by Lars Vogel

Android BroadcastReceiver Tutorial

Lars Vogel

Version 2.8

07.01.2013

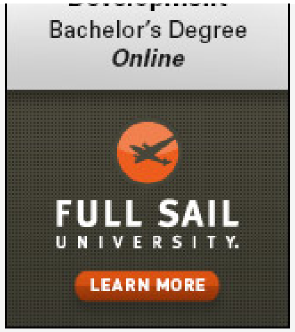
Revision History

Revision 0.1	07.03.2011	Lars Vogel	created
Revision 0.2 - 2.8	08.03.2011 - 07.01.2013	Lars Vogel	bug fixed and enhancements

Using BroadcastReceivers in Android

This tutorial describes how to create and consume Android services. It is based on Eclipse 4.2, Java 1.6 and Android 4.2.

Table of Contents


Bachelor's Degree
Online
FULL SAIL
UNIVERSITY.
LEARN MORE

1. Broadcast receiver

1.1. Definition

A *broadcast receiver* is an Android component which allows to register for system or application events. All registered *receivers* for an event will be notified by Android once this event happens.

For example Android allows that applications can register for the `ACTION_BOOT_COMPLETED` which is fired by the system once the Android system has completed the boot process.

1.2. Implementation

A *broadcast receiver* extends the `BroadcastReceiver` class and which is registered as a receiver in an Android Application via the *`AndroidManifest.xml`* file.

Alternatively to this static registration, you can also register a *broadcast receiver* dynamically via the `Context.registerReceiver()` method.

In an event for which the *broadcast receiver* has registered happens the `onReceive()` method of the *broadcast receiver* is called.

1.3. Unregister dynamically registered receiver

Do not forget to unregister a dynamically created *Broadcast Receiver* by using `Context.unregisterReceiver()` method. Otherwise the system will report a `leaked broadcast receiver` error. For instance if you registered a receive in in `onResume()` methods of your activity, you should unregister it in the `onPause()` method.

1.4. Long running operations

After the `onReceive()` of the `BroadcastReceiver` has finished, the Android system can recycle the `BroadcastReceiver`.

Therefore you cannot perform any asynchronous operation in the `onReceive()` method. If you have potentially long running operations you should trigger a *service* for that.

1.5. Restrictions for defining BroadcastReceiver

As of Android 3.1 the Android system will by default exclude all `BroadcastReceiver` from receiving *Intents* if the corresponding application has never been started by the user or if the user explicitly stopped the application via the Android menu (in Manage Application).

This is an additional security features as the user can be sure that only the applications he started will receive broadcast *Intents*.

1.6. Sticky Broadcast Intents

A normal broadcast `Intent` is not available anymore after is was send and processed by the system. If you use the `sendStickyBroadcast(Intent)` method, the `Intent` is sticky, meaning the `Intent` you are sending stays around after the broadcast is complete.

You can can retrieve that data through the return value of `registerReceiver(BroadcastReceiver, IntentFilter)` . This works also for a null `BroadcastReceiver`.

In all other ways, this behaves the same as `sendBroadcast(Intent)`.

The Android system uses sticky broadcast for certain system information. For example the battery status is send as sticky `Intent` and can get received at any time. The following example demonstrates that.

```
// Register for the battery changed event
IntentFilter filter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);

/ Intent is sticky so using null as receiver works fine
// return value contains the status
Intent batteryStatus = this.registerReceiver(null, filter);

// Are we charging / charged?
int status = batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
boolean isCharging = status == BatteryManager.BATTERY_STATUS_CHARGING
```

```
  || status == BatteryManager.BATTERY_STATUS_FULL;

boolean isFull = status == BatteryManager.BATTERY_STATUS_FULL;

// How are we charging?
int chargePlug = batteryStatus.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
boolean usbCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_USB;
boolean acCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_AC;
```

Sticky *Broadcast Intents* typically require special permissions.

2. Defining custom events and receivers

2.1. Registering Broadcast receiver for custom events

You can register your receivers your your own events in the *`AndroidManifest.xml`* file.

The following *`AndroidManifest.xml`* file shows a `BroadcastReceiver` which is registered to a custom action.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.vogella.android.receiver.own"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <receiver android:name="MyReceiver" >
            <intent-filter>
                <action android:name="de.vogella.android.mybroadcast" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

2.2. Sending Broadcast Intents

The `sendBroadcast()` method allows to send *Intents* to your registered receivers.

You cannot trigger system Broadcasts, the Android system will prevent this.

