Intents in Android

An Intent is a messaging object that you can use to request an action from an app component. An Intent is basically an intention to do an action. It's a way to communicate between Android components to request an action from a component, by different components.

The basic example of implicit Intent is to open any web page

ExampleIntent intentObj = new Intent(Intent.ACTION_VIEW);

intentObj.setData(Uri.parse("https://www.abhiandroid.com"));

startActivity(intentObj);

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

Uses of Intent in Android

There are three fundamental uses of intents:

1. To start an Activity

An Activity represents a single screen in an app. You can start a new instance of an Activity by passing an Intent to startActivity(). The Intent describes the activity to start and carries any necessary data along.

2. To start a Service

A Service is a component that performs operations in the background and does not have a user interface. You can start a service to perform a one-time operation(such as downloading a file) by passing an Intent to startService(). The Intent describes which service to start and carries any necessary data.

3. To deliver a Broadcast

A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an Intent to sendBroadcast() or sendOrderedBroadcast().

Intent Filter:

An intent filter is an instance of the IntentFilter class.

It decide the behavior of an intent. Intent about the navigation of one activity to another, which can be achieve by declaring intent filter. We can declare an Intent Filter for an Activity in manifest file. Intent filters specify the type of intents that an Activity, service or Broadcast receiver can respond to.

• Syntax of Intent Filters:

```
<activity android:name=".MainActivity">
 <intent-filter
        android:icon="@drawable/icon"
        android:label="@string/label" >
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

• icon: This is displayed as icon for activity. You can check or save png image of name

icon in drawable folder. android:icon="@drawable/icon"

• label: The label / title that appears at top in Toolbar of that particular Activity. You can

check or edit label name by opening String XML file present inside Values folder

android:label = "@string/label“

or

android:label = "New Activity“

Just like icon attribute, if you have not declared any label for your activity then it

will be same as your parent activity.

- Elements In Intent Filter:

There are following three elements in an intent filter:

1.Action

2.Data

3.Category

```
<activity android:name=".MainActivity">
        <intent-filter>
           <action android:name="android.intent.action.MAIN" />
           <category android:name="android.intent.category.LAUNCHER" />
           <action android:name="android.intent.action.SEND" />
           <category android:name="android.intent.category.DEFAULT" />
           <data android:mimeType="message/rfc822" />
        </intent-filter>
```

Action

A string naming the action to be performed — or, in the case of broadcast intents, the action that took place and is being reported. The Intent class defines a number of action constants, including these:

Constant	Target component	Action
ACTION_CALL	activity	Initiate a phone call.
ACTION_EDIT	activity	Display data for the user to edit.
ACTION_MAIN	activity	Start up as the initial activity of a task, with no data input and no returned output.
ACTION_SYNC	activity	Synchronize data on a server with

		data on the mobile device.
ACTION_BATTERY_LOW	broadcast receiver	A warning that the battery is low.
ACTION_HEADSET_PLUG	broadcast receiver	A headset has been plugged into the device, or unplugged from it.
ACTION_SCREEN_ON	broadcast receiver	The screen has been turned on.
ACTION_TIMEZONE_CHANGED	broadcast receiver	The setting for the time zone has changed.

See the Intent class description for a list of pre-defined constants for generic actions. Other actions are defined elsewhere in the Android API. You can also define your own action strings for activating the components in your application. Those you invent should include the application package as a prefix — for example: "com.example.project.SHOW_COLOR".

The action largely determines how the rest of the intent is structured — particularly the data and extras fields — much as a method name determines a set of arguments and a return value. For this reason, it's a good idea to use action names that are as specific as possible, and to couple them tightly to the other fields of the intent. In other words, instead of defining an action in isolation, define an entire protocol for the Intent objects your components can handle.

The action in an Intent object is set by the setAction() method and read by getAction().

Data

The URI of the data to be acted on and the MIME type of that data. Different actions are paired with different kinds of data specifications. For example, if the action field is ACTION_EDIT, the data field would contain the URI of the document to be displayed for editing. If the action is ACTION_CALL, the data field would be a tel: URI with the number to call. Similarly, if the action is ACTION_VIEW and the data field is an http: URI, the receiving activity would be called upon to download and display whatever data the URI refers to.

When matching an intent to a component that is capable of handling the data, it's often important to know the type of data (its MIME type) in addition to its URI. For example, a component able to display image data should not be called upon to play an audio file.

In many cases, the data type can be inferred from the URI — particularly content: URIs, which indicate that the data is located on the device and controlled by a content provider (see the separate discussion on content providers). But the type can also be explicitly set in the Intent object. The setData() method specifies data only as a URI, setType() specifies it only as a MIME type, and setDataAndType() specifies it as both a URI and a MIME type. The URI is read by getData() and the type by getType().

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 object. As it does for actions, the Intent class defines several category constants, including these:

Constant	Meaning
CATEGORY_BROWSABLE	The target activity can be safely invoked by the browser to display data referenced by a link — for example, an image or an e-mail message.
CATEGORY_GADGET	The activity can be embedded inside of another activity that hosts gadgets.
CATEGORY_HOME	The activity displays the home screen, the first screen the user sees when the device is turned on or when the HOME key is pressed.
CATEGORY_LAUNCHER	The activity can be the initial activity of a task and is listed in the top-level application launcher.
CATEGORY_PREFERENCE	The target activity is a preference panel.

See the Intent class description for the full list of categories.

The addCategory() method places a category in an Intent object, removeCategory() deletes a category previously added, and getCategories() gets the set of all categories currently in the object.

Extras

Key-value pairs for additional information that should be delivered to the component handling the intent. Just as some actions are paired with particular kinds of data URIs, some are paired with particular extras. For example, an ACTION_TIMEZONE_CHANGED intent has a "time-zone" extra that identifies the new time zone, and ACTION_HEADSET_PLUG has a "state" extra indicating whether the headset is now plugged in or unplugged, as well as a "name" extra for the type of headset. If you were to invent a SHOW_COLOR action, the color value would be set in an extra key-value pair.

The Intent object has a series of put...() methods for inserting various types of extra data and a similar set of get...() methods for reading the data. These methods parallel those for Bundle objects. In fact, the extras can be installed and read as a Bundle using the putExtras() and getExtras() methods.

Flags

Flags of various sorts. Many 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). All these flags are defined in the Intent class.

The Android system and the applications that come with the platform employ Intent objects both to send out system-originated broadcasts and to activate system-defined components. To see how to structure an intent to activate a system component, consult the list of intents in the reference.

Types of Intents

In Android, there are two types of Intents:

1. Explicit Intents
2. Implicit Intents

Explicit Intents

Explicit Intent specifies the component. In such a case, intent provides the external class to be invoked.

you generally use an explicit intent to start a new component in your own app, because you know which exact activity or service you want to start. For example, you can start a new activity in response to a user action or start a service to download a file in the background.

