INTENTS

COMP 41690

DAVID COYLE > D.COYLE@UCD.IE

INTENTS

An Intent is a messaging object the describes:

- An operation to be performed.
- Or an event that has occurred.

Although intents facilitate communication between components in several ways, there are three fundamental use-cases:

- To start an activity
- To start a service
- To deliver a broadcast

https://developer.android.com/guide/components/intents-filters.html

USING INTENTS

You build an Intent and pass it to the Android system

The system decides what to do using a process called Intent Resolution.

```
    /** Called when the user clicks the Send button */
    public void sendMessage(View view) {
        Intent intent = new Intent(this, DisplayMessageActivity.class);
        EditText editText = (EditText) findViewById(R.id.edit_message);
        String message = editText.getText().toString();
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivity(intent);
    }
}
```

BUILDING AN INTENT

An Intent object has properties that the Android system uses to resolve which component to start, plus information that the recipient component uses in order to properly perform the action.

Component name

Action

Data & Type

Category

An intent can carry additional information that does not affect how it is resolved to an app component.

Extras

Flags

THE COMPONENT

Component name: The name of the component to start.

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

The Component name dictates whether an Intent is **Explicit** or **Implicit**

ACTION

A string that specifies the generic action to perform.

ACTION_VIEW : Indicates that you have some information that you want shown to the user by another activity, e.g. a photo to view in a gallery app, or an address to view in a map app.

ACTION_DIAL : dial a number

ACTION_EDIT : display data to edit

ACTIO_SYNC : Snychronise device data with server.

You can specify your own actions for use by intents within your app, but *you should usually use action constants* defined by the Intent class or other framework classes.

Intent class defines more generic actions:
https://developer.android.com/reference/android/content/Intent.html

DATA - URI

The data associated with the Intent.

- Formatted as a URI object

The type of data supplied is generally dictated by the intent's action.

For example:

- If the action is ACTION_VIEW, the data might contain the URI of the photo to view.
- If the action is ACTION_DIAL, it might be Uri.parse(“tel:+353123456789)

DATA - TYPE

Specifies the MIME type of your data to help the Android system find the best component to start.

- image/*, e.g. image/jpeg, image/png
- text/plain, text/html

If no MIME type is specified the Android system will try to infer it from the URI.

Caution: If you want to set both the URI and MIME type, do not call setData() and setType() because they each nullify the value of the other. Always use setDataAndType() to set both URI and MIME type.

CATEGORY

A string containing additional information about the kind of component that should handle the intent. Any number of category descriptions can be placed in an intent, but most intents do not require a category.

CATEGORY_BROWSABLE

The target activity (to which the intent is passed) will be started by a web browser to display data referenced by the URI.

CATEGORY_LAUNCHER

The activity is the initial activity of a task and must be listed in the system's application launcher.

EXTRAS

Extras are key-value pairs that carry additional information required to accomplish the requested action.

You can add extra data with various putExtra() methods, each accepting two parameters: the key name and the value.

Example, when creating an intent to send an email with ACTION_SEND, you can specify the "to" recipient with the EXTRA_EMAIL key, and specify the "subject" with the EXTRA_SUBJECT key.

The Intent class specifies many EXTRA_* constants for standardized data types:

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

FLAGS

Flags function as metadata for the intent.

The flags may instruct the Android system how to launch an activity (for example, which task the activity should belong to) and how to treat it after it's launched (for example, whether it belongs in the list of recent activities).

```
FLAG_GRANT_READ_URI_PERMISSION
FLAG_GRANT_WRITE_URI_PERMISSION
FLAG_GRANT_PERSISTABLE_URI_PERMISSION
FLAG_GRANT_PREFIX_URI_PERMISSION
FLAG_DEBUG_LOG_RESOLUTION
FLAG_FROM_BACKGROUND
FLAG_ACTIVITY_BROUGHT_TO_FRONT
FLAG_ACTIVITY_CLEAR_TASK
FLAG_ACTIVITY_CLEAR_TOP
FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET
FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS
FLAG_ACTIVITY_FORWARD_RESULT
FLAG_ACTIVITY_LAUNCHED_FROM_HISTORY
FLAG_ACTIVITY_MULTIPLE_TASK
```

THE COMPONENT

Component name: The name of the component to start.

This is optional, but it's the critical piece of information that makes an intent **explicit**, meaning that the intent should be delivered only to the app component defined by the component name.

Without a component name, the intent is **implicit** and the system decides which component should receive the intent based on the other intent information.

> **Note:** When starting a `Service`, you should **always specify the component name**. Otherwise, you cannot be certain what service will respond to the intent, and the user cannot see which service starts.

EXPLICIT INTENTS

Explicit intents specify the component to start by name (the fully-qualified class name).

You'll typically use an explicit intent to start a component in your own app.

For example, start a new activity in response to a user action:

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

Or start a service to download a file in the background.

