Programming with Android: Intents and Permissions

Federico Montori

Dipartimento di Scienze dell'Informazione

Università di Bologna

Outline

- What is an **intent**?
- **Intent** description
- Handling **Explicit Intents**
- Handling **implicit Intents**
- **Intent-Resolution** process
- **Intent with results**: Sender side
- **Intent with results**: Receiver side

More on Activities: Activity states

- **Active** (or running)
  - Foreground of the screen (top of the stack)
- **Paused**
  - Lost focus but still visible
  - Can be killed by the system in extreme situations
- **Stopped**
  - Completely obscured by another activity
  - Killed if memory is needed somewhere else

More on Activities: Saving resources

- An activity lifecycle flows between onCreate and onDestroy
- Create, initialize everything you need in onCreate
- Destroy everything that is not used anymore, such as background processes, in onDestroy
- It is fundamental to save the data used by the application in between the state-transitions …

Activities and AndroidManifest.xml

- An Android application can be composed of **multiple Activities** …
- Each activity must be declared in the file: **AndroidManifest.xml**
  - Unless using an external activity...
- Add a **child element** to the <application> tag:

```
<application>
    <activity android:name=".MyActivity" />
    <activity android:name=".SecondActivity" />
</application>
```

Activities and AndroidManifest.xml

- Each activity has its **Java** class and **layout** file.

```
public class MyActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_first);
}
```

```
public class SecondActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
}
```

Intent Definition

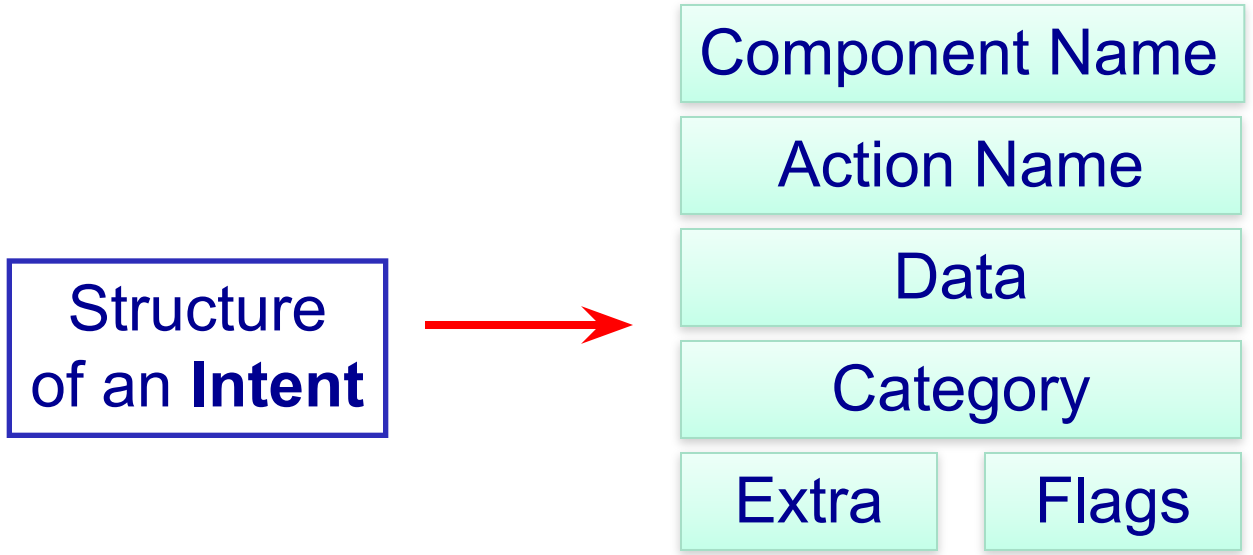
Intent: facility for late run-time binding between components in the same or different applications.

- **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

Intent Definition

- 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).



