UNIT – III
UI DESIGN AND DATA STORAGE

The basic unit of an Android application is an *activity*. An *activity* displays the user interface of your application, which may contain widgets like buttons, labels, text boxes, and so on. Typically, you define your UI using an XML file. During run time, you load the XML UI in the onCreate() event handler in your Activity class, using the setContentView(**R.layout.main**) method of the Activity class. During compilation, each element in the XML fi le is compiled into its equivalent Android GUI class, with attributes represented by methods. The Android system then creates the UI of the activity when it is loaded.

VIEWS AND VIEWGROUPS

An activity contains Views and ViewGroups. A view is a widget that has an appearance on screen. Examples of views are buttons, labels, and text boxes. A view derives from the base class **android.view.View**.One or more views can be grouped together into a ViewGroup. A ViewGroup (which is itself a special type of view) provides the layout in which you can order the appearance and sequence of views. Examples of ViewGroups include LinearLayout and FrameLayout. A ViewGroup derives from the base class **android.view.ViewGroup**.

Android supports the following ViewGroups:

- ➤➤LinearLayout
- ➤➤AbsoluteLayout
- ➤➤TableLayout
- ➤➤RelativeLayout
- ➤➤FrameLayout

LineayourLayout - The LinearLayout arranges views in a single column or a single row. Child views can be arranged either vertically or horizontally.eg: consider the following elements typically contained in the main.xml file:

```
<?xml version=”1.0” encoding=”utf-8”?>
<LinearLayout
xmlns:android=”http://schemas.android.com/apk/res/android”
android:orientation=”vertical”
android:layout_width=”fill_parent”
android:layout_height=”fill_parent”>
<TextView
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”@string/hello”/>
</LinearLayout>
```

In the main.xml file, observe that the root element is <LinearLayout> and it has a <TextView> element contained within it. The <LinearLayout> element controls the order in which the views contained within it appear. Each View and ViewGroup has a set of common attributes, some of which are described below

- layout_width - Specifies the width of the View or ViewGroup
- layout_height - Specifies the height of the View or ViewGroup
- layout_marginTop - Specifies extra space on the top side of the View or ViewGroup
- layout_marginBottom - Specifies extra space on the bottom side of the View or ViewGroup
- layout_marginLeft - Specifies extra space on the left side of the View or ViewGroup
- layout_marginRight- Specifies extra space on the right side of the View or ViewGroup
- layout_gravity - Specifies how child Views are positioned(left,right etc)

The width of the <TextView> element fills the entire width of its parent (which is the screen in this case) using the fill_parent constant. Its height is indicated by the wrap_content constant, which means that its height is the height of its content. If you don't want to have the <TextView> view occupy the entire row, you can set its layout_width attribute to wrap_content. We can also set the width of both the TextView and Button views to an absolute value.(eg:160dp or 150dp). The default orientation layout is horizontal.

AbsoluteLayout

The AbsoluteLayout enables you to specify the exact location of its children. Consider the following:

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
android:layout_width="fill_parent"
android:layout_height="fill_parent"
xmlns:android=http://schemas.android.com/apk/res/android>
<Button android:layout_width="188dp"
android:layout_height="wrap_content"
android:text="Button"
android:layout_x="126px"
android:layout_y="361px"/>
</AbsoluteLayout>
```

TableLayout

The TableLayout groups views into rows and columns. You use the <TableRow> element to designate a row in the table. Each row can contain one or more views. Each view you place within a row forms a cell. The width of each column is determined by the largest width of each cell in that column.Consider the content of main.xml shown here:

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_height="fill_parent"
android:layout_width="fill_parent">
<TableRow>
<TextView
android:text="User Name:"
android:width ="120px"/>
<EditText
```

```
android:id=”@+id/txtUserName”
android:width=”200px” />
</TableRow>
<TableRow>
<TextView
android:text=”Password:”/>
<EditText
android:id=”@+id/txtPassword”
android:password=”true”/>
</TableRow>
<TableRow>
<Button
android:id=”@+id/buttonSignIn”
android:text=”Log In” />
</TableRow>
</TableLayout>
```

RelativeLayout

The RelativeLayout enables you to specify how child views are positioned relative to each other.Consider the following main.xml file:

```
<?xml version=”1.0” encoding=”utf-8”?>
<RelativeLayout
android:id=”@+id/RLayout”
android:layout_width=”fill_parent”
android:layout_height=”fill_parent”
xmlns:android=”http://schemas.android.com/apk/res/android”>
<TextView
android:id=”@+id/lblComments”
android:layout_width=”wrap_content”
android:layout_height=”wrap_content”
android:text=”Comments”
android:layout_alignParentTop=”true”
android:layout_alignParentLeft=”true”/>
<EditText
android:id=”@+id/txtComments”
android:layout_width=”fill_parent”
android:layout_height=”170px”
android:textSize=”18sp”
android:layout_alignLeft=”@+id/lblComments”
android:layout_below=”@+id/lblComments”
android:layout_centerHorizontal=”true”/>
```

```
<Button
android:id="@+id/btnSave"
android:layout_width="125px"
android:layout_height="wrap_content"
android:text="Save"
android:layout_below="@+id/txtComments"
android:layout_alignRight="@+id/txtComments"/>
</RelativeLayout>
```

Notice that each view embedded within the RelativeLayout has attributes that enable it to align with another view. These attributes are layout_alignParentTop, layout_alignParentLeft, layout_alignLeft, layout_alignRight, layout_below and layout_centerHorizontal.

