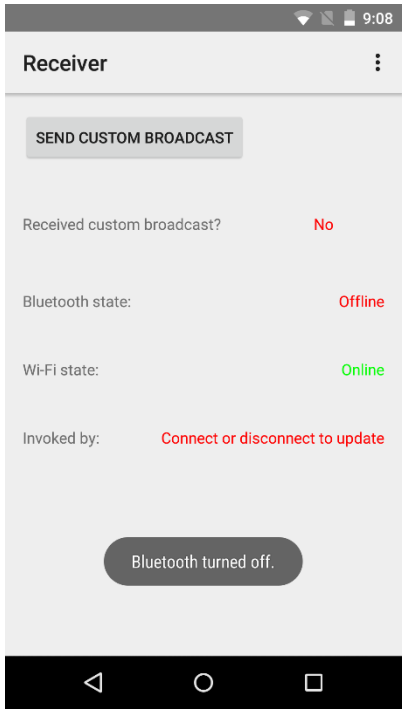
App start screen



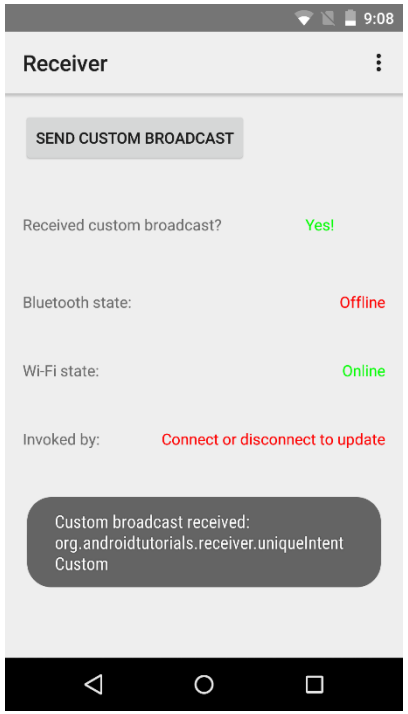
After toggling Wi-Fi



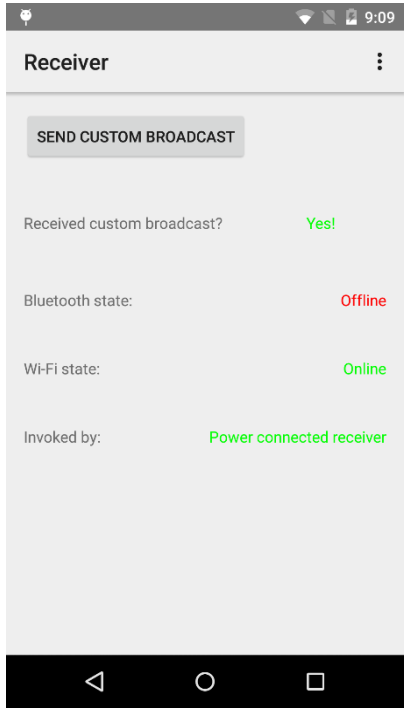
After toggling Bluetooth



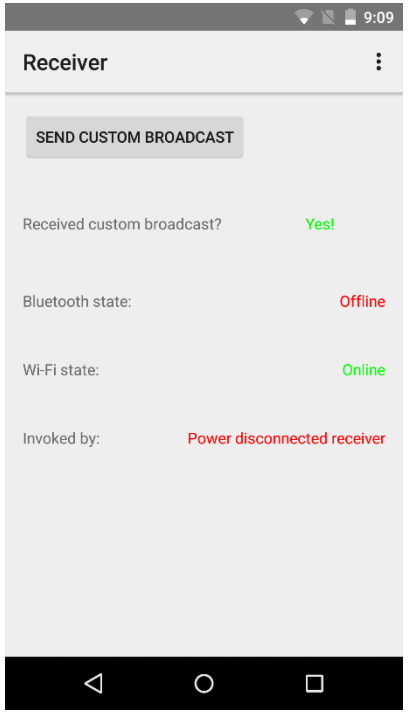
After sending custom broadcast



Activity started automatically after connecting power



Activity started automatically after disconnecting power



- Two major classes of broadcasts.
  - Normal Broadcasts: use Context.sendBroadcast() method.
    - Asynchronous.
    - All the receivers of this broadcast are executed in undefined order.
    - Receivers cannot use result or abort the broadcast.
    - Advantages: Efficient, simple to use.
  - Ordered Broadcasts: use Context.sendOrderedBroadcast() method.
    - Broadcast are delivered one at a time based on priority.
    - Each receiver can propagate result to next receiver.
    - Receivers can abort the broadcast and prevent next receiver from receiving.
- LocalBroadcastManager - Used to send intents to local objects within the app process.
  - We know the data we are sending within our app.
  - Other apps cannot broadcast to our app.
  - More efficient and secure compared to system wide broadcast.

References

- BroadcastReceivers
- Intents and Intent-Filters
- LocalBroadcastManager
- Task and Back Stack

Exercise:
Nothing to do!