Broadcast Receivers and Result Intents

Dieudonne U

Broadcast Receivers

- 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

- All registered receivers for an event are notified by the Android runtime once this event happens
- For example, applications can register for the ACTION_BOOT_COMPLETED system event which is fired once the Android system has completed the boot process

- 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.

Registering Broadcast Receiver

- A receiver can be registered via the AndroidManifest.xml file.
- Alternatively to this static registration, you can also register a receiver dynamically via the Context.registerReceiver() method.
- The implementing class for a receiver extends the BroadcastReceiver class.
- If the event for which the broadcast receiver has registered happens, the onReceive() method of the receiver is called by the Android system

Using Manifest File

```
<receiver android:name=“MyReceiver" >
        <intent-filter>
            <action android:name="android.intent.action.PHONE_STATE" >
            </action>
        </intent-filter>
    </receiver>
```

```
<receiver android:name=".SampleBroadcastReceiver">
    <intent-filter>
        <action
android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

The above statement will fire the defined system broadcast event whenever the boot process is completed.

Dynamic BroadcastReceiver Registration

```
Public void regiterMyReceiver(Context contect,BroadcastReceiver
reciever){
IntentFilter filter=new IntentFilter();
filter.addAction(Intent.ACTION_POWER_CONNECTED);
....
contecxt.getApplicationContext().register(receiver,intent);


}
```

Un Registering Broadcast Receiver

```
public void unRegiterMyReceiver(Context context,BroadcastReceiver
reciever){
context.getApplicationContext().unRegisterReceiver(receiver,intent);
}
```

Creating BroadcastReceiver

```
public class MyBroadcastReceiver extends BroadcastReceiver {
   @Override
    public void onReceive(Context context, Intent intent){
     //do something
       }
}
```

Broadcasting Custom Intent

```
public void broadcastIntent(View view){
 Intent intent = new Intent();
 intent.setAction("com.tutorialspoint.CUSTOM_INTENT");
 sendBroadcast(intent);
}
```

Register Custom Intent in Manifest

```
<receiver android:name="MyReceiver">
        <intent-filter>
          <action android:name="com.tutorialspoint.CUSTOM_INTENT">
          </action>
        </intent-filter>

     </receiver>
```

RESULT INTENTS

Getting a Result from an Activity

- Starting another activity doesn't have to be one-way. You can also start another activity and receive a result back. To receive a result, call **startActivityForResult()** instead of **startActivity().**

- For example, your app can start a camera app and receive the captured photo as a result. Or, you might start the People app in order for the user to select a contact and you'll receive the contact details as a result.

- Of course, the activity that responds must be designed to return a result. When it does, it sends the result as another Intent object. Your activity receives it in the **onActivityResult()** callback.

startActivityForResult()

```
static final int PICK_CONTACT_REQUEST = 1;  // The request code
...
private void pickContact() {
    Intent pickContactIntent = new Intent(Intent.ACTION_PICK,
Uri.parse("content://contacts"));
    pickContactIntent.setType(Phone.CONTENT_TYPE); // Show user only contacts w/
phone numbers
    startActivityForResult(pickContactIntent, PICK_CONTACT_REQUEST);
}
```

Receiving the result

- When the user is done with the subsequent activity and returns, the system calls your activity's onActivityResult() method. This method includes three arguments:
- The request code you passed to startActivityForResult().
- A result code specified by the second activity. This is either RESULT_OK if the operation was successful or RESULT_CANCELED if the user backed out or the operation failed for some reason.
- An Intent that carries the result data.
- For example, here's how you can handle the result for the "pick a contact" intent:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent resultIntent) {
    // Check which request it is that we're responding to
    if (requestCode == PICK_CONTACT_REQUEST) {
        // Make sure the request was successful
        if (resultCode == RESULT_OK) {
            // Get the URI that points to the selected contact
            Uri contactUri = resultIntent.getData();
            // We only need the NUMBER column, because there will be only one row in the result
            String[] projection = {Phone.NUMBER};
            Cursor cursor = getContentResolver()
                    .query(contactUri, projection, null, null, null);
            cursor.moveToFirst();

            // Retrieve the phone number from the NUMBER column
            int column = cursor.getColumnIndex(Phone.NUMBER);
            String number = cursor.getString(column);
            // Do something with the phone number...
        }
```