You can trigger for example an event via the following `sendBroadcast()` method call.

```
Intent intent = new Intent();
intent.setAction("de.vogella.android.mybroadcast");
sendBroadcast(intent);
```

3. System broadcasts

Several system events are defined as final static fields in the `Intent` class. Other Android system classes also define events, e.g. the `TelephonyManager` defines events for the change of the phone state.

The following table lists a few important system events.

Table 1. System Events

Event	Description
Intent.ACTION_BOOT_COMPLETED	Boot completed. Requires the `android.permission.RECEIVE_BOOT_COMPLETED` permission.
Intent.ACTION_POWER_CONNECTED	Power got connected to the device.
Intent.ACTION_POWER_DISCONNECTED	Power got disconnected to the device.
Intent.ACTION_BATTERY_LOW	Battery gets low, typically used to reduce activities in your app which consume power.

Intent.ACTION_BATTERY_OKAY	Battery status good again.

4. Automatically starting Services from a Receivers

To start `Services` automatically after the Android system starts you can register a `BroadcastReceiver` to the Android `android.intent.action.BOOT_COMPLETED` system event. This requires the `android.permission.RECEIVE_BOOT_COMPLETED` permission.

The following AndroidManifest.xml registers a receiver for the `BOOT_COMPLETED` event.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.vogella.android.ownservice.local"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="10" />

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

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name" >
        <activity
            android:name=".ServiceConsumerActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <receiver android:name="MyScheduleReceiver" >
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
        <receiver android:name="MyStartServiceReceiver" >
        </receiver>
    </application>

</manifest>
```

In the `onReceive()` method the corresponding `BroadcastReceiver` would then start the service.

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

public class MyReceiver extends BroadcastReceiver {

  @Override
  public void onReceive(Context context, Intent intent) {
    Intent service = new Intent(context, WordService.class);
    context.startService(service);
  }
}
```

If you application is installed on the SD card, then it is not available after the `android.intent.action.BOOT_COMPLETED` event. Register yourself in this case for the `android.intent.action.ACTION_EXTERNAL_APPLICATIONS_AVAILABLE` event.

Also note that as of Android 3.0 the user needs to have started the application at least once before your application can receive `android.intent.action.BOOT_COMPLETED` events.

5. Pending Intent

A PendingIntent is a token that you give to another application (e.g. Notification Manager, Alarm Manager or other 3rd party applications), which allows this other application to use the permissions of your application to execute a predefined piece of code.

To perform a broadcast via a pending intent so get a PendingIntent via the `getBroadcast()` method of the `PendingIntent` class. To perform an activity via an pending intent you receive the activity via `PendingIntent.getActivity()`.

6. Tutorial: Broadcast Receiver

We will define a broadcast receiver which listens to telephone state changes. If the phone receives a phone call then our receiver will be notified and log a message.

Create a new project `de.vogella.android.receiver.phone`. Create a dummy *activity* as this is required so that the *BroadcastReceiver* also gets activated. Create the following *`AndroidManifest.xml`* file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.vogella.android.receiver.phone"
    android:versionCode="1"
    android:versionName="1.0" >

     <uses-sdk android:minSdkVersion="15" />

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

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name" >
           <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <receiver android:name="MyPhoneReceiver" >
            <intent-filter>
                <action android:name="android.intent.action.PHONE_STATE" >
                </action>
            </intent-filter>
        </receiver>
    </application>



</manifest>
```

Create the `MyPhoneReceiver` class.

```
package de.vogella.android.receiver.phone;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.TelephonyManager;
import android.util.Log;

public class MyPhoneReceiver extends BroadcastReceiver {

  @Override
  public void onReceive(Context context, Intent intent) {
    Bundle extras = intent.getExtras();
    if (extras != null) {
      String state = extras.getString(TelephonyManager.EXTRA_STATE);
      Log.w("DEBUG", state);
      if (state.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
        String phoneNumber = extras
            .getString(TelephonyManager.EXTRA_INCOMING_NUMBER);
        Log.w("DEBUG", phoneNumber);
      }
    }
  }
}
```

Install your application and simulate a phone call via the DDMS perspective in Eclipse. Your receiver is called and logs a message to the console.

7. Tutorial: System Services and BroadcastReceiver

In this chapter we will schedule a `BroadcastReceiver` via the AlertManager. Once called it will use the VibratorManager and a Toast to notify the user.

Create a new project "de.vogella.android.alarm" with the activity "AlarmActivity". Create the following layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <EditText
        android:id="@+id/time"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Number of seconds"
        android:inputType="numberDecimal" >
    </EditText>

    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="startAlert"
        android:text="Start Counter" >
    </Button>

</LinearLayout>
```

