Android UI: Layouts & Widgets

CS 236503

Dr. Oren Mishali

Based on the Official Android Development Guide

Android UI – Outline

Basic

- Layouts

Input Controls

- Button
- EditText
- AutoCompleteTextView
- CheckBox
- RadioButton
- ToggleButton/Switch
- Spinner
- Pickers
- SeekBar

Advanced

- The App Bar
- ListView

Understanding Layouts

"A layout defines the visual structure for a user interface"

Declaring a Layout

- A layout can be declared in two ways, via XML, or programmatically
  - By creating and manipulating **View** and **ViewGroup** objects at runtime
- A combined approach
  - Declaring a base XML layout, and manipulate programmatically
- Use XML whenever possible, better and more flexible design

XML Layout

- Each XML layout file must contain exactly one root element
  - Additional nested layout and widget elements may be added

```
<?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" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```


Loading an XML Layout

- An XML layout file is compiled into a **View** resource
  - Which is loaded in *Activity.onCreate()*

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```


The ID Attribute

- Any View object **may** have an integer ID associated with it, to *uniquely identify* the View within the tree

```
android:id="@+id/my_button"
```

- The @ symbol identifies the string as an ID resource
- The + symbol tells the parser it is a new resource name
  - Hence should be added to R.java
  - Is not needed when referencing previously-defined IDs
- Defining IDs is important when creating a **RelativeLayout**

The ID Attribute (2)

- A common pattern is to define a widget in the layout file with an id:

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

- And then retrieve its instance in *onCreate()*:

```
Button myButton = (Button) findViewById(R.id.my_button);
```


Should an ID be unique?

14

In documentation You may read

> **An ID need not be unique** throughout the entire tree, **but it should be** unique within the part of the tree you are searching (which may often be the entire tree, so it's best to be completely unique when possible).

It means there will be no exceptions if You use same id for all Your views but obviously layout will get useless then.

FindViewById works simply by traversing a tree until it finds first element with searched id and returns it (or null if doesn't find). If You had few elements with same id in tree You will always get same element, the one that is first in tree.

You may have plenty of fragments inflated with same layout just like you have ListView with each element having same layout, that is because inflater doesn't care about id values. It simply reads XML file and create a tree with correct view objects nothing more.

http://stackoverflow.com/questions/18067426/are-android-view-id-supposed-to-be-unique

Width and Height of a View

- Each view is required to define **layout_width** and **layout_height**
- It is possible to define exact width/height, however the use of these constants is more common
  - ***wrap_content*** tells the view to size itself to the dimensions required by its content
  - ***match_parent*** tells the view to become as big as its parent view group will allow
- When an exact width/height is needed
  - Do not use absolute units such as pixels
  - Use relative measurements such as density-independent pixel units (*dp*)

Common Layouts

- **LinearLayout**
  - Organizes its children into a single horizontal or vertical row
  - Creates a scrollbar if the length of the window exceeds the length of the screen
- **RelativeLayout**
  - Enables to specify the location of child objects relative to each other or to the parent
- **WebView**
  - Displays web pages

Tip: for better performance, keep layout hierarchy as shallow as possible

Layout Weight

- LinearLayout supports **layout_weight**
- Remaining space is assigned to children in the proportion of their weight
- Default weight is zero

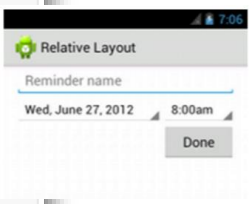
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/re
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```

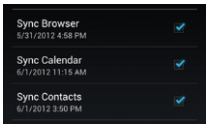
Relative Layout

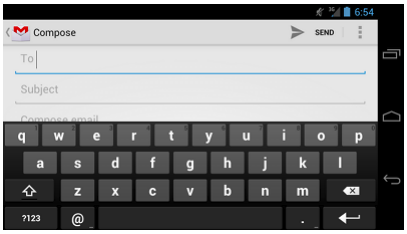
- A very powerful utility for designing a user interface
- Many nested LinearLayouts? consider **RelativeLayout** instead
- By default, all child views are drawn at the top-left of the layout
- So the position of each view must be defined using the various layout properties

See RelativeLayout.LayoutParams for all attributes available

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/r
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```







Input Controls

Button

- A push-button that can be pressed, or clicked, to perform an action
- Consists of text or an icon (or both text and an icon)
- Use ***Button*** element to create a button with text or text + icon
- Use ***ImageButton*** element to create icon-only button

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:drawableLeft="@drawable/button_icon"
    ... />
```

Optional attribute to add an icon

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    ... />
```


Responding to Click Events

1. Add to ***Button*** element the ***android:onClick*** attribute

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

A Button may be styled in various options

2. Add handler method to the ***Activity*** class