Framelayout

The FrameLayout is a placeholder on screen that you can use to display a single view. Views that you add to a FrameLayout are always anchored to the top left of the layout.In eg here, you have a FrameLayout within a RelativeLayout. Within the FrameLayout, you embed anImageView. If you add another view (such as a Button view) within the FrameLayout, the view will overlap the previous view (seebelow Figure):

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
android:id="@+id/RLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/res/android">
<TextView
android:id="@+id/lblComments"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="This is my lovely dog, Ookii"
android:layout_alignParentTop="true"
android:layout_alignParentLeft="true"/>
<FrameLayout
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/lblComments"
android:layout_below="@+id/lblComments"
android:layout_centerHorizontal="true">
<ImageView
android:src = "@drawable/ookii"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
<Button
```

android:layout_width="124dp"
android:layout_height="wrap_content"
android:text="Print Picture"/>
</FrameLayout>
</RelativeLayout>

VIEWS

A view is a widget that has an appearance on screen. Examples of views are buttons, labels, and text boxes. A view derives from the base class **android.view.View**.The different types of Views are:

- **Basic views** — Commonly used views such as the TextView, EditText, and Button views
- **Picker views** — Views that enable users to select from a list, such as the TimePicker and DatePicker views
- **List views** — Views that display a long list of items, such as the ListView and the SpinnerView views

BASIC VIEWS

The basic views that you can use to design the UI of your Android applications:

- TextView
- EditText
- Button
- ImageButton
- CheckBox
- ToggleButton
- RadioButton
- RadioGroup

These basic views enable you to display text information, as well as perform some basic selection.

- **Textview**

  When you create a new Android project, Eclipse always creates the main.xml fi le (located in the res/ layout folder), which contains a <TextView> element:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/hello"/>
</LinearLayout>
```

The TextView view is used to display text to the user. This is the most basic view and one that you will frequently use when you develop Android applications.

- **EditText**

  A subclass of the TextView view, except that it allows users to edit its text content

  Eg:

```
<EditText android:id="@+id/txtName"
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
```

- **Button**

  Represents a push-button widget used to select an action

  Eg:

```
<Button android:id="@+id/btnOpen"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Open" />
```

- **ImageButton**

  Similar to the Button view, except that it also displays an imageThe ImageButton displays a button with an image. The image is set through the src attribute.

  Eg:

```
<ImageButton android:id="@+id/btnImg1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/icon" />
```

- **CheckBox**

  A special type of button that has two states: checked or unchecked

  Eg:

```
<CheckBox android:id="@+id/ChkAutosave"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
```

android:text=”Auto Save” />

- **RadioGroup and RadioButton**

  The RadioButton has two states: either checked or unchecked.Once a RadioButton is checked, it cannot be unchecked. A RadioGroup is used to group togetherone or more RadioButton views, thereby allowing only one RadioButton to be checked withinthe RadioGroup.When a RadioButton in a RadioGroup is selected, all other RadioButton s are automatically unselected: Note that the RadioButtons are listed vertically, one on top of another. If you want to list them hori-zontally, you need to change the orientation attribute to horizontal .

Eg:

```
<RadioGroup android:id=”@+id/rdbGp1”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:orientation=”vertical” >
<RadioButton android:id=”@+id/rdb1”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”Option 1” />
<RadioButton android:id=”@+id/rdb2”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”Option 2” />
</RadioGroup>
```

- **ToggleButton**

  Displays checked/unchecked states using a light indicatorThe ToogleButton displays a rectangular button that users can toggle on and off by clicking it:

  Eg:

```
<ToggleButton android:id=”@+id/toggle1”
android:layout_width=”wrap_content”
android:layout_height=”wrap_content” />
```

GridView

GridView is a view group that display items in two dimensional scrolling grid (rows and columns).

Adapter Is Used To Fill Data In Gridview: To fill the data in a GridView we simply use adapter and grid items are automatically inserted to a GridView using an Adapter which pulls the content from a source such as an arraylist, array or database.

Basic GridView code in XML:

```
<GridView
android:id="@+id/simpleGridView"
```

```
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:numColumns="3”
android:horizontalSpacing="50dp"  <!--50dp horizontal space between grid items-->
android:verticalSpacing="50dp"/>  <!--50dp vertical space set between grid items-->
```

ImageView

ImageView class is used to display an image file in application.

An ImageView code in XML:

Make sure to save lion image in drawable folder

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/lion" />
```

Handling View Events

To handle the events fired by each view, you first have to programmatically locate the view that you created during the onCreate() event. You do so using the Activity.findViewById() method, supplying it with the ID of the view:

- **Button**

  The setOnClickListener() method registers a callback to be invoked later when the view is clicked:

```
Button btnOpen = (Button) findViewById(R.id.btnOpen);
btnOpen.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
                DisplayToast(“You have clicked the Open button”);}});
```

  The onClick() method is called when the view is clicked.

- **CheckBox**

  For the CheckBox, to determine its state you have to typecast the argument of the onClick() method to a CheckBox and then check its isChecked() method to see if it is checked:

```
CheckBox checkBox = (CheckBox) findViewById(R.id.chkAutosave);
checkBox.setOnClickListener(new View.OnClickListener()
{
public void onClick(View v) {
        if (((CheckBox)v).isChecked())
                DisplayToast(“CheckBox is checked”);
        else
                DisplayToast(“CheckBox is unchecked”);
        }
```

