

# Using Intents to Broadcast Events

As a system-level message-passing mechanism, Intents are capable of sending structured messages across process boundaries.

So far you've looked at using Intents to start new application components, but they can also be used to broadcast messages anonymously *between* components with the sendBroadcast method. You can implement

Broadcast Receivers to listen for, and respond to, these broadcast Intents within your applications. Broadcast Intents are used to notify listeners of system or application events, extending the eventdriven programming model between applications.

Broadcasting Intents helps make your application more open; by broadcasting an event using an Intent, you let yourself and thirdparty developers react to events without having to modify your original application. Within your applications, you can listen for Broadcast Intents to replace or enhance native (or third-party) applications or react to system changes and application events.

For example, by listening for the incoming call broadcast, you can modify the ringtone or volume based on the caller.

Android uses Broadcast Intents extensively to broadcast system events like battery-charging levels, network connections, and incoming calls.

### Broadcasting Events with Intents

Broadcasting Intents is actually quite simple. Within your application component, construct the Intent you want to broadcast, and use the sendBroadcast method to send it.

Set the action, data, and category of your Intent in a way that lets Broadcast Receivers accurately determine their interest. In this scenario, the Intent *action string* is used to identify the event being broadcast, so it should be a unique string that identifi es the event. By convention, action strings are constructed using the same form as Java packages, as shown in the following snippet:

public static final String NEW\_LIFEFORM\_DETECTED = "com.paad.action.NEW\_LIFEFORM";

If you wish to include data within the Intent, you can specify a URI using the Intent's data property. You can also include *extras* to add additional primitive values. Considered in terms of an event-driven paradigm, the extras Bundle equates to optional parameters within an event handler.

The skeleton code below shows the basic creation of a Broadcast Intent using the action defi ned previously, with additional event information stored as extras.

Intent intent = new Intent(NEW\_LIFEFORM\_DETECTED); intent.putExtra("lifeformName", lifeformType); intent.putExtra("longitude", currentLongitude); intent.putExtra("latitude", currentLatitude); sendBroadcast(intent);

### Listening for Broadcasts with Broadcast Receivers

Broadcast Receivers are used to listen for Broadcast Intents. To enable a Broadcast Receiver, it needs to be registered, either in code or within the application manifest. When registering a Broadcast Receiver, you must use an Intent Filter to specify which Intents it is listening for.

To create a new Broadcast Receiver, extend the BroadcastReceiver class and override the onReceive event handler as shown in the skeleton code below:

import android.content.BroadcastReceiver; import android.content.Context; import android.content.Intent; public class MyBroadcastReceiver extends BroadcastReceiver { @Override public void onReceive(Context context, Intent intent) { //TODO: React to the Intent received. } } The onReceive method will be executed when a Broadcast Intent is received that matches the Intent Filter used to register the receiver. The onReceive handler must complete within 5 seconds, or the *Application Unresponsive* dialog will be displayed.

Applications with registered Broadcast Receivers do not have to be running when the Intent is broadcast for the receivers to execute. They will be started automatically when a matching Intent is broadcast. This is excellent for resource management as it lets you create event-driven applications that can be closed or killed, safe in the knowledge that they will still respond to broadcast events.

Typically, Broadcast Receivers will update content, launch Services, update Activity UI, or notify the user using the Notifi cation Manager. The 5-second execution limit ensures that major processing cannot, as it should not, be done within the Broadcast Receiver directly.

