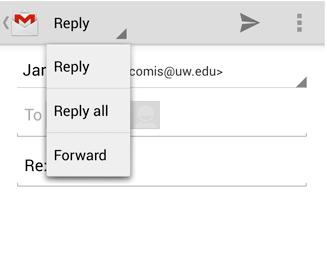
Android - Spinner

For example. When you are using Gmail application you would get drop down menu as shown below, you need to select an item from a drop down menu.



#### SPINNER EXAMPLE

## Example

This example demonstrates the category of computers, you need to select a category from the category.

To experiment with this example, you need to run this on an actual device on after developing the application according to the steps below.

|  |  |
| --- | --- |
| **Steps** | **Description** |
| 1 | You will use Android studio to create an Android application and name it as AndroidSpinnerExample under a package com.example.spinner. |
| 2 | Modify src/AndroidSpinnerExampleActivity.java file to create a simple list view with items which are showing as spinner items |
| 3 | Modify res/layout/activity\_main.xml file to add respective XML code. |
| 4 | No need to define default string constants. Android studio takes care of default string constants at string.xml |
| 5 | Run the application and choose a running android device and install the application on it and verify the results. |

Following is the content of the modified main activity file **src/com.example.spinner/AndroidSpinnerExampleActivity.java.**

package com.example.spinner;

import java.util.ArrayList;

import java.util.List;

import android.app.Activity;

import android.os.Bundle;

import android.view.View;

import android.widget.AdapterView;

import android.widget.ArrayAdapter;

import android.widget.Spinner;

import android.widget.Toast;

import android.widget.AdapterView.OnItemSelectedListener;

class AndroidSpinnerExampleActivity extends Activity implements OnItemSelectedListener{

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

// Spinner element

Spinner spinner = (Spinner) findViewById(R.id.spinner);

// Spinner click listener

spinner.setOnItemSelectedListener(this);

// Spinner Drop down elements

List<String> categories = new ArrayList<String>();

categories.add("Automobile");

categories.add("Business Services");

categories.add("Computers");

categories.add("Education");

categories.add("Personal");

categories.add("Travel");

// Creating adapter for spinner

ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(this, android.R.layout.simple\_spinner\_item, categories);

// Drop down layout style - list view with radio button

dataAdapter.setDropDownViewResource(android.R.layout.simple\_spinner\_dropdown\_item);

// attaching data adapter to spinner

spinner.setAdapter(dataAdapter);

}

@Override

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

// On selecting a spinner item

String item = parent.getItemAtPosition(position).toString();

// Showing selected spinner item

Toast.makeText(parent.getContext(), "Selected: " + item, Toast.LENGTH\_LONG).show();

}

public void onNothingSelected(AdapterView<?> arg0) {

// TODO Auto-generated method stub

}

}

Modify the content of **res/layout/activity\_main.xml** to the following

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

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

android:orientation="vertical"

android:padding="10dip"

android:layout\_width="fill\_parent"

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

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:layout\_marginTop="10dip"

android:text="Category:"

android:layout\_marginBottom="5dp"/>

<Spinner

android:id="@+id/spinner"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:prompt="@string/spinner\_title"/>

</LinearLayout>

Modify the **res/values/string.xml** to the following

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

<resources>

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

</resources>

This is the default **AndroidManifest.xml**

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

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

package="com.example.spinner" >

<application

android:allowBackup="true"

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

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

android:theme="@style/AppTheme" >

<activity

android:name="com.example.spinner.AndroidSpinnerExampleActivity"

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

<intent-filter>

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

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

</intent-filter>

</activity>

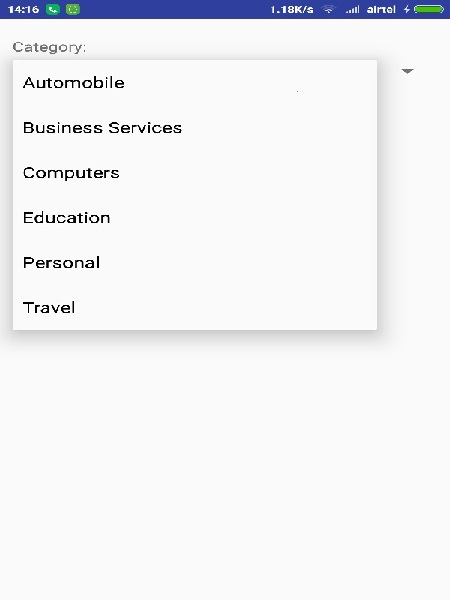
</application>

</manifest>

Let's try to run your AndroidSpinnerExample application. I assume you have connected your actual Android Mobile device with your computer. 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. Before starting your application,Android studio will display following window to select an option where you want to run your Android application.



