Laboratorio di Applicazioni Mobili
Bachelor in Computer Science &
Computer Science for Management

University of Bologna

Intents & Permissions

Federico Montori
federico.montori2@unibo.it

Table of Contents

Reminder about Activities…

- An Android application can be composed of multiple Activities
- Each activity must be declared in the **AndroidManifest.xml** … unless using an external activity

Add a child element to the <application> tag:

```
<application>
    <activity android:name=".MainActivity" />
    <activity android:name=".ActivityTwo" />
</application>
```

Reminder about Activities…

```
class MainActivity : AppCompatActivity() {
    override fun
    onCreate(savedInstanceState: Bundle?){
        super.onCreate(savedInstanceState)
        setContentView
            (R.layout.activity_main)
    }
}
```

```
class ActivityTwo : AppCompatActivity() {
    override fun
    onCreate(savedInstanceState: Bundle?){
        super.onCreate(savedInstanceState)
        setContentView
            (R.layout.activity_two)
    }
}
```

How to handle an app with more than one activity? How to navigate between them?

Overview on Intents

Intent: facility for late run-time binding between components in the same or different applications.

- Call a component from another component
- Possible to pass data between components
  - Components: **Activities, Services, Broadcast receivers**

- Something like: “Android, please do that with this data”
- Reuse already installed applications and components

It’s a message object

Overview on Intents

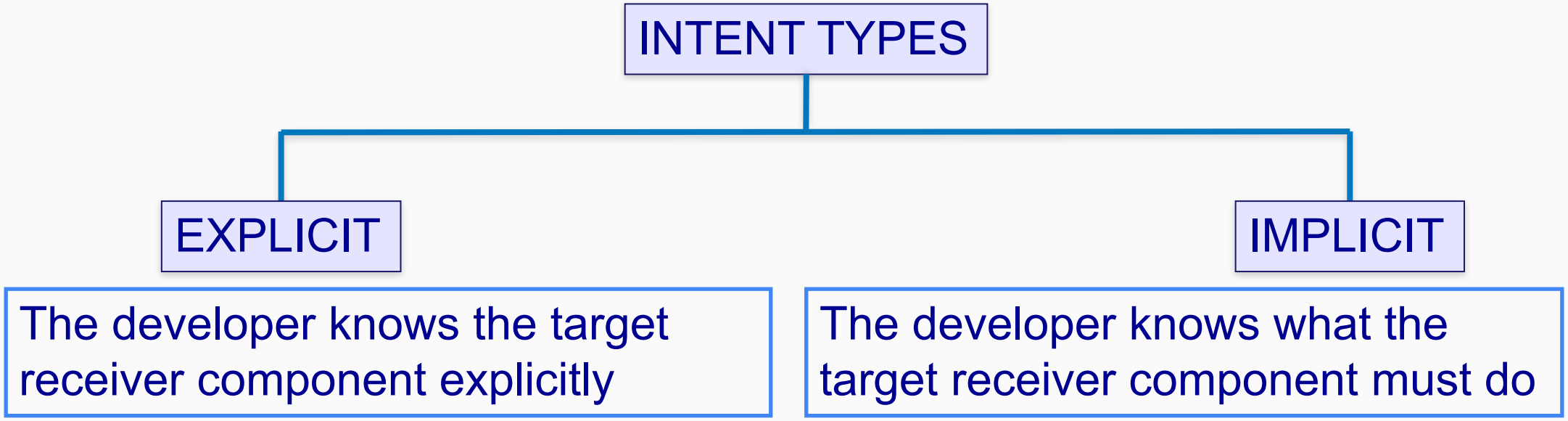
We can think of an "**Intent**" object as a message containing a bundle of information.

- Information of interests for the receiver (e.g. name)
- Information of interests for the Android system (e.g. category).

```
val intent: Intent = Intent()
```

Structure of an Intent

Overview on Intents



Intent Description: Explicit

INTENT TYPES

EXPLICIT

The target receiver is specified through the Component Name
e.g. Used to launch specific Activities

IMPLICIT

The target receiver is specified by data type/names.

