Mobile Applications

3. Intents and broadcast receivers in Android

Boni García

boni.garcia@uc3m.es

Telematic Engineering Department
School of Engineering

2024/2025

uc3m | Universidad Carlos III de Madrid

Table of contents

1. Introduction

- What we have learned so far:
  - Android apps are created using some building block called app components, namely: activities, services, content providers, and broadcast receivers
  - Among them, we learned that activities represents screens on the user interface (UI) in Android apps
  - In Jetpack Compose, we usually implement single-activity apps in which we use the Navigation Component to navigate between different screens
  - Intents are messaging objects use to request an action from one app component to other (e.g., start a second activity)
- What is new in this unit:
  - We continue learning about **intents**
  - We understand how to create **broadcast receivers**

Table of contents

2. Intents

- **Intents** are a way to express a user's intention to perform a given action
  - In the source code (Java/Kotlin), an intent is a messaging object we can use to request an action from another app component
- Three of the four component types (activities, services, and broadcast receivers) are activated by intents:
  - Starting an activity
    - For example, when the app launcher starts the main activity
  - Starting a service
  - Delivering a broadcast message

https://developer.android.com/training/basics/intents

2. Intents

- There are two types on intents in Android:
  - **Explicit intents**
    - Used for communication within the same app
    - We specify the exact component (by class name) to be invoked
  - **Implicit intents**
    - Used for communication between different apps
    - Do not specify the exact component. Instead, they declare a general action to perform, and the system determines the appropriate component to handle it
- When intent is sent to all apps in the system, we call them as **broadcast intents**
  - Broadcast intents can be explicit or implicit
- When the intents are defined by the Android system, we call them as **system intents**

Fork me on GitHub

2. Intents - Explicit intents

- We already have seen an app using an **explicit intent** to start a second activity:

MainActivity.kt

```kotlin
@Composable
fun MyLayout(modifier: Modifier = Modifier) {
    var text by rememberSaveable { mutableStateOf("") }
    val context = LocalContext.current

    Column(modifier = modifier) {
        Text(text = stringResource(R.string.text_msg))
        TextField(
            value = text,
            onValueChange = { text = it },
            modifier = Modifier.fillMaxWidth()
        )
        Button(
            onClick = {
                val intent = Intent(context, SecondActivity::class.java).apply {
                    putExtra("name", text)
                }
                context.startActivity(intent)
            },
        ) {
            Text(stringResource(R.string.button_msg))
        }
    }
}
```

SecondActivity.kt

```kotlin
class SecondActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MyAppTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    val name = intent.getStringExtra("name")
                    val hello = String.format(stringResource(R.string.hello_msg), name)
                    MyLayout(modifier = Modifier.padding(innerPadding), hello)
                }
            }
        }
    }

}
```

Type of navigation is not necessary when using the Navigation Component

2. Intents - Attributes

- Intents can be created as Kotlin objects as follows:

```
val intent = Intent
```

- Some relevant attributes of the `Intent` objects are:
  - **Extras**: key-value pairs that can be attached to an intent to pass additional data

```
intent.putExtra("name", value)
```

  - **Actions**: desired operation or behavior that we want to perform, represented with a string that unequivocally identified the intent

```
intent.setAction("es.uc3m.android.sendbroadcast")
intent.setAction("android.intent.action.DIAL")
```

Actions can be custom (i.e., defined by an app) or system (i.e. defined by the Android system)

  - **Category**: additional information about the intent purpose

```
intent.addCategory("android.intent.category.DEFAULT")
```

Also, categories can be custom or system

  - **Data**: data on which the action should be performed, represented with and Uniform Resource Identifier (URI)

```
intent.setData(Uri.parse("https://www.google.com"))
```

2. Intents - Actions

System actions are defined as string constants in the class `Intent`, e.g.:

Constant	Value	Description
ACTION_MAIN	`"android.intent.action.MAIN"`	Entry point for an app
ACTION_VIEW	`"android.intent.action.VIEW"`	Display some data to the user (e.g., using a web browser)
ACTION_DIAL	`"android.intent.action.DIAL"`	Dial some number to make a phone call
ACTION_ANSWER	`"android.intent.action.ANSWER"`	Handle an incoming phone call
ACTION_SEARCH	`"android.intent.action.ACTION_SEARCH"`	Perform a search
ACTION_AIRPLANE_MODE_CHANGED	`"android.intent.action.AIRPLANE_MODE"`	Change to airplane mode
ACTION_BATTERY_LOW	`"android.intent.action.BATTERY_LOW"`	Battery level <= 15%
ACTION_POWER_CONNECTED	`"android.intent.action.POWER_CONNECTED"`	The external power has been connected
ACTION_POWER_DISCONNECTED	`"android.intent.action.ACTION_POWER_DISCONNECTED"`	The external power has been removed

https://developer.android.com/reference/kotlin/android/content/Intent

2. Intents - Categories

System categories are defined as string constants in the class `Intent`, e.g.:

Constant	Value	Description
CATEGORY_LAUNCHER	`"android.intent.category.LAUNCHER"`	This category is used with the main activity of an application. It indicates that the activity should be a primary entry point for the app and appear in the app launcher (part of the Android system that manages the home screen and app icons)
CATEGORY_DEFAULT	`"android.intent.category.DEFAULT"`	The default category is often used in conjunction with actions to indicate that the intent is a general-purpose action
BROWSABLE	`"android.intent.category.BROWSABLE"`	This category is often used with activities that can be launched from a web browser. It indicates that the activity can be safely invoked from a browser, allowing deep linking from web pages, i.e., custom URL schemes (e.g., `myapp://`) that, when clicked, open the app directly

