**3. Android – Fragments and Intents/Filters**

# Android – Fragments

A **Fragment**is a piece of an activity which enable more modular activity design. It will not be wrong if we say, a fragment is a kind of **sub-activity**.

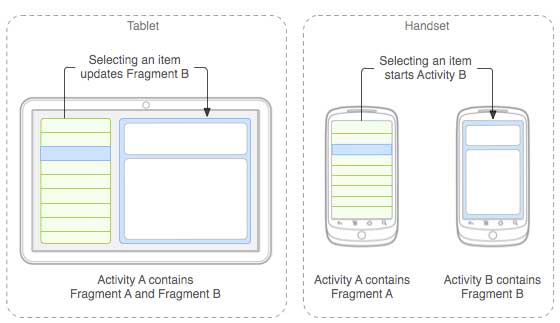
Following are important points about fragment −

* A fragment has its own layout and its own behaviour with its own life cycle callbacks.
* You can add or remove fragments in an activity while the activity is running.
* You can combine multiple fragments in a single activity to build a multi-pane UI.
* A fragment can be used in multiple activities.
* Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
* A fragment can implement a behaviour that has no user interface component.
* Fragments were added to the Android API in Honeycomb version of Android which API version 11.

You create fragments by extending **Fragment** class and You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a **<fragment>** element.

Prior to fragment introduction, we had a limitation because we can show only a single activity on the screen at one given point in time. So we were not able to divide device screen and control different parts separately. But with the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time. Now we can have a single activity but each activity can comprise of multiple fragments which will have their own layout, events and complete life cycle.

Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.



The application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article.

## Fragment Life Cycle

Android fragments have their own life cycle very similar to an android activity. This section briefs different stages of its life cycle.



#### FRAGMENT LIFECYCLE

Here is the list of methods which you can to override in your fragment class −

* **onAttach()**The fragment instance is associated with an activity instance.The fragment and the activity is not fully initialized. Typically you get in this method a reference to the activity which uses the fragment for further initialization work.
* **onCreate()** The system calls this method when creating the fragment. You should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
* **onCreateView()** The system calls this callback when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a **View** component from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
* **onActivityCreated()**The onActivityCreated() is called after the onCreateView() method when the host activity is created. Activity and fragment instance have been created as well as the view hierarchy of the activity. At this point, view can be accessed with the findViewById() method. example. In this method you can instantiate objects which require a Context object
* **onStart()**The onStart() method is called once the fragment gets visible.
* **onResume()**Fragment becomes active.
* **onPause()** The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.
* **onStop()**Fragment going to be stopped by calling onStop()
* **onDestroyView()**Fragment view will destroy after call this method
* **onDestroy()**onDestroy() called to do final clean up of the fragment's state but Not guaranteed to be called by the Android platform.

## How to use Fragments?

This involves number of simple steps to create Fragments.

* First of all decide how many fragments you want to use in an activity. For example let's we want to use two fragments to handle landscape and portrait modes of the device.
* Next based on number of fragments, create classes which will extend the *Fragment* class. The Fragment class has above mentioned callback functions. You can override any of the functions based on your requirements.
* Corresponding to each fragment, you will need to create layout files in XML file. These files will have layout for the defined fragments.
* Finally modify activity file to define the actual logic of replacing fragments based on your requirement.

## Types of Fragments

Basically fragments are divided as three stages as shown below.

* [Single frame fragments](https://www.tutorialspoint.com/android/android_single_fragments.htm) − Single frame fragments are using for hand hold devices like mobiles, here we can show only one fragment as a view.
* [List fragments](https://www.tutorialspoint.com/android/android_list_fragment.htm) − fragments having special list view is called as list fragment
* [Fragments transaction](https://www.tutorialspoint.com/android/android_fragment_transitions.htm) − Using with fragment transaction. we can move one fragment to another fragment.

# Android - Single Fragments

**Single Frame Fragment**

Single frame fragment is designed for small screen devices such as hand hold devices(mobiles) and it should be above android 3.0 version.

## Example

This example will explain you how to create your own *Fragments*. Here we will create two fragments and one of them will be used when device is in landscape mode and another fragment will be used in case of portrait mode. So let's follow the following steps to similar to what we followed while creating *Hello World Example* −

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use Android StudioIDE to create an Android application and name it as *MyFragments* under a package *com.example.myfragments*, with blank Activity. |
| 2 | Modify main activity file *MainActivity.java* as shown below in the code. Here we will check orientation of the device and accordingly we will switch between different fragments. |
| 3 | Create a two java files *PM\_Fragment.java* and *LM\_Fragement.java* under the package *com.example.myfragments* to define your fragments and associated methods. |
| 4 | Create layouts files *res/layout/lm\_fragment.xml* and *res/layout/pm\_fragment.xml* and define your layouts for both the fragments. |
| 5 | Modify the default content of *res/layout/activity\_main.xml* file to include both the fragments. |
| 6 | Define required constants in *res/values/strings.xml* file |
| 7 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

Following is the content of the modified main activity file **MainActivity.java** −

package com.example.myfragments;

import android.app.Activity;

import android.app.FragmentManager;

import android.app.FragmentTransaction;

import android.content.res.Configuration;

import android.os.Bundle;

public class MainActivity extends Activity {

/\*\* Called when the activity is first created. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

Configuration config = getResources().getConfiguration();

FragmentManager fragmentManager = getFragmentManager();

FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

/\*\*

\* Check the device orientation and act accordingly

\*/

if (config.orientation == Configuration.ORIENTATION\_LANDSCAPE) {

/\*\*

\* Landscape mode of the device

\*/

LM\_Fragement ls\_fragment = new LM\_Fragement();

fragmentTransaction.replace(android.R.id.content, ls\_fragment);

}else{

/\*\*

\* Portrait mode of the device

\*/

PM\_Fragement pm\_fragment = new PM\_Fragement();

fragmentTransaction.replace(android.R.id.content, pm\_fragment);

}

fragmentTransaction.commit();

}

}

Create two fragment files **LM\_Fragement.java** and **PM\_Fragment.java**

Following is the content of **LM\_Fragement.java** file −

package com.example.myfragments;

import android.app.Fragment;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

/\*\*

\* Created by TutorialsPoint7 on 8/23/2016.