Intent types

INTENT **TYPES**

EXPLICIT

The target receiver is specified through the **Component Name**

Used to launch specific Activities

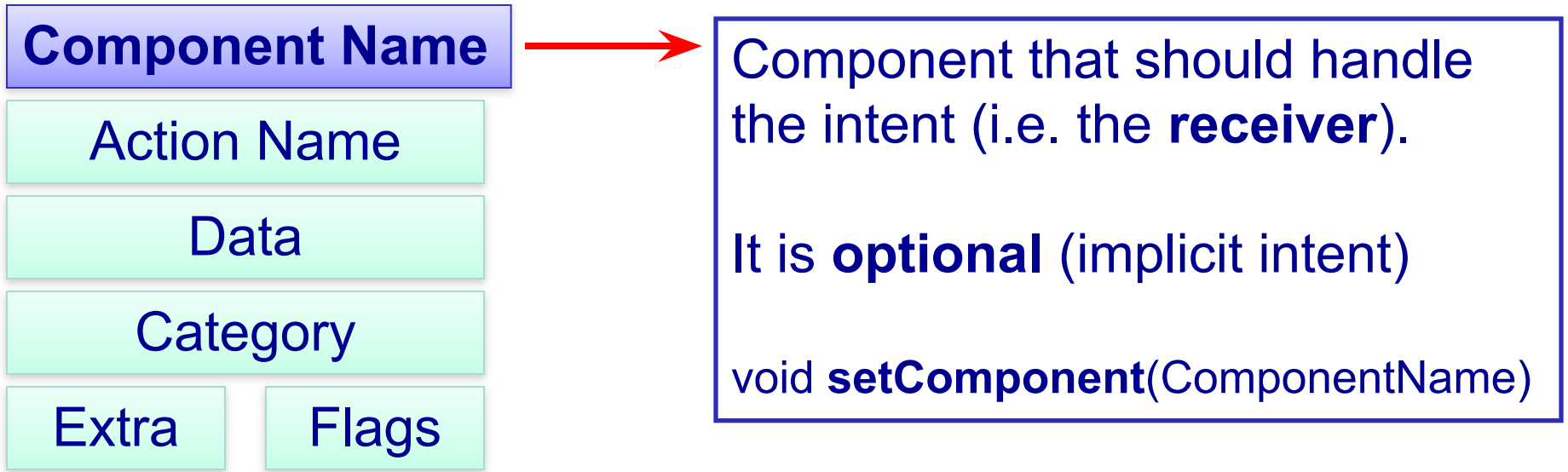
IMPLICIT

The target receiver is specified by **data type/names**.

The system chooses the receiver that matches the request.

Intent Components

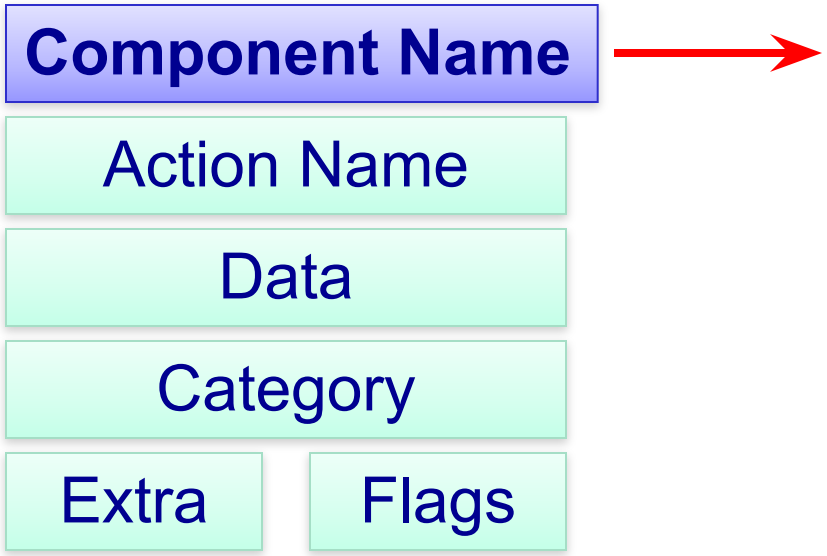
- We can think of an "**Intent**" object as a **message** containing a bundle of information.
  - Information of interests for the receiver (e.g. data)
  - Information of interests for the Android system (e.g. category).



Intent Components

Mostly three typical targets:

- Activities (*you know what they are don't you?*)
- Services (background components)
- BroadcastReceivers (listening components)



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

It is **optional** (implicit intent)

void **setComponent**(ComponentName)
… less used:
void **setClass**(Context, Class)
void **setClassName**(Package, ClassName)

Intent types: Explicit Intents

- **Explicit** Intent: Specify the name of the Activity that will handle the intent.

```
Intent intent = new Intent(this, SecondActivity.class);
startActivity(intent);
```

```
Intent intent = new Intent();
ComponentName component =
    new ComponentName(this, SecondActivity.class);
intent.setComponent(component);
startActivity(intent);
```

