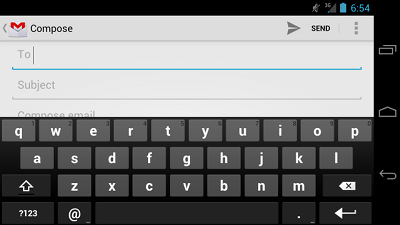
Text Fields

A text field allows the user to type text into your app. It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard. In addition to typing, text fields allow for a variety of other activities, such as text selection (cut, copy, paste) and data look-up via auto-completion.

You can add a text field to you layout with the EditText object. You should usually do so in your XML layout with a <EditText> element.



Specifying the Keyboard Type



**Figure 1.** The default text input type.



**Figure 2.** The textEmailAddress input type.



**Figure 3.** The phone input type.

Text fields can have different input types, such as number, date, password, or email address. The type determines what kind of characters are allowed inside the field, and may prompt the virtual keyboard to optimize its layout for frequently used characters.

You can specify the type of keyboard you want for your [EditText](http://developer.android.com/reference/android/widget/EditText.html)  object with the [android:inputType](http://developer.android.com/reference/android/widget/TextView.html" \l "attr_android:inputType) attribute. For example, if you want the user to input an email address, you should use the textEmailAddress input type:

<EditText  
    android:id="@+id/email\_address"  
    android:layout\_width="match\_parent"  
    android:layout\_height="wrap\_content"  
    android:hint="@string/email\_hint"  
    android:inputType="textEmailAddress" />

There are several different input types available for different situations. Here are some of the more common values for [android:inputType](http://developer.android.com/reference/android/widget/TextView.html#attr_android:inputType):

"text"

Normal text keyboard.

"textEmailAddress"

Normal text keyboard with the @ character.

"textUri"

Normal text keyboard with the / character.

"number"

Basic number keypad.

"phone"

Phone-style keypad.

Controlling other behaviors

The [android:inputType](http://developer.android.com/reference/android/widget/TextView.html" \l "attr_android:inputType) also allows you to specify certain keyboard behaviors, such as whether to capitalize all new words or use features like auto-complete and spelling suggestions.

The [android:inputType](http://developer.android.com/reference/android/widget/TextView.html" \l "attr_android:inputType) attribute allows bitwise combinations so you can specify both a keyboard layout and one or more behaviors at once.

Here are some of the common input type values that define keyboard behaviors:

"textCapSentences"

Normal text keyboard that capitalizes the first letter for each new sentence.

"textCapWords"

Normal text keyboard that capitalizes every word. Good for titles or person names.

"textAutoCorrect"

Normal text keyboard that corrects commonly misspelled words.

"textPassword"

Normal text keyboard, but the characters entered turn into dots.

"textMultiLine"

Normal text keyboard that allow users to input long strings of text that include line breaks (carriage returns).

For example, here's how you can collect a postal address, capitalize each word, and disable text suggestions:

<EditText  
    android:id="@+id/postal\_address"  
    android:layout\_width="match\_parent"  
    android:layout\_height="wrap\_content"  
    android:hint="@string/postal\_address\_hint"  
    android:inputType="textPostalAddress|  
                       textCapWords|  
                       textNoSuggestions" />

All behaviors are also listed with the [android:inputType](http://developer.android.com/reference/android/widget/TextView.html" \l "attr_android:inputType) documentation.

Specifying Keyboard Actions

http://developer.android.com/images/ui/edittext-actionsend.png

**Figure 4.** If you declareandroid:imeOptions="actionSend", the keyboard includes the Send action.

In addition to changing the keyboard's input type, Android allows you to specify an action to be made when users have completed their input. The action specifies the button that appears in place of the carriage return key and the action to be made, such as "Search" or "Send."