https://developer.android.com/reference/kotlin/android/content/Intent

2. Intents - Filters

- Intent filters (`<intent-filter>` in the manifest) define the types of intents that a app component (e.g., an activity) can respond to
- We use intent filters to:
  - Specify the main activity (i.e., the app entry point)
  - Indicate the implicit intents that an activity is interested
  - Indicate the broadcast intents that a Broadcast Receiver is interested
- For instance:

```
<activity
    android:name=".MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

The main activity of an app is defined using this intent filter:

- `android.intent.action.MAIN` : This activity will be the home page
- `android.intent.category.LAUNCHER` : This activity is listed in the app launcher

2. Intents - Implicit intents

- **Implicit intents** in Android are a type of intent that does not specify the exact component (e.g., activity, service, or broadcast receiver) to be invoked
  - Instead, it describes a general action to be performed, and the Android system determines the appropriate component to handle the intent based on the action and data provided
- Implicit intents are commonly used to request actions from other apps or system components, such as opening a web page, sending an email, or sharing content
  - The Android system finds the appropriate component to start by comparing the contents of the intent to the intent filters declared in the manifest file of other apps on the device
  - If the intent matches an intent filter, the system starts that component and delivers it the Intent object

2. Intents - Implicit intents

Fork me on GitHub

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:theme="@style/Theme.MyApp">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".ImplicitActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="es.uc3m.android.implicit" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

The examples repository contains a sample app that manages **implicit intents** (both custom and system)

The activity will be started when using the action `"es.uc3m.android.implicit"` in an implicit intent

Fork me on GitHub

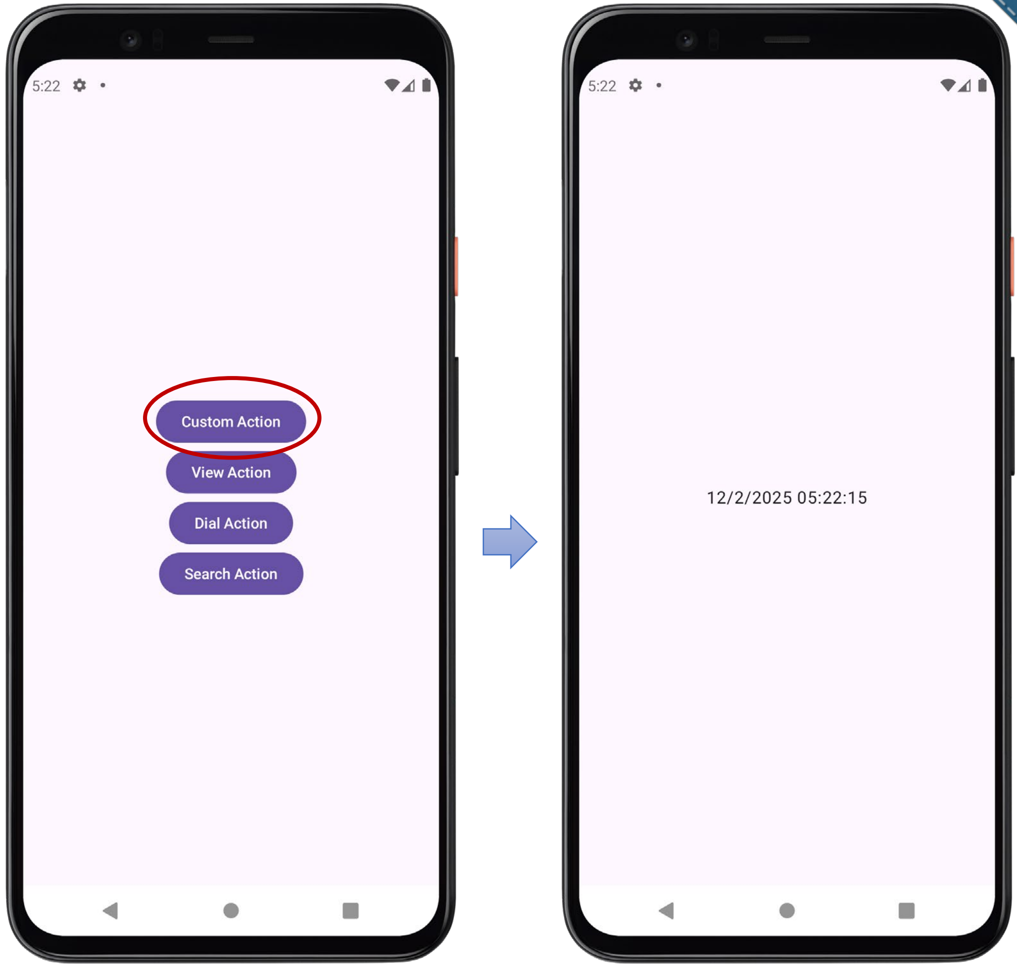
2. Intents - Implicit intents

MainActivity.kt

```
Button(
    onClick = {
        val intent = Intent("es.uc3m.android.implicit")
        val currentDate =
            SimpleDateFormat("dd/M/yyyy hh:mm:ss", Locale.ROOT).format(Date())
        intent.putExtra("date", currentDate)
        context.startActivity(intent)
    },
) {
    Text(stringResource(R.string.button1))
}
```

ImplicitActivity.kt

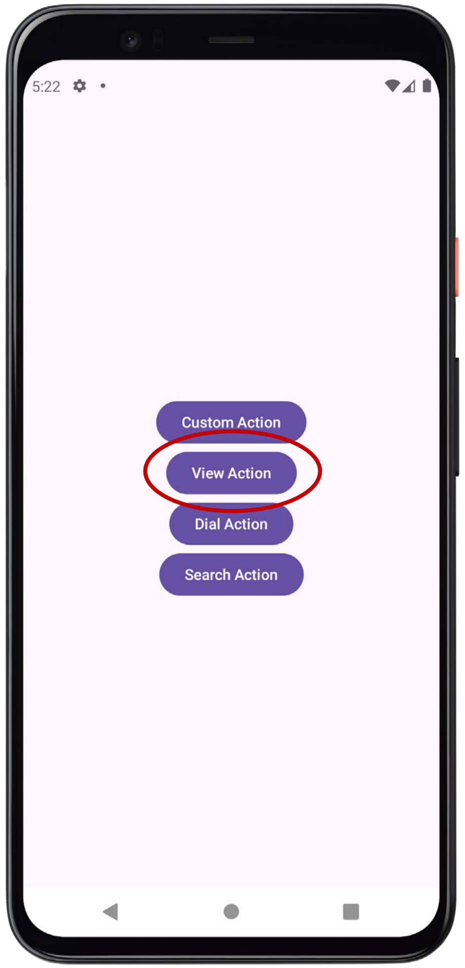
```
class ImplicitActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MyAppTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    val date = intent.getStringExtra("date")
                    MyLayout(String.format("%s", date))
                }
            }
        }
    }