Intent with Results

- Activities can return results (e.g. data)
- Sender side: invoke the **startActivityForResult**()
  - **onActivityResult**(int requestCode, int resultCode, Intent data)
  - **startActivityForResult**(Intent intent, int requestCode);

```
Intent intent = new Intent(this, SecondActivity.class);
startActivityForResult(intent, CHOOSE_ACTIVITY_CODE);
…
public void onActivityResult(int requestCode, int resultCode, Intent data)
{
   // Invoked when SecondActivity completes its operations …
}
```

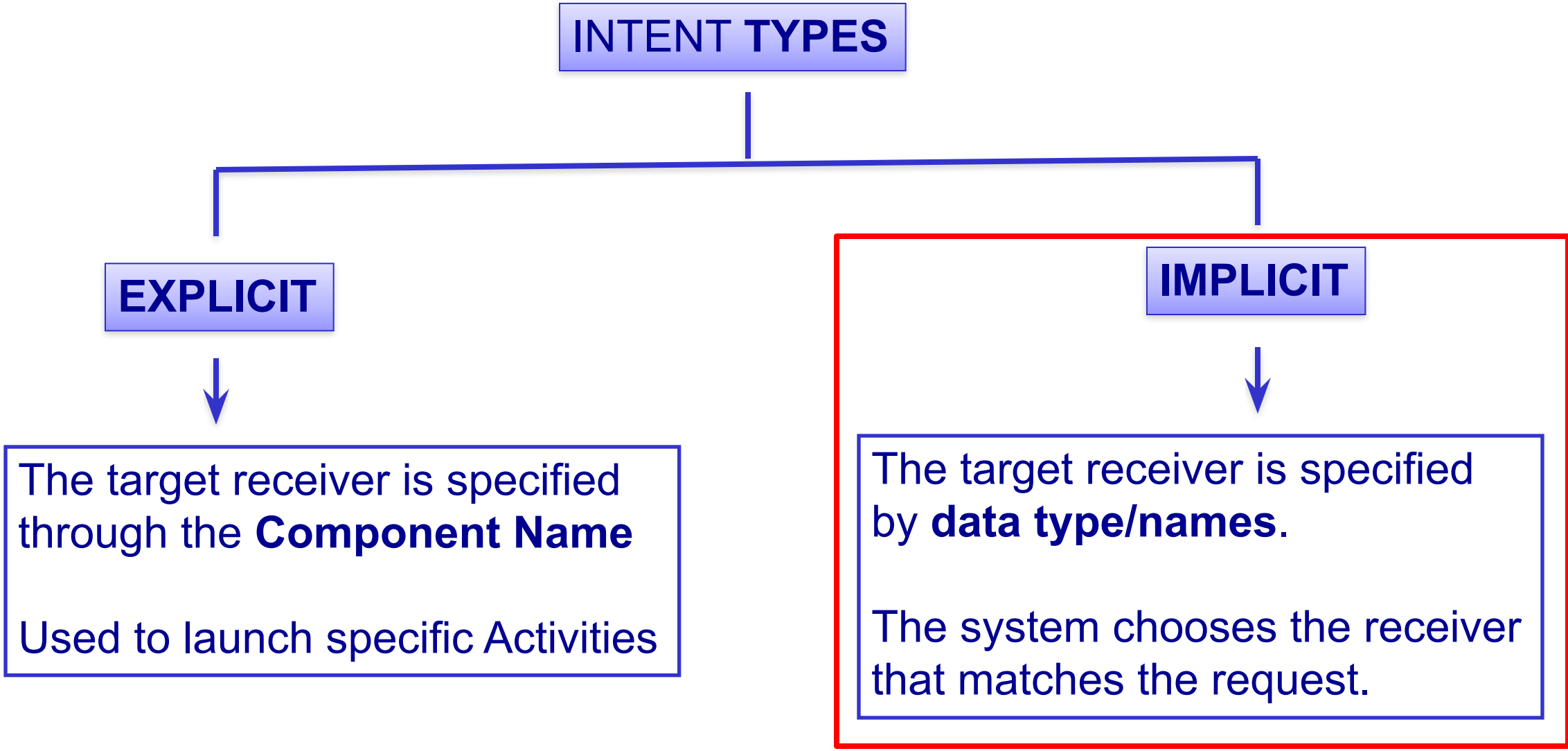
Intent with Results

- Activities can return results (e.g. data)
- Receiver side: invoke the **setResult()**
  - void **setResult**(int resultCode, Intent data)

```
Intent intent = getIntent();
setResult(RESULT_OK, intent);
intent.putExtra("result", resultValue);
finish();
```

- The result is delivered to the caller component only after invoking the **finish()** method!