You can specify the action by setting the [android:imeOptions](http://developer.android.com/reference/android/widget/TextView.html#attr_android:imeOptions) attribute. For example, here's how you can specify the Send action:

<EditText  
    android:id="@+id/search"  
    android:layout\_width="match\_parent"  
    android:layout\_height="wrap\_content"  
    android:hint="@string/search\_hint"  
    android:inputType="text"  
    android:imeOptions="actionSend" />

If you do not explicitly specify an input action then the system attempts to determine if there are any subsequent[android:focusable](http://developer.android.com/reference/android/view/View.html#attr_android:focusable) fields. If any focusable fields are found following this one, the system applies the"actionNext" action to the current [EditText](http://developer.android.com/reference/android/widget/EditText.html) so the user can select Next to move to the next field. If there's no subsequent focusable field, the system applies the "actionDone" action. You can override this by setting the [android:imeOptions](http://developer.android.com/reference/android/widget/TextView.html#attr_android:imeOptions)  attribute to any other value such as "actionSend" or "actionSearch" or suppress the default behavior by using the "actionNone" action.

Responding to action button events

If you have specified a keyboard action for the input method using [android:imeOptions](http://developer.android.com/reference/android/widget/TextView.html" \l "attr_android:imeOptions) attribute (such as"actionSend"), you can listen for the specific action event using an [TextView.OnEditorActionListener](http://developer.android.com/reference/android/widget/TextView.OnEditorActionListener.html). The[TextView.OnEditorActionListener](http://developer.android.com/reference/android/widget/TextView.OnEditorActionListener.html) interface provides a callback method called [onEditorAction()](http://developer.android.com/reference/android/widget/TextView.OnEditorActionListener.html" \l "onEditorAction(android.widget.TextView, int, android.view.KeyEvent)) that indicates the action type invoked with an action ID such as [IME\_ACTION\_SEND](http://developer.android.com/reference/android/view/inputmethod/EditorInfo.html#IME_ACTION_SEND) or [IME\_ACTION\_SEARCH](http://developer.android.com/reference/android/view/inputmethod/EditorInfo.html#IME_ACTION_SEARCH).

For example, here's how you can listen for when the user clicks the Send button on the keyboard:

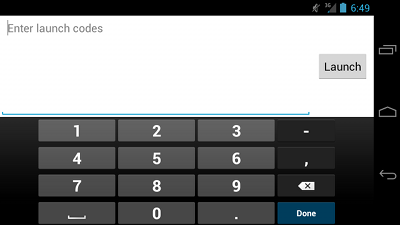
EditText editText = (EditText) findViewById(R.id.search);  
editText.setOnEditorActionListener(new OnEditorActionListener()

{  
    @Override  
    public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {  
        boolean handled = false;  
        if (actionId == EditorInfo.IME\_ACTION\_SEND) {  
            sendMessage();  
            handled = true;  
        }  
        return handled;  
    }  
});

Setting a custom action button label

If the keyboard is too large to reasonably share space with the underlying application (such as when a handset device is in landscape orientation) then fullscreen ("extract mode") is triggered. In this mode, a labeled action button is displayed next to the input. You can customize the text of this button by setting the [android:imeActionLabel](http://developer.android.com/reference/android/widget/TextView.html#attr_android:imeActionLabel) attribute:

<EditText  
    android:id="@+id/launch\_codes"  
    android:layout\_width="match\_parent"  
    android:layout\_height="wrap\_content"  
    android:hint="@string/enter\_launch\_codes"  
    android:inputType="number"  
    android:imeActionLabel="@string/launch" />

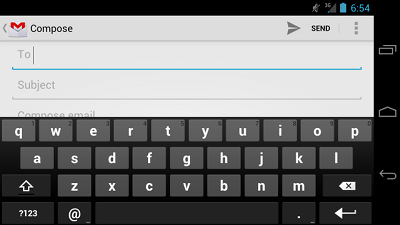


**Figure 5.** A custom action label with [android:imeActionLabel](http://developer.android.com/reference/android/widget/TextView.html" \l "attr_android:imeActionLabel).