}
```

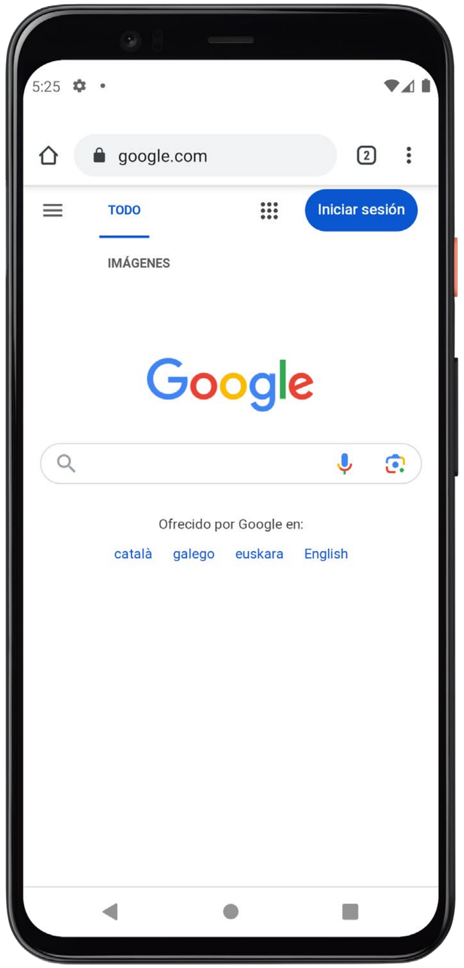


2. Intents - Implicit intents

```kotlin
Button(
    onClick = {
        val intent = Intent(Intent.ACTION_VIEW, Uri.parse("https://www.google.com"))
        context.startActivity(intent)
    },
) {
    Text(stringResource(R.string.button2))
}
```

This app also sends several implicit intents

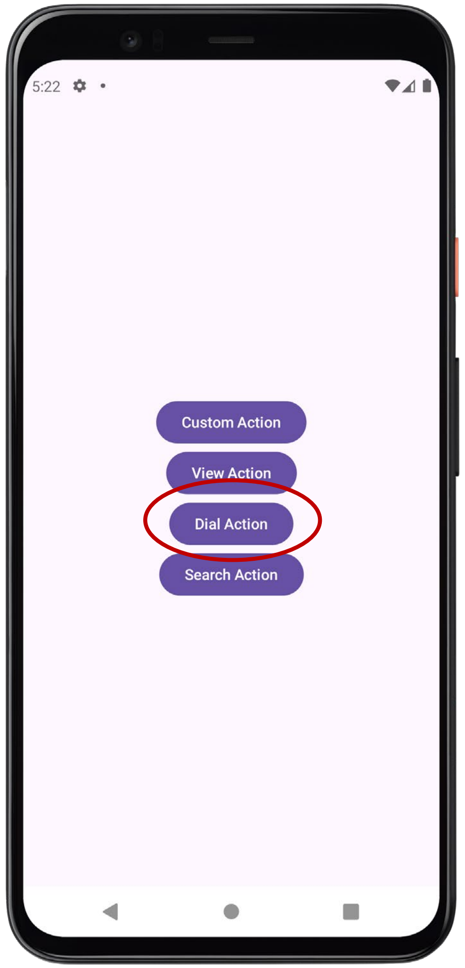


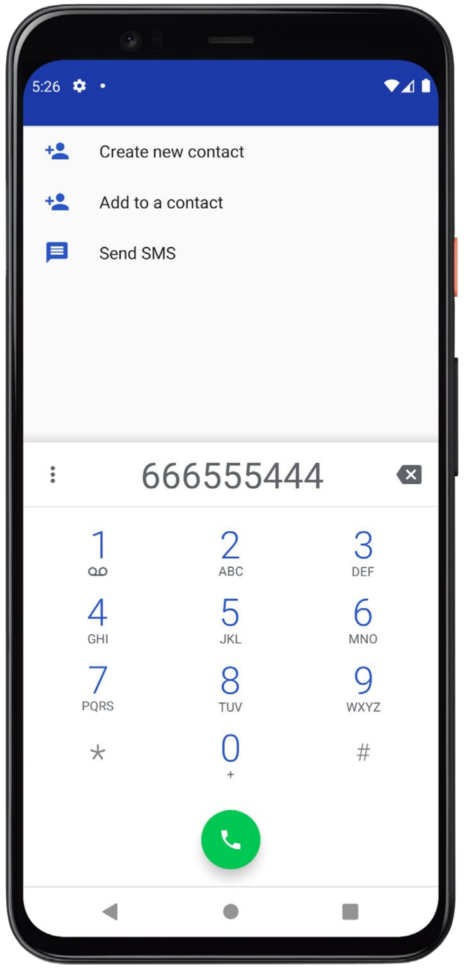


2. Intents - Implicit intents

