## Android Adapters

## What are Adapters?

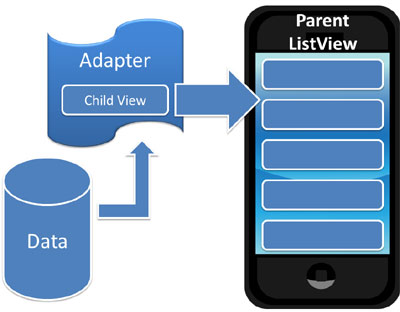
Adapters are the link between a set of data and the AdapterView that displays the data.

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter holds the data and send the data to adapter view, the view can takes the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.

##### **AdapterView**

AdapterViews are ViewGroups that display child views given to it by an adapter. An example of an AdapterView is a ListView.

Adapters also provide the child views that display the data in the AdapterView. Adapters are responsible for supplying the data and creating the views representing each item.



Adapters get the data and pass it, along with a child view, to the parent AdapterView which displays the child view and the data

The Android Framework has a set of native adapters that are easy to use. You can also create your own custom adapters if you wish.

Two of Android’s common adapters are:

## ArrayAdapter

An ArrayAdapter is an adapter backed by an array of objects. It links the array to the Adapter View.

The default ArrayAdapter converts an array item into a String object putting it into a TextView. The text view is then displayed in the AdapterView (a ListView for example).

When we create the adapter, we need to supply the layout for displaying each array string. We can define our own or use one of Android’s, such as:

**android.R.layout.simple\_list\_item\_1**

## SimpleCursorAdapter

The SimpleCursorAdapter links the data contained in a Cursor to an Adapter View.

##### **Cursors**

A cursor is a set of data. We usually get a cursor when we do a database query. The result of our query is contained in the cursor.

# Cursor is the Interface which represents a 2 dimensional table of any database. When we try to retrieve some data using SELECT statement, then the database will first create a CURSOR object and return its reference to us.

In simple words, Cursor is a Interface which returns collection of our query data. moveToFirst() is used to point the cursor position from where we want to get data from our cursor. There are methods

moveToLast(), moveToNext(), moveToPrevious(), moveToPosition(position)

by which we can iterate through our cursor by desired way.

### CursorAdapter

A CursorAdapter links a Cursor’s data to a List View. You must include the database’s \_id column as it’s used in processing the list item’s selection.

### SimpleAdapter

The SimpleAdapter links static data to views defined in a layout file. You specify the data as an ArrayList of Maps. Each entry in the ArrayList will display as a row in a list.

The SimpleCursorAdapter binds the Cursor data to an Adapter View. We define a layout that controls how each row of data is displayed.

Each row’s view is populated using the column values of the corresponding row in the cursor.

This layout is then displayed in the Adapter View, like a ListView for example.



The SimpleCursorAdapter gets the data out of the Cursor, puts each row of data in a layout that you define and then displays it in the Adapter View

When we construct the SimpleCursorAdapter, we specify which column’s data is to be retrieved from the cursor. We also specify which fields in our layout are to display this data. The adapter then creates a new view for each cursor entry and populates it with the corresponding cursor column values.

**Other Adapters**

### BaseAdapter

The BaseAdapter is a common base class for an Adapter that can be used in a ListView and a Spinner.

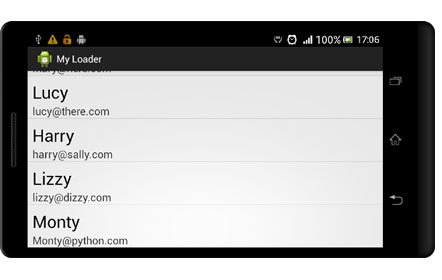
### ListAdapter

The ListAdapter links the data and a ListView displaying the data. The List View can display any data type provided it’s wrapped in a ListAdapter.

#### ListView

A ListView is a ViewGroup that displays a list of scrollable items. Items are inserted in the list using an Adapter.