Adding Other Keyboard Flags

In addition to the actions you can specify with the [android:imeOptions](http://developer.android.com/reference/android/widget/TextView.html" \l "attr_android:imeOptions) attribute, you can add additional flags to specify other keyboard behaviors. All available flags are listed along with the actions in the[android:imeOptions](http://developer.android.com/reference/android/widget/TextView.html#attr_android:imeOptions) documentation.

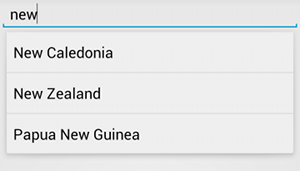
For example, figure 5 shows how the system enables a fullscreen text field when a handset device is in landscape orientation (or the screen space is otherwise constrained for space). You can disable the fullscreen input mode with flagNoExtractUi in the [android:imeOptions](http://developer.android.com/reference/android/widget/TextView.html" \l "attr_android:imeOptions) attribute, as shown in figure 6.



**Figure 6.** The fullscreen text field ("extract mode") is disabled with android:imeOptions="flagNoExtractUi".

Providing Auto-complete Suggestions

If you want to provide suggestions to users as they type, you can use a subclass of [EditText](http://developer.android.com/reference/android/widget/EditText.html) called[AutoCompleteTextView](http://developer.android.com/reference/android/widget/AutoCompleteTextView.html). To implement auto-complete, you must specify an [Adapter](http://developer.android.com/reference/android/widget/Adapter.html) that provides the text suggestions. There are several kinds of adapters available, depending on where the data is coming from, such as from a database or an array.



**Figure 7.** Example of [AutoCompleteTextView](http://developer.android.com/reference/android/widget/AutoCompleteTextView.html)  with text suggestions.

The following procedure describes how to set up an [AutoCompleteTextView](http://developer.android.com/reference/android/widget/AutoCompleteTextView.html) that provides suggestions from an array, using [ArrayAdapter](http://developer.android.com/reference/android/widget/ArrayAdapter.html):

1. Add the [AutoCompleteTextView](http://developer.android.com/reference/android/widget/AutoCompleteTextView.html) to your layout. Here's a layout with only the text field:

<?xml version="1.0" encoding="utf-8"?>  
<AutoCompleteTextView xmlns:android="http://schemas.android.com/apk/res/android"   
    android:id="@+id/autocomplete\_country"  
    android:layout\_width="match\_parent"  
    android:layout\_height="wrap\_content" />

1. Define the array that contains all text suggestions. For example, here's an array of country names that's defined in an XML resource file (res/values/strings.xml):

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string-array name="countries\_array">  
        <item>Afghanistan</item>  
        <item>Albania</item>  
        <item>Algeria</item>  
        <item>American Samoa</item>  
        <item>Andorra</item>  
        <item>Angola</item>  
        <item>Anguilla</item>  
        <item>Antarctica</item>  
        ...  
    </string-array>  
</resources>

1. In your [Activity](http://developer.android.com/reference/android/app/Activity.html) or [Fragment](http://developer.android.com/reference/android/app/Fragment.html), use the following code to specify the adapter that supplies the suggestions:

// Get a reference to the AutoCompleteTextView in the layout  
AutoCompleteTextView textView = (AutoCompleteTextView) findViewById(R.id.autocomplete\_country);  
// Get the string array  
String[] countries = getResources().getStringArray(R.array.countries\_array);  
// Create the adapter and set it to the AutoCompleteTextView   
ArrayAdapter<String> adapter =   
        new ArrayAdapter<String>(this, android.R.layout.simple\_list\_item\_1, countries);  
textView.setAdapter(adapter);

Here, a new [ArrayAdapter](http://developer.android.com/reference/android/widget/ArrayAdapter.html) is initialized to bind each item in the COUNTRIES string array to a [TextView](http://developer.android.com/reference/android/widget/TextView.html) that exists in the simple\_list\_item\_1 layout (this is a layout provided by Android that provides a standard appearance for text in a list).

