Broadcast Receivers and Long-Running Services

Through previous chapters, you have been exposed to activities, content providers, and services. We haven’t talked much about broadcast receivers, so we will do that in this chapter.

We’ll show you first how to invoke a simple broadcast receiver and then extend the idea to invoking multiple broadcast receivers. We will also explore how broadcast receivers can reside in processes outside of the client processes. We will demonstrate how a broadcast receiver can send notification messages via the notification manager.

We will talk about the 10-second limit on a broadcast receiver to respond before the system throws “application not responding” (ANR) messages and suggest known mechanisms to work around this. We will develop a framework where you can start viewing a long-running service as a special abstraction of a broadcast intent, and finally, we’ll talk about wake locks in the context of long running services.

Let’s start this extensive coverage on broadcast receivers by starting with coding a simple broadcast receiver.

Broadcast Receivers

In Chapter 13, we talked about broadcast receiver being another component of an Android process, along with activities, content providers, and services. As the name indicates, a broadcast receiver is a component that can respond to a broadcast message sent by a client. The message itself is an Android broadcast intent, and a broadcast message can be received by more than one receiver.

A component such as an activity or a service (or anything that is eventually implementing the Context class) wanting to broadcast an event (intent) uses the sendBroadcast() method available on the Context class. The argument to this method is an intent.
Receiving components of the broadcast intent will need to inherit from a Receiver class available in the Android SDK. These receiving components (broadcast receivers) then need to be registered in the manifest file as a receiver that is interested in the broadcast intent.

**NOTE:** You can register receivers at run time as well without mentioning them in the manifest file. Please note that we are not covering that aspect in this chapter, and we recommended that you see the API documentation URL indicated in the “References” section of this chapter for further information.

## Sending a Broadcast

Listing 14–1 shows sample code, taken from an activity class, that sends a broadcast. This code creates an intent with a unique, specific action, puts an extra message on it, and calls the sendBroadcast() Method. Putting an extra message on the intent is optional; many times, receiving an intent is sufficient for a receiver, and an extra is not needed.

### Listing 14–1. Broadcasting an Intent

```java
private void testSendBroadcast(Activity activity) {
    //Create an intent with an action
    String uniqueActionString = "com.androidbook.intents.testbc";
    Intent broadcastIntent = new Intent(uniqueActionString);
    broadcastIntent.putExtra("message", "Hello world");
    activity.sendBroadcast(broadcastIntent);
}
```

In the code in Listing 14–1, the action is an arbitrary identifier that is suitable for your needs. To make this action string unique, you may want to use a namespace similar to a Java class. Now, let’s look at how we can respond to this broadcast intent.

## Coding a Simple Receiver: Sample Code

Listing 14–2 shows how you can code a receiver to respond to the broadcasted intent in Listing 14–1.

### Listing 14–2. Sample Receiver Code

```java
public class TestReceiver extends BroadcastReceiver {
    private static final String tag = "TestReceiver";
    @Override
    public void onReceive(Context context, Intent intent) {
        Utils.logThreadSignature(tag);
        Log.d("TestReceiver", "intent=", intent);
        String message = intent.getStringExtra("message");
        Log.d(tag, message);
    }
}
```