If you click on spinner button, It will a drop down menu as shown below



# Android Toast Example

Andorid Toast can be used to display information for the short period of time. A toast contains message to be displayed quickly and disappears after sometime.

The android.widget.Toast class is the subclass of java.lang.Object class.

You can also create custom toast as well for example toast displaying image. You can visit next page to see the code for custom toast.

## Toast class

Toast class is used to show notification for a particular interval of time. After sometime it disappears. It doesn't block the user interaction.

#### Constants of Toast class

There are only 2 constants of Toast class which are given below.

|  |  |
| --- | --- |
| **Constant** | **Description** |
| public static final int LENGTH\_LONG | displays view for the long duration of time. |
| public static final int LENGTH\_SHORT | displays view for the short duration of time. |

#### Methods of Toast class

The widely used methods of Toast class are given below.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public static Toast makeText(Context context, CharSequence text, int duration) | makes the toast containing text and duration. |
| public void show() | displays toast. |
| public void setMargin (float horizontalMargin, float verticalMargin) | changes the horizontal and vertical margin difference. |

## Android Toast Example

1. Toast.makeText(getApplicationContext(),"Hello Javatpoint",Toast.LENGTH\_SHORT).show();

Another code:

1. Toast toast=Toast.makeText(getApplicationContext(),"Hello Javatpoint",Toast.LENGTH\_SHORT);
2. toast.setMargin(50,50);
3. toast.show();

Here, getApplicationContext() method returns the instance of Context.

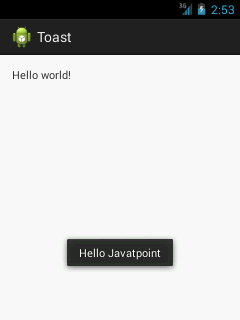
## Full code of activity class displaying Toast

Let's see the code to display the toast.

*File: MainActivity.java*

1. **package** com.example.toast;
2. **import** android.os.Bundle;
3. **import** android.app.Activity;
4. **import** android.view.Menu;
5. **import** android.view.View;
6. **import** android.widget.Toast;
8. **public** **class** MainActivity **extends** Activity {
9. @Override
10. **public** **void** onCreate(Bundle savedInstanceState) {
11. **super**.onCreate(savedInstanceState);
12. setContentView(R.layout.activity\_main);
14. //Displaying Toast with Hello Javatpoint message
15. Toast.makeText(getApplicationContext(),"Hello Javatpoint",Toast.LENGTH\_SHORT).show();
16. }
18. @Override
19. **public** **boolean** onCreateOptionsMenu(Menu menu) {
20. getMenuInflater().inflate(R.menu.activity\_main, menu);
21. **return** **true**;
22. }
24. }

#### Output:



# 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:"));

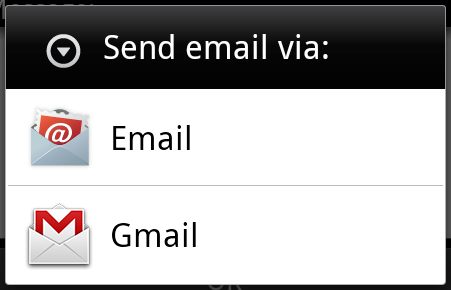
email.putExtra(Intent.EXTRA\_EMAIL, recipients);

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. |

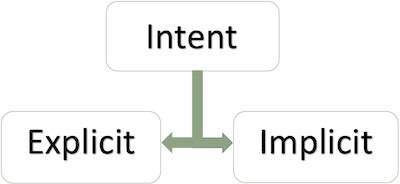
### Component Name

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().