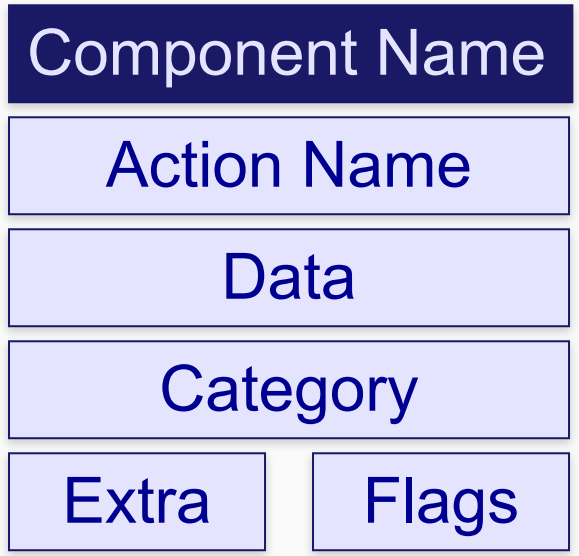
The system chooses the receiver that matches the request.

Intent Description: Explicit

Component that should handle the intent (i.e. the receiver).

- It is optional (set **only** for Explicit intents)

```
intent.setComponent(ComponentName(
    "com.example.MyApplication",
    "com.example.MyApplication.MyActivity")
)
```

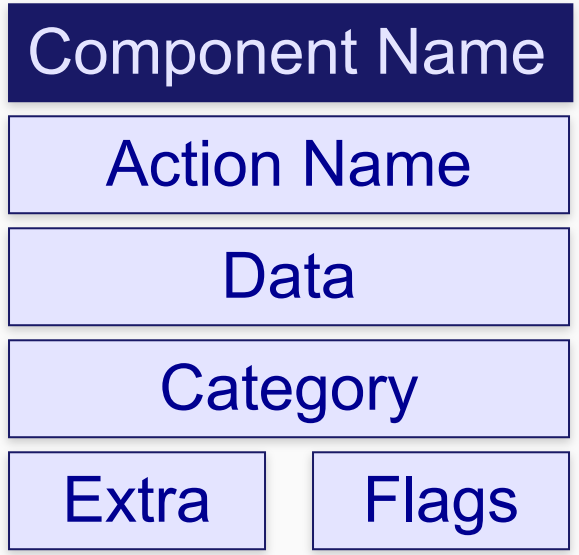


Intent Description: Explicit

Component that should handle the intent (i.e. the receiver).

- It is optional (set **only** for Explicit intents)

```
intent.setComponent(ComponentName(
    this,
    MyActivity::class.java)
)
```



Intent Description: Explicit

How to navigate to a new activity within the same application:

```
val intent: Intent = Intent()
intent.component = ComponentName(this, ActivityTwo::class.java)
startActivity(intent)
```

... or simply:

```
val intent: Intent = Intent(this, ActivityTwo::class.java)
startActivity(intent)
```

With startActivity we are explicitly saying that the component handling the intent **must** be an Activity

Intent Description: Implicit

INTENT TYPES

EXPLICIT

The target receiver is specified through the Component Name
e.g. Used to launch specific Activities