\*/

public class LM\_Fragement extends Fragment {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

/\*\*

\* Inflate the layout for this fragment

\*/

return inflater.inflate(R.layout.lm\_fragment, container, false);

}

}

Following is the content of **PM\_Fragement.java** file −

package com.example.myfragments;

import android.app.Fragment;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

/\*\*

\* Created by TutorialsPoint7 on 8/23/2016.

\*/

public class PM\_Fragement extends Fragment {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

/\*\*

\* Inflate the layout for this fragment

\*/

return inflater.inflate(R.layout.pm\_fragment, container, false);

}

}

Create two layout files **lm\_fragment.xml** and **pm\_fragment.xml** under *res/layout* directory.

Following is the content of **lm\_fragement.xml** file −

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:background="#7bae16">

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/landscape\_message"

android:textColor="#000000"

android:textSize="20px" />

<!-- More GUI components go here -->

</LinearLayout>

Following is the content of **pm\_fragment.xml** file −

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="horizontal"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:background="#666666">

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/portrait\_message"

android:textColor="#000000"

android:textSize="20px" />

<!-- More GUI components go here -->

</LinearLayout>

Following will be the content of **res/layout/activity\_main.xml** file which includes your fragments −

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="horizontal">

<fragment

android:name="com.example.fragments"

android:id="@+id/lm\_fragment"

android:layout\_weight="1"

android:layout\_width="0dp"

android:layout\_height="match\_parent" />

<fragment

android:name="com.example.fragments"

android:id="@+id/pm\_fragment"

android:layout\_weight="2"

android:layout\_width="0dp"

android:layout\_height="match\_parent" />

</LinearLayout>

Make sure you have following content of **res/values/strings.xml** file −

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">My Application</string>

<string name="landscape\_message">This is Landscape mode fragment</string>

<string name="portrait\_message">This is Portrait mode fragment></string>

</resources>

Let's try to run our modified **MyFragments** application we just created. I assume you had created your **AVD** while doing environment set-up. To run the app from Android Studio, open one of your project's activity files and click Run icon from the tool bar. Android Studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display Emulator window where you will click on Menu button to see the following window. Be patience because it may take sometime based on your computer speed −



To change the mode of the emulator screen, let's do the following −

* **fn+control+F11** on Mac to change the landscape to portrait and vice versa.
* **ctrl+F11** on Windows.
* **ctrl+F11** on Linux.

Once you changed the mode, you will be able to see the GUI which you have implemented for landscape mode as below −



This way you can use same activity but different GUI's through different fragments. You can use different type of GUI components for different GUI's based on your requirements.

# Android - list Fragment

Static library support version of the framework's ListFragment. Used to write apps that run on platforms prior to Android 3.0. When running on Android 3.0 or above, this implementation is still used.

**The basic implementation of list fragment is for creating list of items in fragments**



#### LIST IN FRAGMENTS

## Example

This example will explain you how to create your own list fragment based on arrayAdapter. So let's follow the following steps to similar to what we followed while creating Hello World Example −

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use Android Studio to create an Android application and name it as *SimpleListFragment* under a package *com.example.tutorialspoint7.myapplication*, with blank Activity. |
| 2 | Modify the string file, which has placed at *res/values/string.xml* to add new string constants |
| 3 | Create a layout called *list\_fragment.xml* under the directory *res/layout* to define your list fragments. and add fragment tag(<fragment>) to your activity\_main.xml |
| 4 | Create a myListFragment.java, which is placed at *java/myListFragment.java* and it contained *onCreateView()*,*onActivityCreated()* and*OnItemClickListener()* |
| 5 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

Before start coding i will initialize of the string constants inside *string.xml* file under *res/values directory*

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">ListFragmentDemo</string>

<string name="action\_settings">Settings</string>

<string name="hello\_world">Hello world!</string>

<string name="imgdesc">imgdesc</string>

<string-array name="Planets">

<item>Sun</item>

<item>Mercury</item>

<item>Venus</item>

<item>Earth</item>

<item>Mars</item>

<item>Jupiter</item>

<item>Saturn</item>

<item>Uranus</item>

<item>Neptune</item>

</string-array>

</resources>

Following will be the content of **res/layout/activity\_main.xml** file. it contained linear layout and fragment tag.

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical" >

<fragment

android:id="@+id/fragment1"

android:name="com.example.tutorialspoint7.myapplication.MyListFragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

</LinearLayout>

Following will be the content of **res/layout/list\_fragment.xml** file. it contained linear layout,list view and text view

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical" >

<ListView

android:id="@android:id/list"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" >

</ListView>

<TextView

android:id="@android:id/empty"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" >

</TextView>

</LinearLayout>

following will be the content of **src/main/java/myListFragment.java**file.before writing to code, need to follow few steps as shown below

* Create a class MyListFragment and extend it to ListFragment.
* Inside the **onCreateView()** method , inflate the view with above defined list\_fragment xml layout.
* Inside the **onActivityCreated()** method , create a arrayadapter from resource ie using String array R.array.planet which you can find inside the string.xml and set this adapter to listview and also set the onItem click Listener.
* Inside the **OnItemClickListener()** method , display a toast message with Item name which is being clicked.