```
});
```

- **RadioButton**

For RadioButton, you need to use the setOnCheckedChangeListener() method on the RadioGroup to register a callback to be invoked when the checked RadioButton changes in this group:

```
RadioGroup radioGroup = (RadioGroup) findViewById(R.id.rdbGp1);
radioGroup.setOnCheckedChangeListener(new OnCheckedChangeListener()
{
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        RadioButton rb1 = (RadioButton) findViewById(R.id.rdb1);
        if (rb1.isChecked()) {
            DisplayToast("Option 1 checked!");}
            else {
            DisplayToast("Option 2 checked!");}  }});
```

When a RadioButton is selected, the onCheckedChanged() method is fired. Within it, you locate individual RadioButtons and then call their isChecked() method to determine which RadioButton is selected. Alternatively, the onCheckedChanged() method contains a second argument that contains a unique identifier of the RadioButton selected.

- **ToggleButton**

```
ToggleButton tb = (ToggleButton) findViewById(R.id.togBtn);
tb.setOnClickListener(new View.OnClickListener()
{
public void onClick(View v) {
if (((ToggleButton)v).isChecked())
    DisplayToast("ToggleButton is ON");
else
    DisplayToast("ToggleButton is OFF");}});
```

PICKER VIEWS

Android provides controls for the user to pick a time or pick a date as ready-to-use dialogs. Each picker provides controls for selecting each part of the time (hour, minute, AM/PM) or date (month, day, year). Android supports this functionality through the TimePicker and DatePicker views. The following sections show how to make use of these views in your activity.

- **TimePicker View**

The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM
mode.

Eg:

```
<TimePicker android:id="@+id/timePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

Methods used In java program

timePicker.setIs24HourView(true); - setting time format
timePicker.getCurrentHour() - Get Hour// or timePicker.getHour();
timePicker.getCurrentMinute() - Get Minute // or timePicker.getMinute()

- **DatePicker view**

Using the DatePicker , you can enable users to select a particular date on the activity.

Eg:

```
<DatePicker android:id="@+id/datePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

Methods used In java program

datePicker.getMonth()
datePicker.getDayOfMonth()
datePicker.getYear()

LIST VIEWS

List views are views that enable you to display a long list of items. In Android, there are two types of list views: ListView and SpinnerView. Both are useful for displaying long lists of items.

- **ListView View**

The ListView displays a list of items in a vertically scrolling list. You can allow multiple items in the ListView to be selected.For that first we have to declare variable **array** as String array with values.

```
public class MainActivity extends ListActivity {
    ListView lstView = getListView();
    //lstView.setChoiceMode(0); //CHOICE_MODE_NONE
    //lstView.setChoiceMode(1); //CHOICE_MODE_SINGLE
    lstView.setChoiceMode(2); //CHOICE_MODE_MULTIPLE
    lstView.setTextFilterEnabled(true);
setListAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, array));
```

The onListItemClick() method is fired whenever an item in the ListView has been clicked:

```
public void onListItemClick(    ListView parent, View v, int position, long id){
Toast.makeText(this,"You have selected " + array[position],
Toast.LENGTH_SHORT).show();
}
```

The ArrayAdapter object manages the array of strings that will be displayed by the ListView. In the preceding example, you set the ListView to display in the simple_list_item_1 mode:

setListAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, presidents));

The onListItemClick() method is fired whenever an item in the ListView has been clicked.Multiple selection can be allowed susing setChoiceMode() method.

- **SpinnerView**

The ListView displays a long list of items in an activity, but sometimes you may want your user interface to display other views, and hence you do not have the additional space for a full-screen viewlike the ListView . In such cases, you should use the SpinnerView . The SpinnerView displays one item at a time from a list and enables users to choose among them.

```
<Spinner
android:id=”@+id/spinner1”
android:layout_width=”wrap_content”
android:layout_height=”wrap_content”
android:drawSelectorOnTop=”true” />
```

Add the following lines in bold to the strings.xml file located in the res/values folder:

```
<string-array name=”student_array”>
<item>Arun</item>
<item>John </item>
<item> Johnson</item>
<item>Manu</item>
</string-array>
```

Add the following statements in bold to the MainActivity.java file

```
String[] st =getResources().getStringArray(R.array.student_array);
Spinner s1 = (Spinner) findViewById(R.id.spinner1);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, st);
s1.setAdapter(adapter);
s1.setOnItemSelectedListener(new OnItemSelectedListener()
{@Override
public void onItemSelected(AdapterView<?> arg0, View arg1,int arg2, long
arg3)
{
        int index = arg0.getSelectedItemPosition();
        Toast.makeText(getBaseContext(),“You have selected item : “ +
        presidents[index],
        Toast.LENGTH_SHORT).show();
}
@Override
public void onNothingSelected(AdapterView<?> arg0) {}
});
```

Using Menus with Views

Menus are useful for displaying additional options that are not directly visible on the main UI of an application. There are two main types of menus in Android:

- **Options menu** — Displays information related to the current activity. In Android, you activate the options menu by pressing the MENU key.
- **Context menu** — Displays information related to a particular view on an activity. In Android,to activate a context menu you tap and hold on to it.

Before you go ahead and create your options and context menus, you need to create two helper methods. One creates a list of items to show inside a menu, while the other handles the event that is fired when the user selects an item inside the menu.Inside MainActivity and after onCreate() define as follows:

Options Menu

Below eg: display the options menu when the user presses the MENU button on the Android device. Inside MainActivity and after onCreate() add following.