```
Button(
    onClick = {
        val intent = Intent(Intent.ACTION_DIAL, Uri.parse("tel:666555444"))
        context.startActivity(intent)
    },
) {
    Text(stringResource(R.string.button3))
}
```

This app also sends several implicit intents

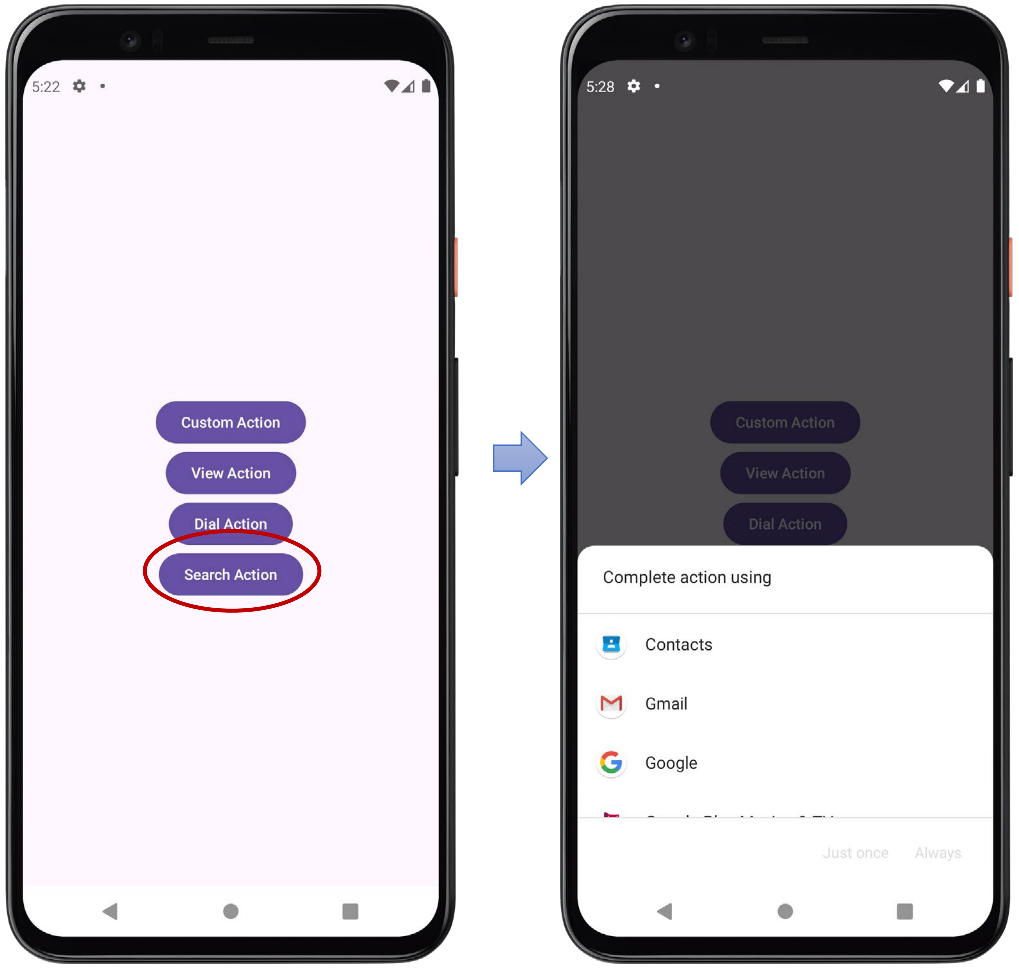




2. Intents - Implicit intents

```
Button(
    onClick = {
        val intent = Intent(Intent.ACTION_SEARCH)
        intent.putExtra(SearchManager.QUERY, "Developing Android apps")
        context.startActivity(intent)
    },
) {
    Text(stringResource(R.string.button4))
}
```

This app also sends several implicit intents



2. Intents - Activity result

- We can use intents to send back a result to the parent activity. This feature can be useful, for example:
  - An app can start a camera app and receive the captured photo as a result
  - An app start the Contacts app and receive the contact details as a result
- The procedure to getting results from activities is the following:
  - From the first activity:
    1. Create an instance of an activity result launcher
    2. Registering a callback for receiving the result
    3. Launching an intent using activity for result
  - From the second activity:
    4. Create a result intent
    5. Define a result code
    6. Finish the activity

2. Intents - Activity result

MainActivity.kt

```
@Composable
fun MyLayout(modifier: Modifier = Modifier) {
    var text by rememberSaveable { mutableStateOf("") }
    val context = LocalContext.current
    val launcher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.StartActivityForResult()
    ) { result ->
        println("The result code is " + result.resultCode)
        val data = result.data?.getStringExtra("message")
        Toast.makeText(context, data, Toast.LENGTH_SHORT).show()
    }

    Column(modifier = modifier) {
        Text(text = stringResource(R.string.text_msg))
        TextField(
            value = text,
            onValueChange = { text = it },
            modifier = Modifier.fillMaxWidth()
        )
        Button(
            onClick = {
                val intent = Intent(context, SecondActivity::class.java).apply {
                    putExtra("name", text)
                }
                launcher.launch(intent)
            },
        ) {
            Text(stringResource(R.string.button1))
        }
    }
}
```

1. Create an instance of an activity result launcher

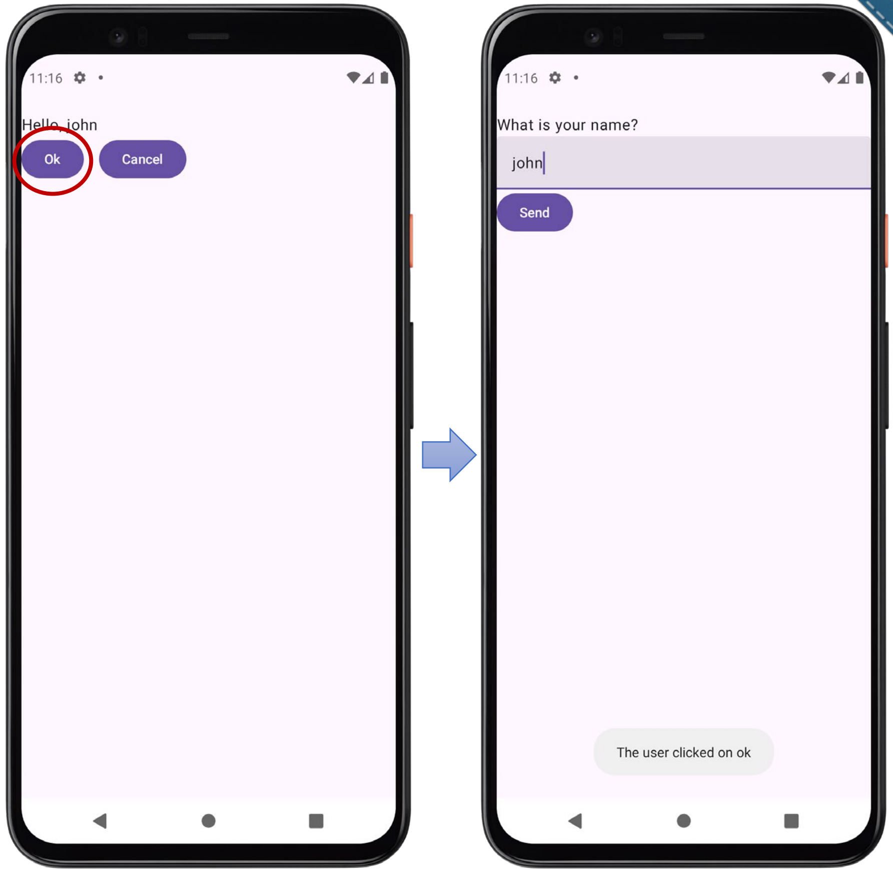
2. Registering a callback for receiving the result

3. Launching an intent using activity for result

2. Intents - Activity result

SecondActivity.kt

