Broadcast Receivers

By: Noah Whited

Cool Stuff

- Enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements.
- It can broadcast things to other apps or the system
  - Ex: An alarm to post a notification to inform about an upcoming event
- By delivering the alarm to a BroadcastReceiver of the app, there is no need for the app to remain running until the alarm goes off

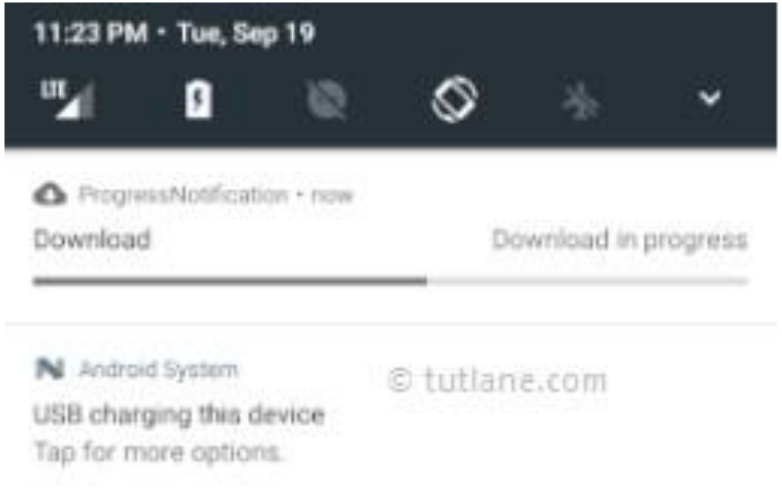
Some more Cool Stuff

Many broadcasts originate from the system...

- ➔ Screen being turned off
- ➔ Booting
- ➔ Battery low
- ➔ Picture captured
- ➔ Volume control
- ➔ Overheating

Broadcasts also come from apps

Downloading:



11:23 PM · Tue, Sep 19

ProgressNotification · now

Download

Download in progress

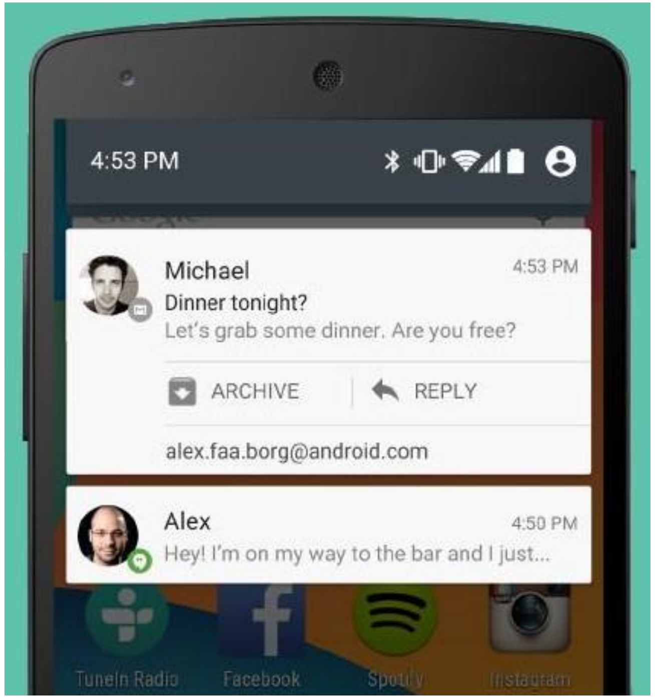
Android System

USB charging this device

Tap for more options

© tutlane.com

Notification:



4:53 PM

Michael 4:53 PM

Dinner tonight?

Let's grab some dinner. Are you free?

ARCHIVE REPLY

alex.faa.borg@android.com

Alex 4:50 PM

Hey! I'm on my way to the bar and I just...

Receiving broadcasts

Manifest-declared

Context-registered

Specify the `<receiver>` element in your app's manifest.