```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

Use the exact method signature

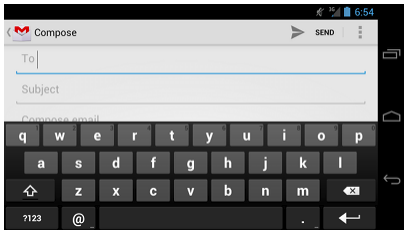
Responding to Click Events (2)

- Alternatively, define an event handler programmatically

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```


EditText

- A text field, allows the user to type text into the app
- Can be either single line or multi-line



EditText – *android:inputType* attribute

- Specifies the input type
- The type determines what kind of characters are allowed
- May prompt the virtual keyboard to optimize its layout

```
<EditText
    android:id="@+id/email_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/email_hint"
    android:inputType="textEmailAddress" />
```

- More types: “text”, “number”, “phone”…

EditText – *android:imeOptions* attribute

- Allows to specify an action to be made when users have completed their input, e.g., “Search”, “Send”
- The action’s button replaces the carriage return key
- In the code, you may listen for the specific action event

```
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;
    }
});
```

Interface

Callback method

```
<EditText
    android:id="@+id/search"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/search_hint"
    android:inputType="text"
    android:imeOptions="actionSend" />
```

The keyboard now includes a **Send** action

AutoCompleteTextView extends EditText

- Use it to provide suggestions to users as they type
- An **Adapter** should be specified that provides the suggestions

1

```
<?xml version="1.0" encoding="utf-8"?>
<AutoCompleteTextView xmlns:android="http://schema
    android:id="@+id/autocomplete_country"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

Use match_parent

2

```
<?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>
```

Note the string array definition in res/values/strings.xml

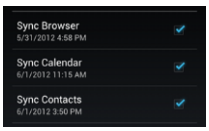
3

```
// 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);
```

This is an example of an array adapter. Data may come from other sources as well

CheckBox

- Allows the user to select one or more options from a set
- Typically, you should present each checkbox option in a vertical list



1

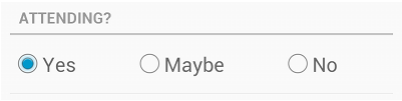
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.androi
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <CheckBox android:id="@+id/checkbox_meat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat"
        android:onClick="onCheckboxClicked"/>
    <CheckBox android:id="@+id/checkbox_cheese"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheese"
        android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

2

```
public void onCheckboxClicked(View view) {
    // Is the view now checked?
    boolean checked = ((CheckBox) view).isChecked();

    // Check which checkbox was clicked
    switch(view.getId()) {
        case R.id.checkbox_meat:
            if (checked)
                // Put some meat on the sandwich
            else
                // Remove the meat
            break;
        case R.id.checkbox_cheese:
            if (checked)
                // Cheese me
            else
                // I'm lactose intolerant
            break;
        // TODO: Veggie sandwich
    }
}
```

RadioButton



- Allows the user to select one option from a set
- Use it when the user needs to see the options side-by-side
  - Otherwise, use a **Spinner**
- The method ***onRadioButtonClicked*** handles the click event

RadioGroup ensures that only one radio button can be selected at a time

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```


ToggleButton / Switch

- Allow the user to change a setting between two states
- From API 14, you may use a ***Switch***, that provides a slider control

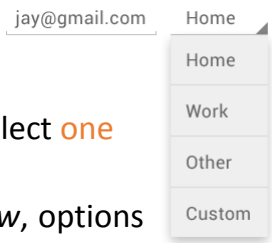
Toggle button

Switches

Responding to button presses

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            // The toggle is enabled
        } else {
            // The toggle is disabled
        }
    }
});
```

Spinner



- Provides a quick way to select one value from a set
- Like *AutoCompleteTextView*, options may be provided via a string-array, and populated via an *Adapter*
  - See an example here

```
<Spinner
    android:id="@+id/planets_spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

When an item is selected, an *onItemSelected* event is received

```
public class SpinnerActivity extends Activity implements OnItemSelectedListener {
    ...

    public void onItemSelected(AdapterView<?> parent, View view,
            int pos, long id) {
        // An item was selected. You can retrieve the selected item using
        // parent.getItemAtPosition(pos)
    }

    public void onNothingSelected(AdapterView<?> parent) {
        // Another interface callback
    }
}
```

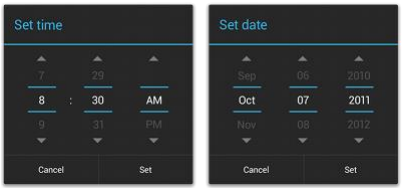
The interface implementation should be specified

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
spinner.setOnItemSelectedListener(this);
```

Where?