```kotlin
@Composable
fun MyLayout(modifier: Modifier = Modifier, text: String) {
    val okMessage = stringResource(R.string.ok_message)
    val cancelMessage = stringResource(R.string.cancel_message)
    val context = LocalContext.current

    Column(modifier = modifier) {
        Text(text = text)
        Row() {
            Button(
                onClick = {
                    if (context is Activity) {
                        finishActivity(context, okMessage, Activity.RESULT_OK)
                    }
                },
            ) {
                Text(stringResource(R.string.button2))
            }
            Button(
                modifier = Modifier.padding(start = 16.dp),
                onClick = {
                    if (context is Activity) {
                        finishActivity(context, cancelMessage, Activity.RESULT_CANCELED)
                    }
                },
            ) {
                Text(stringResource(R.string.button3))
            }
        }
    }
}

private fun finishActivity(context: Activity, resultData: String, resultCode: Int) {
    val resultIntent = Intent().apply {
        putExtra("message", resultData)
    }
    context.setResult(resultCode, resultIntent)
    context.finish()
}
```

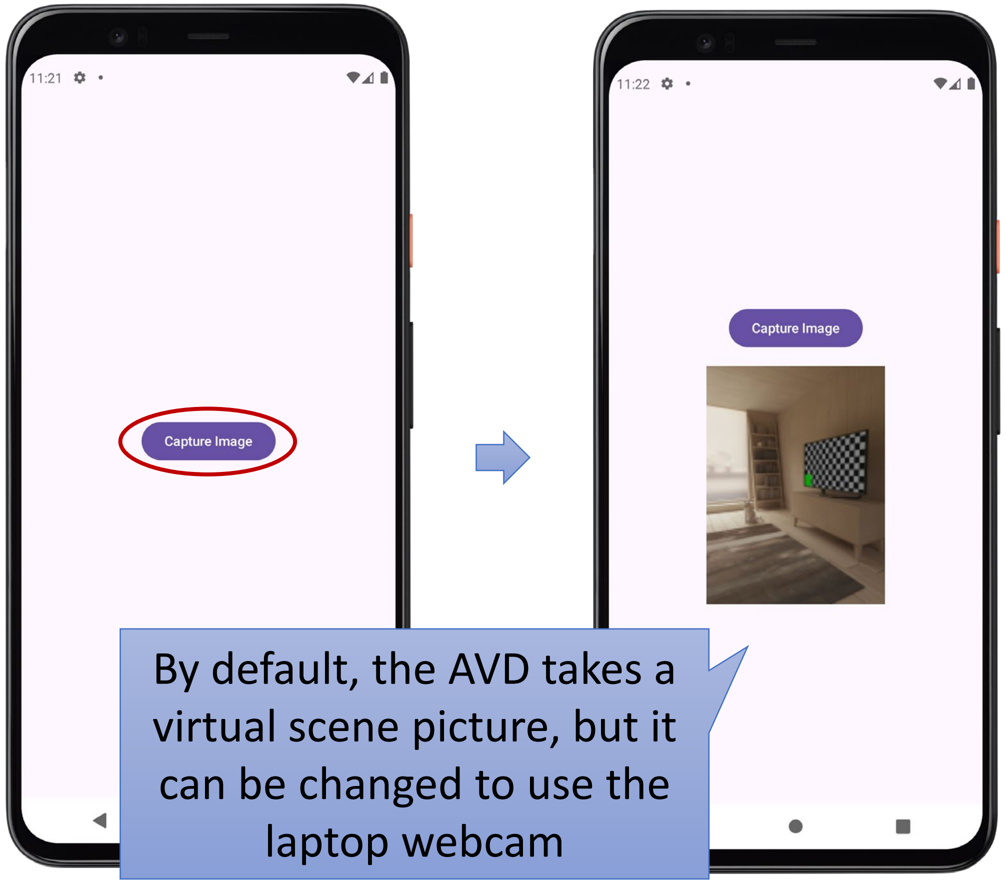


4. Create a result intent
5. Define a result code
6. Finish the activity

2. Intents - Activity result

- A common use case for using activity results is when accesing the device camera from our app to take a picture:

```kotlin
@Composable
fun CameraCaptureScreen() {
    var imageBitmap by remember { mutableStateOf<Bitmap?>(null) }
    val context = LocalContext.current
    val cameraLauncher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.StartActivityForResult()
    ) { result ->
        if (result.resultCode == Activity.RESULT_OK) {
            val bitmap = result.data?.extras?.get("data") as? Bitmap
            imageBitmap = bitmap
        }
    }

    // ...