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

Subclass `BroadcastReceiver` and implement `onReceive(Context, Intent)`. The broadcast receiver in the following example logs and displays the contents of the broadcast:

```
private const val TAG = "MyBroadcastReceiver"

class MyBroadcastReceiver : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        StringBuilder().apply {
            append("Action: ${intent.action}\n")
            append("URI: ${intent.toUri(Intent.URI_INTENT_SCHEME)}\n")
            toString().also { log ->
                Log.d(TAG, log)
                Toast.makeText(context, log, Toast.LENGTH_LONG).show()
            }
        }
    }
}
```

The system package manager registers the receiver when the app is installed. The receiver then becomes a separate entry point into your app which means that the system can start the app and deliver the broadcast if the app is not currently running.

The system creates a new BroadcastReceiver component object to handle each broadcast that it receives.

Create an instance of `BroadcastReceiver`

```kotlin
val br: BroadcastReceiver = MyBroadcastReceiver()
```

Create an `IntentFilter` and register the receiver by calling `registerReceiver(BroadcastReceiver, IntentFilter)`:

```kotlin
val filter = IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION).apply {
    addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED)
}
registerReceiver(br, filter)
```

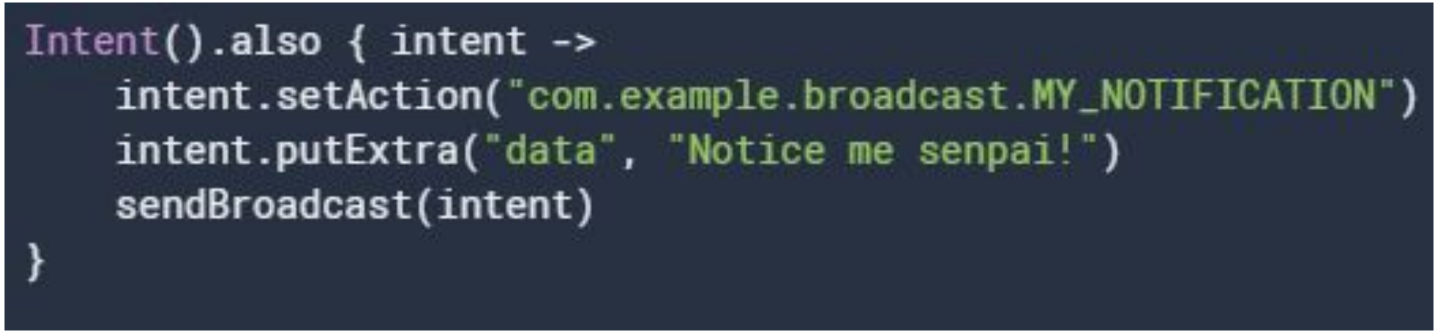
Context-registered receivers receive broadcasts as long as their registering context is valid. For an example, if you register within an `Activity` context, you receive broadcasts as long as the activity is not destroyed. If you register with the Application context, you receive broadcasts as long as the app is running.

To stop receiving broadcasts, call `unregisterReceiver(android.content.BroadcastReceiver)`. Be sure to unregister the receiver when you no longer need it or the context is no longer valid.

Sending broadcasts

- The `sendOrderedBroadcast(Intent, String)` method sends broadcasts to one receiver at a time. As each receiver executes in turn, it can propagate a result to the next receiver, or it can completely abort the broadcast so that it won't be passed to other receivers. The order receivers run in can be controlled with the android:priority attribute of the matching intent-filter; receivers with the same priority will be run in an arbitrary order.
- The `sendBroadcast(Intent)` method sends broadcasts to all receivers in an undefined order. This is called a Normal Broadcast. This is more efficient, but means that receivers cannot read results from other receivers, propagate data received from the broadcast, or abort the broadcast.
- The `LocalBroadcastManager.sendBroadcast` method sends broadcasts to receivers that are in the same app as the sender. If you don't need to send broadcasts across apps, use local broadcasts. The implementation is much more efficient (no interprocess communication needed) and you don't need to worry about any security issues related to other apps being able to receive or send your broadcasts.

