Designing User Interfaces with Layouts

Learning Objectives

In this module you learn about

- Layout and its use
- How to create layout using XML
- How to create layout programmatically
- Different types of built-in layouts
- Use of List View

Introduction

One special type of control found within the android.widget package is called a layout. A layout control is still a View object, but it doesn't actually draw anything specific on the screen. Instead, it is a parent container for organizing other controls (children). Layout controls determine how and where on the screen child controls are drawn. Each type of layout control draws its children using particular rules. For instance, the LinearLayout control draws its child controls in a single horizontal row or a single vertical column. Similarly, a TableLayout control displays each child control in tabular format (in cells within specific rows and columns).

Application user interfaces can be simple or complex, involving many different screens or only a few. Layouts and user interface controls can be defined as application resources or created programmatically at runtime.

Creating Layouts Using XML Resources

Android provides a simple way to create layout files in XML as resources provided in the /res/layout project directory. This is the most common and convenient way to build Android user interfaces and is especially useful for defining static screen elements and control properties that you know in advance, and to set default attributes that you can modify programmatically at runtime.

You can configure almost any ViewGroup or View (or View subclass) attribute using the XML layout resource files. This method greatly simplifies the user interface design process, moving much of the static creation and layout of user interface controls, and basic definition of control attributes, to the XML, instead of littering the code. Developers reserve the ability to alter these layouts programmatically as necessary, but they can set all the defaults in the XML template. You will recognize the following as a simple layout file with a LinearLayout and a single TextView control.

```
<?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>
```

This block of XML shows a basic layout with a single TextView. The first line, which you might recognize from most XML files, is required.

Creating only an XML file, though, won’t actually draw anything on the screen. A particular layout is usually associated with a particular Activity. In your default Android project, there is only one activity, which sets the main.xml layout by default. To associate the main.xml layout with the activity, use the method call setContentView() with the identifier of the main.xml layout. The ID of the layout matches the XML filename without the extension. In this case, the preceding example came from main.xml, so the identifier of this layout is simply main:

```
setContentView(R.layout.main);
```

Creating Layouts Programmatically

You can create user interface components such as layouts at runtime programmatically, but for organization and maintainability, it's best that you leave this for the odd case rather than the norm. The main reason is because the creation of layouts programmatically is onerous and difficult to maintain, whereas the XML resource method is visual, more organized, and could be done by a separate designer with no Java skills.

The following example shows how to programmatically have an Activity instantiate a LinearLayout view and place two TextView objects within it. No resources whatsoever are used; actions are done at runtime instead.

```
public void onCreate (Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView text1 = new TextView(this);
        text1.setText("Hi there!");
        TextView text2 = new TextView(this);
        text2.setText("I'm second. I need to wrap.");
        text2.setTextSize((float) 60);
        LinearLayout ll = new LinearLayout(this);
        ll.setOrientation(LinearLayout.VERTICAL);
        ll.addView(text1);
        ll.addView(text2);
        setContentView(ll);
}
```

The onCreate() method is called when the Activity is created. The first thing this method does is some normal Activity housekeeping by calling the constructor for the base class. Next, two TextView controls are instantiated. The Text property of each TextView is set using the setText() method. All TextView attributes, such as TextSize, are set by making method calls on the TextView object. These actions perform the same function that you have in the past by setting the properties Text and TextSize

using the layout resource designer, except these properties are set at runtime instead of defined in the layout files compiled into your application package.

Built in Layouts

We talked a lot about the LinearLayout layout, but there are several other types of layouts. Each layout has a different purpose and order in which it displays its child View controls on the screen. Layouts are derived from android.view.ViewGroup.