package com.example.tutorialspoint7.myapplication;

import android.annotation.SuppressLint;

import android.app.ListFragment;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.AdapterView;

import android.widget.AdapterView.OnItemClickListener;

import android.widget.ArrayAdapter;

import android.widget.Toast;

public class MyListFragment extends ListFragment implements OnItemClickListener {

@Override

public View onCreateView(LayoutInflater inflater,

ViewGroup container, Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.list\_fragment, container, false);

return view;

}

@Override

public void onActivityCreated(Bundle savedInstanceState) {

super.onActivityCreated(savedInstanceState);

ArrayAdapter adapter = ArrayAdapter.createFromResource(getActivity(),

R.array.Planets, android.R.layout.simple\_list\_item\_1);

setListAdapter(adapter);

getListView().setOnItemClickListener(this);

}

@Override

public void onItemClick(AdapterView<?> parent, View view, int position,long id) {

Toast.makeText(getActivity(), "Item: " + position, Toast.LENGTH\_SHORT).show();

}

}

Following code will be the content of MainActivity.java

package com.example.tutorialspoint7.myapplication;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

}

following code will be the content of manifest.xml, which has placed at res/AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.tutorialspoint7.myapplication">

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".MainActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

## Running the Application

Let's try to run our **SimpleListFragment** application we just created. I assume you had created your **AVD** while doing environment set-up. To run the app from Android Studio, open one of your project's activity files and click Run Eclipse Run Icon icon from the toolbar. Android installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

# Android list fragment

# Android - Fragment Transition

## What is a Transition?

Activity and Fragment transitions in Lollipop are built on top of a relatively new feature in Android called Transitions. Introduced in KitKat, the transition framework provides a convenient API for animating between different UI states in an application. The framework is built around two key concepts: scenes and transitions. A scene defines a given state of an application's UI, whereas a transition defines the animated change between two scenes.

When a scene changes, a Transition has two main responsibilities −

* Capture the state of each view in both the start and end scenes.
* Create an Animator based on the differences that will animate the views from one scene to the other.

## Example

This example will explain you how to create your custom animation with fragment transition . So let's follow the following steps to similar to what we followed while creating Hello World Example −

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use Android Studio to create an Android application and name it as *fragmentcustomanimations* under a package *com.example.fragmentcustomanimations*, with blank Activity. |
| 2 | Modify the activity\_main.xml, which has placed at *res/layout/activity\_main.xml* to add a Text View |
| 3 | Create a layout called *fragment\_stack.xml.xml* under the directory *res/layout* to define your fragment tag and button tag |
| 4 | Create a folder, which is placed at *res/* and name it as animation and add fragment\_slide\_right\_enter.xml fragment\_slide\_left\_exit.xml, ,fragment\_slide\_right\_exit.xml and fragment\_slide\_left\_enter.xml |
| 5 | In MainActivity.java, need to add fragment stack, fragment manager, and onCreateView() |
| 6 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

following will be the content of *res.layout/activity\_main.xml* it contained TextView

<?xml version="1.0" encoding="utf-8"?>

<TextView xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/text"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:gravity="center\_vertical|center\_horizontal"

android:text="@string/hello\_world"

android:textAppearance="?android:attr/textAppearanceMedium" />

Following will be the content of **res/animation/fragment\_stack.xml** file. it contained frame layout and button

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical" >

<fragment

android:id="@+id/fragment1"

android:name="com.pavan.listfragmentdemo.MyListFragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

</LinearLayout>

Following will be the content of **res/animation/fragment\_slide\_left\_enter.xml** file. it contained set method and object animator

<?xml version="1.0" encoding="utf-8"?>

<set xmlns:android="http://schemas.android.com/apk/res/android">

<objectAnimator

android:interpolator="@android:interpolator/decelerate\_quint"

android:valueFrom="100dp" android:valueTo="0dp"

android:valueType="floatType"

android:propertyName="translationX"

android:duration="@android:integer/config\_mediumAnimTime" />

<objectAnimator

android:interpolator="@android:interpolator/decelerate\_quint"

android:valueFrom="0.0" android:valueTo="1.0"

android:valueType="floatType"

android:propertyName="alpha"

android:duration="@android:integer/config\_mediumAnimTime" />

</set>

following will be the content of **res/animation/fragment\_slide\_left\_exit.xml** file.it contained set and object animator tags.

<?xml version="1.0" encoding="utf-8"?>

<set xmlns:android="http://schemas.android.com/apk/res/android">

<objectAnimator

android:interpolator="@android:interpolator/decelerate\_quint"

android:valueFrom="0dp" android:valueTo="-100dp"

android:valueType="floatType"

android:propertyName="translationX"

android:duration="@android:integer/config\_mediumAnimTime" />

<objectAnimator

android:interpolator="@android:interpolator/decelerate\_quint"

android:valueFrom="1.0" android:valueTo="0.0"

android:valueType="floatType"

android:propertyName="alpha"

android:duration="@android:integer/config\_mediumAnimTime" />

</set>

Following code will be the content of **res/animation/fragment\_slide\_right\_enter.xml**file.it contained set and object animator tags

<?xml version="1.0" encoding="utf-8"?>

<set xmlns:android="http://schemas.android.com/apk/res/android">

<objectAnimator

android:interpolator="@android:interpolator/decelerate\_quint"

android:valueFrom="-100dp" android:valueTo="0dp"

android:valueType="floatType"

android:propertyName="translationX"

android:duration="@android:integer/config\_mediumAnimTime" />

<objectAnimator

android:interpolator="@android:interpolator/decelerate\_quint"

android:valueFrom="0.0" android:valueTo="1.0"

android:valueType="floatType"

android:propertyName="alpha"

android:duration="@android:integer/config\_mediumAnimTime" />

</set>

following code will be the content of **res/animation/fragment\_slide\_right\_exit.xml**file, it contained set and object animator tags