IMPLICIT

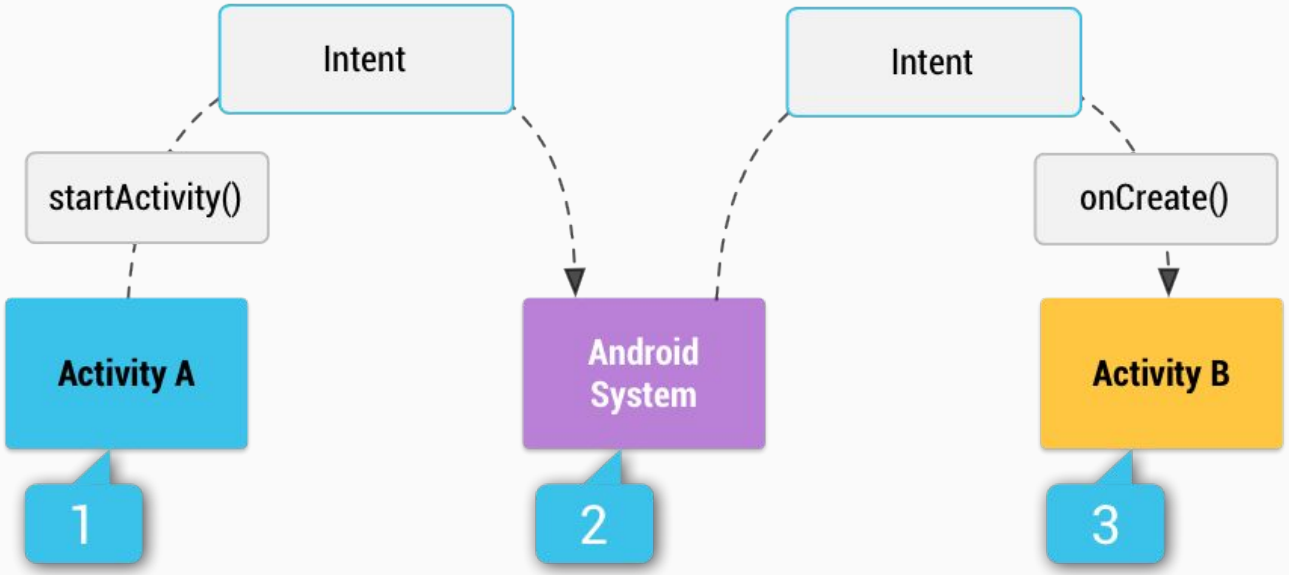
The target receiver is specified by data type/names.

The system chooses the receiver that matches the request.

Intent Description: Implicit

- **Implicit Intents** do not name a target (component name is left blank)
- When an Intent is launched, Android checks out which Component can handle the Intent
  - If one is found, such component is started
  - If two or more are found, the user must choose which one
- Binding does not occur at compile time, nor at install time, but at run-time (late run-time binding)

Intent Description: Implicit



- Activity A fires an Intent
- Android System looks for suitable activities by looking at the manifests of all apps
- When one is found, it is called
- If multiple are found, a choice dialog is displayed

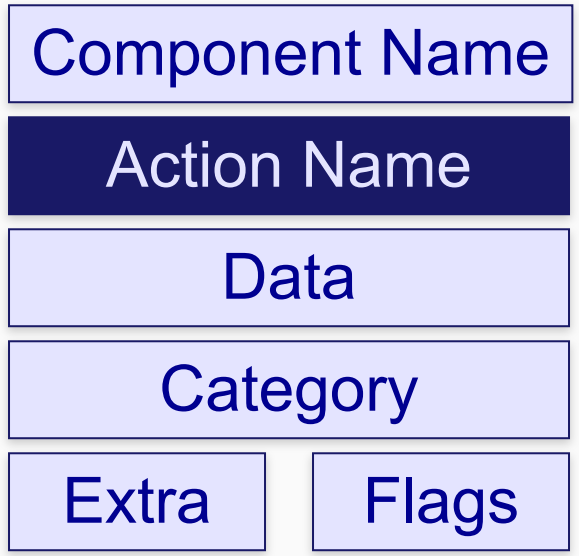
Intent Action

A String naming the action to be performed

- Mandatory for Implicit intents
- Can be defined by the developer or one of the many predefined ones

```
intent.action = Intent.ACTION_EDIT

intent.action =
    "com.example.MyApplication.MY_ACTION"
```



Intent Action

You can use Actions to determine what the called Activity must do. Simple examples:

- **ACTION_VIEW** is called when the receiving Activity shows something to the user (e.g. a photo in the gallery, an address on the map).
- **ACTION_SEND** is called when the receiving Activity is able to send the data received through the Intent using some dedicated channel (e.g. an e-mail or a message in a social app).