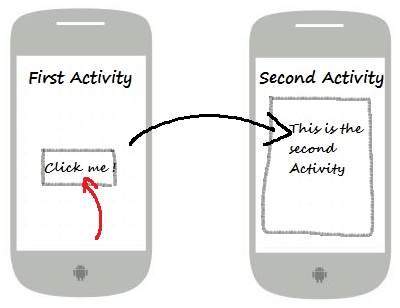
## Types of Intents

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 −

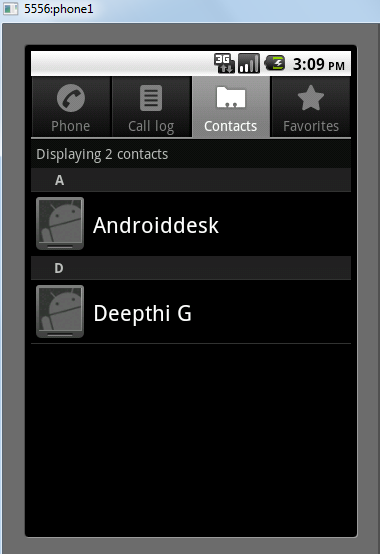
Intent read1=new Intent();

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" />

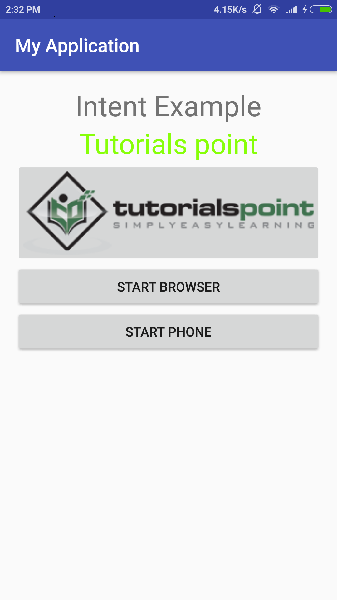
</intent-filter>

</activity>

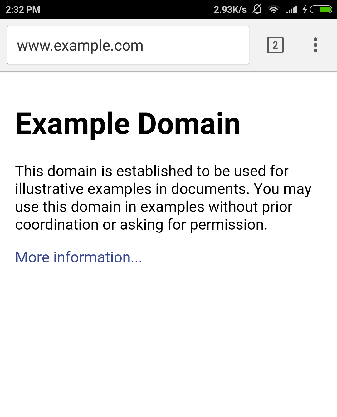
</application>

</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" />

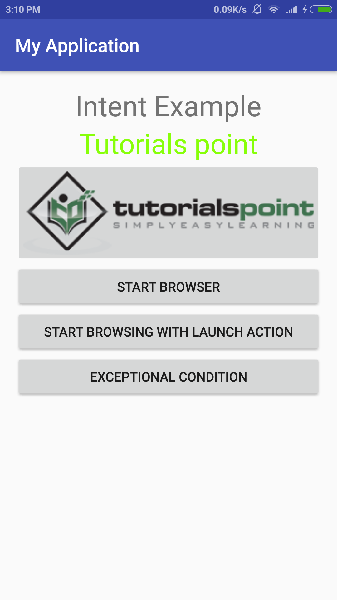
</intent-filter>

</activity>

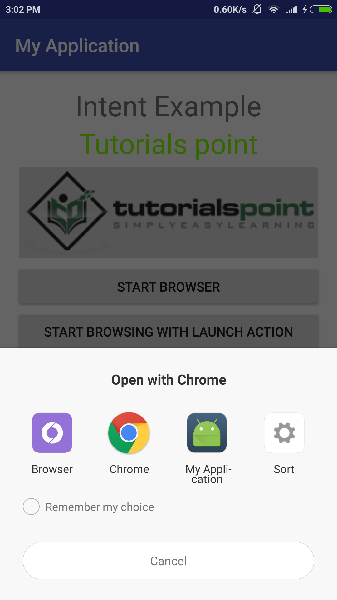
</application>

</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.