<?xml version="1.0" encoding="utf-8"?>

<set xmlns:android="http://schemas.android.com/apk/res/android">

<objectAnimator

android:interpolator="@android:interpolator/decelerate\_quint"

android:valueFrom="0dp" android:valueTo="100dp"

android:valueType="floatType"

android:propertyName="translationX"

android:duration="@android:integer/config\_mediumAnimTime" />

<objectAnimator

android:interpolator="@android:interpolator/decelerate\_quint"

android:valueFrom="1.0" android:valueTo="0.0"

android:valueType="floatType"

android:propertyName="alpha"

android:duration="@android:integer/config\_mediumAnimTime" />

</set>

following code will be the content of **src/main/java/MainActivity.java** file. it contained button listener, stack fragment and onCreateView

package com.example.fragmentcustomanimations;

import android.app.Activity;

import android.app.Fragment;

import android.app.FragmentTransaction;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.View.OnClickListener;

import android.view.ViewGroup;

import android.widget.Button;

import android.widget.TextView;

/\*\*

\* Demonstrates the use of custom animations in a FragmentTransaction when

\* pushing and popping a stack.

\*/

public class FragmentCustomAnimations extends Activity {

int mStackLevel = 1;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.fragment\_stack);

// Watch for button clicks.

Button button = (Button)findViewById(R.id.new\_fragment);

button.setOnClickListener(new OnClickListener() {

public void onClick(View v) {

addFragmentToStack();

}

});

if (savedInstanceState == null) {

// Do first time initialization -- add initial fragment.

Fragment newFragment = CountingFragment.newInstance(mStackLevel);

FragmentTransaction ft = getFragmentManager().beginTransaction();

ft.add(R.id.simple\_fragment, newFragment).commit();

}

else

{

mStackLevel = savedInstanceState.getInt("level");

}

}

@Override

public void onSaveInstanceState(Bundle outState) {

super.onSaveInstanceState(outState);

outState.putInt("level", mStackLevel);

}

void addFragmentToStack() {

mStackLevel++;

// Instantiate a new fragment.

Fragment newFragment = CountingFragment.newInstance(mStackLevel);

// Add the fragment to the activity, pushing this transaction

// on to the back stack.

FragmentTransaction ft = getFragmentManager().beginTransaction();

ft.setCustomAnimations(R.animator.fragment\_slide\_left\_enter,

R.animator.fragment\_slide\_left\_exit,

R.animator.fragment\_slide\_right\_enter,

R.animator.fragment\_slide\_right\_exit);

ft.replace(R.id.simple\_fragment, newFragment);

ft.addToBackStack(null);

ft.commit();

}

public static class CountingFragment extends Fragment {

int mNum;

/\*\*

\* Create a new instance of CountingFragment, providing "num"

\* as an argument.

\*/

static CountingFragment newInstance(int num) {

CountingFragment f = new CountingFragment();

// Supply num input as an argument.

Bundle args = new Bundle();

args.putInt("num", num);

f.setArguments(args);

return f;

}

/\*\*

\* When creating, retrieve this instance's number from its arguments.

\*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

mNum = getArguments() != null ? getArguments().getInt("num") : 1;

}

/\*\*

\* The Fragment's UI is just a simple text view showing its

\* instance number.

\*/

@Override

public View onCreateView(LayoutInflater inflater,

ViewGroup container,Bundle savedInstanceState) {

View v = inflater.inflate(R.layout.hello\_world, container, false);

View tv = v.findViewById(R.id.text);

((TextView)tv).setText("Fragment #" + mNum);

tv.setBackgroundDrawable(getResources().

getDrawable(android.R.drawable.gallery\_thumb));

return v;

}

}

}

following will be the content of**AndroidManifest.xml**

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.fragmentcustomanimations"

android:versionCode="1"

android:versionName="1.0" >

<application

android:allowBackup="true"

android:icon="@drawable/ic\_launcher"

android:label="@string/app\_name"

android:theme="@style/AppTheme" >

<activity

android:name="com.example.fragmentcustomanimations.MainActivity"

android:label="@string/app\_name" >

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

## Running the Application

Let's try to run our **Fragment Transitions** application we just created. I assume you had created your **AVD** while doing environment set-up. To run the app from Android Studio, open one of your project's activity files and click Run Eclipse Run Icon icon from the toolbar. Android installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window:



If click on new fragment, it going to be changed the first fragment to second fragment as shown below

# Android - Intents and Filters

An Android **Intent** is an abstract description of an operation to be performed. It can be used with **startActivity** to launch an Activity, **broadcastIntent** to send it to any interested BroadcastReceiver components, and **startService(Intent)** or **bindService(Intent, ServiceConnection, int)**to communicate with a background Service.

**The intent itself, an Intent object, is a passive data structure holding an abstract description of an operation to be performed.**

For example, let's assume that you have an Activity that needs to launch an email client and sends an email using your Android device. For this purpose, your Activity would send an ACTION\_SEND along with appropriate **chooser**, to the Android Intent Resolver. The specified chooser gives the proper interface for the user to pick how to send your email data.