The types of layouts built-in to the Android SDK framework include:

- FrameLayout
- LinearLayout
- RelativeLayout
- TableLayout

All layouts, regardless of their type, have basic layout attributes. Layout attributes apply to any child View within that layout. You can set layout attributes at runtime programmatically, but ideally you set them in the XML layout files using the following syntax: android:layout_attribute_name=”value”

There are several layout attributes that all ViewGroup objects share.These include size attributes and margin attributes. You can find basic layout attributes in the ViewGroup.LayoutParams class.The margin attributes enable each child View within a layout to have padding on each side. Find these attributes in the ViewGroup.MarginLayoutParams classes. There are also a number of ViewGroup attributes for handling child View drawing bounds and animation settings.

Frame Layout

FrameLayout is designed to block out an area on the screen to display a single item. Generally, FrameLayout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other. You can, however, add multiple children to a

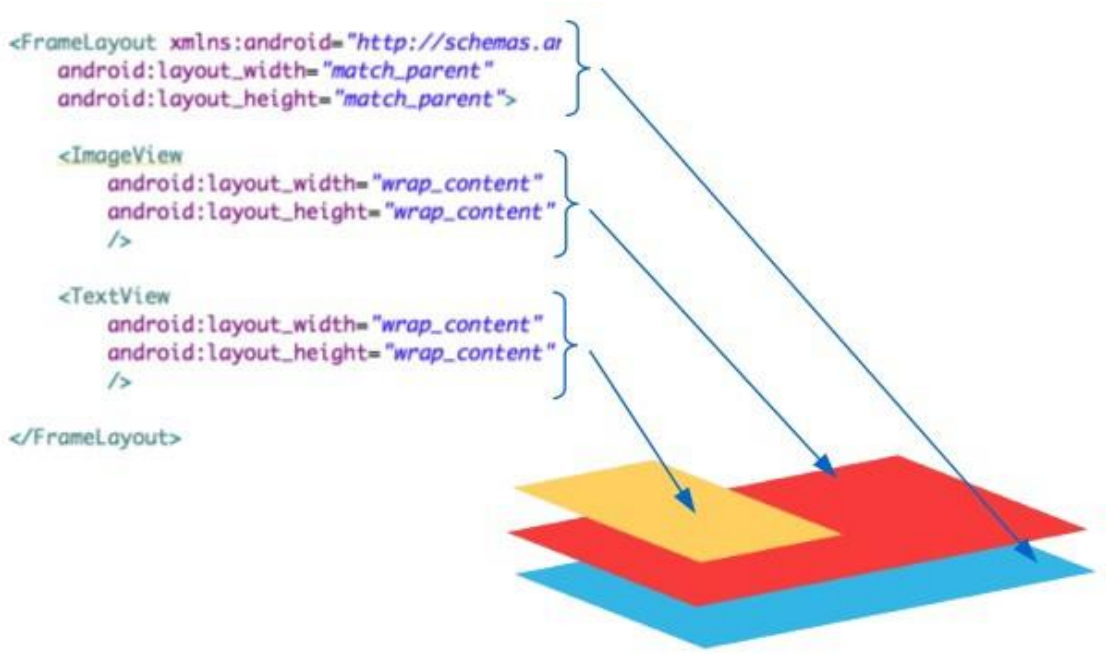
FrameLayout and control their position within the FrameLayout by assigning gravity to each child, using the android:layout_gravity attribute.

Child views are drawn in a stack, with the most recently added child on top. The size of the FrameLayout is the size of its largest child (plus padding), visible or not (if the FrameLayout's parent permits). Views that are View.GONE are used for sizing only if setConsiderGoneChildrenWhenMeasuring() is set to true.

Following Table describes some of the important attributes specific to FrameLayout views.

Attribute Name	Applies To	Description	Value
android: foreground	Parent view	Drawable to draw over the content	Drawable resource.
android: foreground-Gravity	Parent view	Gravity of foreground drawable.	One or more constants separated by "|". The constants available are top, bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, and fill.
android: measureAll-Children	Parent view	Restrict size of layout to all child views or just the child views set to VISIBLE (and not those set to INVISIBLE).	True or false.
android: layout_ gravity	Child view	A gravity constant that describes how to place the child View within the parent.	One or more constants separated by "|". The constants available are top, bottom, left, right, center_vertical, fill_ vertical, center_horizontal, fill_horizontal, center, and fill.