```
Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
startActivity(i);
```

Create an Explicit Intent

To create an explicit intent,

1. You need to make an Intent object. The constructor of the Explicit Intent's object needs two parameters as follows:

Context c: This represents the object of the Activity from where you are calling the intent.

Java file name: This represents the name of the java file of the Activity you want to open.

Note: *You need to mention the java file name with .class extension*

```
Intent i  = new Intent(this, MyJavaFile.class);
```

2. Call startActivity() method and pass the intent's object as the parameter. This method navigates to the java file mentioned in the Intent's object.

```
startActivity(i);
```

3. If you need to pass some information or data to the new Activity you are calling, you can do this by calling putExtra() method before the startActivity() method. This method accepts key-value pair as its parameter.
4. i.putExtra("key1", "I am value1");
5. i.putExtra("key2", "I am value2");

```
startActivity(i);
```

Note: *To receive the data in the new Activity and use it accordingly, you need to call the getIntent() method and then getStringExtra() method in the java class of the Activity you want to open through explicit intent. getStringExtra() method takes the key as the parameter.*

String a = getIntent().getStringExtra("key1");

Doing this, stores the value stored at **key1** into the string variable a.

Implicit Intent

Implicit Intent doesn't specifiy the component. In such case, intent provides information of available components provided by the system that is to be invoked. It declare a general action to perform, which allows a component from another app to handle it.

For example, you may write the following code to view the webpage.

```
Intent intent=new Intent(Intent.ACTION_VIEW);
        intent.setData(Uri.parse("http://www.sangolacollege.org/"));
startActivity(intent);
```

Create an Implicit Intent

To create an implicit intent,

1. You need to make an Intent object. The constructor of the Implicit Intent's object needs a **type of action** you want to perform.

   An **action** is a string that specifies the generic action to be performed. The action largely determines how the rest of the intent is structured, particularly the information that is contained as data and extras in the intent object. For example,

   ACTION_VIEW: This action is used when you have some information that an activity can show to the user, such as a photo to view in a **Gallery** app, or an address to view in a **Map** app.

   ACTION_SEND: This action is used when you have some data that the user can share through another app, such as an **Email** app or some **Social Networking** app.

Note: *You can specify your own actions for getting used by intents within your own app (or for getting used by other apps to invoke components in your app), but you usually specify action constants defined by the Intent class or other framework classes.*

```
Intent i = new Intent(Intent.ACTION_VIEW);
```

2. You need to provide some data for the action to be performed. Data is typically expressed as a URI(Uniform Resource Identifier) which provides data to the other app so that any other app which is capable of handling the URI data can perform the desired action. For example, if you want to open a website through your app, you can pass the Uri data using **setData()** method as follows:

   ```
   i.setData(Uri.parse("http://www.google.co.in"));
   ```

3. Call startActivity() method in the end with the intent object as the parameter.

   ```
   startActivity(i);
   ```