Then assign the adapter to the [AutoCompleteTextView](http://developer.android.com/reference/android/widget/AutoCompleteTextView.html) by calling [setAdapter()](http://developer.android.com/reference/android/widget/AutoCompleteTextView.html" \l "setAdapter(T)).

# Android Option Menu Example

**Android Option Menus** are the primary menus of android. They can be used for settings, search, delete item etc.

Here, we are going to see two examples of option menus. First, the simple option menus and second, options menus with images.

Here, we are inflating the menu by calling the **inflate()** method of **MenuInflater** class. To perform event handling on menu items, you need to override **onOptionsItemSelected()** method of Activity class.

### Android Option Menu Example

Let's see how to create menu in android. Let's see the simple option menu example that contains three menu items.

#### activity\_main.xml

We have only one textview in this file.

*File: activity\_main.xml*

1. **<RelativeLayout** xmlns:androclass="http://schemas.android.com/apk/res/android"
2. xmlns:tools="http://schemas.android.com/tools"
3. android:layout\_width="match\_parent"
4. android:layout\_height="match\_parent"
5. android:paddingBottom="@dimen/activity\_vertical\_margin"
6. android:paddingLeft="@dimen/activity\_horizontal\_margin"
7. android:paddingRight="@dimen/activity\_horizontal\_margin"
8. android:paddingTop="@dimen/activity\_vertical\_margin"
9. tools:context=".MainActivity" **>**
11. **<TextView**
12. android:layout\_width="wrap\_content"
13. android:layout\_height="wrap\_content"
14. android:text="@string/hello\_world" **/>**
16. **</RelativeLayout>**

#### menu\_main.xml

It contains three items as show below. It is created automatically inside the res/menu directory.

*File: menu\_main.xml*

1. **<menu** xmlns:androclass="http://schemas.android.com/apk/res/android" **>**
2. **<item**  android:id="@+id/item1"
3. android:title="Item 1"**/>**
4. **<item**  android:id="@+id/item2"
5. android:title="Item 2"**/>**
6. **<item**  android:id="@+id/item3"
7. android:title="Item 3"**/>**
8. **</menu>**

#### Activity class

This class displays the content of menu.xml file and performs event handling on clicking the menu items.

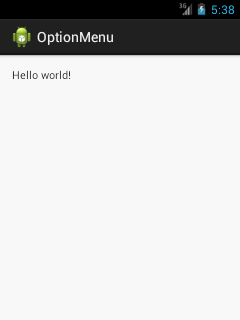
*File: MainActivity.java*

1. **package** com.javatpoint.optionmenu;
2. **import** android.os.Bundle;
3. **import** android.app.Activity;
4. **import** android.view.Menu;
5. **import** android.view.MenuItem;
6. **import** android.widget.Toast;
7. **public** **class** MainActivity **extends** Activity {
8. @Override
9. **protected** **void** onCreate(Bundle savedInstanceState) {
10. **super**.onCreate(savedInstanceState);
11. setContentView(R.layout.activity\_main);
12. }
13. @Override
14. **public** **boolean** onCreateOptionsMenu(Menu menu) {
15. // Inflate the menu; this adds items to the action bar if it is present.
16. getMenuInflater().inflate(R.menu.main, menu);//Menu Resource, Menu
17. **return** **true**;
18. }
19. @Override
20. **public** **boolean** onOptionsItemSelected(MenuItem item) {
21. **switch** (item.getItemId()) {
22. **case** R.id.item1:
23. Toast.makeText(getApplicationContext(),"Item 1 Selected",Toast.LENGTH\_LONG).show();
24. **return** **true**;
25. **case** R.id.item2:
26. Toast.makeText(getApplicationContext(),"Item 2 Selected",Toast.LENGTH\_LONG).show();
27. **return** **true**;
28. **case** R.id.item3:
29. Toast.makeText(getApplicationContext(),"Item 3 Selected",Toast.LENGTH\_LONG).show();
30. **return** **true**;
31. **default**:
32. **return** **super**.onOptionsItemSelected(item);
33. }
34. }
35. }