Intent Action

Predefined actions
(https://developer.android.com/reference/android/content/Intent)

Action Name	***Description***
ACTION_EDIT	*Display data to edit*
ACTION_MAIN	*Start as a main entry point, does not expect to receive data.*
ACTION_PICK	*Pick an item from the data, returning what was selected.*
ACTION_VIEW	*Display the data to the user*
ACTION_SEARCH	*Perform a search*
ACTION_SEND	*Send some data through another component*

Intent Action

Predefined actions
(https://developer.android.com/reference/android/content/Intent)

Action Name	*Description*
ACTION_IMAGE_CAPTION	*Open the camera and receive a photo*
ACTION_VIDEO_CAPTION	*Open the camera and receive a video*
ACTION_DIAL	*Open the phone app and dial a phone number*
ACTION_SENDTO	*Send an email (email data contained in the extra)*
ACTION_SETTINGS	*Open the system setting*
ACTION_WIRELESS_SETTINGS	*Open the system setting of the wireless interfaces*
ACTION_DISPLAY_SETTINGS	*Open the system setting of the display*

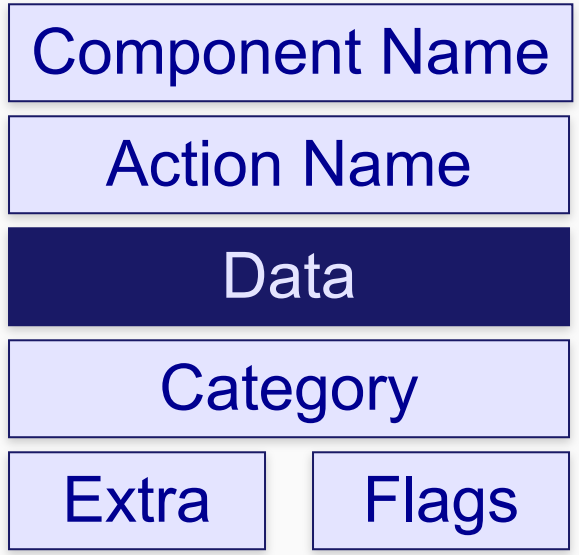
Intent Data

Data passed from the caller to the called Component.

- Def. of the data (URI) - setData()
- Type of the data (MIME type) - setType()
  - Multipurpose Internet Mail Extension

```
intent.data = "https://www.unibo.it/"
intent.type = "text/html"
```

Do not call setData() and setType() if you need to set both because they nullify each other: call setDataAndType()



Intent Data

Data is specified by a name and/or type

name: Uniform Resource Identifier (URI):

scheme://host:port/path

- tel://+1-330-555-0125
- content://contacts/people
- http://www.cs.unibo.it/

An Uri starting with “content” means that the data is stored on the device.

Intent Data

Data is specified by a name and/or type

type: MIME (Multipurpose Internet Mail Extensions)-type
Composed by two parts: a type and a subtype

image/gif	image/jpeg	image/png	image/tiff
text/html	text/plain	text/javascript	text/css
video/mp4	video/mpeg4	video/quicktime	video/ogg

application/vnd.google-earth.kml+xml

Intent Category

A String that gives additional information about the action to execute.

- addCategory()
- for special intents that have additional features to consider

```
intent.addCategory(Intent.CATEGORY_BROWSABLE)
```

Component Name

Action Name

Data

Category

Extra

Flags

Intent Category

Predefined categories
(https://developer.android.com/reference/android/content/Intent)

Category Name	*Description*
CATEGORY_HOME	**The activity displays the HOME screen.**
CATEGORY_LAUNCHER	**The activity is listed in the top-level application launcher, and can be displayed.**
CATEGORY_PREFERENCE	**The activity is a preference panel.**
CATEGORY_BROWSABLE	**The activity can be invoked by the browser to display data referenced by a link.**

Intent Extras

Additional information that should be delivered to the handler (e.g. parameters).