Intent types



Intent types: Implicit Intents

- **Implicit** Intents: do not name a target (*component name is left blank*) …
- When an Intent is launched, Android checks out which Activities (not only) can handle the Intent...
- If at least one is found, then that Activity is started!
- Binding does not occur at compile time, nor at install time, but at run-time …(***late run-time binding***)

Intent Components: Action

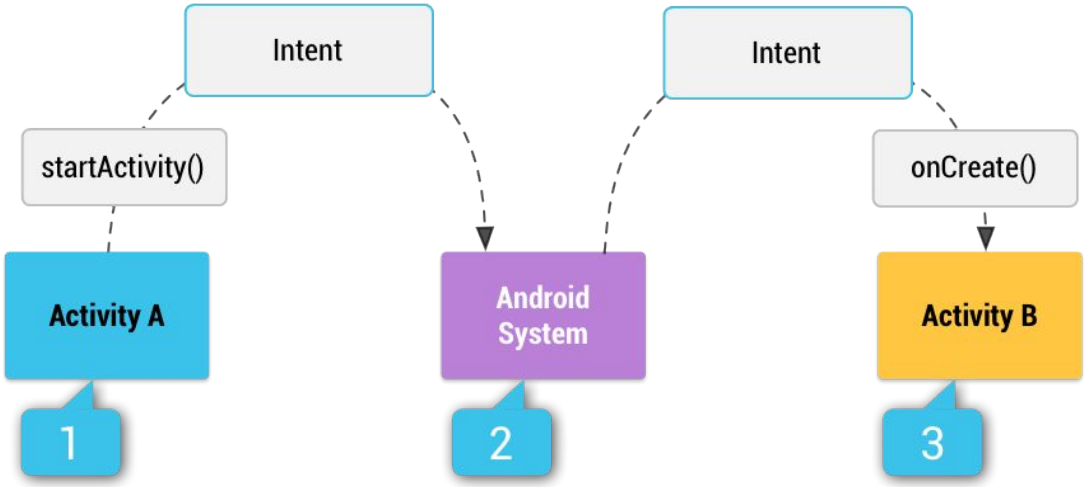
- We can think of an "**Intent**" object as a **message** containing a bundle of information.
  - Information of interests for the receiver (e.g. data)
  - Information of interests for the Android system (e.g. category).

A string naming the **action** to be performed.

Pre-defined, or can be specified by the programmer.

void **setAction**(String)

Intent Components: Action



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

Intent Components: Action

You can use Actions to determine what the called Activity is supposed to 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 Components: Action

- **Predefined** actions (http://developer.android.com/reference/android/content/Intent.html)

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

- **Defined** by the programmer
  - it.example.projectpackage.FILL_DATA (package prefix + name action)

Intent Components: Action

- **Special** actions (http://developer.android.com/reference/android/content/Intent.html)

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 Components: Action

- Example of Implicit Intent that **initiates a web search**.

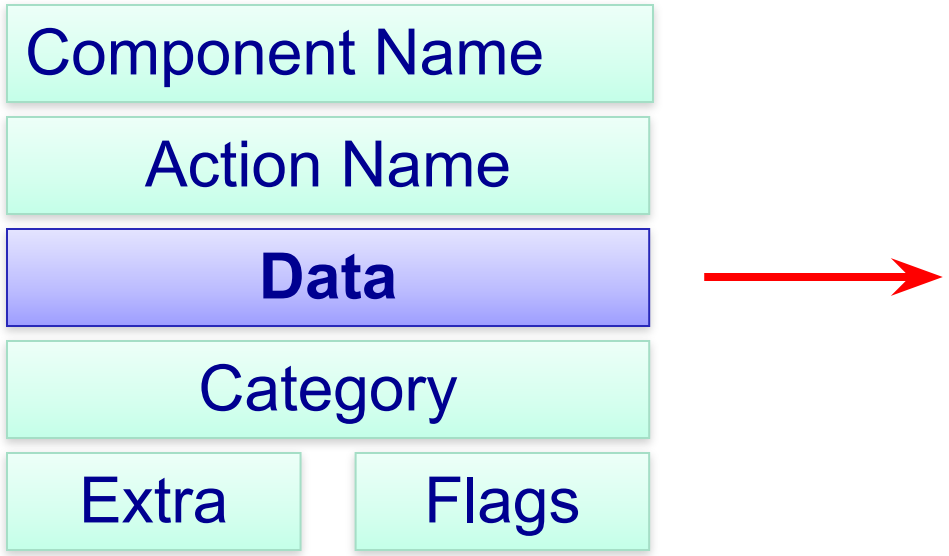
```
public void doSearch(String query) {
    Intent intent = new Intent(Intent.ACTION_SEARCH);
    Intent.putExtra(SearchManager.QUERY, query);
    if (intent.resolveActivity(getPackageManager()) != null)
        startActivity(intent);
}
```