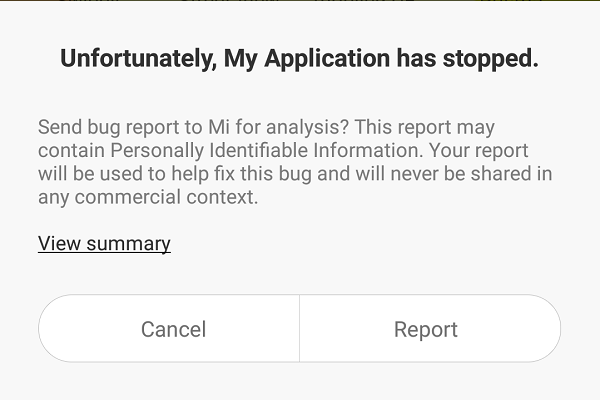


Now if you select Browser, then Android will launch web browser and open example.com website but if you select IndentDemo option then Android will launch CustomActivity which does nothing but just capture passed data and displays in a text view as follows −



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 −



# Android - Sending SMS

In Android, you can use SmsManager API or devices Built-in SMS application to send SMS's. In this tutorial, we shows you two basic examples to send SMS message −

**SmsManager API**

SmsManager smsManager = SmsManager.getDefault();

smsManager.sendTextMessage("phoneNo", null, "sms message", null, null);

**Built-in SMS application**

Intent sendIntent = new Intent(Intent.ACTION\_VIEW);

sendIntent.putExtra("sms\_body", "default content");

sendIntent.setType("vnd.android-dir/mms-sms");

startActivity(sendIntent);

Of course, both need **SEND\_SMS permission**.

<uses-permission android:name="android.permission.SEND\_SMS" />

Apart from the above method, there are few other important functions available in SmsManager class. These methods are listed below −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **ArrayList<String> divideMessage(String text)**  This method divides a message text into several fragments, none bigger than the maximum SMS message size. |
| 2 | **static SmsManager getDefault()**  This method is used to get the default instance of the SmsManager |
| 3 | **void sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)**  This method is used to send a data based SMS to a specific application port. |
| 4 | **void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliveryIntents)**  Send a multi-part text based SMS. |
| 5 | **void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)**  Send a text based SMS. |

## Example

Following example shows you in practical how to use SmsManager object to send an SMS to the given mobile number.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use Android Studio IDE to create an Android application and name it as *tutorialspoint* under a package *com.example.tutorialspoint*. |
| 2 | Modify *src/MainActivity.java* file and add required code to take care of sending sms. |
| 3 | Modify layout XML file *res/layout/activity\_main.xml* add any GUI component if required. I'm adding a simple GUI to take mobile number and SMS text to be sent and a simple button to send SMS. |
| 4 | No need to define default string constants at res/values/strings.xml. Android studio takes care of default constants. |
| 5 | Modify *AndroidManifest.xml* as shown below |
| 6 | 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.tutorialspoint/MainActivity.java**.

package com.example.tutorialspoint;

import android.Manifest;

import android.content.pm.PackageManager;

import android.os.Bundle;

import android.app.Activity;

import android.support.v4.app.ActivityCompat;

import android.support.v4.content.ContextCompat;

import android.telephony.SmsManager;

import android.util.Log;

import android.view.Menu;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.Toast;

public class MainActivity extends Activity {

private static final int MY\_PERMISSIONS\_REQUEST\_SEND\_SMS =0 ;

Button sendBtn;

EditText txtphoneNo;

EditText txtMessage;

String phoneNo;

String message;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

sendBtn = (Button) findViewById(R.id.btnSendSMS);

txtphoneNo = (EditText) findViewById(R.id.editText);

txtMessage = (EditText) findViewById(R.id.editText2);

sendBtn.setOnClickListener(new View.OnClickListener() {

public void onClick(View view) {

sendSMSMessage();

}

});

}

protected void sendSMSMessage() {

phoneNo = txtphoneNo.getText().toString();

message = txtMessage.getText().toString();

if (ContextCompat.checkSelfPermission(this,

Manifest.permission.SEND\_SMS)

!= PackageManager.PERMISSION\_GRANTED) {

if (ActivityCompat.shouldShowRequestPermissionRationale(this,

Manifest.permission.SEND\_SMS)) {

} else {

ActivityCompat.requestPermissions(this,

new String[]{Manifest.permission.SEND\_SMS},

MY\_PERMISSIONS\_REQUEST\_SEND\_SMS);

