

# Android Intents

## how to create a new activity

1. right click on the package you would like to create the new activity
2. choose "java class" and write the name of the class
3. inside the new file add "extends Activity"
4. add the selected overridden functions (such as "onCreate" - must)
5. Create a new xml file in res/layout folder that contains the UI for that Activity.
6. In the java file in the "onCreate" function type in the "setContentView("xmlFileName") to connect the xml file to the java file.
7. inside the manifest xml in the <application> tag add a new "<Activity>"
8. Add "android:name=".ActivityName" - the location is relative in the package(no need to enter the full package).
9. Optional - Add "android:label="@String/anythingYouWant"

## Intent

After creating the activity we need to tell the system to create an instance when needed. This is done with the Intent class, which is designed to send message throughout the system. To open the second activity from the main activity you must:

Create a new "Intent" instance which will need a context (this) and the class (instance of the class from the class loader) of the second activity (CLASS\_NAME.class - for example - SecondActivity.class)

Then start the activity with the **startActivity** function.

Example:

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

OR

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

## Extra

This is the way to send information from one activity to another. (`intent.putExtra()`) the Extras are simply an hashtable that contains keys and values.

inside this function you need to put a key which will give a name to the sent information (this way you will be able to receive the information with the same key)

example:

```
intent.putExtra(key,information)
```

To receive the information you must go to the receiving activity and use the **`getIntent()`** function (each activity is triggered by an intent, this is how the get that intent). this will give you the intent which started this activity. after getting the intent you can then use the `getStringExtra`, `getIntExtra`, or else.

example:

```
String information = getIntent().getStringExtra(key);
```

## explicit intent

This is what we have done until now. We tell the system the name of activity which we want to open - **explicitly** - this is the way to open a specific class (or activity). We broadcast the intent and the system creates an instance of that SPECIFIC activity and calls `oncreate`.

## implicit intent

Instead of telling the system what to open we tell her what ACTION we want to do and let her choose the proper way to handle the call and open whomever can do the action we wanted. It can be a chooser of activities to choose from. This implicit intent is an intent that is sent out to all activities that are able to do the action declared in the intent. The activity can tell the system which actions he can perform with the intent filter in the manifest file.

## Action

A string passed to the implicit intent that defines a unique action in the system. We can use the android constants actions such as `ACTION_VIEW`, `ACTION_CALL`, `ACTION_DIAL` or create our own unique action string. To create our own activity that can perform an action and can be

launched from the system as a response to an action and to implicit intent we should use the **intent filter** we discussed before. For example if we send the following custom action with intent:

```
Intent intent = new Intent("com.pzlapps.translate");
```

and we want an activity that can respond to this action, we can add an activity to the manifest file with the following intent filter (intent filter can also be added programmatically) :

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

if there are few activities respond to the same action the system will create an activities chooser.

## Android's constant actions

There are constant actions in the android api(here are some of them):

### ACTION\_DIAL

Open the dialer with the number specified through the intent's setData function (setData(Uri.parse("tel:" + number)).

### ACTION\_CALL

Dial to a specific number but need the permission to call in the manifest (since we don't show the usual dialer and let the user decide, but instead perform the call ourself)

### ACTION\_SEND

Deliver some data to someone else.

For example in order to send an e-mail the necessary data to be added to the intent:

```
Intent intent = new Intent(Intent.ACTION_SEND);  
intent.putExtra(Intent.EXTRA_EMAIL, new String[]{"ababab@gamil.com"});  
intent.putExtra(Intent.EXTRA_SUBJECT, "This is the subject");  
intent.putExtra(Intent.EXTRA_TEXT, "This is the email body");  
intent.setType("text/plain");
```

## ACTION\_VIEW

The most flexible action for many uses. Display the data to the user. This is the most common action performed on data -- it is the generic action you can use on a piece of data to get the most reasonable thing to occur. For example, when used on a contacts entry it will view the entry; when used on a mailto: URI it will bring up a compose window filled with the information supplied by the URI; when used with a tel: URI it will invoke the dialer.

Example open the browser to specific site:

```
Intent intent = new Intent(Intent.ACTION_VIEW,Uri.parse("http://www.cnn.com"))
```

## permissions

To perform any special action inside a user's phone without the known UI(Dial without using the regular dialer) you must add the specific permission for that action.

To do that you must add in the manifest above the "application" tag add the "<uses-permission>" tag. Inside that you add the "name" information of that action.

For example:

```
android:name="android.permission.CALL_PHONE" to be able to immediately start a call.
```

Before the user installs our app all the permissions written there will be presented for him to approve. After approving them he doesn't have any other access controls(prior to Android 6.0).

# StartActivityForResult - OnActivityResult

We can start an activity for specific task and when completed get back the data. for this mechanism we use a function called **startActivityResult** and override the function **onActivityResult**.

the OnActivityResult has three variables to input: requestCode, resultCode, data

requestCode is the sent code that was sent from the activating activity, a code identifying the action we want, that way if we start a few activities for result from the same activity, all of them returned to the same onActivityResult and we distinguish one from the other.

resultCode is the code that is returned from the second activity (the one you preformed the specific action, there are two options RESULT\_OK or RESULT\_CANCELED)

data is the data you will be transferring from the two activities

## How to startActivityResult

In the first activity - the one requesting the action

1. Define an int constant variable that will represent the action (requestCode).
2. use the startActivityResult function which takes an intent (implicit or explicit) and the defining requestResult (put data extra in the intent if needed).
3. override the OnActivityResult function while using the three parameters. First check the request code to detect which activity has returned and then check the result code to see if the activity completed successfully or cancelled. After that and on successful completion use the data bundle the get the requested result (photo for example).

In the second activity - the one performing the action

1. input the information of which you would like to return into the intent
2. use the function setResult and input the parameter: the first parameter will be the resultCode (RESULT\_OK or RESULT\_CANCELLED for example) and the second parameter should be the intent with the data you would like to return (intent.putExtra()).
3. Add finish for the second activity to visually close and the first activity to appear with the new data.

## finish

An activity can terminate himself by calling this function

## finishActivity(request\_code)

This function allows you to terminate an activity with the request\_code sent as a parameter. Used for finishing an activity started with the startActivityForResult from the calling activity, without waiting for him to finish himself.

## Take photo with native camera app

Use the following action: MediaStore.ACTION\_IMAGE\_CAPTURE

Start the activity for result and get the thumbnail of the photo taken this way:

```
Bitmap bitmap = (Bitmap)data.getExtras().get("data");
```

If you want to full size photo you should put an extra EXTRA\_OUTPUT to control where this image will be written, the full-sized image will be written to the Uri value of EXTRA\_OUTPUT. If the EXTRA\_OUTPUT is not present, then a small sized image is returned as a Bitmap object in the extra field.

## Use the Voice recognition intent

Use the following action : RecognizerIntent.ACTION\_RECOGNIZE\_SPEECH

Here are some of the additional extras you can add to the intent:

```
intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE,add a show two language  
coed("iw","en"...))
```

```
intent.putExtra(RecognizerIntent.EXTRA_MAX_RESULTS,max results desired);
```

```
intent.putExtra(RecognizerIntent.EXTRA_PROMPT,extra text to be shown to the user  
on the google's recognition window);
```

In the onActivityResult, get the recognition results like this:

```
ArrayList<String> results =
```

```
data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
```