An Example of FlowLayout is shown below



LinearLayout

A LinearLayout view organizes its child View objects in a single row, shown in Figure below, or column, depending on whether its orientation attribute is set to horizontal or vertical. This is a very handy layout method for creating forms.

You can find the layout attributes available for LinearLayout child View objects in android.control.LinearLayout.LayoutParams. Following table describes some of the important attributes specific to LinearLayout views.

Attribute Name	Applies To	Description	Value
android: orientation	Parent view	Layout is a single row (horizontal) or single column (vertical).	Horizontal or Vertical
android: gravity	Parent view	Gravity of child views within layout.	One or more constants separated by "|". The constants available are top, bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, and fill.
android: layout_	Child View	The gravity for a specific child view. Used for	One or more constants separated by "|". The constants

Attribute Name	Applies To	Description	Value
gravity		positioning of views.	available are top, bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, and fill.
android: layout_ weight	Child view	The weight for a specific child view. Used to provide ratio of screen space used within the parent control.	The sum of values across all child views in a parent view must equal 1. For example, one child control might have a value of .3 and another have a value of .7.

An Example of Linear Layout is shown below

RelativeLayout

The RelativeLayout view enables you to specify where the child view controls are in relation to each other. For instance, you can set a child View to be positioned “above” or “below” or “to the left of ” or “to the right of ” another View, referred to by its unique identifier. You can also align child View objects relative to one another or the parent layout edges. Combining RelativeLayout attributes can simplify creating interesting user interfaces without resorting to multiple layout groups to achieve a desired effect. You can find the layout attributes available for RelativeLayout child View objects in

android.control.RelativeLayout.LayoutParams. Following Table describes some of the important attributes specific to RelativeLayout views.

Attribute Name	Applies To	Description	Value
android:gravity	Parent view	Gravity of child views within layout.	One or more constants separated by "|". The constants available are top, bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, and fill.
android: layout_centerInParent	Child view	Centers child view horizontally and vertically within parent view.	True or False
android: layout_centerHorizontal	Child view	Centers child view horizontally within parent view	True or False
android: layout_centerVertical	Child view	Centers child view vertically within parent view.	True or False
android: layout_alignParentTop	Child view	Aligns child view with top edge of parent view	True or False
android: layout_alignParentBottom	Child view	Aligns child view with bottom edge of parent view.	True or False
android: layout_alignParentLeft	Child View	Aligns child view with left edge of parent view.	True or False
android: layout_alignParentRight	Child View	Aligns child view with right edge of parent view.	True or False
android: layout_alignRight	Child View	Aligns child view with right edge of another child view, specified by ID.	A view ID; for example, @id/Button1
android: layout_alignLeft	Child View	Aligns child view with left edge of another	A view ID; for example,

Attribute Name	Applies To	Description	Value
		child view, specified by ID.	@id/Button1
android: layout_alignTop	Child View		A view ID; for example, @id/Button1
android: layout_alignBottom	Child View		A view ID; for example, @id/Button1
android: layout_above	Child View	Positions bottom edge of child view above another child view, specified by ID.	A view ID; for example, @id/Button1
android: layout_below	Child View	Positions top edge of child view below another child view, specified by ID.	A view ID; for example, @id/Button1
android: layout_toLeftOf	Child View	Positions right edge of child view to the left of another child view, specified by ID.	A view ID; for example, @id/Button1
android: layout_toRightOf	Child View	Positions left edge of child view to the right of another child view, specified by ID.	A view ID; for example, @id/Button1

Following figure shows how each of the button controls is relative to each other.