}

}

}

@Override

public void onRequestPermissionsResult(int requestCode,String permissions[], int[] grantResults) {

switch (requestCode) {

case MY\_PERMISSIONS\_REQUEST\_SEND\_SMS: {

if (grantResults.length > 0

&& grantResults[0] == PackageManager.PERMISSION\_GRANTED) {

SmsManager smsManager = SmsManager.getDefault();

smsManager.sendTextMessage(phoneNo, null, message, null, null);

Toast.makeText(getApplicationContext(), "SMS sent.",

Toast.LENGTH\_LONG).show();

} else {

Toast.makeText(getApplicationContext(),

"SMS faild, please try again.", Toast.LENGTH\_LONG).show();

return;

}

}

}

}

}

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

Here abc indicates about tutorialspoint logo

<?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="MainActivity">

<TextView

android:id="@+id/textView1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Sending SMS 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\_alignRight="@+id/imageButton"

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

<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:hint="Enter Phone Number"

android:phoneNumber="true"

android:textColorHint="@color/abc\_primary\_text\_material\_dark"

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

android:layout\_centerHorizontal="true" />

<EditText

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/editText2"

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

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

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

android:textColorHint="@color/abc\_primary\_text\_material\_dark"

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

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

android:hint="Enter SMS" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Send Sms"

android:id="@+id/btnSendSMS"

android:layout\_below="@+id/editText2"

android:layout\_centerHorizontal="true"

android:layout\_marginTop="48dp" />

</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">tutorialspoint</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.tutorialspoint" >

<uses-permission android:name="android.permission.SEND\_SMS" />

<application

android:allowBackup="true"

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

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

android:theme="@style/AppTheme" >

<activity

android:name="com.example.tutorialspoint.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>

Let's try to run your **tutorialspoint** application. I assume you have connected your actual Android Mobile device with your computer. 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. Before starting your application, Android studio installer will display following window to select an option where you want to run your Android application.



Now you can enter a desired mobile number and a text message to be sent on that number. Finally click on **Send SMS** button to send your SMS. Make sure your GSM/CDMA connection is working fine to deliver your SMS to its recipient.

You can take a number of SMS separated by comma and then inside your program you will have to parse them into an array string and finally you can use a loop to send message to all the given numbers. That's how you can write your own SMS client. Next section will show you how to use existing SMS client to send SMS.

## Using Built-in Intent to send SMS

You can use Android Intent to send SMS by calling built-in SMS functionality of the Android. Following section explains different parts of our Intent object required to send an SMS.

## Intent Object - Action to send SMS

You will use **ACTION\_VIEW** action to launch an SMS client installed on your Android device. Following is simple syntax to create an intent with ACTION\_VIEW action.

Intent smsIntent = new Intent(Intent.ACTION\_VIEW);

## Intent Object - Data/Type to send SMS

To send an SMS you need to specify **smsto:** as URI using setData() method and data type will be to **vnd.android-dir/mms-sms** using setType() method as follows −

smsIntent.setData(Uri.parse("smsto:"));

smsIntent.setType("vnd.android-dir/mms-sms");

## Intent Object - Extra to send SMS

Android has built-in support to add phone number and text message to send an SMS as follows −

smsIntent.putExtra("address" , new String("0123456789;3393993300"));

smsIntent.putExtra("sms\_body" , "Test SMS to Angilla");

Here address and sms\_body are case sensitive and should be specified in small characters only. You can specify more than one number in single string but separated by semi-colon (;).