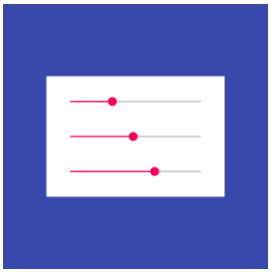
Pickers

- Allow the user to pick a time or a date as ready-to-use dialogs
  - Using them helps ensure that a valid time or date is picked
- Pickers are defined programmatically
  - And not via XML
  - Wrapped in a *DialogFragment*
  - Added to the UI, e.g., as a Button that opens the dialog
- See implementation examples here



SeekBar (slider)

- ***max*** attribute (integer) defines the maximum it can take
- ***progress*** (integer) defines the default progress value
- Listeners are set to read user selection (see example code in the tutorial below)



```
<SeekBar
android:id="@+id/simpleSeekBar"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:max="200"
android:progress="50"/><!-- set 150 maximum value for the progress -->
```

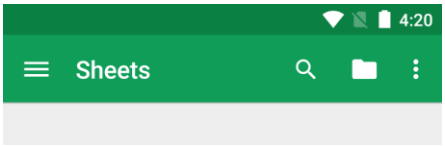
Tutorial: http://abhiandroid.com/ui/seekbar

The App Bar (Action Bar)

- The App Bar, previously called Action Bar, "*is one of the most important design elements in your app's activities*"
- http://developer.android.com/training/appbar/index.html

App Bar's Key Functions

- A dedicated space for giving your app an identity and indicating the user's location in the app
- Access to important actions in a predictable way, such as search
- Support for navigation and view switching (with tabs or drop-down lists)



The *ToolBar* Widget

- We will use the ***ToolBar*** widget as an app bar
  - From the **v7 appcompat** support library
- There are other ways such as using the ***ActionBar***
- However ***ToolBar*** works on the widest range of devices and its usage is more recommended

Add a Toolbar to an Activity

1. Add the v7 appcompat support library to the project
2. Make sure the activity extends ***AppCompatActivity***
3. In the app manifest, set the <application> element to use one of appcompat's NoActionBar themes
4. Add a ***ToolBar*** to the activity's layout
5. In ***onCreate()***, call ***setSupportActionBar()***, and pass it the activity's toolbar

This prevents the app from using the native ***ActionBar***

To access various utility methods:
ActionBar bar = getSupportActionBar();
bar.hide()
// and many more…

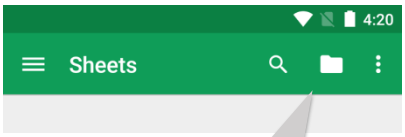
See Example Code

The *options menu*

- The options menu is the primary collection of menu items for an activity
- It is where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."
- To specify the options menu for an activity, override *onCreateOptionsMenu*:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

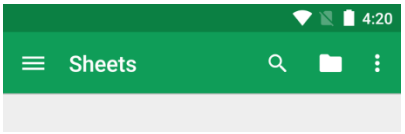
This is an XML menu resource (next slides)



The options menu includes visible buttons and hidden ones (overflow)

Adding and Handling Actions

- The app bar allows to add buttons for user actions
- Most important actions should be visible
- Others should be placed in the *overflow* menu
- To define an action, add an ***<item>*** element in the corresponding XML menu resource under res/menu/
- When the user selects an action item
  - The ***onOptionsItemSelected()*** callback is called
  - Provided with a ***MenuItem*** object



Defining Action Items in XML

showAsAction: Whether the action should be shown on the app bar

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

    <!-- "Mark Favorite", should appear as action button if possible -->
    <item
        android:id="@+id/action_favorite"
        android:icon="@drawable/ic_favorite_black_48dp"
        android:title="@string/action_favorite"
        app:showAsAction="ifRoom"/>

    <!-- Settings, should always be in the overflow -->
    <item android:id="@+id/action_settings"
          android:title="@string/action_settings"
          app:showAsAction="never"/>

</menu>
```

You can find many useful icons in the Materials Icons page

Responding to an Action Item

More on menus in this API guide

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_settings:
            // User chose the "Settings" item, show the app settings UI...
            return true;

        case R.id.action_favorite:
            // User chose the "Favorite" action, mark the current item
            // as a favorite...
            return true;

        default:
            // If we got here, the user's action was not recognized.
            // Invoke the superclass to handle it.
            return super.onOptionsItemSelected(item);

    }
}
```

If the method does not recognize the user's action, it invokes the superclass method

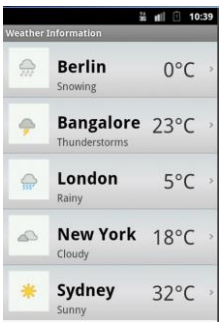
Adding an Up Action

- Help users to find their way back to the app's main screen
- One way to do this is to provide an *Up* button on the app bar for all activities except the main one
- The implementation involves
  - Declaring the activity's parent in the manifest
  - enabling (in the code) the app bar's *Up* button
- See example code in this tutorial page