- Key-value pairs
  - putExtras()
  - getExtras()

```
val intent: Intent = Intent(Intent.ACTION_SEND)
intent.putExtra
    (Intent.EXTRA_EMAIL,"federico.montori2@unibo.it")
```

Intent Extras

Extras can be predefined (for most Actions there are defined extras that are expected)

- e.g. for ACTION_SEND you can specify the recipient with EXTRA_EMAIL and the subject with EXTRA_SUBJECT

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

You can specify your own as long as the package is specified

```
const val EXTRA_BASS = "com.example.MyApplication.BASS_NOTE"
```

Intent Flags

A bitwise OR of Integers containing additional information that instructs Android how to launch a component, and how to treat it after executed.

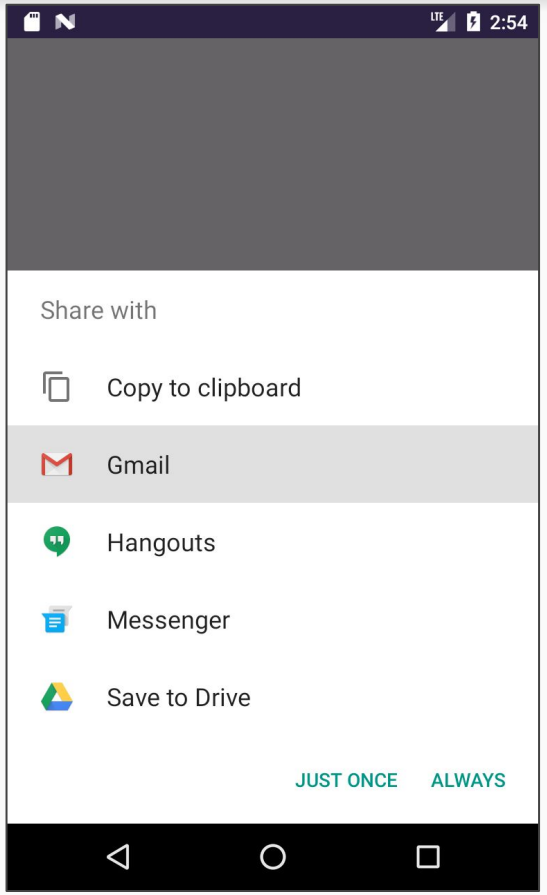
```
intent.flags =
    Intent.FLAG_ACTIVITY_NEW_TASK or
    Intent.FLAG_ACTIVITY_NO_ANIMATION
```

- Component Name
- Action Name
- Data
- Category
- Extra
- **Flags**

Intent Resolution: sender side

Implicit intents are very useful to re-use code and to launch external applications … how do we know who responds?

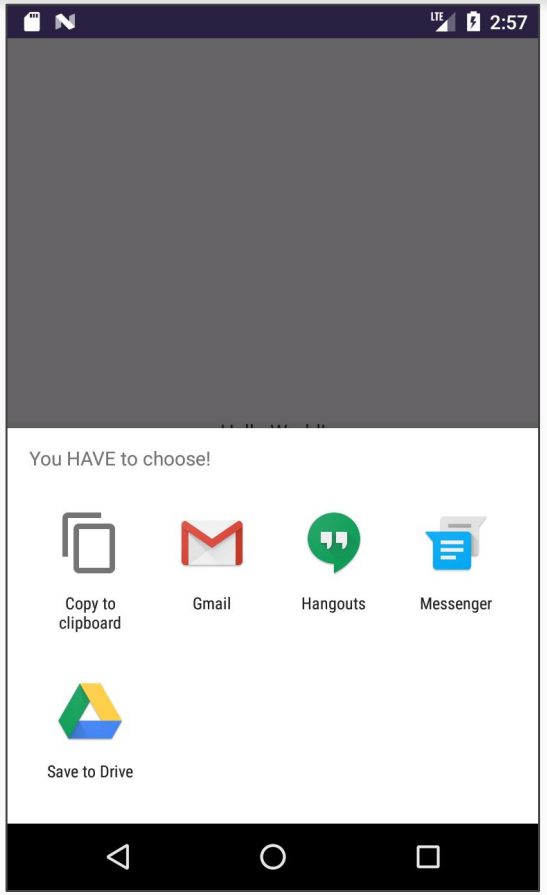
```
val intent: Intent = Intent(Intent.ACTION_SEND)
intent.putExtra(Intent.EXTRA_TEXT, "Hello World!")
intent.type = "text/plain"
if (intent.resolveActivity(packageManager) != null)
    startActivity(intent)
```



