CMSC436: Programming Handheld Systems
The BroadcastReceiver Class
Today’s Topics

The BroadcastReceiver Class
Registering for events
Broadcasting events
Processing events
BroadcastReceiver

Base class for components that receive and react to events
BroadcastReceiver

BroadcastReceivers register to receive events in which they are interested
BroadcastReceiver

When Events occur at runtime they are represented as Intents
Those Intents are then broadcast to the system
BroadcastReceiver

Android routes the Intents to BroadcastReceivers that have registered to receive them. BroadcastReceivers receive the Intent via a call to onReceive().
Typical Use Case

Register BroadcastReceivers to receive specific events

When event occurs, broadcast an Intent

Android delivers Intent to registered recipients by calling their onReceive() method

Event handled in onReceive()
Registering for Intents

BroadcastReceivers can register in two ways:

- Statically, in AndroidManifest.XML
- Dynamically, by calling a registerReceiver() method
Static Registration

Put `<receiver>` and `<intent-filter>` tags in `AndroidManifest.xml`
<Receiver> Tag Format

<receiver
    android:enabled=["true" | "false"]
    android:exported=["true" | "false"]
    android:icon="drawable resource"
    android:label="string resource"
    android:name="string"
    android:permission="string"
    android:process="string" >
    
    ...

</receiver>
Intent Filter

Specify `<intent-filter>` tag within a `<receiver>`
See lecture on Intent class
Static Registration

Receivers can be registered in AndroidManifest.xml

Will be woken to receive broadcasts, if needed

In API 26 or higher, statically registered receivers cannot receive most implicit intents

<receiver
    android:name=".Receiver"
    android:exported="false"
    android:permission="android.permission.VIBRATE">
    <intent-filter>
        <action android:name="course.examples.broadcastreceiver.
            singlebroadcaststaticregistration.SHOW_TOAST" />
    </intent-filter>
</receiver>
public class SimpleBroadcastActivity extends Activity {
    private static final String CUSTOM_INTENT =
            "course.examples.broadcastreceiver.
            singlebroadcaststaticregistration.SHOW_TOAST";

    …

    public void onClick(@ SuppressWarnings("unused") View v) {
        Log.i(TAG, "Broadcast sent");
        Intent intent = new Intent(CUSTOM_INTENT);
        intent.setPackage("course.examples.broadcastreceiver.
                             singlebroadcaststaticregistration");
        sendBroadcast(intent, Manifest.permission.VIBRATE);
    }
}
public class Receiver extends BroadcastReceiver {
    @SuppressWarnings("FieldCanBeLocal")
    private final String TAG = "Receiver";

    public void onReceive(Context context, Intent intent) {
        Log.i(TAG, "Broadcast Received");
        Vibrator v = (Vibrator) context
            .getSystemService(Context.VIBRATOR_SERVICE);
        v.vibrate(500);
        Toast.makeText(context, "Broadcast Received by Receiver",
                        Toast.LENGTH_LONG).show();
    }
}
Dynamic Registration

Create an IntentFilter
Create a BroadcastReceiver
Register BroadcastReceiver using registerReceiver()
   LocalBroadcastManager
   Context
Call unRegisterReceiver() to unregister BroadcastReceiver
public class SingleBroadcastActivity extends Activity {

    private static final String CUSTOM_INTENT = ...;
    private final IntentFilter intentFilter = new IntentFilter(CUSTOM_INTENT);
    private final Receiver receiver = new Receiver();
    private LocalBroadcastManager mBroadcastMgr;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mBroadcastMgr = LocalBroadcastManager.getInstance(getActivity());
        setContentView(R.layout.main);
    }
}
// Called when Button is clicked
public void onClick(@SuppressWarnings("unused") View v) {
    mBroadcastMgr.sendBroadcast(new Intent(CUSTOM_INTENT));
}

protected void onStart() {
    super.onStart();
    mBroadcastMgr.registerReceiver(receiver, intentFilter);
}

protected void onStop() {
    mBroadcastMgr.unregisterReceiver(receiver);
    super.onStop();
}
private static final String CUSTOM_INTENT = ...
private final Receiver1 mReceiver1 = new Receiver1();
private final IntentFilter mIntentFilter = new IntentFilter(CUSTOM_INTENT);
...
public void onClick(View v) {
    Intent intent = new Intent(CUSTOM_INTENT)
        .setPackage("course.examples.broadcastreceiver.compoundbroadcast");
    sendBroadcast(intent, Manifest.permission.VIBRATE);
}
protected void onStart() {
    super.onStart();
    registerReceiver(mReceiver1, mIntentFilter);
}
protected void onStop() {
    unregisterReceiver(mReceiver1);
    super.onStop();
}
<receiver
    android:name=".Receiver3"
    android:exported="false">
    <intent-filter>
      <action android:name="...SHOW_TOAST" />
    </intent-filter>
</receiver>

<receiver
    android:name=".Receiver2"
    android:exported="false">
    <intent-filter>
      <action android:name="...SHOW_TOAST" />
    </intent-filter>
</receiver>
Event Broadcast