```
public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        CreateMenu(menu);
        return true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
        return MenuChoice(item);
}
private void CreateMenu(Menu menu)
{       MenuItem mnu1 = menu.add(0, 0, 0, “Item 1”);
        {
                mnu1.setAlphabeticShortcut(‘a’);
                mnu1.setIcon(R.drawable.icon);
        }
        MenuItem mnu2 = menu.add(0, 1, 1, “Item 2”);
        {
                mnu2.setAlphabeticShortcut(‘b’);
                mnu2.setIcon(R.drawable.icon);
        }
        menu.add(0, 3, 3, “Item 3”);
        menu.add(0, 3, 3, “Item 4”);
}
private boolean MenuChoice(MenuItem item)
{
```

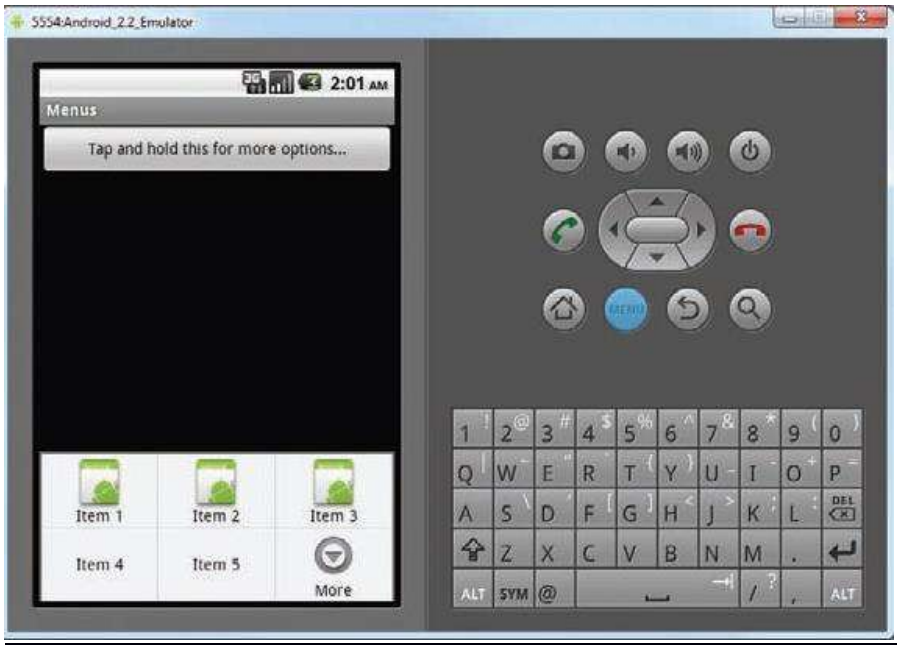
```
switch (item.getItemId()) {
case 0:
        Toast.makeText(this, "You clicked on Item 1",
        Toast.LENGTH_LONG).show();
        return true;
case 1:    . . . . . .
return false;
}
```

The preceding example creates two methods: CreateMenu() and MenuChoice(). The CreateMenu() method takes a Menu argument and adds a series of menu items to it. To add a menu item to the menu, you create an instance of the MenuItem class and use the add() methodof the Menu object. The four arguments of the add() method are as follows:

- groupId — The group identifier that the menu item should be part of. Use 0 if an item is not in a group.
- itemId — Unique item ID
- order — The order in which the item should be displayed
- title — The text to display for the menu item

The setIcon() method sets an image to be displayed on the menu item. The MenuChoice() method takes a MenuItem argument and checks its ID to determine the menu item that is clicked. It then displays a Toast message to let the user know which menu item was clicked.

To display the options menu for your activity, you need to override two methods in your activity:onCreateOptionsMenu() and onOptionsItemSelected(). The onCreateOptionsMenu() method is called when the MENU button is pressed. In this event, you call the CreateMenu() helper method to display the options menu. When a menu item is selected, the onOptionsItemSelected() method is called. In this case, you call the MenuChoice() method to display the menu item selected (and do whatever you want to do).



Context Menu

The previous section showed how the options menu is displayed when the user presses the MENU button. Besides the options menu, you can also display a context menu. A context menu is usually associated with a view on an activity, and it is displayed when the user long clicks an item. For example, if the user taps on a Button view and hold it for a few seconds, a context menu can be displayed. If you want to associate a context menu with a view on an activity, you need to call the setOnCreateContextMenuListener() method of that particular view. Inside MainActivity and inside onCreate() add following.

```
Button btn = (Button) findViewById(R.id.btn1);
btn.setOnCreateContextMenuListener(this);
```

After onCreate() add following:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        CreateMenu(menu);
        return true;}
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    return MenuChoice(item);}
@Override
public void onCreateContextMenu(ContextMenu menu, View view,
ContextMenuInfo menuInfo)
    {super.onCreateContextMenu(menu, view, menuInfo);
    CreateMenu(menu);
}
        @Override
public boolean onContextItemSelected(MenuItem item)
    {return MenuChoice(item);}
private void CreateMenu(Menu menu){//...}
private boolean MenuChoice(MenuItem item)
{//...}}
```

Below figure shows the context menu that is displayed when you long-click the Button view.