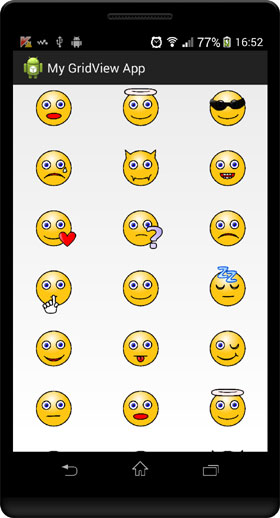
The Adapter gets the data from a source such as an array, converts each item into a view and places the view in the list.



A ListView displaying a list of items

#### GridView

A GridView is a ViewGroup that display items in a 2 dimensional scrollable grid.An Adapter gets the items from a data source, creates a view to enclose the item and then inserts the view in the parent grid.



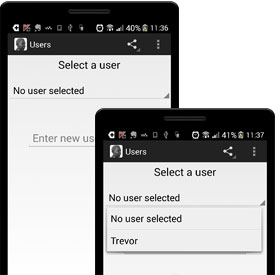
Use an Adapter to populate a GridView

### SpinnerAdapter

The SpinnerAdapter links the data to a Spinner.

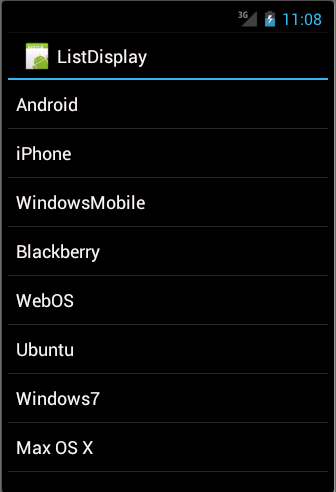
The data can come from any source but must be provided by a SpinnerAdapter (for example an ArrayAdapter or a CursorAdapter).

Note that the BaseAdapter implements SpinnerAdapter. Both the CursorAdapter and the ArrayAdapter are subclasses of BaseAdpater so we can also use any of these to load our Spinner.



**Android List View**

Android **ListView** is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an **Adapter** that pulls content from a source such as an array or database.



#### LIST VIEW

The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

We will see separate examples for both the adapters.

## ListView Attributes

Following are the important attributes specific to GridView –

|  |  |
| --- | --- |
| **Sr.No** | **Attribute & Description** |
| 1 | **android:id**  This is the ID which uniquely identifies the layout. |
| 2 | **android:divider**  This is drawable or color to draw between list items. |
| 3 | **android:dividerHeight**  This specifies height of the divider. This could be in px, dp, sp, in, or mm. |
| 4 | **android:entries**  Specifies the reference to an array resource that will populate the ListView. |
| 5 | **android:footerDividersEnabled**  When set to false, the ListView will not draw the divider before each footer view. The default value is true. |
| 6 | **android:headerDividersEnabled**  When set to false, the ListView will not draw the divider after each header view. The default value is true. |

## ArrayAdapter

We can use this adapter when our data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a **TextView**. Consider we have an array of strings we want to display in a ListView, initialize a new **ArrayAdapter** using a constructor to specify the layout for each string and the string array −

ArrayAdapter adapter = new ArrayAdapter<String>(this,R.layout.ListView,StringArray);

Here are arguments for this constructor −

* First argument **this** is the application context. Most of the case, keep it **this**.
* Second argument will be layout defined in XML file and having **TextView** for each string in the array.
* Final argument is an array of strings which will be populated in the text view.

Once we have array adapter created, then simply call **setAdapter()** on your **ListView** object as follows −

ListView listView = (ListView) findViewById(R.id.listview);

listView.setAdapter(adapter);

We will define our list view under res/layout directory in an XML file. For our example we are going to using activity\_main.xml file.