The following example shows how to implement a Broadcast Receiver. In the following sections, you will learn how to register it in code or in your application manifest.

public class LifeformDetectedBroadcastReceiver extends BroadcastReceiver { public static final String BURN = "com.paad.alien.action.BURN\_IT\_WITH\_FIRE"; @Override public void onReceive(Context context, Intent intent) { // Get the lifeform details from the intent. Uri data = intent.getData(); String type = intent.getStringExtra("type"); double lat = intent.getDoubleExtra("latitude", 0); double lng = intent.getDoubleExtra("longitude", 0); Location loc = new Location("gps"); loc.setLatitude(lat); loc.setLongitude(Ing); if (type.equals("alien")) { Intent startIntent = new Intent(BURN, data); startIntent.putExtra("latitude", lat); startIntent.putExtra("longitude". Ing): context.startActivity(startIntent); } } }

#### Registering Broadcast Receivers in Your Application Manifest

To include a Broadcast Receiver in the application manifest, add a receiver tag within the application node specifying the class name of the Broadcast Receiver to register. The receiver node needs to include an intent-filter tag that specifi es the action string being listened for, as shown in the XML snippet below:

<receiver android:name=".LifeformDetectedBroadcastReceiver"> <intent-filter> <action android:name="com.paad.action.NEW\_LIFEFORM"/> </intent-filter> </receiver> Broadcast Receivers registered this way are always active.

#### Registering Broadcast Receivers in Code

You can control the registration of Broadcast Receivers in code. This is typically done when the receiver is being used to update UI elements in an Activity. In this case, it's good practice to unregister Broadcast Receivers when the Activity isn't visible (or active).

The following code snippet shows how to register a Broadcast Receiver using an IntentFilter:

// Create and register the broadcast receiver. IntentFilter filter = new IntentFilter(NEW\_LIFEFORM\_DETECTED); LifeformDetectedBroadcastReceiver r = new LifeformDetectedBroadcastReceiver(); registerReceiver(r, filter); To unregister a Broadcast Receiver, use the unregisterReceiver method on your application context, passing in a Broadcast Receiver instance, as shown below:

unregisterReceiver(r);

Further examples can also be found in Chapter 8 when you learn to create your own background Services and use Intents to broadcast events back to your Activities.

## Native Android Broadcast Actions

Android broadcasts Intents for many of the system Services. You can use these messages to add functionality to your own projects based on system events such as time-zone changes, data-connection status, incoming SMS messages, or phone calls.

The following list introduces some of the native actions exposed as constants in the Intents class; these actions are used primarily to track device status changes:

□ ACTION\_BOOT\_COMPLETED Fired once when the device has completed its start-up sequence. Requires the RECEIVE\_BOOT\_COMPLETED permission.

□ ACTION\_CAMERA\_BUTTON Fired when the Camera button is clicked.

□ ACTION\_DATE\_CHANGED and ACTION\_TIME\_CHANGED These actions are broadcast if the date or time on the device is manually changed (as opposed to them changing through the natural progress of time).

□ ACTION\_GTALK\_SERVICE\_CONNECTED and ACTION\_GTALK\_SERVICE\_DISCONNECTED These two actions are broadcast each time a GTalk connection is made or lost. More specifi c handling of GTalk messaging is discussed in more detail in Chapter 9.

□ ACTION\_MEDIA\_BUTTON Fired when the Media button is clicked.

□ ACTION\_MEDIA\_EJECT If the user chooses to eject the external storage media, this event is fi red fi rst. If your application is reading or writing to the external media storage, you should listen for this event in order to save and close any open file handles.

ACTION\_MEDIA\_MOUNTED and ACTION\_MEDIA\_UNMOUNTED These two events are broadcast whenever new external storage media are successfully added or removed from the device.

□ ACTION\_SCREEN\_OFF and ACTION\_SCREEN\_ON Broadcasts when the screen turns off or on.

□ ACTION\_TIMEZONE\_CHANGED This action is broadcast whenever the phone's current time zone changes. The Intent includes a time-zone extra that returns the ID of the new java.util.TimeZone.

A comprehensive list of the broadcast actions used and transmitted natively by Android to notify applications of system state changes is available at http://code.google.com/android/reference/ android/content/Intent.html.

Android also uses Broadcast Intents to monitor application-specific events like incoming SMS messages. The actions and Intents associated with these events are discussed in more detail in later chapters when you learn more about the associated Services.