In the preceding example, you call the setOnCreateContextMenuListener() method of the Button view to associate it with a context menu. When the user long-clicks the Button view, the onCreateContextMenu() method is called. In this method, you call the CreateMenu() method to display the context menu. Similarly, when an item inside the context menu is selected, the onContextItemSelected() method is called, where you call the MenuChoice() method to display a message to the user.

DATA STORAGE IN ANDROID

For Android, there are primarily three basic ways of persisting data:

- A lightweight mechanism known as *shared preferences* to save small chunks of data
- Traditional file systems
- A relational database management system through the support of SQLite databases

- **SharedPreferences**

SharedPreferences object to help you save simple application data. For example, your application may have an option to allow users to specify the font size of the text displayed in your application. The application can set the size appropriately and can be used at the usage of next time.

- **Traditional file systems –** We can save the data to a file, but you have to perform some file management routines, such as writing the data to the file, indicating how many characters to read from it, and so on.

Disadvantage: - If we have several pieces of information to save, such as text size, font name, preferred background color, and so on, then the task of writing to a file becomes more diificult.

- **INTERNAL STORAGE**

The first way to save files in your Android application is to write to the device's internal storage .

Steps to write:

1. To save text into a file, we use the FileOutputStream class.

2. The openFileOutput() method opens a named file for writing, with the mode specified.
eg.

FileOutputStream fOut =openFileOutput("textfile.txt",MODE_WORLD_READABLE);

- ❖ MODE_PRIVATE – file can only be accessed by the application that created it
- ❖ MODE_APPEND - for appending to an existing file
- ❖ MODE_WORLD_WRITEABLE - all other applications have write access to the file

3. To convert a character stream into a byte stream, use an instance of the OutputStreamWriter class. ie. **OutputStreamWriter osw = new OutputStreamWriter(fOut);**

4. Use its write() method to write the string to the file.

5. To ensure that all the bytes are written to the file, use the flush() method.

6. Finally, use the close() method to close the file:

```
osw.write(str);
osw.flush();
osw.close();
```

Steps to read:

1. To read the content of a file from the device'sinternal storage in android app, use the FileInputStream class, together with the InputStreamReader class:

```
FileInputStream fIn = openFileInput("textfile.txt");
InputStreamReader isr = new InputStreamReader(fIn);
```

2. The read() method of the InputStreamReader object reads the number of characters read and returns -1, if the end of the file is reached.

```
char[] inputBuffer = new char[READ_BLOCK_SIZE];
String s = "";
int charRead;
while ((charRead = isr.read(inputBuffer))>0)
{String readString =  String.copyValueOf(inputBuffer, 0,charRead);
s += readString;
inputBuffer = new char[READ_BLOCK_SIZE];
}
```

● EXTERNAL STORAGE

Sometimes, it would be useful to save them to external storage (such as an SD card) because of its larger capacity, as well as the capability to share the files easily with other users (by removing the SD card and passing it to somebody else).

Steps to write

1. getExternalStorageDirectory() method is used to return the full path to the external storage.

```
File sdCard = Environment.getExternalStorageDirectory();
```

2. Create a new directory

File directory = new File (sdCard.getAbsolutePath() +"/MyFiles");

3. Finally, you save the file into this directory.

File file = new File(directory, "textfile.txt");

4. Continue similar to internal storage writing as follows

```
OutputStreamWriter osw = new  OutputStreamWriter(fOut);
 osw.write(str);
 osw.flush();
osw.close();
```

NB: In order to write to the external storage, we need to add the WRITE_EXTERNAL_STORAGE permission in your AndroidManifest.xml file:.

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE">
</uses-permission>
```

Steps to read

1. To load the file from the external storage, modify the onClick() method for the Load button:

```
 File sdCard = Environment.getExternalStorageDirectory();
File directory = new File (sdCard.getAbsolutePath() +"/MyFiles");
File file = new File(directory, "textfile.txt");
FileInputStream fIn = new FileInputStream(file);
InputStreamReader isr = new InputStreamReader(fIn);
```

2. Remaining code is similar to reading from internal storage. ie.
   The read() method of the InputStreamReader object reads the number of characters read and returns -1, if the end of the file is reached.