- Example of Implicit Intent that **plays a music file**.

```
public void playMedia(Uri file) {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    if (intent.resolveActivity(getPackageManager()) !=null)
        startActivity(intent);
}
```

Intent Components: Data

- We can think of an "**Intent**" object as a **message** containing a bundle of information.
  - Information of interests for the receiver (e.g. data)
  - Information of interests for the Android system (e.g. category).



Data passed from the caller to the called Component.

Def. of the data (**URI**)
Type of the data (**MIME** type)
Multipurpose Internet Mail Extension

void **setData**(Uri)
void **setType**(String)

Intent Components: Data

- Each data is specified by a **name** and/or **type**.
- **name**: Uniform Resource Identifier (**URI**)
- **scheme://host:port/path**

EXAMPLEs

tel://003-232-234-678

content://contacts/people

http://www.cs.unibo.it/

Intent Components: Data

- Each 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

EXAMPLEs

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 Components: Data

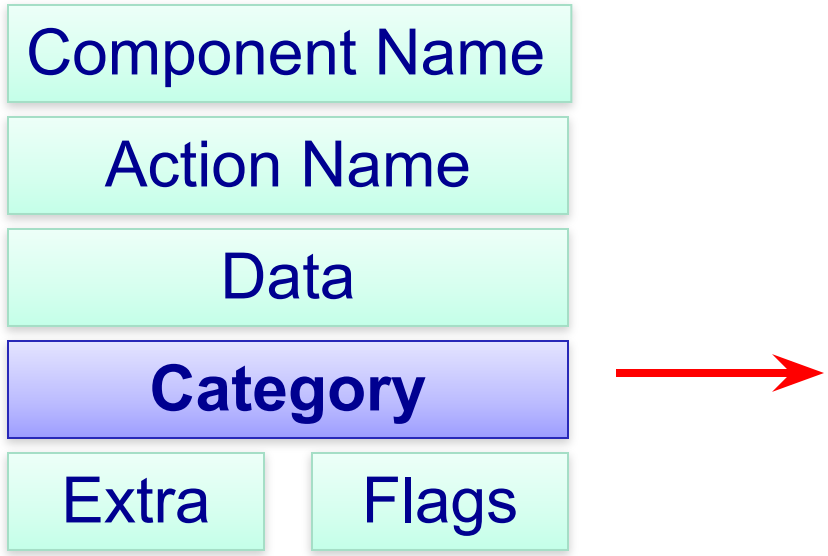
MIME type is important to help the system handle better the intent… although the type of data supplied is often dictated by the Action (e.g. if Action is ACTION_EDIT, then the data is the Uri of the document to edit).

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

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

Intent Components: Category

- We can think of an "**Intent**" object as a **message** containing a bundle of information.
  - Information of interests for the receiver (e.g. data)
  - Information of interests for the Android system (e.g. category).



A string containing information about the **kind of component** that should handle the Intent.

> 1 can be specified for an Intent
often none...

void **addCategory**(String)

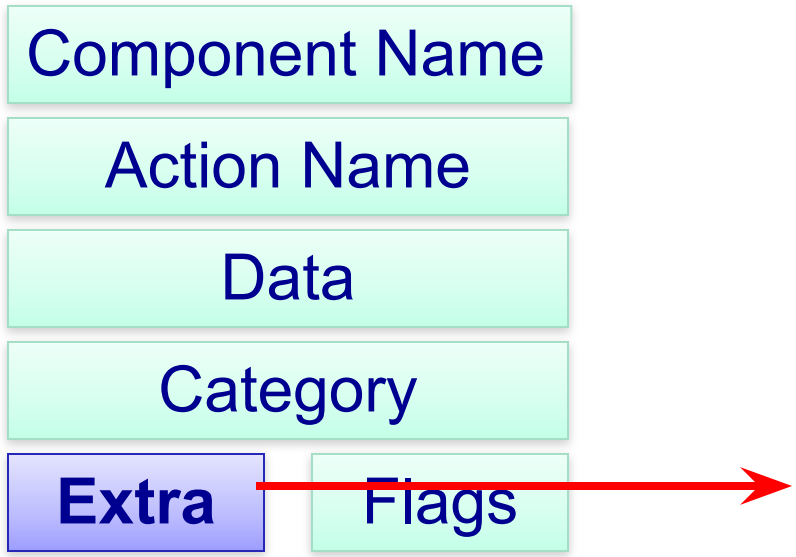
Intent Components: Category

- **Category**: string describing the **kind of component** that should handle the 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 Components: Extra