### Example

Following is the example which will take you through simple steps to show how to create your own Android application using ListView. Follow the following steps to modify the Android application we created in *Hello World Example* chapter –

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | We will use Android Studio IDE to create an Android application and name it as *ListDisplay* under a package *com.example.ListDisplay* as explained in the *Hello World Example* chapter. |
| 2 | Modify the default content of *res/layout/activity\_main.xml* file to include ListView content with the self explanatory attributes. |
| 3 | No need to change string.xml, Android studio takes care of default string constants. |
| 4 | Create a Text View file *res/layout/activity\_listview.xml*. This file will have setting to display all the list items. So we can customize its fonts, padding, color etc. using this file. |
| 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.ListDisplay/ListDisplay.java**. This file can include each of the fundamental life cycle methods.

package com.example.ListDisplay;

import android.os.Bundle;

import android.app.Activity;

import android.view.Menu;

import android.widget.ArrayAdapter;

import android.widget.ListView;

public class ListDisplay extends Activity {

// Array of strings...

String[] mobileArray = {"Android","IPhone","WindowsMobile","Blackberry",

"WebOS","Ubuntu","Windows7","Max OS X"};

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

ArrayAdapter adapter = new ArrayAdapter<String>(this,

R.layout.activity\_listview, mobileArray);

ListView listView = (ListView) findViewById(R.id.mobile\_list);

listView.setAdapter(adapter);

}

}

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

<LinearLayout 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:orientation="vertical"

tools:context=".ListActivity" >

<ListView

android:id="@+id/mobile\_list"

android:layout\_width="match\_parent"

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

</ListView>

</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">ListDisplay</string>

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

</resources>

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

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

<!-- Single List Item Design -->

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

android:id="@+id/label"

android:layout\_width="fill\_parent"

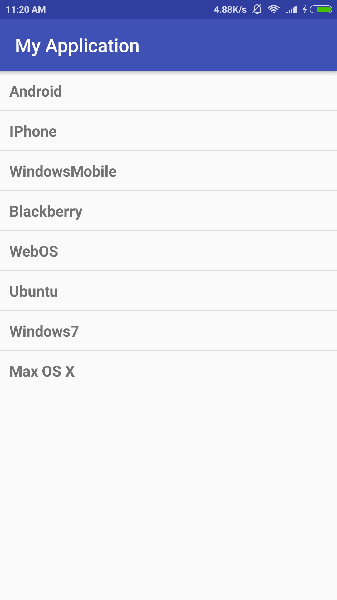
android:layout\_height="fill\_parent"

android:padding="10dip"

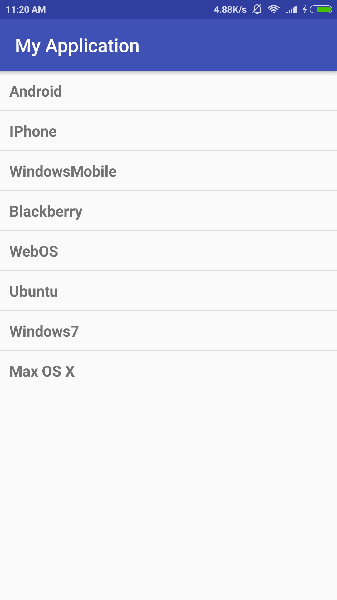
android:textSize="16dip"

android:textStyle="bold" >

</TextView>



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



## SimpleCursorAdapter

We can use this adapter when our data source is a database Cursor. When using *SimpleCursorAdapter*, we must specify a layout to use for each row in the **Cursor** and which columns in the Cursor should be inserted into which views of the layout.

For example, if we want to create a list of people's names and phone numbers, we can perform a query that returns a Cursor containing a row for each person and columns for the names and numbers. We then create a string array specifying which columns from the Cursor we want in the layout for each result and an integer array specifying the corresponding views that each column should be placed −

String[] fromColumns = {ContactsContract.Data.DISPLAY\_NAME,

ContactsContract.CommonDataKinds.Phone.NUMBER};

int[] toViews = {R.id.display\_name, R.id.phone\_number};

When we instantiate the SimpleCursorAdapter, pass the layout to use for each result, the Cursor containing the results, and these two arrays −

SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,

R.layout.person\_name\_and\_number, cursor, fromColumns, toViews, 0);

ListView listView = getListView();

listView.setAdapter(adapter);

The SimpleCursorAdapter then creates a view for each row in the Cursor using the provided layout by inserting each from Columns item into the corresponding **toViews** view.