```
char[] inputBuffer = new char[READ_BLOCK_SIZE];
String s = "";
int charRead;
while ((charRead = isr.read(inputBuffer))>0)
{String readString =  String.copyValueOf(inputBuffer, 0,charRead);
s += readString;
inputBuffer = new char[READ_BLOCK_SIZE];
}
```

- **SQLITE DATABASE**

An alternative to writing to a text file is to use a database.For saving relational data, using a database is much more efficient. For example, if you want to store the results of all the students in a school, it is much more efficient to use a database to represent them because you can use database querying to retrieve the results of the specific students. Moreover, using databases enables you to enforce data integrity by specifying the relationships between different sets of data.

Data storage using SQLite

Android uses the SQLite database system. The database that you create for an application is only accessible to itself; other applications will not be able to access it.For Android, the SQLite database that you create programmatically in an application is always stored in the /data/data/<package_name>/databases folder.

SQLite is a relational database management system (RDBMS). If most RDBMSs such as MySQL, Oracle, etc. are standalone server processes, then SQLite is embedded because it is provided in the form of a library that is linked in applications.Like other RDBMSs, data is accessed in a SQLite database by using Structured Query Language (SQL).

Android SQLite Java Classes

1. **Cursor**: a class provides access to the results of a database query. Its methods include:

   - close(): release all resources used by cursor and close it.
   - getCount(): returns the number of rows contained within the result set.
   - moveToFirst(): moves to the first row in the result set.
   - moveToLast(): moves to the last row in the result set.
   - moveToNext(): moves to the next row in the result set.
   - move(): moves by a specified offset from the current position in the result set.
   - get<type>() (such as getInt(), getDouble(), so on): returns the value of the specified <type> contained at the specified column index of the row at the current cursor position.

2. **SQLiteDatabase** provides the primary interface between the application code and underlying SQLite database. Its methods include:

   - insert(): inserts a new row into a database table.
   - delete(): deletes rows from a database table
   - query(): performs a specified database query and returns matching results via a Cursor object.
   - execSQL(): executes a single SQL Statement that does not return result data.
   - rawQuery(): executes an SQL query statement and returns matching results in the form of a Cursor object.

3. **SQLiteOpenHelper** is designed to make it easier to create and update databases. Its methods include:

   - onCreate(): called when the database is created for the first time.
   - onUpgrade(): called in the event that the application code contains a more recent database version number reference.
   - onOpen(): called when the database is opened.
   - getWritableDatabase(): opens or creates a database for reading and writing.
   - getReadableDatabase(): creates or opens a database for reading only.
   - close(): closes the database.

4. **ContentValues** allows key/value pairs to be declared consisting of table column identifiers and the values to be stored in each column. Its methods include:

    - put(): adds a value to the set.

Steps of creating a database application

1. Database creation:

Create a database with name "MyDb" with version number "1". Also Create a table "user" with field (id, username, email and password). Steps are:

a) Create a new class (DataBase) as sub set of " SQLiteOpenHelper" class
b) Set version number (1)
c) Call the context of this class (DataBase) to create database (MyDb)
d) In onCreate method, create table making quey, and call execSQL to create the table

DataBase.java

```
public class DataBase extends SQLiteOpenHelper {
  private static final int DATABASE_VERSION = 1;
  private static final String DATABASE_NAME = "MyDb";
  public DataBase(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
  }
        @Override
        public void onCreate(SQLiteDatabase db) {
        String CREATE_TBL = "CREATE TABLE user(id INTEGER PRIMARY KEY
        AUTOINCREMENT ,username TEXT, password TEXT ,email TEXT )";
        db.execSQL(CREATE_TBL);
  }
```

2. Insert data into SQLite database

For inserting to database steps are:

1. Declare database object
   SQLiteDatabase db;
2. Declare object of DataBase class and open the database.
   DataBase objdatabase = new DataBase(this);
   db = objdatabase.getWritableDatabase();
3. Load the values from activity into variables as required data type
   eg. userName = edUserName.getText().toString();
4. Create object of Contentvalues class
   ContentValues values = new ContentValues();

5. Use “put method” to map values to fields

```
values.put("username", _userName);
values.put("email", _userEmail);
values.put("password", _userPword);
```

6. use insert method to insert a row of data

```
long id = db.insert("user", null, values);
```

value of id is the last inserted record’s id.

3. Read data from SQLite database

For reading data steps are:

1. Declare object of SQLiteDatabase class and open the database.

```
DataBaseStudent objdatabase = new DataBaseStudent(this);
SQLiteDatabase db = this.getReadableDatabase();
```

2. Create a **cursor** , run the query to read data, and store into cursor

```
Cursor c = db.query(DataBaseStudent.TABLE_STUDENT, null,
null, null, null, null,null);
```

3. Use **getString** method to get values into a variable, from the corresponding field of present row in the cursor