}

fun launchCamera(launcher: ActivityResultLauncher<Intent>) {
    launcher.launch(Intent(MediaStore.ACTION_IMAGE_CAPTURE))
}
```



By default, the AVD takes a virtual scene picture, but it can be changed to use the laptop webcam

Table of contents

3. Broadcast receivers

- A **Broadcast Receiver** is a type of Android app component that allows to listen and responds to broadcast intents
  - Broadcast Receivers operate in the background and do not have a UI
- **Broadcast intents** are system wide messages sent out from another app or the system to all apps that have registered an interested broadcast receiver
  - Broadcast intents are `Intent` objects that are broadcast via a call to the following Activity methods:
    - *sendBroadcast()* : normal broadcast. This operation is asynchronous, i.e., any receivers receive these intents in no particular order
    - *sendOrderedBroadcast()* : ordered broadcast. This operation is synchronous, i.e., receivers should receive these intents in the order they were sent

https://developer.android.com/develop/background-work/background-tasks/broadcasts

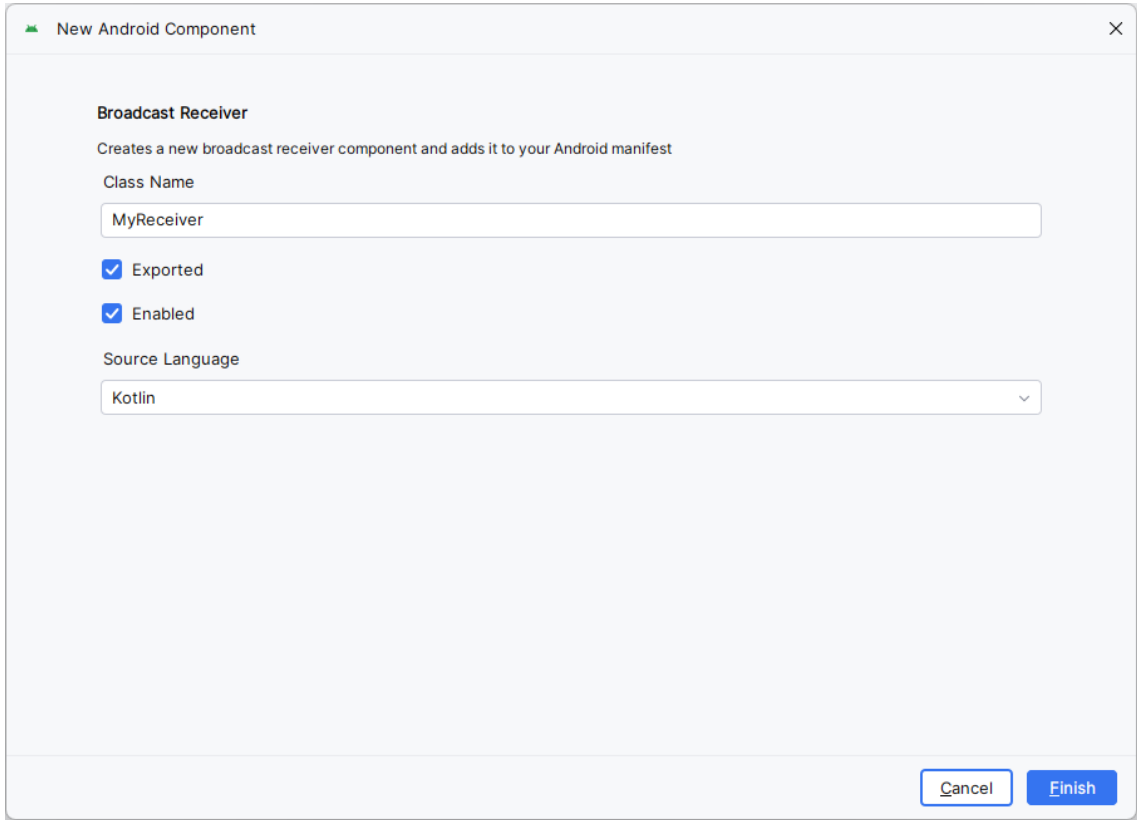
3. Broadcast receivers

- The procedure to create a Broadcast Receiver in an Android app is the following:

1. Create a **Java/Kotlin class** that extends `BroadcastReceiver`
2. Register this Java class in the **manifest** using an `receiver` tag
3. Register an **intent filter** to indicate the types of broadcast intents the Broadcast Receiver it is interested
   - To improve the overall system performance, since Android 8, broadcast receivers must be registered in runtime (i.e., in Java/Kotlin, and not in the manifest)
   - When broadcast intent match, the receiver is invoked by the Android system

3. Broadcast receivers

- We can create an Broadcast Receiver in Android Studio using the wizard File→New→Other→Broadcast Receiver



New Android Component

Broadcast Receiver

Creates a new broadcast receiver component and adds it to your Android manifest

Class Name: MyReceiver

[✓] Exported

[✓] Enabled

Source Language: Kotlin

3. Broadcast receivers

- We can create an Broadcast Receiver in Android Studio using the wizard File→New→Other→Broadcast Receiver

```
class MyReceiver : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        // This method is called when the BroadcastReceiver is receiving an Intent broadcast.
        TODO("MyReceiver.onReceive() is not implemented")
    }
}
```

```
<receiver
    android:name=".MyReceiver"
    android:enabled="true"
    android:exported="true"></receiver>
```

Using this wizard, Android Studio first created the Java class for the receiver and includes a `receiver` tag in the Manifest

Fork me on GitHub

3. Broadcast receivers

```
class MainActivity : ComponentActivity() {

    private val myBroadcastIntent = "es.uc3m.android.sendbroadcast"

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Register my broadcast receiver
        val receiver = MyReceiver()
        val filter = IntentFilter().apply {
            addAction(myBroadcastIntent)
            addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED)
            addAction(Intent.ACTION_BATTERY_LOW)
        }

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            registerReceiver(receiver, filter, RECEIVER_EXPORTED)
        } else {
            @Suppress("UnspecifiedRegisterReceiverFlag")
            registerReceiver(receiver, filter)
        }

        enableEdgeToEdge()
        setContent {
            MyAppTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    MyScreen(action = myBroadcastIntent)
                }
            }
        }
```

The intent filter must be specified in the Java/Kotlin logic. In this example, a custom intent plus two native events will be handled by the broadcast receiver (called `MyReceiver`)

```
@Composable
fun MyScreen(action: String) {
    val context = LocalContext.current