Multiple broadcast methods supported

Normal vs. Ordered

Normal: processing order undefined

Ordered: sequential processing in priority order
Some Debugging Tips

Log extra Intent resolution information
   Intent.setFlag(FLAG_DEBUG_LOG_RESOLUTION)
List registered BroadcastReceivers
Dynamically registered
   % adb shell dumpsys activity b
Statically registered
   % adb shell dumpsys package
Event Delivery

Intents are delivered to BroadcastReceiver by calling `onReceive(Context, Intent)`

- The Context in which the receiver is running
- The Intent that was broadcast
Event Handling in `onReceive()`

Hosting process has high priority while `onReceive()` is executing.
`onReceive()` runs on the main Thread.
So `onReceive` should be short-lived.
Event Handling in `onReceive()`

Note: If event handling is lengthy, consider starting a Service, rather than performing complete operation in `onReceive()`

Will cover the Service class later in the course
Event Handling in onReceive()

BroadcastReceiver is not considered valid once onReceive() returns

Normally, BroadcastReceivers can’t start asynchronous operations

  e.g., showing a Dialog, starting an Activity via startActivityForResult()
Ordered Broadcasts

// send Intent to BroadcastReceiver in priority order
void sendOrderedBroadcast (Intent intent, String receiverPermission)

// send Intent to BroadcastReceiver in priority order. Includes multiple
// parameters for greater control
void sendOrderedBroadcast (Intent intent,
    String receiverPermission,
    BroadcastReceiver resultReceiver,
    Handler scheduler,
    int initialCode,
    String initialData,
    Bundle initialExtras)
<receiver
    android:name=".Receiver2"
    android:exported="false">
    <intent-filter android:priority="1">
        <action android:name="...SHOW_TOAST" />
    </intent-filter>
</receiver>

<receiver
    android:name=".Receiver3"
    android:exported="false">
    <intent-filter android:priority="10">
        <action android:name="...SHOW_TOAST" />
    </intent-filter>
</receiver>
public void onClick(View v) {
    sendOrderedBroadcast(new Intent(CUSTOM_INTENT)
        .setPackage("course.examples.broadcastreceiver.compoundorderedbroadcast"),
        android.Manifest.permission.VIBRATE);
}

protected void onStart() {
    super.onStart();
    IntentFilter intentFilter = new IntentFilter(CUSTOM_INTENT);
    intentFilter.setPriority(3);
    registerReceiver(mReceiver, intentFilter);
}

protected void onStop() {
    unregisterReceiver(mReceiver);
    super.onStop();
}
public class Receiver1 extends BroadcastReceiver {
    private final String TAG = "Receiver1";
    public void onReceive(Context context, Intent intent) {
        Log.i(TAG, "INTENT RECEIVED");
        if (isOrderedBroadcast()) {
            Log.i(TAG, "Calling abortBroadcast()");
            abortBroadcast();
        }
        ...
    }
}
Final Result is: Receiver 3: Receiver 1: Receiver 2:
public void onClick(View v) {
    sendOrderedBroadcast(new Intent("CUSTOM_INTENT")
            .setPackage("course.examples.broadcastreceiver.resultreceiver"),
            null,
            new BroadcastReceiver() {
                public void onReceive(Context context, Intent intent) {
                    Toast.makeText(context, "Final Result is " + getResultData(),
                            Toast.LENGTH_LONG).show();
                }
            },
            null, 0, null, null);
}
public class Receiver3 extends BroadcastReceiver {
  ...

  public void onReceive(Context context, Intent intent) {
    Log.i(TAG, "INTENT RECEIVED by Receiver3");

    String tmp = getResultData() == null ? "" : getResultData();
    setResultData(tmp + ":Receiver 3");
  }
}
Long-Running Operations

After `onReceive()` exits, system can kill BroadcastReceiver

Don’t’ start long-running Threads from `onReceive()`

Options

Call `goAsync()`

Schedule a JobService with JobScheduler. (Will discuss Services later in course)
goAsync()

BroadcastReceiver is generally valid only until `onReceive()` exits.

Use `goAsync()` to allow asynchronous processing from `onReceive()`.

Method returns an object of `PendingResult`.

Receiver considered alive until `PendingResult.finish()`
BcastRecGoAsync
public class Receiver extends BroadcastReceiver {
    public void onReceive(final Context context, final Intent intent) {
        ...
        final PendingResult pendingResult = goAsync();
        new Thread(new Runnable() {
            public void run() {
                try { /* long-running operation */
                ...
            }
            // Must call finish() so the BroadcastReceiver can be recycled.
            pendingResult.finish();
        }
    }).start();
    ...
}
Additional Notes

BroadcastReceiver’s original design has changed to improve security, performance and UX

- Prefer LocalBroadcastManager to Context
- Prefer Context registration over Manifest registration
- Don’t put sensitive info in implicit Intents you broadcast
- Don’t start Activities from onReceive()
Next Time

User Notifications
Example Applications

BcastRecSinBcastStatReg
BcastRecSinBcastDynReg
BcastRecCompBcast
BcastRecCompOrdBcast
BcastRecCompOrdBcastWithResRec
BcastRecGoAsync