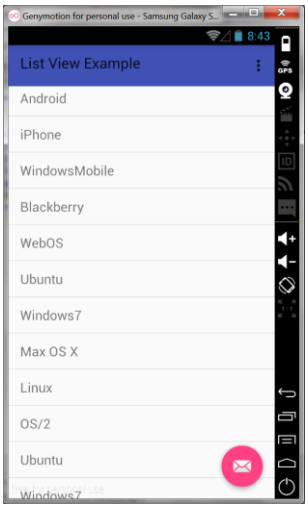
App Bar – Additional Reading

- Action Views and Action Providers
  - Actions with richer functionality
- Creating a search interface

List View



How to Create a Simple Text-Based List View?



Define a *ListView* In the XML

MainActivity.java × content_main.xml × activity_listview.xml ×

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/and
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".ListActivity" >

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?android:attr/actionBarSize"
        android:theme="@style/ThemeOverlay.AppCompat.ActionBar"/>

    <ListView
        android:id="@+id/listview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```


Setting an Array Adapter

```
public class MainActivity extends AppCompatActivity {

    private String[] values = new String[] { "Android", "iPhone", "WindowsMobile",
            "Blackberry", "WebOS", "Ubuntu", "Windows7", "Max OS X",
            "Linux", "OS/2", "Ubuntu", "Windows7", "Max OS X", "Linux",
            "OS/2", "Ubuntu", "Windows7", "Max OS X", "Linux", "OS/2",
            "Android", "iPhone", "WindowsMobile" };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);
        setSupportActionBar((Toolbar)findViewById(R.id.toolbar));

        ListView listview = (ListView) findViewById(R.id.listview);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, R.layout.row_text, values);
        listview.setAdapter(adapter);
    }
```


About ArrayAdapter

- A concrete *BaseAdapter*
- Backed by an array of arbitrary objects
- By default, fills a provided *TextView* with an object's ***toString()***
  - The layout we have provided has a *TextView* element
  - You may reuse a provided layout, e.g., *android.R.layout.simple_list_item_1*
- To use a more complex row layout override ***getView()***
  - We will see an example

Handling a Click Event

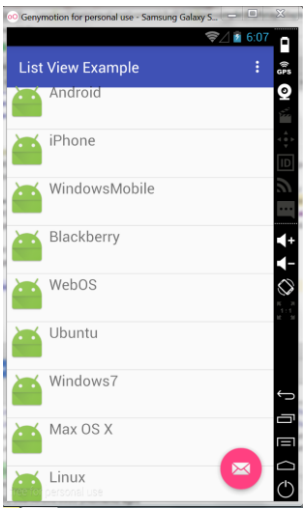
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);
    setSupportActionBar((Toolbar)findViewById(R.id.toolbar));

    ListView listview = (ListView) findViewById(R.id.listview);
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, R.layout.row_text, values);
    listview.setAdapter(adapter);

    listview.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view,
                                int position, long id) {
            Toast.makeText(getApplicationContext(),
                    ((TextView)view).getText() + " clicked, item number " + position, Toast.LENGTH_LONG)
                    .show();
        }
    });
}
```


How to Create a Complex Row Layout?



Define a Complex Row Layout in XML

```
<?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="?android:attr/listPreferredItemHeight"
    android:orientation="horizontal">

    <ImageView
        android:id="@+id/icon"
        android:layout_width="100px"
        android:layout_height="100px"
        android:layout_marginLeft="4px"
        android:layout_marginRight="10px"
        android:layout_marginTop="4px">
    </ImageView>

    <TextView
        android:id="@+id/label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@+id/label"
        android:textSize="20sp" >
    </TextView>

</LinearLayout>
```


Define a Custom Adapter

```
public class MyArrayAdapter extends ArrayAdapter<String> {
    private final Context context;
    private final String[] values;

    public MyArrayAdapter(Context context, String[] values) {
        super(context, R.layout.row_complex, values);
        this.context = context;
        this.values = values;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater) context
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        View rowView = inflater.inflate(R.layout.row_complex, parent, false);
        TextView textView = (TextView) rowView.findViewById(R.id.label);
        ImageView imageView = (ImageView) rowView.findViewById(R.id.icon);

        textView.setText(values[position]);
        imageView.setImageResource(R.mipmap.ic_launcher);

        return rowView;
    }
}
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);
    setSupportActionBar((Toolbar)findViewById(R.id.toolbar));

    ListView listview = (ListView) findViewById(R.id.listview);
    MyArrayAdapter adapter = new MyArrayAdapter(this, values);
    listview.setAdapter(adapter);
```

Read *getView*'s Javadoc for more info

Additional Reading

- http://www.vogella.com/tutorials/AndroidListView/article.html