## Example

Following example shows you in practical how to use Intent object to launch SMS client to send an SMS to the given recipients.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use Android studio IDE to create an Android application and name it as *tutorialspoint* under a package *com.example.tutorialspoint*. |
| 2 | Modify *src/MainActivity.java* file and add required code to take care of sending SMS. |
| 3 | Modify layout XML file *res/layout/activity\_main.xml* add any GUI component if required. I'm adding a simple button to launch SMS Client. |
| 4 | No need to define default constants.Android studio takes care of default constants. |
| 5 | Modify *AndroidManifest.xml* as shown below |
| 6 | 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.tutorialspoint/MainActivity.java**.

package com.example.tutorialspoint;

import android.net.Uri;

import android.os.Bundle;

import android.app.Activity;

import android.content.Intent;

import android.util.Log;

import android.view.Menu;

import android.view.View;

import android.widget.Button;

import android.widget.Toast;

public class MainActivity extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

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

startBtn.setOnClickListener(new View.OnClickListener() {

public void onClick(View view) {

sendSMS();

}

});

}

protected void sendSMS() {

Log.i("Send SMS", "");

Intent smsIntent = new Intent(Intent.ACTION\_VIEW);

smsIntent.setData(Uri.parse("smsto:"));

smsIntent.setType("vnd.android-dir/mms-sms");

smsIntent.putExtra("address" , new String ("01234"));

smsIntent.putExtra("sms\_body" , "Test ");

try {

startActivity(smsIntent);

finish();

Log.i("Finished sending SMS...", "");

} catch (android.content.ActivityNotFoundException ex) {

Toast.makeText(MainActivity.this,

"SMS faild, please try again later.", Toast.LENGTH\_SHORT).show();

}

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

// Inflate the menu; this adds items to the action bar if it is present.

getMenuInflater().inflate(R.menu.main, menu);

return true;

}

}

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

Here abc indicates about tutorialspoint logo

<?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:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Drag and Drop Example"

android:id="@+id/textView"

android:layout\_alignParentTop="true"

android:layout\_centerHorizontal="true"

android:textSize="30dp" />

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Tutorials Point "

android:id="@+id/textView2"

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

android:layout\_centerHorizontal="true"

android:textSize="30dp"

android:textColor="#ff14be3c" />

<ImageView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/imageView"

android:src="@drawable/abc"

android:layout\_marginTop="48dp"

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

android:layout\_centerHorizontal="true" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Compose SMS"

android:id="@+id/button"

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

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

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

android:layout\_marginTop="54dp"

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

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

</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">tutorialspoint</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.tutorialspoint" >

<application

android:allowBackup="true"