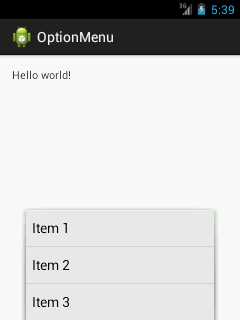
[download this android example](http://www.javatpoint.com/src/android/OptionMenu.zip)

#### Output:

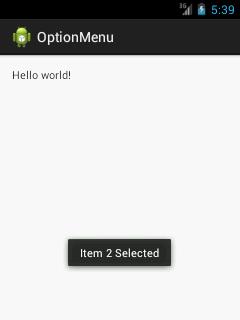
Output without clicking on the menu button.



Output after clicking on the menu button.



Output after clicking on the second menu item .



### Option Menu with Icon

You need to have icon images inside the res/drawable directory. The android:icon element is used to display the icon on the option menu. You can write the string information in the strings.xml file. But we have written it inside the menu\_main.xml file.

*File: menu\_main.xml*

1. **<menu** xmlns:androclass="http://schemas.android.com/apk/res/android" **>**
2. **<item**  android:id="@+id/item1"
3. android:icon="@drawable/add"
4. android:title="Item 1"**/>**
5. **<item**  android:id="@+id/item2"
6. android:icon="@drawable/minus"
7. android:title="Item 2"**/>**
8. **<item**  android:id="@+id/item3"
9. android:icon="@drawable/delete"
10. android:title="Item 3"**/>**
11. **</menu>**

Android Video Player Example

By the help of **MediaController** and **VideoView** classes, we can play the video files in android.

**MediaController class**

The **android.widget.MediaController** is a view that contains media controls like play/pause, previous, next, fast-forward, rewind etc

**VideoView class**

The **android.widget.VideoView** class provides methods to play and control the video player. The commonly used methods of VideoView class are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| **public void setMediaController(MediaController controller)** | sets the media controller to the video view. |
| **public void setVideoURI (Uri uri)** | sets the URI of the video file. |
| **public void start()** | starts the video view. |
| **public void stopPlayback()** | stops the playback. |
| **public void pause()** | pauses the playback. |
| **public void suspend()** | suspends the playback. |
| **public void resume()** | resumes the playback. |
| **public void seekTo(int millis)** | seeks to specified time in miliseconds. |

Android Media Player Example

We can play and control the audio files in android by the help of **MediaPlayer class**.

Here, we are going to see a simple example to play the audio file. In the next page, we will see the example to control the audio playback like start, stop, pause etc.

MediaPlayer class

The **android.media.MediaPlayer** class is used to control the audio or video files.

**Methods of MediaPlayer class**

There are many methods of MediaPlayer class. Some of them are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| **public void setDataSource(String path)** | sets the data source (file path or http url)  to use. |
| **public void prepare()** | prepares the player for playback synchronously. |
| **public void start()** | it starts or resumes the playback. |
| **public void stop()** | it stops the playback. |
| **public void pause()** | it pauses the playback. |
| **public boolean isPlaying()** | checks if media player is playing. |
| **public void seekTo(int millis)** | seeks to specified time in miliseconds. |
| **public void setLooping(boolean looping)** | sets the player for looping or non-looping. |
| **public boolean isLooping()** | checks if the player is looping or non-looping. |
| **public void selectTrack(int index)** | it selects a track for the specified index. |
| **public int getCurrentPosition()** | returns the current playback position. |
| **public int getDuration()** | returns duration of the file. |
| **public void setVolume(float leftVolume,float rightVolume)** | sets the volume on this player. |