Create the following broadcast receiver class. This class will get the Vibrator service.

```
package de.vogella.android.alarm;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Vibrator;
import android.widget.Toast;

public class MyBroadcastReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    Toast.makeText(context, "Don't panik but your time is up!!!!.",
        Toast.LENGTH_LONG).show();
    // Vibrate the mobile phone
    Vibrator vibrator = (Vibrator) context.getSystemService(Context.VIBRATOR_SERVICE);
    vibrator.vibrate(2000);
  }

}
```

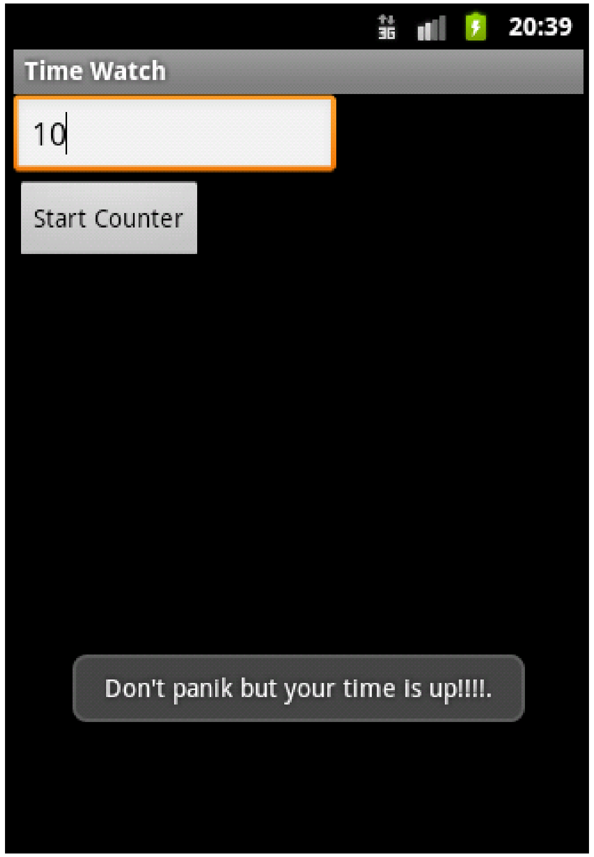
Maintain this class as broadcast receiver in *AndroidManifest.xml* and allow the vibrate authorization.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.vogella.android.alarm"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="15" />

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

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name" >
        <activity
            android:name=".AlarmActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <receiver android:name="MyBroadcastReceiver" >
        </receiver>
    </application>

</manifest>
```

Change the code of your Activity "AlarmActivity" to the following. This activity will create an Intent for the Broadcast receiver and get the AlarmManager service.

```
package de.vogella.android.alarm;

import android.app.Activity;
import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class AlarmActivity extends Activity {

/** Called when the activity is first created. */

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }

  public void startAlert(View view) {
    EditText text = (EditText) findViewById(R.id.time);
    int i = Integer.parseInt(text.getText().toString());
    Intent intent = new Intent(this, MyBroadcastReceiver.class);
    PendingIntent pendingIntent = PendingIntent.getBroadcast(this.getApplicationContext(
), 234324243, intent, 0);
    AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
    alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis()
        + (i * 1000), pendingIntent);
    Toast.makeText(this, "Alarm set in " + i + " seconds",
        Toast.LENGTH_LONG).show();
  }

}
```

Run your application on the device. Set your time and start the alarm. After the defined number of seconds a Toast should be displayed. Keep in mind that the vibrator alarm does not work on the Android emulator.



8. Thank you

Please help me to support this article:

Flattr this!	Donate

9. Questions and Discussion

Before posting questions, please see the **vogella FAQ**. If you have questions or find an error in this article please use the **www.vogella.com Google Group**. I have created a short list **how to create good questions** which might also help you.

10. Links and Literature

10.1. Source Code

Source Code of Examples

10.2. Android Resources

Android Development Tutorial

Android ListView and ListActivity

Android Location API and Google Maps

Android Intents

Android and Networking

Android Background processing with Threads and Asynchronous Task

Remote Messenger Service from Google

10.3. vogella Resources

vogella Training Android and Eclipse Training from the vogella team

Android Tutorial Introduction to Android Programming

GWT Tutorial Program in Java and compile to JavaScript and HTML

Eclipse RCP Tutorial Create native applications in Java

JUnit Tutorial Test your application

Git Tutorial Put everything you have under distributed version control system