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

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

android:theme="@style/AppTheme" >

<activity

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

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

<intent-filter>

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

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

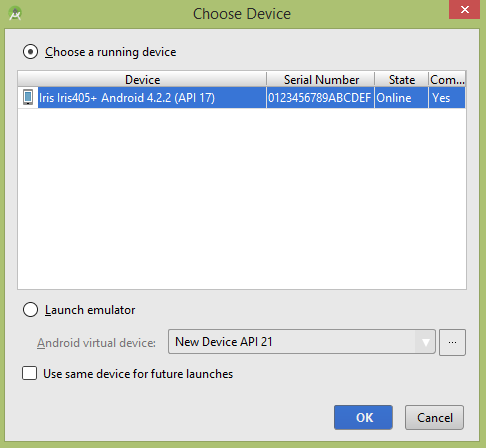
</intent-filter>

</activity>

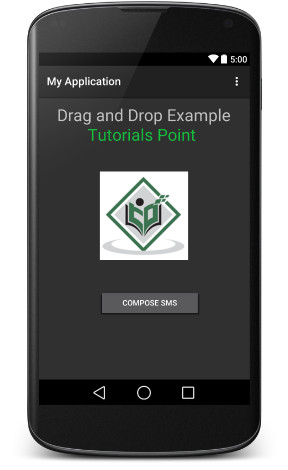
</application>

</manifest>

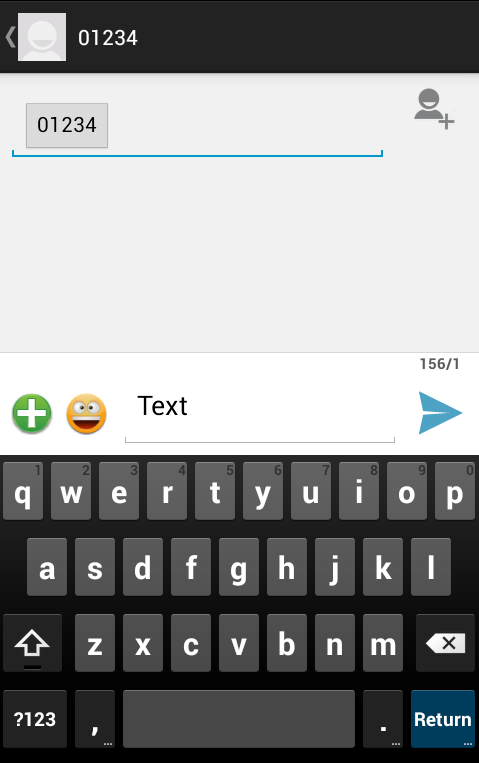
Let's try to run your **tutorialspoint** application. I assume you have connected your actual Android Mobile device with your computer. To run the app from Android studio, open one of your project's activity files and click Run Eclipse Run Icon icon from the toolbar. Before starting your application, Android studio will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen −



Now use **Compose SMS** button to launch Android built-in SMS clients which is shown below −



You can modify either of the given default fields and finally use send SMS button to send your SMS to the mentioned recipient.

# Android - Phone Calls

Android provides Built-in applications for phone calls, in some occasions we may need to make a phone call through our application. This could easily be done by using implicit Intent with appropriate actions. Also, we can use PhoneStateListener and TelephonyManager classes, in order to monitor the changes in some telephony states on the device.

This chapter lists down all the simple steps to create an application which can be used to make a Phone Call. You can use Android Intent to make phone call by calling built-in Phone Call functionality of the Android. Following section explains different parts of our Intent object required to make a call.

## Intent Object - Action to make Phone Call

You will use **ACTION\_CALL** action to trigger built-in phone call functionality available in Android device. Following is simple syntax to create an intent with ACTION\_CALL action

Intent phoneIntent = new Intent(Intent.ACTION\_CALL);

You can use **ACTION\_DIAL** action instead of ACTION\_CALL, in that case you will have option to modify hardcoded phone number before making a call instead of making a direct call.

## Intent Object - Data/Type to make Phone Call

To make a phone call at a given number 91-000-000-0000, you need to specify **tel:** as URI using setData() method as follows −

phoneIntent.setData(Uri.parse("tel:91-000-000-0000"));

The interesting point is that, to make a phone call, you do not need to specify any extra data or data type.

## Example

Following example shows you in practical how to use Android Intent to make phone call to the given mobile number.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

|  |  |
| --- | --- |
| **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/MainActivity.java* file and add required code to take care of making a call. |
| 3 | Modify layout XML file *res/layout/activity\_main.xml* add any GUI component if required. I'm adding a simple button to Call 91-000-000-0000 number |
| 4 | No need to define default string constants.Android studio takes care of default constants. |
| 5 | Modify *AndroidManifest.xml* as shown below |
| 6 | 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.saira\_000.myapplication;

import android.Manifest;

import android.content.Intent;

import android.content.pm.PackageManager;

import android.net.Uri;

import android.os.Bundle;

import android.support.v4.app.ActivityCompat;

import android.support.v7.app.AppCompatActivity;

import android.view.View;

import android.widget.Button;

public class MainActivity extends AppCompatActivity {

private Button button;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

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

button.setOnClickListener(new View.OnClickListener() {

public void onClick(View arg0) {

Intent callIntent = new Intent(Intent.ACTION\_CALL);

callIntent.setData(Uri.parse("tel:0377778888"));

if (ActivityCompat.checkSelfPermission(MainActivity.this,

Manifest.permission.CALL\_PHONE) != PackageManager.PERMISSION\_GRANTED) {

return;

}

startActivity(callIntent);

}

});

}

}

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

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

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

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="vertical" >

<Button

android:id="@+id/buttonCall"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="call 0377778888" />

</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.saira\_000.myapplication" >

<uses-permission android:name="android.permission.CALL\_PHONE" />

<application

android:allowBackup="true"

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

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

android:theme="@style/AppTheme" >

<activity

android:name="com.example.saira\_000.myapplication.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>

Let's try to run your **My Application** application. I assume you have connected your actual Android Mobile device with your computer. 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.Select your mobile device as an option and then check your mobile device which will display following screen −



Now use **Call**button to make phone call as shown below −

# Android Mobile Call Progress