Intent email = new Intent(Intent.ACTION\_SEND, Uri.parse("mailto:"));

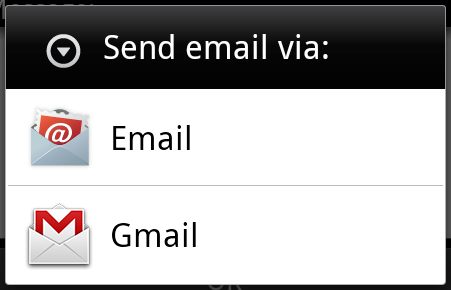
email.putExtra(Intent.EXTRA\_EMAIL, recipients);

email.putExtra(Intent.EXTRA\_SUBJECT, subject.getText().toString());

email.putExtra(Intent.EXTRA\_TEXT, body.getText().toString());

startActivity(Intent.createChooser(email, "Choose an email client from..."));

Above syntax is calling startActivity method to start an email activity and result should be as shown below −



For example, assume that you have an Activity that needs to open URL in a web browser on your Android device. For this purpose, your Activity will send ACTION\_WEB\_SEARCH Intent to the Android Intent Resolver to open given URL in the web browser. The Intent Resolver parses through a list of Activities and chooses the one that would best match your Intent, in this case, the Web Browser Activity. The Intent Resolver then passes your web page to the web browser and starts the Web Browser Activity.

String q = "tutorialspoint";

Intent intent = new Intent(Intent.ACTION\_WEB\_SEARCH );

intent.putExtra(SearchManager.QUERY, q);

startActivity(intent);

Above example will search as **tutorialspoint** on android search engine and it gives the result of tutorialspoint in your an activity

There are separate mechanisms for delivering intents to each type of component − activities, services, and broadcast receivers.

|  |  |
| --- | --- |
| **Sr.No** | **Method & Description** |
| 1 | **Context.startActivity()**  The Intent object is passed to this method to launch a new activity or get an existing activity to do something new. |
| 2 | **Context.startService()**  The Intent object is passed to this method to initiate a service or deliver new instructions to an ongoing service. |
| 3 | **Context.sendBroadcast()**  The Intent object is passed to this method to deliver the message to all interested broadcast receivers. |

## Intent Objects

An Intent object is a bundle of information which is used by the component that receives the intent as well as information used by the Android system.

An Intent object can contain the following components based on what it is communicating or going to perform −

### Action

This is mandatory part of the Intent object and is 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 action largely determines how the rest of the intent object is structured . The Intent class defines a number of action constants corresponding to different intents. Here is a list of [Android Intent Standard Actions](https://www.tutorialspoint.com/android/android_intent_standard_actions.htm)

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

### Data

Adds a data specification to an intent filter. The specification can be just a data type (the mimeType attribute), just a URI, or both a data type and a URI. A URI is specified by separate attributes for each of its parts −

These attributes that specify the URL format are optional, but also mutually dependent −

* If a scheme is not specified for the intent filter, all the other URI attributes are ignored.
* If a host is not specified for the filter, the port attribute and all the path attributes are ignored.

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().

Some examples of action/data pairs are −

|  |  |
| --- | --- |
| **Sr.No.** | **Action/Data Pair & Description** |
| 1 | **ACTION\_VIEW content://contacts/people/1**  Display information about the person whose identifier is "1". |
| 2 | **ACTION\_DIAL content://contacts/people/1**  Display the phone dialer with the person filled in. |
| 3 | **ACTION\_VIEW tel:123**  Display the phone dialer with the given number filled in. |
| 4 | **ACTION\_DIAL tel:123**  Display the phone dialer with the given number filled in. |
| 5 | **ACTION\_EDIT content://contacts/people/1**  Edit information about the person whose identifier is "1". |
| 6 | **ACTION\_VIEW content://contacts/people/**  Display a list of people, which the user can browse through. |
| 7 | **ACTION\_SET\_WALLPAPER**  Show settings for choosing wallpaper |
| 8 | **ACTION\_SYNC**  It going to be synchronous the data,Constant Value is **android.intent.action.SYNC** |
| 9 | **ACTION\_SYSTEM\_TUTORIAL**  It will start the platform-defined tutorial(Default tutorial or start up tutorial) |
| 10 | **ACTION\_TIMEZONE\_CHANGED**  It intimates when time zone has changed |
| 11 | **ACTION\_UNINSTALL\_PACKAGE**  It is used to run default uninstaller |

### Category

The category is an optional part of Intent object and it's a string containing additional information about the kind of component that should handle the intent. 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. Here is a list of [Android Intent Standard Categories](https://www.tutorialspoint.com/android/android_intent_standard_categories.htm).

You can check detail on Intent Filters in below section to understand how do we use categories to choose appropriate activity corresponding to an Intent.

### Extras

This will be in key-value pairs for additional information that should be delivered to the component handling the intent. The extras can be set and read using the putExtras() and getExtras() methods respectively. Here is a list of [Android Intent Standard Extra Data](https://www.tutorialspoint.com/android/android_intent_standard_extra_data.htm)

### Flags

These flags are optional part of Intent object and instruct the Android system how to launch an activity, and how to treat it after it's launched etc.

|  |  |
| --- | --- |
| **Sr.No** | **Flags & Description** |
| 1 | **FLAG\_ACTIVITY\_CLEAR\_TASK**  If set in an Intent passed to Context.startActivity(), this flag will cause any existing task that would be associated with the activity to be cleared before the activity is started. That is, the activity becomes the new root of an otherwise empty task, and any old activities are finished. This can only be used in conjunction with FLAG\_ACTIVITY\_NEW\_TASK. |
| 2 | **FLAG\_ACTIVITY\_CLEAR\_TOP**  If set, and the activity being launched is already running in the current task, then instead of launching a new instance of that activity, all of the other activities on top of it will be closed and this Intent will be delivered to the (now on top) old activity as a new Intent. |
| 3 | **FLAG\_ACTIVITY\_NEW\_TASK**  This flag is generally used by activities that want to present a "launcher" style behavior: they give the user a list of separate things that can be done, which otherwise run completely independently of the activity launching them. |

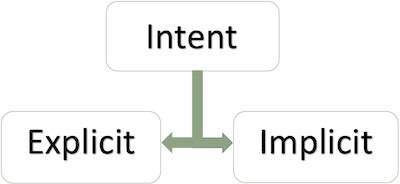
### Component Name

This optional field is an android **ComponentName** object representing either Activity, Service or BroadcastReceiver class. If it is set, the Intent object is delivered to an instance of the designated class otherwise Android uses other information in the Intent object to locate a suitable target.