    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Button(
            onClick = {
                val intent = Intent(action)
                context.sendBroadcast(intent)
            },
        ) {
            Text(stringResource(R.string.button))
        }
    }
}
```

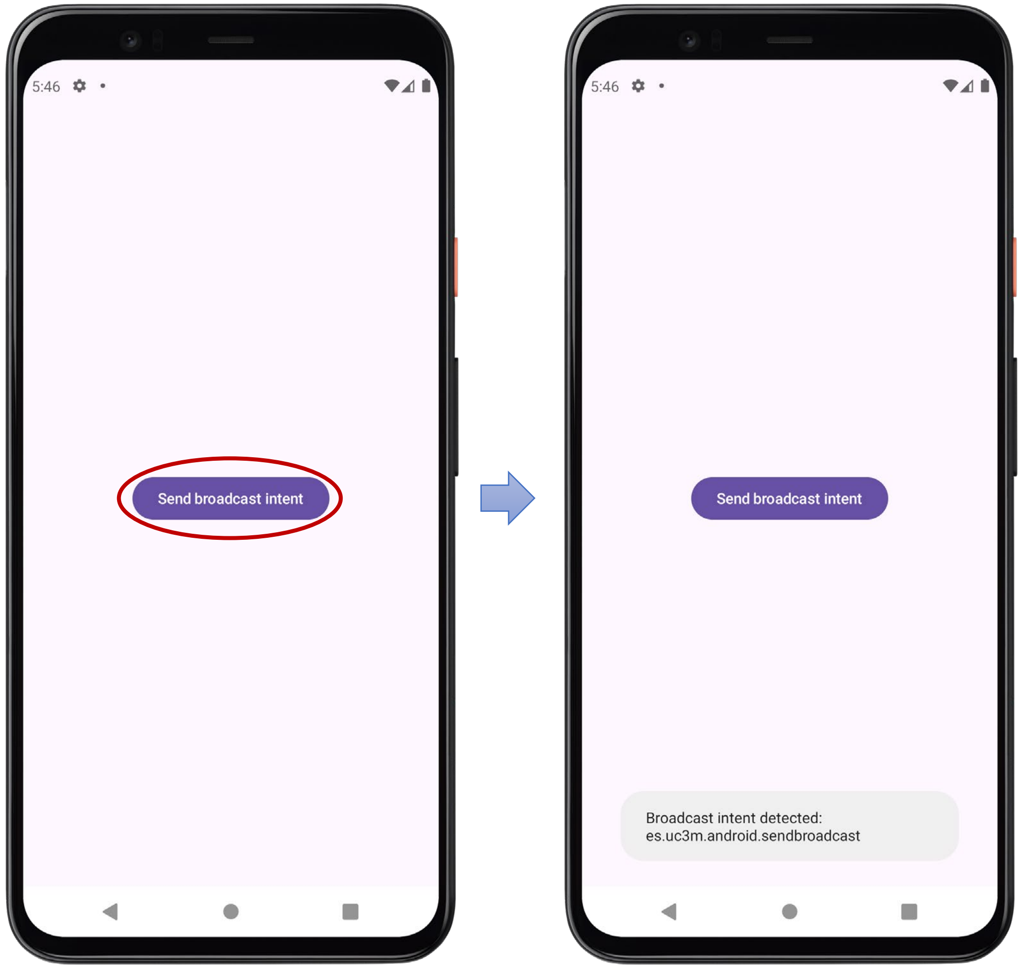
When the user clicks on the UI button, the intent with action `es.uc3m.android.sendbroadcast` is send by broadcast

Fork me on GitHub

3. Broadcast receivers

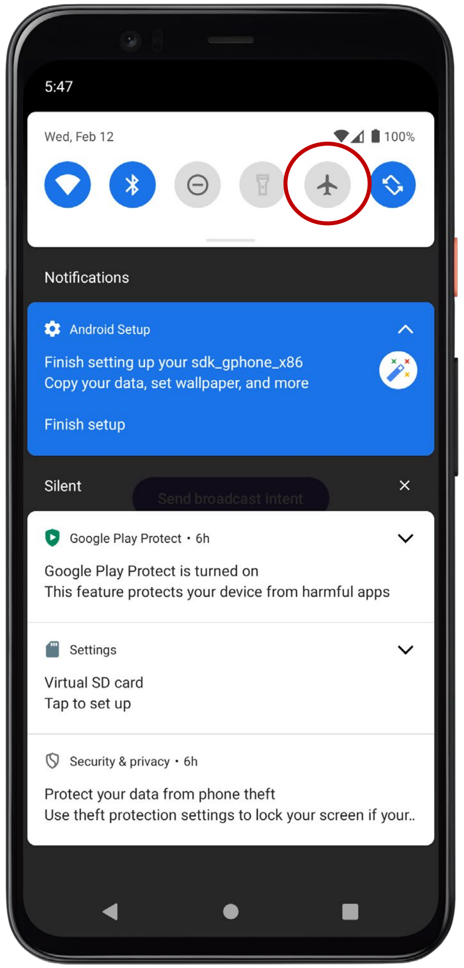
```kotlin
class MyReceiver : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        val message = "Broadcast intent detected: " + intent.action
        Toast.makeText(context, message, Toast.LENGTH_LONG).show()
    }