- We can think to an "**Intent**" object as a **message** containing a bundle of information.
  - Information of interests for the receiver (e.g. data)
  - Information of interests for the Android system (e.g. category).



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

Key-value pairs

void **putExtras**() **getExtras**()

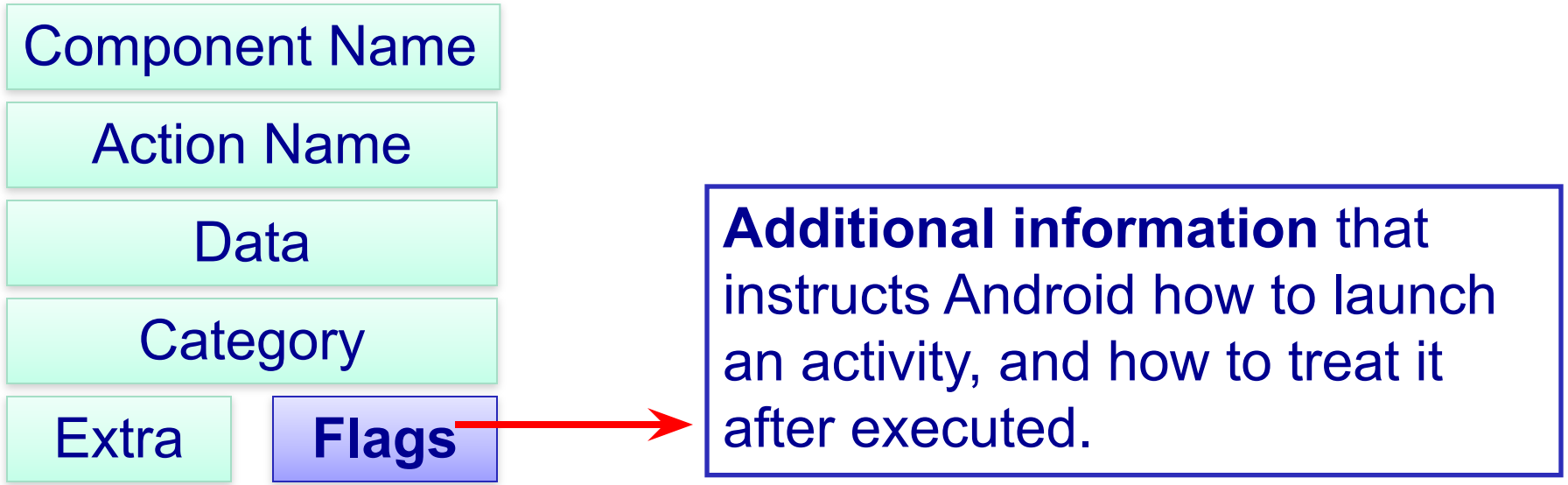
Intent Components: Extra

- Extras are not freely addressable by the developer.
  - They have defined types
- Some Action might use particular Uri, in the same way they use particular Extras
  - e.g. for ACTION_SEND you can specify the recipient with EXTRA_EMAIL and the subject with EXTRA_SUBJECT
- Yes, you can specify your own, but be sure to put the package:

```
static final String EXTRA_BASS = “it.example.BASS_NOTE”;
```

Intent Components: Flags

- We can think to an "**Intent**" object as a **message** containing a bundle of information.
  - Information of interests for the receiver (e.g. data)
  - Information of interests for the Android system (e.g. category).



Intent types: Implicit Intents

```
Intent i = new
    Intent(android.content.Intent.ACTION_VIEW,
    Uri.parse("http://informatica.unibo.it"));
startActivity(i);
```

Action to perform

Data to perform the action on

- Implicit intents are very useful to **re-use code** and to **launch external applications …**
- *More than a component can match the Intent request …*
- **How to define the target component**?