Intent Resolution: sender side

We can force the generation of the chooser even if the user selected a default choice.

```
val intent: Intent = Intent(Intent.ACTION_SEND)
intent.putExtra(Intent.EXTRA_TEXT, "Hello World!")
intent.type = "text/plain"
val chooser =
      Intent.createChooser(intent, "You HAVE to choose!")
if (intent.resolveActivity(packageManager) != null)
            startActivity(chooser)
```



Intent Resolution: receiver side

How to declare which intents a component is able to handle?
<intent-filter> tag in **AndroidManifest.xml**
This activity can capture intents with action *com.example.ACTION_ECHO*:

```
<activity
    android:name=".MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="com.example.ACTION_ECHO" />
    </intent-filter>
</activity>
```

exported indicates whether the activity can be invoked by another application

Intent Resolution: receiver side

It can be much more articulated:

- If you specify more than one intent-filter →
  - Your activity should handle intent received differently (e.g. view or edit an image) as they are different entry points to the activity.
- If you specify more than one instance of the same tag (e.g. more than one action) within the same intent-filter →
  - Your activity should handle each combination of these.

Intent Resolution: receiver side

The **intent resolution process** resolves the Intent-Filter that can handle a given Intent.

Three tests to be passed:

- **Action** field test
- **Category** field test
- **Data** field test

If the Intent-filter passes all the three test, then it is selected to handle the Intent.

Intent Resolution: receiver side

ACTION Test: An intent filter always needs to specify at least one action. The action specified in the Intent must match one of the actions listed in the filter.

- If the filter does not specify any action → FAIL
- An intent that does not specify an action → SUCCESS as as long as the filter contains at least one Action.

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
</intent-filter>
```

Intent Resolution: receiver side

CATEGORY Test: An intent filter *for Activities* always needs to specify at least one category. Every category in the Intent must match a category of the filter.

- If the category is not specified in the Intent, Android assumes it is CATEGORY_DEFAULT → the filter must include this category to handle the intent.

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

Intent Resolution: receiver side

DATA Test: The URI of the intent is compared with the parts of the URI mentioned in the filter (this part might be incomplete or using wildcards such as *).

- Both URI and MIME-types are compared (4 different sub-cases).
- All parts specified by the filter need to be matched by the Intent (not vice-versa).

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.item/*" />
</intent-filter>
```

Intent with results

Activities can be invoked to return results (e.g. pick an image from the gallery)

Sender side: invoke the startActivityForResult (deprecated)

```
val ACTIVITY_CODE = 0
val intent: Intent = Intent(this, ActivityPrefix::class.java)
startActivityForResult(intent, ACTIVITY_CODE)
…
override fun onActivityResult
    (requestCode: Int, resultCode: Int, data: Intent?) {
    // Invoked when SecondActivity completes its operations
}
```

Intent with results

Activities can be invoked to return results (e.g. pick an image from the gallery)

Receiver side: invoke the setResult()

```
val intent = getIntent() // in Kotlin this line is not even needed
setResult(RESULT_OK, intent)
intent.putExtra("response", "whatever you wanted")
finish() // The result is not returned until finish() is called
```

Intent with results

Since **Androidx**, *startActivityForResult()* has been wrapped by

Activity Result API: https://developer.android.com/training/basics/intents/result

- To overcome cases when the calling activity is destroyed and recreated while the called one is running.
- With the Activity Result API basically the callback is registered whenever the caller is recreated, decoupling it from the call itself.
- A whole set of new callback functions

Intent with results

Registering can be called when declaring state variables

- **GetContent** is a default contract constructor that is used to return...content! In this case you get back a Uri from the called Activity
  - The contract specifies the type of input and the type of output
  - You can create your own contracts
- **registerForActivityResult** finally returns a launcher that we can fire

```
val mLauncher: ActivityResultLauncher<String> =
    registerForActivityResult(
        ActivityResultContracts.GetContent(),
        ActivityResultCallback<Uri?>() { uri: Uri? -> /* Handle the Uri */ }
    )
```

Intent with results

```
mLauncher.launch("image/*")
```

Fire it by passing in the data type that you want the user to choose from (following this example, you'd probably open the gallery).