```
String stName =c.getString(c.getColumnIndex(DataBaseStudent.KEY_NAME));
String stRNo =c.getString(c.getColumnIndex(DataBaseStudent.KEY_ROLL_NUMBER));
```

This cursor data can also be used as an array to show in ListView.

CONTENT PROVIDERS

Content providers can help an application manage access to data stored by itself, stored by other apps, and provide a way to share data with other apps. They encapsulate the data, and provide mechanisms for defining data security. Content providers are the standard interface that connects data in one process with code running in another process. Implementing a content provider has many advantages. Most importantly you can configure a content provider to allow other applications to securely access and modify your app data**.**

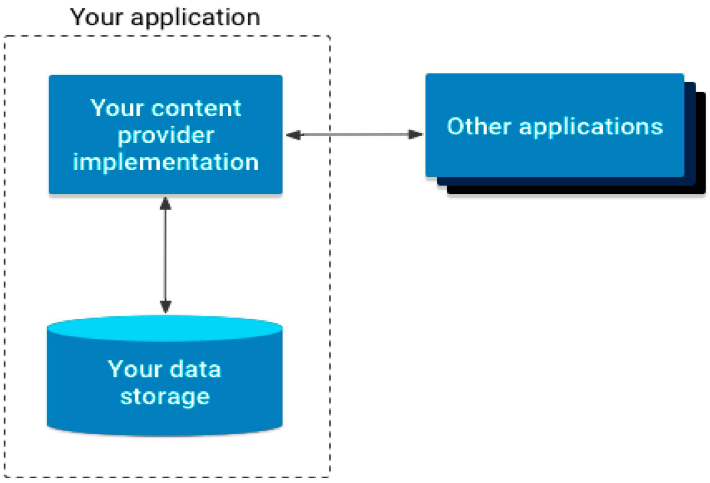
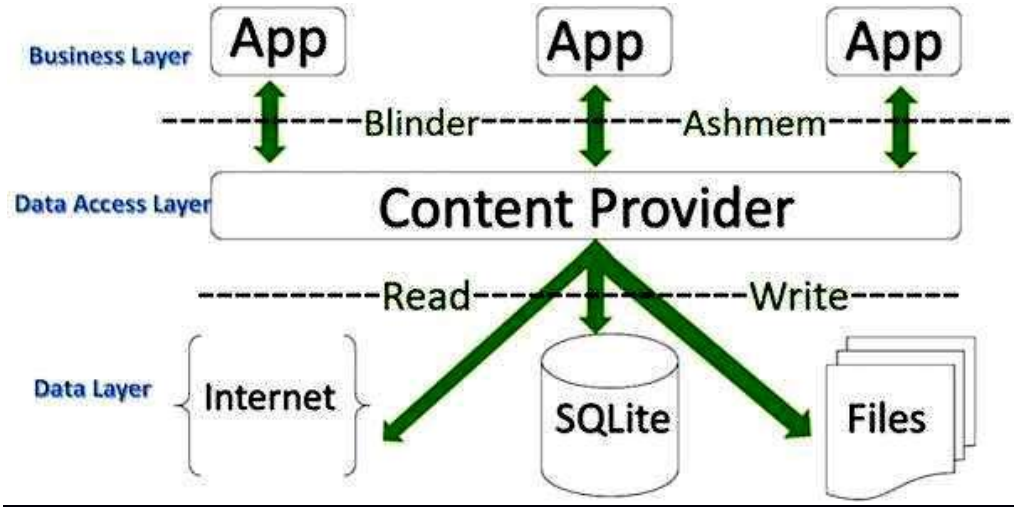


Fig : Overview diagram of how content providers manage access to storage.

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.

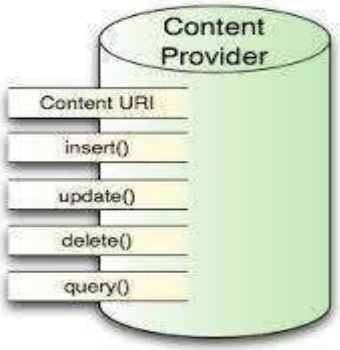


Sometimes it is required to share data across applications. This is where content providers become very useful. Content providers let you centralize content in one place and have many different applications access it as needed. A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods. In most cases this data is stored in an SQlite database.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class My Application extends  ContentProvider {
}
```

Here is the list of methods which you need to override in Content Provider class to have your Content Provider working −



ContentProvider

- **onCreate()** This method is called when the provider is started.
- **query()** This method receives a request from a client. The result is returned as a Cursor object.
- **insert()**This method inserts a new record into the content provider.
- **delete()** This method deletes an existing record from the content provider.
- **update()** This method updates an existing record from the content provider.
- **getType()** This method returns the MIME type of the data at the given URI.

Advantages of content providers

- Content providers offer granular control over the permissions for accessing data.

You can choose to restrict access to a content provider from solely within your application, grant blanket permission to access data from other applications, or configure different permissions for reading and writing data.

- You can use a content provider to abstract away the details for accessing different data sources in your application.

For example, your application might store structured records in a SQLite database, as well as video and audio files. You can use a content provider to access all of this data, if you implement this development pattern in your application.

Dusadvantages of content providers

- content providers can only use SQLite as a storage mechanism.
- This may change in the future, though, because there are no hard-coded SQLite constraints.
- If using object-oriented databases, a proxy could be used between the content provider and the database interface. Or a temporary SQLite database could be a solution
- Resources are crucial on a mobile environment and you might be better off using SQLite.
- More complicated queries, with joins and distinct, are not possible with content provider.

SMS SERVICE IN ANDROID

SMS messaging is one of the main *killer applications* on a mobile phone today — for some users as necessary as the phone itself. Any mobile phone you buy today should have at least SMS messaging capabilities, and nearly all users of any age know how to send and receive.

Sending SMS Messages Programmatically

Android uses a permission-based policy where all the permissions needed by an application need to be specified in the AndroidManifest.xml file. By doing so, when the application is installed it will be clear to the user what specific access permissions are required by the application. For example, as sending SMS messages will potentially incur additional cost on the user's end, indicating the SMS permissions in the AndroidManifest.xml file will let the user decide whether to allow the application to install or not.

In the AndroidManifest.xml file, we can add following two permissions – SEND_SMS and RECEIVE_SMS whichever is needed.:

```
<uses-permission android:name="android.permission.SEND_SMS">
  </uses-permission>
  <uses-permission android:name="android.permission.RECEIVE_SMS">
  </uses-permission>
```

In the main.xml file located in the res/layout folder, add code so that the user can enter a phone number as well as a message to send using edittexts and button.