The component name is set by setComponent(), setClass(), or setClassName() and read by getComponent().

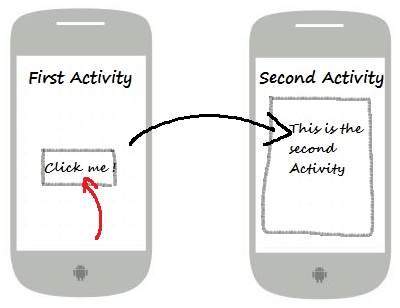
## Types of Intents

There are following two types of intents supported by Android



### Explicit Intents

Explicit intent going to be connected internal world of application,suppose if you wants to connect one activity to another activity, we can do this quote by explicit intent, below image is connecting first activity to second activity by clicking button.



These intents designate the target component by its name and they are typically used for application-internal messages - such as an activity starting a subordinate service or launching a sister activity. For example −

// Explicit Intent by specifying its class name

Intent i = new Intent(FirstActivity.this, SecondActivity.class);

// Starts TargetActivity

startActivity(i);

### Implicit Intents

These intents do not name a target and the field for the component name is left blank. Implicit intents are often used to activate components in other applications. For example −

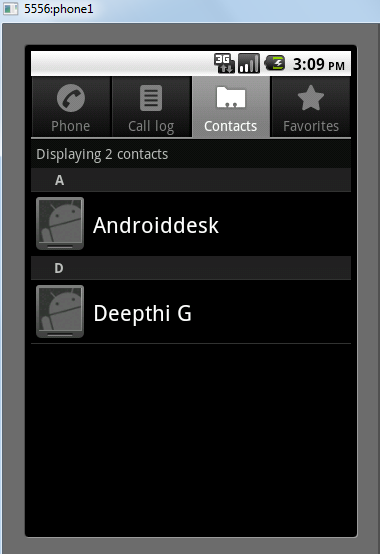
Intent read1=new Intent();

read1.setAction(android.content.Intent.ACTION\_VIEW);

read1.setData(ContactsContract.Contacts.CONTENT\_URI);

startActivity(read1);

Above code will give result as shown below



The target component which receives the intent can use the **getExtras()**method to get the extra data sent by the source component. For example −

// Get bundle object at appropriate place in your code

Bundle extras = getIntent().getExtras();

// Extract data using passed keys

String value1 = extras.getString("Key1");

String value2 = extras.getString("Key2");

## Example

Following example shows the functionality of a Android Intent to launch various Android built-in applications.

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use Android studio IDE to create an Android application and name it as *My Application* under a package *com.example.saira\_000.myapplication*. |
| 2 | Modify *src/main/java/MainActivity.java* file and add the code to define two listeners corresponding two buttons ie. Start Browser and Start Phone. |
| 3 | Modify layout XML file *res/layout/activity\_main.xml* to add three buttons in linear layout. |
| 4 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

Following is the content of the modified main activity file **src/com.example.My Application/MainActivity.java**.

package com.example.saira\_000.myapplication;

import android.content.Intent;

import android.net.Uri;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

public class MainActivity extends AppCompatActivity {

Button b1,b2;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

b1=(Button)findViewById(R.id.button);

b1.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent i = new Intent(android.content.Intent.ACTION\_VIEW,

Uri.parse("http://www.example.com"));

startActivity(i);

}

});

b2=(Button)findViewById(R.id.button2);

b2.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent i = new Intent(android.content.Intent.ACTION\_VIEW,

Uri.parse("tel:9510300000"));

startActivity(i);

}

});

}

}

Following will be the content of **res/layout/activity\_main.xml** file −

<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

android:paddingBottom="@dimen/activity\_vertical\_margin"

tools:context=".MainActivity">

<TextView

android:id="@+id/textView1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Intent Example"

android:layout\_alignParentTop="true"

android:layout\_centerHorizontal="true"

android:textSize="30dp" />

<TextView

android:id="@+id/textView2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Tutorials point"

android:textColor="#ff87ff09"

android:textSize="30dp"

android:layout\_below="@+id/textView1"

android:layout\_centerHorizontal="true" />

<ImageButton

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/imageButton"

android:src="@drawable/abc"

android:layout\_below="@+id/textView2"

android:layout\_centerHorizontal="true" />

<EditText

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/editText"

android:layout\_below="@+id/imageButton"

android:layout\_alignRight="@+id/imageButton"

android:layout\_alignEnd="@+id/imageButton" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Start Browser"

android:id="@+id/button"

android:layout\_alignTop="@+id/editText"

android:layout\_alignRight="@+id/textView1"

android:layout\_alignEnd="@+id/textView1"

android:layout\_alignLeft="@+id/imageButton"

android:layout\_alignStart="@+id/imageButton" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Start Phone"

android:id="@+id/button2"

android:layout\_below="@+id/button"

android:layout\_alignLeft="@+id/button"

android:layout\_alignStart="@+id/button"

android:layout\_alignRight="@+id/textView2"

android:layout\_alignEnd="@+id/textView2" />

</RelativeLayout>

Following will be the content of **res/values/strings.xml** to define two new constants −

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">My Applicaiton</string>

</resources>

Following is the default content of **AndroidManifest.xml** −

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.saira\_000.myapplication">

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".MainActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