- There are several default contracts…
  - https://developer.android.com/reference/androidx/activity/result/contract/ActivityResultContracts
- What if I want to use it the "classic" way (no need of a contract)?

```
val mLauncher: ActivityResultLauncher<Intent> =
    registerForActivityResult(ActivityResultContracts.StartActivityForResult())
        { result: ActivityResult ->
            if (result.resultCode == RESULT_OK) { intent = result.data } }
mLauncher.launch(Intent(this, ActivityPrefix::class.java))
```

Intent with results

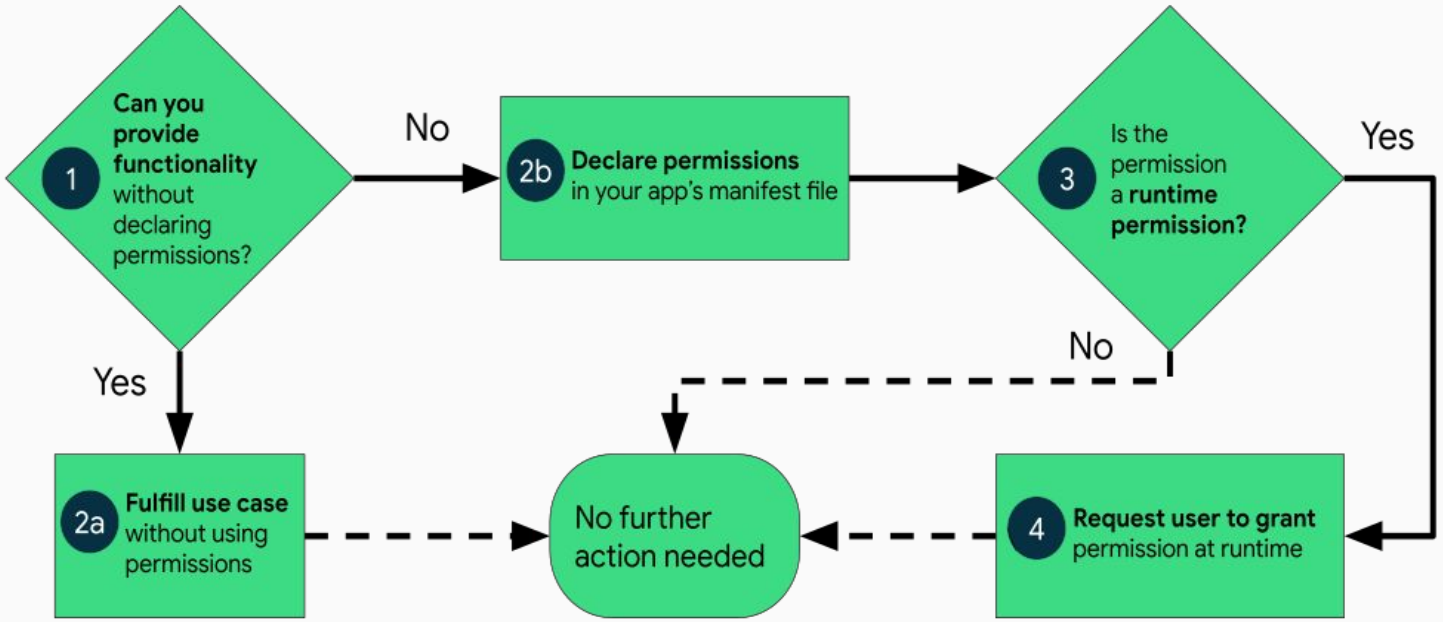
```
/* Creating a custom contract… */
class PickRingtone : ActivityResultContract<Int, Uri?>() {
   override fun createIntent(context: Context, ringtoneType: Int) =
      Intent(RingtoneManager.ACTION_RINGTONE_PICKER).apply {
         putExtra(RingtoneManager.EXTRA_RINGTONE_TYPE, ringtoneType)
      }

   override fun parseResult(resultCode: Int, result: Intent?) : Uri? {
      if (resultCode != Activity.RESULT_OK) {
         return null
      }
      return result?.getParcelableExtra(RingtoneManager.EXTRA_RINGTONE_PICKED_URI)
   }
}
```