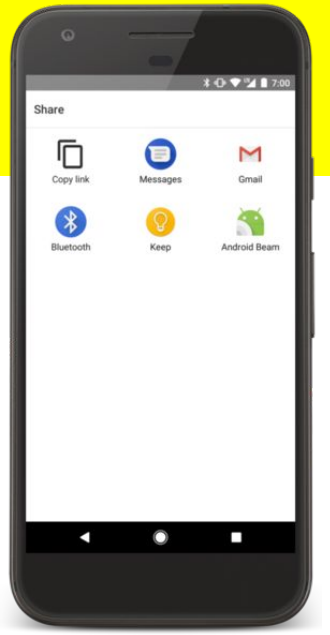
Intent types: More Implicit Intents

```
String msg = “This has been send through an Intent!”;
Intent i = new Intent();
i.setAction(Intent.ACTION_SEND);
i.putExtra(Intent.EXTRA_TEXT, msg);
i.setType(“text/plain”);

if (i.resolveActivity(getPackageManager()) != null) {
    startActivity(i);
}
```

- Note that receiver is not determined by the data Uri, but by the ACTION_SEND and the data Type.

Intent types: More Implicit Intents

```
Intent i = new Intent(Intent.ACTION_SEND);
String title = “Share this photo with...”;
Intent choose = Intent.createChooser(i, title);

if (i.resolveActivity(getPackageManager()) != null) {
      startActivity(chooser);
}
```



- User here is forced to choose the app to open every time with no chance to set a default.

Intent types: Implicit Intents

- How to declare which intents I'm able to handle?

  <**intent-filter**> tag in AndroidManifest.xml

- How?

```
<intent-filter>
        <action android:name="my.project.ACTION_ECHO" />
</intent-filter>
```

- If a component creates an Intent with "my.project.ACTION_ECHO" as action, the corresponding activity will be executed …

Intent types: Implicit Intents

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)
- If you specify more than one instance of the same tag (e.g. more than one action) →
  - Your activity should handle each combination of these

Intent types: Intent Resolution

- 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 types: Intent Resolution

- (**ACTION** Test): 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.

Side takeaway message: An intent filter always needs to specify at least one Action.

```
<intent-filter>
    <action android:name="com.example.it.ECHO"/>
</intent-filter>
```

Intent types: Intent Resolution

- (**CATEGORY** Test): 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, thus the filter must include this category to handle the intent [only for Activities] Side takeaway message: An intent filter for Activities always needs to specify at least one Category.

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

Intent types: Intent Resolution

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

```
<intent-filter>
    <data android:mimeType="audio/*  android:scheme="http"/>
    <data android:mimeType="video/mpeg  android:scheme="http"/>
</intent-filter>
```

- Both URI and MIME-types are compared (4 different sub-cases …)
  - All parts specified by the filter need to be matched by the intent.

Common Intents

- Setting an Alarm – ACTION_SET_ALARM
- Timer – ACTION_SET_TIMER
- Calendar Events – ACTION_INSERT and data
- Camera – ACTION_IMAGE_CAPTURE / ACTION_VIDEO_CAPTURE
- Email – ACTION_SEND / ACTION_SENDTO
- Files – ACTION_GET_CONTENT
- Various meanings – ACTION_VIEW

More at: https://developer.android.com/guide/components/intents-common.html

Activity Result API

From Androix 1.2.0 you can use the Activity Result API which leverages a whole new set of functions (all async).

Set :

implementation "androidx.activity:activity:1.2.0"

Learn:

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

Activity Result API

Why so?

- 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 basically registered whenever the caller is recreated, decoupling it from the call itself.

Still using the **startActivityForResult** underneath…

Activity Result API: Step 1

```
ActivityResultLauncher<String> mGetContent = registerForActivityResult(
    new GetContent(),
    new ActivityResultCallback<Uri>() {
        @Override
        public void onActivityResult(Uri uri) { /* Handle the returned Uri */   }
});
```

- 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…
  - 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 use to trigger the process

Activity Result API: Step 2

```
mGetContent.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)?