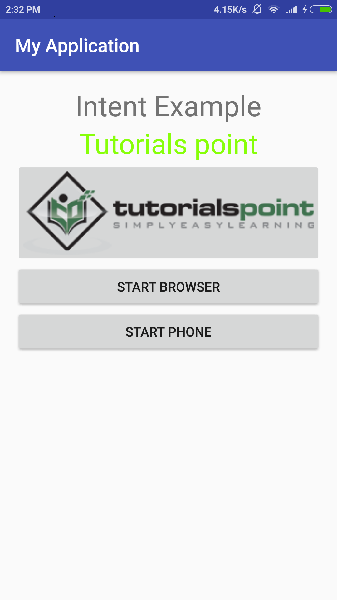
</intent-filter>

</activity>

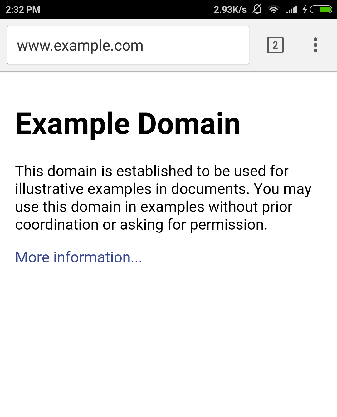
</application>

</manifest>

Let's try to run your **My Application** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run Eclipse Run Icon icon from the toolbar.Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −



Now click on **Start Browser** button, which will start a browser configured and display http://www.example.com as shown below −



Similar way you can launch phone interface using Start Phone button, which will allow you to dial already given phone number.

## Intent Filters

You have seen how an Intent has been used to call an another activity. Android OS uses filters to pinpoint the set of Activities, Services, and Broadcast receivers that can handle the Intent with help of specified set of action, categories, data scheme associated with an Intent. You will use **<intent-filter>** element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast receiver.

Following is an example of a part of **AndroidManifest.xml** file to specify an activity **com.example.My Application.CustomActivity** which can be invoked by either of the two mentioned actions, one category, and one data −

<activity android:name=".CustomActivity"

android:label="@string/app\_name">

<intent-filter>

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

<action android:name="com.example.My Application.LAUNCH" />

<category android:name="android.intent.category.DEFAULT" />

<data android:scheme="http" />

</intent-filter>

</activity>

Once this activity is defined along with above mentioned filters, other activities will be able to invoke this activity using either the **android.intent.action.VIEW**, or using the **com.example.My Application.LAUNCH** action provided their category is **android.intent.category.DEFAULT**.

The **<data>** element specifies the data type expected by the activity to be called and for above example our custom activity expects the data to start with the "http://"

There may be a situation that an intent can pass through the filters of more than one activity or service, the user may be asked which component to activate. An exception is raised if no target can be found.

There are following test Android checks before invoking an activity −

* A filter <intent-filter> may list more than one action as shown above but this list cannot be empty; a filter must contain at least one <action> element, otherwise it will block all intents. If more than one actions are mentioned then Android tries to match one of the mentioned actions before invoking the activity.
* A filter <intent-filter> may list zero, one or more than one categories. if there is no category mentioned then Android always pass this test but if more than one categories are mentioned then for an intent to pass the category test, every category in the Intent object must match a category in the filter.
* Each <data> element can specify a URI and a data type (MIME media type). There are separate attributes like **scheme, host, port**, and **path** for each part of the URI. An Intent object that contains both a URI and a data type passes the data type part of the test only if its type matches a type listed in the filter.

## Example

Following example is a modification of the above example. Here we will see how Android resolves conflict if one intent is invoking two activities defined in , next how to invoke a custom activity using a filter and third one is an exception case if Android does not file appropriate activity defined for an intent.

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use android studio to create an Android application and name it as *My Application* under a package *com.example.tutorialspoint7.myapplication;*. |
| 2 | Modify *src/Main/Java/MainActivity.java* file and add the code to define three listeners corresponding to three buttons defined in layout file. |
| 3 | Add a new *src/Main/Java/CustomActivity.java* file to have one custom activity which will be invoked by different intents. |
| 4 | Modify layout XML file *res/layout/activity\_main.xml* to add three buttons in linear layout. |
| 5 | Add one layout XML file *res/layout/custom\_view.xml* to add a simple <TextView> to show the passed data through intent. |
| 6 | Modify *AndroidManifest.xml* to add <intent-filter> to define rules for your intent to invoke custom activity. |
| 7 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

Following is the content of the modified main activity file **src/MainActivity.java**.

package com.example.tutorialspoint7.myapplication;

import android.content.Intent;

import android.net.Uri;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

public class MainActivity extends AppCompatActivity {

Button b1,b2,b3;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

b1=(Button)findViewById(R.id.button);

b1.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent i = new Intent(android.content.Intent.ACTION\_VIEW,

Uri.parse("http://www.example.com"));

startActivity(i);

}

});

b2 = (Button)findViewById(R.id.button2);

b2.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent i = new Intent("com.example.

tutorialspoint7.myapplication.

LAUNCH",Uri.parse("http://www.example.com"));

startActivity(i);

}

});

b3 = (Button)findViewById(R.id.button3);

b3.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent i = new Intent("com.example.

My Application.LAUNCH",

Uri.parse("https://www.example.com"));

startActivity(i);

}

});

}

}

Following is the content of the modified main activity file **src/com.example.My Application/CustomActivity.java**.

package com.example.tutorialspoint7.myapplication;

import android.app.Activity;

import android.net.Uri;

import android.os.Bundle;

import android.widget.TextView;

/\*\*

\* Created by TutorialsPoint7 on 8/23/2016.