}
```

The broadcast receiver simply shows a toast to the user display the intent's actions



Broadcast intent detected:
es.uc3m.android.sendbroadcast

3. Broadcast receivers


5:47

Wed, Feb 12 100%

Notifications

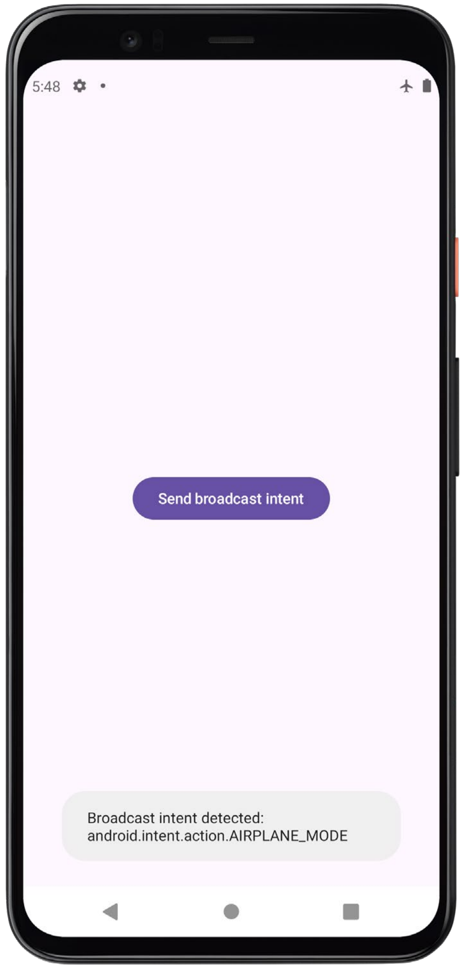
Android Setup
Finish setting up your sdk_gphone_x86
Copy your data, set wallpaper, and more
Finish setup

Silent

Google Play Protect • 6h
Google Play Protect is turned on
This feature protects your device from harmful apps

Settings
Virtual SD card
Tap to set up

Security & privacy • 6h
Protect your data from phone theft
Use theft protection settings to lock your screen if your..


5:48

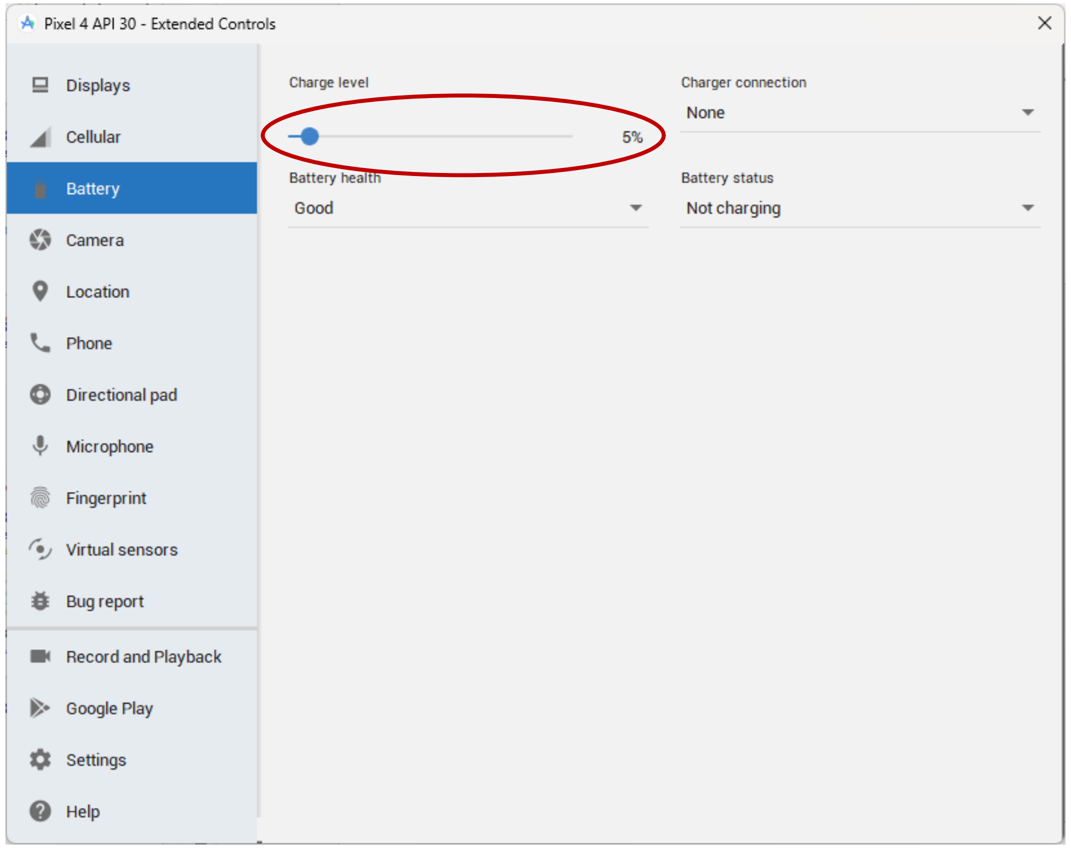
Send broadcast intent

Broadcast intent detected:
android.intent.action.AIRPLANE_MODE

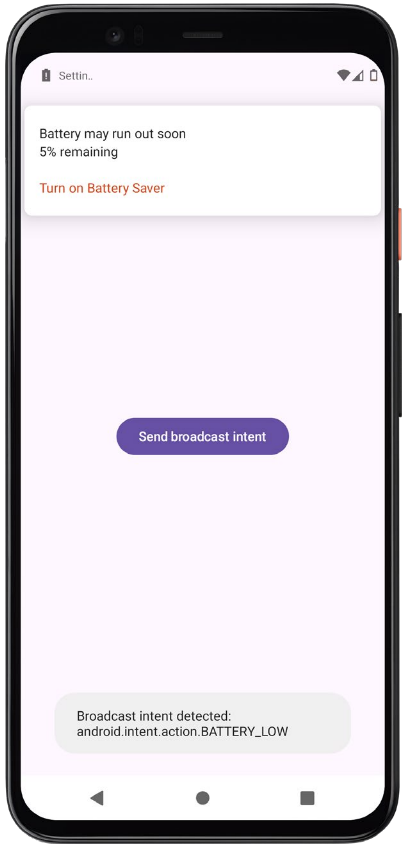
Also, when setting the airplane mode, the broadcast system intent is also received by our broadcast receiver

Fork me on GitHub

3. Broadcast receivers


Pixel 4 API 30 - Extended Controls

Charge level: 5%
Charger connection: None
Battery health: Good
Battery status: Not charging


Battery may run out soon
5% remaining
Turn on Battery Saver

Broadcast intent detected:
android.intent.action.BATTERY_LOW

We can change the battery level with using the extended controls of the virtual device

3. Broadcast receivers

- To simulate the power plug and unplug (and other options), we can use the command-line tool adb
- **Android Debug Bridge** (adb) is a tool for debugging Android devices. It is part of the Android SDK, and it is installed on:
  - Windows: `C:\Users\[user]\AppData\Local\Android\sdk\platform-tools`
  - Mac: `~/Library/Android/sdk/platform-tools/`
  - Linux: `/usr/share/android-sdk/platform-tools/`

```
adb shell dumpsys battery set level 15
```

https://developer.android.com/studio/command-line/adb

Table of contents

4. Takeaways

- Intents are messaging object we use to request an action from one app component to other
- Explicit intents are used to start activities or services defining explicitly the target component that should be invoked
- Implicit intents are used to for communication between different apps by requesting an action without specifying the target component
- Broadcast intents are system wide messages sent out from another app or the system to all app that have registered an interested broadcast receiver
- A broadcast receiver is a type of app component that allows us to listen and responds to broadcast intents
- Android Debug Bridge (adb) is a command-line tool for debugging Android devices