How to send a broadcast by creating an Intent and calling `sendBroadcast(Intent)`



```
Intent().also { intent ->
    intent.setAction("com.example.broadcast.MY_NOTIFICATION")
    intent.putExtra("data", "Notice me senpai!")
    sendBroadcast(intent)
}
```

The broadcast message is wrapped in an `Intent` object. The intent's action string must provide the app's Java package name syntax and uniquely identify the broadcast event. You can attach additional information to the intent with `putExtra(String, Bundle)`. You can also limit a broadcast to a set of apps in the same organization by calling `setPackage(String)` on the intent.

Sending with permissions

When you call `sendBroadcast(Intent, String)` or `sendOrderedBroadcast(Intent, String, BroadcastReceiver, Handler, int, String, Bundle)`, you can specify a permission parameter. Only receivers who have requested that permission with the tag in their manifest (and subsequently been granted the permission if it is dangerous) can receive the broadcast. For example, the following code sends a broadcast:

```
sendBroadcast(Intent("com.example.NOTIFY"), Manifest.permission.SEND_SMS)
```

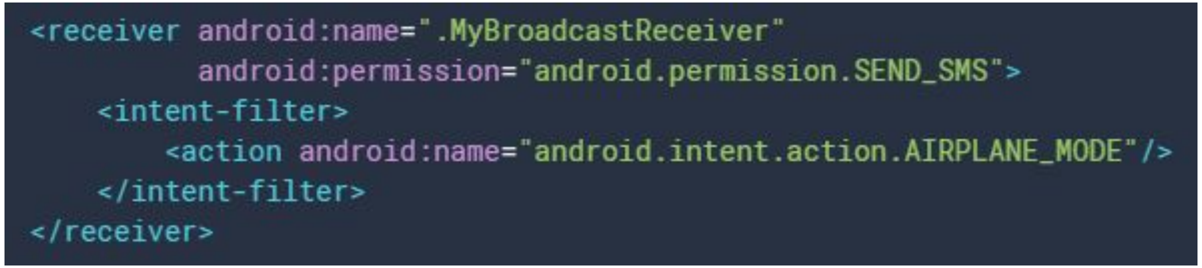
To receive the broadcast, the receiving app must request the permission as shown:

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

Receiving with permissions

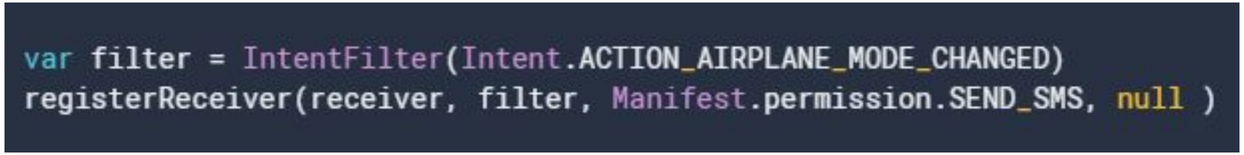
If you specify a permission parameter when registering a broadcast receiver (either with `registerReceiver(BroadcastReceiver, IntentFilter, String, Handler)` or in `<receiver>` tag in your manifest), then only broadcasters who have requested the permission with the `<uses-permission>` tag in their manifest (and subsequently been granted the permission if it is dangerous) can send an Intent to the receiver.

Manifest-declared receiver



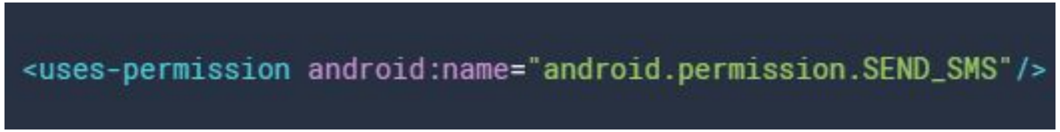
```
<receiver android:name=".MyBroadcastReceiver"
          android:permission="android.permission.SEND_SMS">
    <intent-filter>
        <action android:name="android.intent.action.AIRPLANE_MODE"/>
    </intent-filter>
</receiver>
```

Context-registered receiver



```
var filter = IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED)
registerReceiver(receiver, filter, Manifest.permission.SEND_SMS, null )
```

To be able to send broadcasts to those receivers, the sending app must request the permission as shown below:



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

Broadcasters are kinda cool and they're useful