\*/

public class CustomActivity extends Activity {

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.custom\_view);

TextView label = (TextView) findViewById(R.id.show\_data);

Uri url = getIntent().getData();

label.setText(url.toString());

}

}

Following will be the content of **res/layout/activity\_main.xml** file −

<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context="com.example.tutorialspoint7.myapplication.MainActivity">

<TextView

android:id="@+id/textView1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Intent Example"

android:layout\_alignParentTop="true"

android:layout\_centerHorizontal="true"

android:textSize="30dp" />

<TextView

android:id="@+id/textView2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Tutorials point"

android:textColor="#ff87ff09"

android:textSize="30dp"

android:layout\_below="@+id/textView1"

android:layout\_centerHorizontal="true" />

<ImageButton

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/imageButton"

android:src="@drawable/abc"

android:layout\_below="@+id/textView2"

android:layout\_centerHorizontal="true" />

<EditText

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/editText"

android:layout\_below="@+id/imageButton"

android:layout\_alignRight="@+id/imageButton"

android:layout\_alignEnd="@+id/imageButton" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Start Browser"

android:id="@+id/button"

android:layout\_alignTop="@+id/editText"

android:layout\_alignLeft="@+id/imageButton"

android:layout\_alignStart="@+id/imageButton"

android:layout\_alignEnd="@+id/imageButton" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Start browsing with launch action"

android:id="@+id/button2"

android:layout\_below="@+id/button"

android:layout\_alignLeft="@+id/button"

android:layout\_alignStart="@+id/button"

android:layout\_alignEnd="@+id/button" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Exceptional condition"

android:id="@+id/button3"

android:layout\_below="@+id/button2"

android:layout\_alignLeft="@+id/button2"

android:layout\_alignStart="@+id/button2"

android:layout\_toStartOf="@+id/editText"

android:layout\_alignParentEnd="true" />

</RelativeLayout>

Following will be the content of **res/layout/custom\_view.xml** file −

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical" android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView android:id="@+id/show\_data"

android:layout\_width="fill\_parent"

android:layout\_height="400dp"/>

</LinearLayout>

Following will be the content of **res/values/strings.xml** to define two new constants −

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">My Application</string>

</resources>

Following is the default content of **AndroidManifest.xml** −

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.tutorialspoint7.myapplication">

<application

android:allowBackup = "true"

android:icon = "@mipmap/ic\_launcher"

android:label = "@string/app\_name"

android:supportsRtl = "true"

android:theme = "@style/AppTheme">

<activity android:name = ".MainActivity">

<intent-filter>

<action android:name = "android.intent.action.MAIN" />

<category android:name = "android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

<activity android:name="com.example.tutorialspoint7.myapplication.CustomActivity">

<intent-filter>

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

<action android:name = "com.example.tutorialspoint7.myapplication.LAUNCH" />

<category android:name = "android.intent.category.DEFAULT" />

<data android:scheme = "http" />

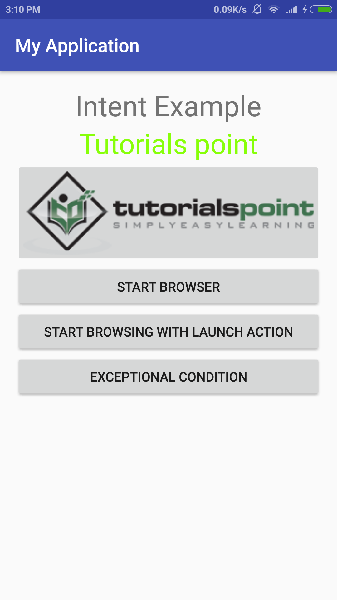
</intent-filter>

</activity>

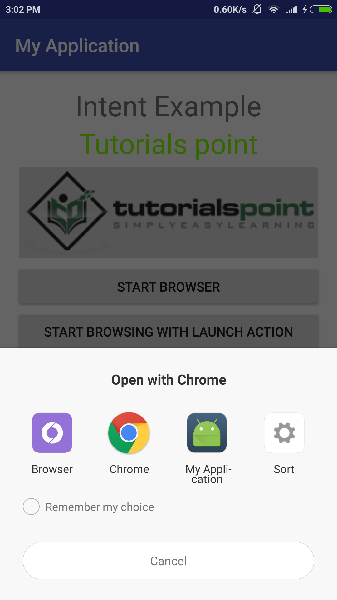
</application>

</manifest>

Let's try to run your **My Application** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run Eclipse Run Icon icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −



Now let's start with first button "Start Browser with VIEW Action". Here we have defined our custom activity with a filter "android.intent.action.VIEW", and there is already one default activity against VIEW action defined by Android which is launching web browser, So android displays following two options to select the activity you want to launch.

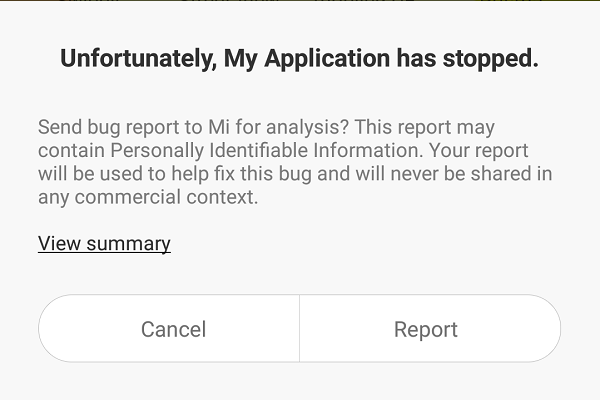


Now if you select Browser, then Android will launch web browser and open example.com website but if you select IndentDemo option then Android will launch CustomActivity which does nothing but just capture passed data and displays in a text view as follows −



Now go back using back button and click on "Start Browser with LAUNCH Action" button, here Android applies filter to choose define activity and it simply launch your custom activity

Again, go back using back button and click on "Exception Condition" button, here Android tries to find out a valid filter for the given intent but it does not find a valid activity defined because this time we have used data as **https**instead of **http** though we are giving a correct action, so Android raises an exception and shows following screen −



# second fragment