```
ActivityResultLauncher<Intent> mLauncher = registerForActivityResult(
    new StartActivityForResult(),
    new ActivityResultCallback<ActivityResult>() {
    @Override
    public void onActivityResult(ActivityResult result) {
        if (result.getResultCode() == Activity.RESULT_OK) { Intent intent = result.getData(); }
    }
});      —>          mLauncher.launch(new Intent(this, resultActivity.class));
```

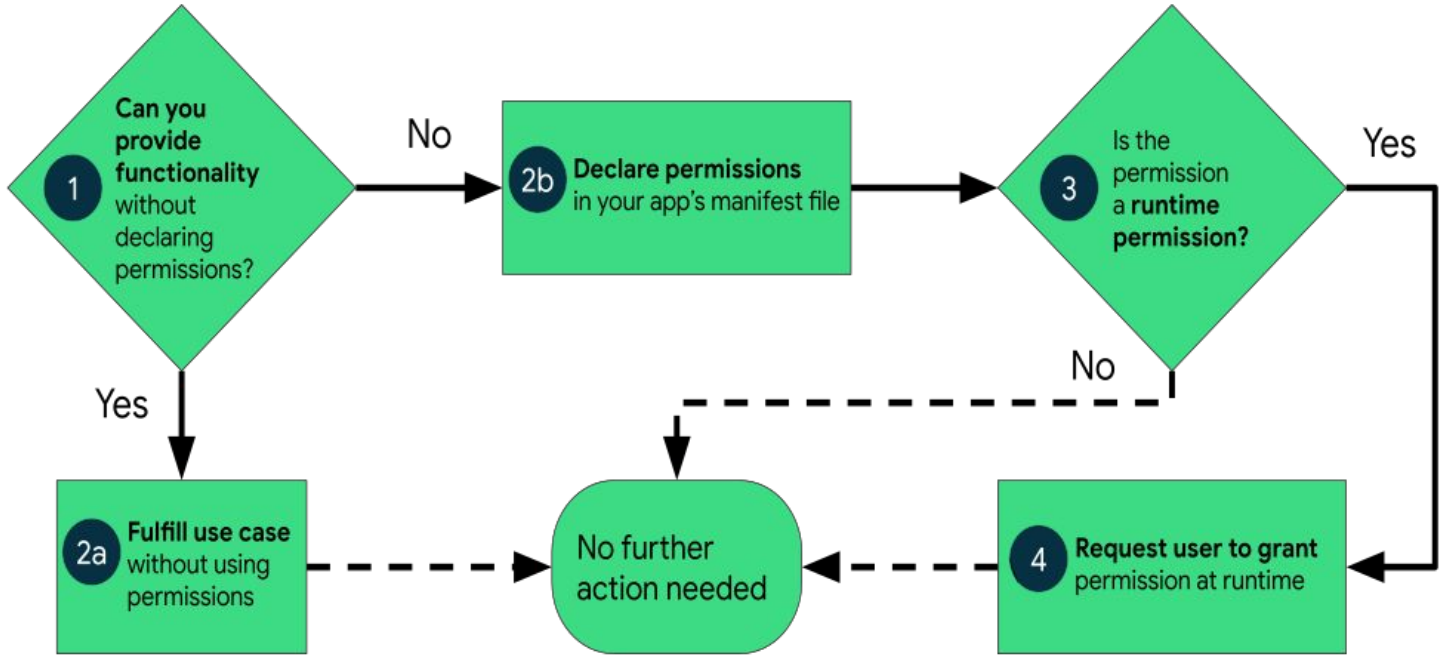
Activity Result API: Custom Contract

- Need to override the underlying intent processing functions…

```
public class PickRingtone extends ActivityResultContract<Integer, Uri> {
    @Override
    public Intent createIntent(@NonNull Context context, @NonNull Integer ringtoneType) {
        Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
        intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TYPE, ringtoneType.intValue());
        return intent;
    }

    @Override
    public Uri parseResult(int resultCode, @Nullable Intent result) {
        if (resultCode != Activity.RESULT_OK || result == null) {
            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

Up to 6.0 (excluded)

- Just declare them in the manifest

```
<uses-permission android:name="android.permission.ACCESS_FINE_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 >= 6.0

Check if permission is granted

When: before performing a permission needed action

```
if (ContextCompat.checkSelfPermission(thisActivity, Manifest.permission.ACCESS_FINE_LOCATION)
!= PackageManager.PERMISSION_GRANTED) {
    // Permission is not granted
}
```

If it is not requested, ask for it

```
ActivityCompat.requestPermissions(thisActivity,
            new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
            MY_PERMISSIONS_LOCATION);
```

Android Permission System >= 6.0

Wait for the user response asynchronously

When: before performing a permission needed action

```
@Override
public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_LOCATION: {
            if (grantResults.length > 0
                && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                // GRANTED
            } else {              // DENIED          }
            return;
        }
    }
}
```