Next, in the SMS activity, using the Button view on click, we will check to see that the phone number of the recipient and the message is entered before we send the message using the sendSMS() function

```
btnSendSMS.setOnClickListener(new View.OnClickListener()
    {
      public void onClick(View v)
      {
        String phoneNo = txtPhoneNo.getText().toString();
        String message = txtMessage.getText().toString();
        if (phoneNo.length()>0 && message.length()>0)
          sendSMS(phoneNo, message);
        else
          Toast.makeText(getBaseContext(),"Please enter both phone number and message.",
            Toast.LENGTH_SHORT).show();
      }
    });
```

Where sendSMS is defined as below:

```
    private void sendSMS(String phoneNumber, String message)
    {
    SmsManager sms = SmsManager.getDefault();
    sms.sendTextMessage(phoneNumber, null, message, null, null);
    }
```

Following are the five arguments to the sendTextMessage() method:

➤➤destinationAddress — Phone number of the recipient

➤➤scAddress — Service center address; use null for default SMSC

➤➤text — Content of the SMS message

➤➤sentIntent — Pending intent to invoke when the message is sent

➤➤deliveryIntent — Pending intent to invoke when the message has been delivered

NOTIFICATIONS

A **notification** is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.Notifications are implemented using NotificationManager to display a persistent message at the top of the device, commonly known as the *status bar* (sometimes also referred to as the *notification bar*).

Step 1 - Create Notification Builder - As a first step is to create a notification builder using NotificationCompat.Builder.build(). You will use Notification Builder to set various Notification properties like its small and large icons, title, priority etc.

NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)

Step 2 - Setting Notification Properties - Once you have **Builder** object, you can set its Notification properties using Builder object as per your requirement. But this is mandatory to set at least following –

- A small icon, set by **setSmallIcon()**
- A title, set by **setContentTitle()**
- Detail text, set by **setContentText()**

Step 3 - Attach Actions

The action is defined by a **PendingIntent** containing an **Intent** that starts an Activity in your application.

Step 4 - Issue the notification

Finally, you pass the Notification object to the system by calling NotificationManager.notify() to send your notification.

Step 5 : Cancelling Android Notification

We can also call the cancel() for a specific notification ID on the NotificationManager. The cancelAll() method call removes all of the notifications you previously issued

DIFFERENCIATE BETWEEN TOAST AND NOTIFICATION/STATUS BAR

Toast:

- We can use the Toast class to display messages to the user.
- While the Toast class is a eeasy way to show users alerts, it is not persistent.
- It flashes on the screen for a few seconds and then disappears.

- If it contains important information, users may easily miss it if they are not looking at the screen.

Notification/status bar

- For messages that are important, we should use a more persistent method.
- In this case, we should use the NotificationManager to display a persistent message at the top of the device, commonlyknown as the status bar (or the notification bar).

STEPS TO PUBLISH APPLICATION IN GOOGLE PLAY STORE

Here are quick steps to prepare your mobile application for publishing in the Google Play Store.

1. Create an account

Firstly, To publish your app on google play store, you need to have account on google. You may have already personal email account with them but for your app it's better to separate one for manage your app(s) easily. You would have to pay a registration fees of 25 USD, using Google payments, While registering your publisher account. After that, a verification mail would be sent to you and then you sign in to your developer console, where all action would take place.

2. Familiarise yourself with Developer Console

Developer console is starting point and main dashboard for publishing tools and operations. Before you go ahead, familiarise yourself with list of merchant countries and developer countries. You need to review list of merchant countries if you want to sell apps or in app purchases or have subscriptions and list of developer countries will tell you about locations where distribution to Google play users is supported.

3. Fill in the necessary account details

After this, you have to provide your complete account details like you have to provide your developer name, the name which would be displayed in Google Play Store. You will have to wait for anything between just a little and 48 hours, after filling the details for Google play developer registration to be processed.

4. Link your merchant account

If you have a paid app or subscription or in app purchases, then you have to inter link your google payments merchant account and developer profile. It can be used for tax and financial identification as well as monthly payout from sale.

5. Upload your app

When you are sign in your Google Play developer console, click on “Add new application” from “All Applications tab”. After this select correct “Default Language” from drop down menu and then type

"title" of your app which would be appear on Google Play store. Then, select "Upload APK" button to view on new page which would be your homepage for app. Time taken to upload file depend on your app size. App will be in drafts until or unless you publish it.

6. Alpha and beta testing of app

It is essential to test it with sample of end users to get feedback of app before launch your app even Google play take care of this as well. In section of your app "APK" of developer console, you will find option related to "Alpha Testing" and "Beta Testing". After you uploaded your app's "APK" file you can use these options for receive URL that can be shared with testers. By using this link, Testers can download app beta or alpha version. Your testers can not provide reviews and feedback on app page. Now you use this feedback to make relevant changes and optimise your app before publishing in app store.

7. Provide details for store listing

After uploading "APK" file go to "Store listing" tab. Over there you need to add details of app like "Full description" and "Short description". Along with this add categorisation, contact details, link of promo video if you have, add screenshots and other necessary details to app. After complete mandatory fields click on "Save" button. You can update your store listing anytime.

8. Add pricing and distribution details

Now move on next tab, which is "Pricing and Distribution" and select it is "Paid" or "Free" app. You also select distribution countries and check boxes stating your app complies with content guidelines. If your app is an game one, then you can put in limelight using "Google Play for Game" option. Now save changes and move on next step.

9. Publishing the application

When all three tabs "Pricing and Distribution", "Store Listing" and "APK" have been filled then appear a green check mark next to them you are all ready to publish app in Google Play. After then click on "Publish this app" button under "Ready to Publish" drop down menu at top right corner of Developer console. A confirmation bar would show up stating your app would appear shortly in Google Play Store.

10. Device Filtering option

There are series of extra option that might not seem to be important to publish the app but they can prevent app from negative feedback. There is also option to manually filter non compatible devices or problematic so make the most use of it to filter out any negativities and stay on the top.