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData( Uri.parse (fileUrl));
startService(downloadIntent);
```

IMPLICIT INTENT

Implicit intents do not name a specific component name, but instead declare a general action to perform.

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

IMPLICIT INTENT

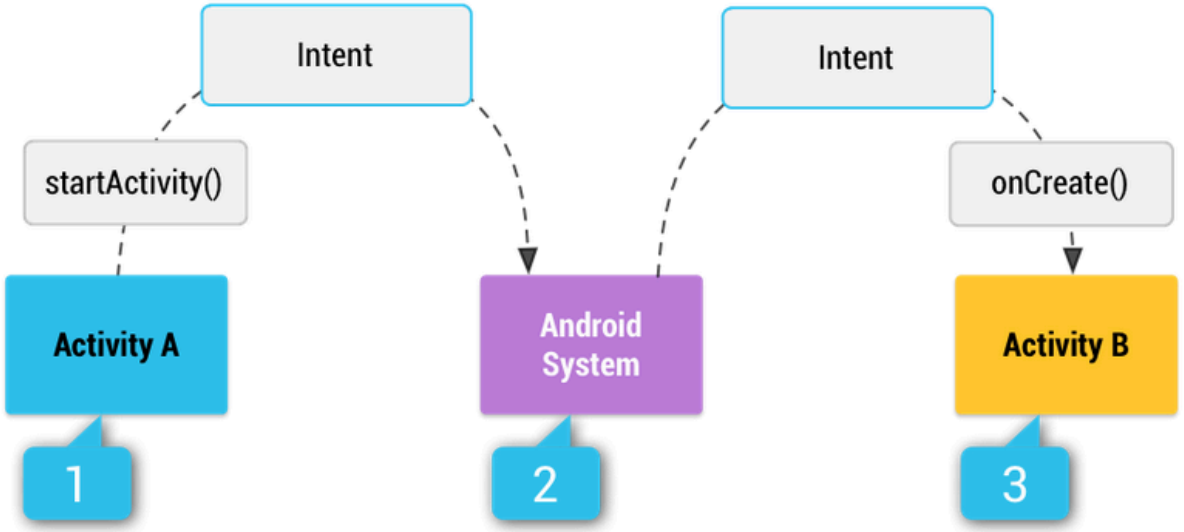

Figure 1. Illustration of how an implicit intent is delivered through the system to start another activity: **[1]** *Activity A* creates an `Intent` with an action description and passes it to `startActivity()`. **[2]** The Android System searches all apps for an intent filter that matches the intent. When a match is found, **[3]** the system starts the matching activity (*Activity B*) by invoking its `onCreate()` method and passing it the `Intent`.

INTENT RESOLUTION

When you create an implicit intent, Android uses a process call **intent resolution** to find a component which matches the intent.

It does this by trying to match the properties 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.

If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.

INTENT FILTERS

To declare which implicit intents your app can receive, declare one or more intent filters for each of your app components with an **<intent-filter>** element in your manifest file.

When the system receives an implicit intent to start it searches for the best component by comparing the intent to intent filters based on three tests:

- The action test
- The data test (both URI and data type)
- The category test

The system will deliver an implicit intent to your app component only if the intent can pass through all tests on one of your intent filters.

INTENT FILTERS

<action>

Declares the intent action accepted, in the name attribute.

<data>

Declares the type of data accepted, using one or more attributes that specify various aspects of the data URI (scheme, host, port, path, etc.) and MIME type.

<category>

Declares the intent category accepted, in the name attribute. The value must be the literal string value of an action, not the class constant.

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

INTENT FILTERS

<action>

Declares the intent action accepted, in the name attribute.

> **Note:** In order to receive implicit intents, you **must include** the `CATEGORY_DEFAULT` category in the intent filter. The methods `startActivity()` and `startActivityForResult()` treat all intents as if they declared the `CATEGORY_DEFAULT` category. If you do not declare this category in your intent filter, no implicit intents will resolve to your activity.

<category>

Declares the intent category accepted, in the name attribute. The value must be the literal string value of an action, not the class constant.

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

NOTES ON INTENT FILTERS

An app component should declare separate filters for each unique job it can do. For example, one activity in an image gallery app may have two filters: one filter to view an image, and another filter to edit an image.

It's okay to create a filter that includes more than one instance of <action>, <data>, or <category>. If you do, you simply need to be certain that the component can handle any and all combinations of those filter elements.

When you want to handle multiple kinds of intents, but only in specific combinations of action, data, and category type, then you need to create multiple intent filters.

```xml
<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>
```

Create an alarm

To create a new alarm, use the `ACTION_SET_ALARM` action and specify alarm details such as the time and message using extras defined below.

> **Note:** Only the hour, minutes, and message extras are available in Android 2.3 (API level 9) and higher. The other extras were added in later versions of the platform.

Google Now

- *"set an alarm for 7 am"*

```
public void createAlarm(String message, int hour, int minutes) {
    Intent intent = new Intent(AlarmClock.ACTION_SET_ALARM)
            .putExtra(AlarmClock.EXTRA_MESSAGE, message)
            .putExtra(AlarmClock.EXTRA_HOUR, hour)
            .putExtra(AlarmClock.EXTRA_MINUTES, minutes);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

View a contact:

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

Edit a contact:

```
public void editContact(Uri contactUri, String email) {
    Intent intent = new Intent(Intent.ACTION_EDIT);
    intent.setData(contactUri);
    intent.putExtra(Intents.Insert.EMAIL, email);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Show a location on a map

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

Example intent filter:

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

Make a phone call

```
public void dialPhoneNumber(String phoneNumber) {
    Intent intent = new Intent(Intent.ACTION_DIAL);
    intent.setData(Uri.parse("tel:" + phoneNumber));
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

For more examples of common intents in Android see

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

QUESTIONS?

Contact:

d.coyle@ucd.ie

Please ask in the Discussion Forum.

Next:

Practical – Android setup and first app.