Android Tutorial: BroadcastReceiver

Wolfram Rittmeyer August 8th, 2012

Whenever you want to know about system wide events you need to implement and register a `BroadcastReceiver`. From then on your receiver gets notifications whenever the system event, for which it is registered, occurs.
`BroadcastReceivers` are one of Android's four standard components.

What are BroadcastReceivers good for?

The fact that Android informs you about system events, offers the possibility for some nice apps as well as for user-friendly additions to existing apps.

Three examples might illustrate this:

1. App2SD uses the ACTION_PACKAGE_ADDED event to be notified whenever a new package has been installed or an existing one has been updated. If this package can be moved to the SDcard, the app adds a notification so that the user knows about this and can move the app easily.

2. Battery Widget Reborn uses the ACTION_BOOT_COMPLETED event to start right away and to display the notification icon containing the current charge level of your device.

3. AirDroid uses the CONNECTIVITY_ACTION event for its widget to display established WIFI connections. If your device has a WIFI connection established, the widget changes its state and with one click on the widget the user sees AirDroid's connection details.

Given the vast amount of broadcast events, the possibilities are sheer endless. It's up to your imagination to make use of broadcast events in ways that support the main task of your app.

Implementing the BroadcastReceiver

When implementing a broadcast receiver you have to do two steps:

- You have to create a subclass of Android's BroadcastReceiver
- You have to implement the `onReceive()` method

I will detail all the necessary steps in the next sections.

Creating a subclass of BroadcastReceiver

Every broadcast receiver must subclass Android's `BroadcastReceiver`. This base class is abstract, which means that you have to provide an implementation of the abstract method `onReceive()`. I will cover this method in the next section.

For now I simply create an empty implementation so that the class compiles - but it doesn't do anything useful yet:

```
package com.grokkingandroid.connectivity;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class ConnectivityChangeReceiver
      extends BroadcastReceiver {

   @Override
   public void onReceive
         (Context context, Intent intent) {
      // react to the event
   }

}
```

Implement the onReceive() method

Android calls the `onReceive()` method on all registered broadcast receivers whenever the event occurs. Say you want to be notified whenever the battery level is getting low. In this case you can register your receiver to the event `Intent.ACTION_BATTERY_LOW`. Now as soon as the battery level falls below a defined threshold your `onReceive()` method will be called.

This method takes two arguments:

The arguments of the onReceive() method

Argument	Meaning
context	The `Context` object you can use to access additional information or to start services or activities
intent	The `Intent` object with the action you used to register your receiver. This object contains additional information that you can use in your implementation.

The Intent object passed into the `onReceive()` method often holds additional information that you can use to determine more about the event. For an example see my post about the connectivity change event.

The Context object on the other hand is necessary if you want to start an Activity or a Service.

Services are probably more common with broadcast receivers as it's quite likely that your user isn't actively using your app at the time the event occurs. Services deserve a blog post on their own. Thus I only show you how to start a service from within your receiver:

```
context.startService
      (new Intent(context, YourService.class));
```

System-wide events that you can register for

A lot of system events are defined as final static fields of the Intent class. Furthermore throughout the API there are many more classes that offer specific broadcast events themself. Some examples are BluetoothDevice or TextToSpeech.Engine and nearly all the Manager classes like UsbManager or AudioManager. Android really offers plenty of events that we can make use of in our apps.

The following list is only a small sample of all available events.

Examples for system events

Event	Usage
Intent.ACTION_BATTERY_LOW	The battery level has fallen below a threshold
Intent.ACTION_BATTERY_OKAY	The battery level has risen again
Intent.ACTION_BOOT_COMPLETED	Android is up and running
Intent.ACTION_DEVICE_STORAGE_LOW	Storage space on the device is getting limited
Intent.ACTION_DEVICE_STORAGE_OK	The storage situation has improved again
Intent.ACTION_HEADSET_PLUG	A headset was plugged in or a previously plugged headset was removed
Intent.ACTION_LOCALE_CHANGED	The user changed the language of the device
Intent.ACTION_MY_PACKAGE_REPLACED	Your app has been updated
Intent.ACTION_PACKAGE_ADDED	A new app has been installed
Intent.ACTION_POWER_CONNECTED	The device has been plugged in
Intent.ACTION_POWER_DISCONNECTED	The device has been disconnected again
KeyChain.ACTION_STORAGE_CHANGED	The keystore changed
BluetoothDevice.ACTION_ACL_CONNECTED	A Bluetooth ACL connection has been established
	The internal audio

AudioManager.ACTION_AUDIO_BECOMING_NOISY	speaker is about to be used instead of other output means (like a headset)

Registering a BroadcastReceiver in the manifest file

As usual broadcast receivers can be configured in the manifest file `AndroidManifest.xml`. A `BroadcastReceiver` that is configured in this way is called statically registered.