Android Permission System

If your app offers functionality that might require access to restricted data or restricted actions, you need to ask permissions.

Permissions have seen changes in the history of Android…



Android Permission System

Always declare them in the manifest:

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

Starting from 6.0

- User can only grant a subset of the permission set
- User can revoke permission after installing the app
- Declare them in the manifest and check if the permission is granted

Android Permission System

Request the permission using the Activity Result API for a solid experience. The launcher should look like this:

```
val requestPermissionLauncher = registerForActivityResult(
    ActivityResultContracts.RequestPermission()
    ) { isGranted: Boolean ->
        if (isGranted) {
            // Permission is granted.
        } else {
            // Permission is denied.
        }
    }
```

You can still use separately, although obsolete:

- **requestPermission**()
- **onRequestPermissionsResult**()

Android Permission System

Request the permission:

```
val REQUEST_CODE = 0
when {
    ContextCompat.checkSelfPermission(this,
        android.Manifest.permission.ACCESS_COARSE_LOCATION
    ) == PackageManager.PERMISSION_GRANTED -> {
        // Permission is already granted.
    }
    ActivityCompat.shouldShowRequestPermissionRationale(
        this, android.Manifest.permission.ACCESS_COARSE_LOCATION) -> {
            // Show an explanation.
        }
    else -> { requestPermissionLauncher.launch(
        Manifest.permission.REQUESTED_PERMISSION) } }
```

WebView

A View that displays web pages, including simple browsing methods (history, zoom in/out/ search, etc).

It is the container for Capacitor Hybrid apps..

Main methods:

- *loadUrl(url)* → load the HTML page at url
- *loadData(data, mimeType, encoding)* → load the HTML page contained in data

WebView

It needs **android.permission.INTERNET**

(one of the few that does not need runtime permissions)

- All it does is pretty much showing the content of a Web page. It’s NOT a browser.
- Useful when you quickly need content that is always up to date.
- In some case better than getting data, parsing and displaying in a layout.

WebView

It is possible to modify the visualization options of a WebView through the **WebSettings** class.

Some options:

- setJavaScriptEnabled(boolean)
- setBuildInZoomControls(boolean)
- setDefaultFontSize(int)

Also, bear in mind that cleartext data is not allowed by default. If you really need it then add to your manifest (**application** tag):

```
android:usesCleartextTraffic="true"
```

WebView

Override the behavior for which links in the WebView open in the WebView (they in fact don’t throw an intent) with a WebViewClient

```
webView.webViewClient = object: WebViewClient() {
    override fun shouldOverrideUrlLoading(
        view: WebView?,
        request: WebResourceRequest?
    ): Boolean {
        if (request?.url?.host == Uri.parse(WEBSITE).host) {
            // This is my website, let the webView handle it
            return false
        } else return super.shouldOverrideUrlLoading(view, request)
    }
}
```

WebView

By default, the WebView UI does not include any navigation button
…However, callbacks methods are defined:

- public void goBack()
- public void goForward()
- public void reload()
- public void clearHistory()

```
override fun onKeyDown
        (keyCode: Int, event: KeyEvent?): Boolean {
        /* Is there a page in the history? */
                if (keyCode == KeyEvent.KEYCODE_BACK &&
                        initialized &&
                        webView.canGoBack()) {
                                webView.goBack()
                                return true
                } else return super.onKeyDown(keyCode, event)
}
```

Questions?

federico.montori2@unibo.it