You can register your receiver in the manifest file by using the `&lt;receiver&gt;` element:

```
<receiver
      android:name=".BootCompletedReceiver"
      android:enabled="true"
      android:exported="true">
   <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED" />
   </intent-filter>
</receiver>
```

The nested element `&lt;intent-filter&gt;` is used to specify the event the receiver should react to.

You have to refer to the documentation to find the correct value to put here. As mentioned above, Android defines final static fields for broadcast actions and the API always states the value of this constant. Within the manifest file you have to use the value.

If you register your receiver in the manifest file, your receiver will be called whenever the event occurs. This might be what you want - or it might be too often. It depends on the type of event and what you need it for. See more about this below in the section "When to use which method to register".

As soon as the `onReceive()` method is finished, your `BroadcastReceiver` terminates.

Register your BroadcastReceiver programmatically

As an alternative you can register your `BroadcastReceiver` implementation dynamically in your code. You just need to call the `registerReceiver()` method on your `Context` object.

The `registerReceiver()` method takes two parameters:

The arguments of the registerReceiver() method

Argument	Meaning
receiver	The `BroadcastReceiver` you want to register
filter	The `IntentFilter` object that specifies which event your receiver should listen to.

When you register your receiver in this way, it lives for as long as the component lives and Android sends events to this receiver until the creating component itself gets destroyed.

It's your task to handle the lifecycle correctly. Thus when you add a receiver dynamically, take care to unregister the same receiver in the `onPause()` method of your Activity!

I suggest to register the receiver in the `onResume()` method of your Activity and to unregister it in your `onPause()` method:

```
@Override
protected void onPause() {
   unregisterReceiver(mReceiver);
   super.onPause();
}

@Override
protected void onResume() {
   this.mReceiver = new ConnectivityChangeReceiver();
   registerReceiver(
         this.mReceiver,
         new IntentFilter(
               ConnectivityManager.CONNECTIVITY_ACTION));
   super.onResume();
}
```

When to use which method to register

Which method to use for registering your `BroadcastReceiver` depends on what your app does with the system event. I think there are basically two reasons why your app wants to know about system-wide events:

- Your app offers some kind of service around these events
- Your app wants to react graciously to state changes

Examples for the first category are apps that need to work as soon as the device is booted or that must start some kind of work whenever an app is installed. Battery Widget Pro or App2SD are good examples for these kinds of apps. For this type you must register the BroadcastReceiver in the Manifest file.

Examples for the second category are events that signal a change to circumstances your app might rely on. Say your app depends on an established Bluetooth connection. You have to react to a state change - but only when your app is active. In this case there is no need for a statically registered broadcast receiver. A dynamically registered one would be more reasonable.

There are also a few events that you are not even allowed to statically register for. An example for this is the Intent.ACTION_TIME_TICK event which is broadcast every minute. Which is a wise decision because a static receiver would unnecessarily drain the battery.

Avoid long-lasting jobs

You should avoid any long-lasting tasks in your `BroadcastReceiver`. Statically and dynamically registered receivers are treated slightly different. But in both cases you should only do minor tasks in the receiver itself. For any longer tasks you should start a service from within your receiver.

Dynamically registered receivers are called on the UI thread. This means that your receivers blocks any UI handling and thus the `onReceive()` method should be as fast as possible. An "Application Not Responding" error is the worst case - but even without users might get frustrated with a seemingly sluggish app.

Permissions

For some events you need to have the appropriate permissions. For example your app needs the permission `"android.permission.RECEIVE_BOOT_COMPLETED"` if you want to be notified of the `Intent.ACTION_BOOT_COMPLETED` event.

To configure this permission add the following line to your manifest file:

```
<uses-permission
      android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

For most events though no permission is needed. Check the API for the appropriate event to see if you need to configure a permission in your manifest file.

Lessons learned

In this blog post I covered the basics you need to know to use broadcast receivers. Whenever you need to know about a state change of a device, there probably is an event available.

You can register dynamically or statically depending on the situation and use your receiver to start Services and Activities of your app.

I did not cover the ability to send broadcast events yourself. You would do so if certain events are about to happen at different places within your app and if you want to untangle different parts of your app. I will cover this in another post.

Stay tuned!

Wolfram Rittmeyer lives in Germany and has been developing with Java for many years.

He has been interested in Android for quite a while and has been blogging about all kind of topics around Android.

You can find him on Google+ and Twitter.

Tags: Android, BroadcastReceiver, onReceive(), registerReceiver(), System-Events, unregisterReceiver()

« Wrapping Your Head Around Android's Plurals
Android Quick Tip: Formatting Text with Html.fromHtml() »