Here’s an example of an XML layout resource with a RelativeLayout and two child View objects, a Button object aligned relative to its parent, and an ImageView aligned and positioned relative to the Button (and the parent):

```
<?xml version=”1.0” encoding=”utf-8”?>
<RelativeLayout xmlns:android= “http://schemas.android.com/apk/res/android”
        android:id=”@+id/RelativeLayout01”
        android:layout_height=”fill_parent”
        android:layout_width=”fill_parent”>
        <Button android:id=”@+id/ButtonCenter”
                android:text=”Center”
                android:layout_width=”wrap_content”
                android:layout_height=”wrap_content”
                android:layout_centerInParent=”true” />
        <ImageView android:id=”@+id/ImageView01”
                android:layout_width=”wrap_content”
                android:layout_height=”wrap_content”
                android:layout_above=”@id/ButtonCenter”
                android:layout_centerHorizontal=”true”
                android:src=”@drawable/arrow” />
</RelativeLayout>
```

TableLayout

A TableLayout view organizes children into rows, as shown in following Figure-68. You add individual View objects within each row of the table using a TableRow layout View (which is basically a horizontally oriented LinearLayout) for each row of the table. Each column of the TableRow can contain one View (or layout with child View objects).You place View items added to a TableRow in columns in the order they are added. You can specify the column number (zero-based) to skip columns as necessary (the bottom row shown in above figure demonstrates this); otherwise, the View object is put in the next column to the right. Columns scale to the size of the largest View of that column. You can also include normal View objects instead of TableRow elements, if you want the View to take up an entire row.

You can find the layout attributes available for TableLayout child View objects in android.control.TableLayout.LayoutParams.You can find the layout attributes available for TableRow child View objects in android.control.TableRow.LayoutParams. Following Table describes some of the important attributes specific to TableLayout View objects.

Attribute Name	Applies To	Description	Value
android: collapseColumns	TableLayout	A comma-delimited list of column indices to collapse (0-based)	String or string resource. For example, “0,1,3,5”
android: shrinkColumns	TableLayout	A comma-delimited list of column indices to shrink (0-based)	String or string resource. Use “*” for all columns. For example, “0,1,3,5”
andriod: stretchColumns	TableLayout	A comma-delimited list of column indices to stretch (0-based)	String or string resource. Use “*” for all columns. For example, “0,1,3,5”

Attribute Name	Applies To	Description	Value
android: layout_column	TableRow child view	Index of column this child view should be displayed in (0-based)	Integer or integer resource. For example, 1
android: layout_span	TableRow child view	Number of columns this child view should span across	Integer or integer resource greater than or equal to 1. For example, 3

Here's an example of an XML layout resource with a TableLayout with two rows (two TableRow child objects).The TableLayout is set to stretch the columns to the size of the screen width. The first TableRow has three columns; each cell has a Button object. The second TableRow puts only one Button view into the second column explicitly:

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/TableLayout01"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:stretchColumns="*">
        <TableRow android:id="@+id/TableRow01">
                <Button android:id="@+id/ButtonLeft"
                        android:text="Left Door" />
                <Button android:id="@+id/ButtonMiddle"
                        android:text="Middle Door" />
                <Button android:id="@+id/ButtonRight"
                         android:text="Right Door" />
        </TableRow>
        <TableRow android:id="@+id/TableRow02">
                <Button android:id="@+id/ButtonBack"
                        android:text="Go Back"
                        android:layout_column="1" />
        </TableRow>
</TableLayout>
```

Using Data-Driven Containers

Some of the View container controls are designed for displaying repetitive View objects in a particular way. Examples of this type of View container control include ListView, GridView, and GalleryView:

- ListView: Contains a vertically scrolling, horizontally filled list of View objects, each of which typically contains a row of data; the user can choose an item to perform some action upon.
- GridView: Contains a grid of View objects, with a specific number of columns; this container is often used with image icons; the user can choose an item to perform some action upon.
- GalleryView: Contains a horizontally scrolling list of View objects, also often used with image icons; the user can select an item to perform some action upon.

These containers are all types of AdapterView controls. An AdapterView control contains a set of child View controls to display data from some data source. An Adapter generates these child View controls from a data source. As this is an important part of all these container controls, we talk about the Adapter objects first.

In this section, you learn how to bind data to View objects using Adapter objects. In the Android SDK, an Adapter reads data from some data source and provides a View object based on some rules, depending on the type of Adapter used. This View is used to populate the child View objects of a particular AdapterView.

The most common Adapter classes are the CursorAdapter and the ArrayAdapter. The CursorAdapter gathers data from a Cursor, whereas the ArrayAdapter gathers data from an array. A CursorAdapter is a good choice to use when using data from a database. The ArrayAdapter is a good choice to use when there is only a single column of data or when the data comes from a resource array.

There are some common elements to know about Adapter objects. When creating an Adapter, you provide a layout identifier.This layout is the template for filling in each

row of data. The template you create contains identifiers for particular controls that the Adapter assigns data to.A simple layout can contain as little as a single TextView control.

When making an Adapter, refer to both the layout resource and the identifier of the TextView control. The Android SDK provides some common layout resources for use in your application.

How to Use the Adapter

An ArrayAdapter binds each element of the array to a single View object within the layout resource. Here is an example of creating an ArrayAdapter:

```
private String[] items = { “Item 1”, “Item 2”, “Item 3” };
ArrayAdapter adapt = new ArrayAdapter<String> (this, R.layout.textview, items);
```

In this example, we have a String array called items. This is the array used by the ArrayAdapter as the source data. We also use a layout resource, which is the View that is repeated for each item in the array. This is defined as follows:

```
<TextView xmlns:android= “http://schemas.android.com/apk/res/android”
        android:layout_width=”fill_parent”
        android:layout_height=”wrap_content”
        android:textSize=”20px” />
```

This layout resource contains only a single TextView. However, you can use a more complex layout with the constructors that also take the resource identifier of a TextView within the layout. Each child View within the AdapterView that uses this Adapter gets one TextView instance with one of the strings from the String array.

If you have an array resource defined, you can also directly set the entries attribute for an AdapterView to the resource identifier of the array to automatically provide the ArrayAdapter.

Binding Data to the AdapterView

Now that you have an Adapter object, you can apply this to one of the AdapterView controls. Any of them works.Although the Gallery technically takes a SpinnerAdapter, the instantiation of SimpleCursorAdapter also returns a SpinnerAdapter. Here is an example of this with a ListView, continuing on from the previous sample code:

```
((ListView)findViewById(R.id.list)).setAdapter(adapter);
```

The call to the setAdapter() method of the AdapterView, a ListView in this case, should come after your call to setContentView().This is all that is required to bind data to your AdapterView. Figure given below shows the same data in a ListView, Gallery, and GridView.

Handling Selection Events

You often use AdapterView controls to present data from which the user should select. All three of the discussed controls—ListView, GridView, and Gallery—enable your application to monitor for click events in the same way.You need to call setOnItemClickListener() on your AdapterView and pass in an implementation of the AdapterView.OnItemClickListener class.

Following is an example implementation of this class:

```
av.setOnItemClickListener(
        new AdapterView.OnItemClickListener() {
        public void onItemClick(
```

```
            AdapterView<?> parent, View view,
            int position, long id) {
                Toast.makeText(Scratch.this, “Clicked _id=”+id,
                    Toast.LENGTH_SHORT).show();
        }
});
```

In the preceding example, av is our AdapterView. The implementation of the onItemClick() method is where all the interesting work happens. The parent parameter is the AdapterView where the item was clicked. This is useful if your screen has more than one AdapterView on it. The View parameter is the specific View within the item that was clicked. The position is the zero-based position within the list of items that the user selects.

Finally, the id parameter is the value of the _id column for the particular item that the user selects. This is useful for querying for further information about that particular row of data that the item represents.

Your application can also listen for long-click events on particular items. Additionally, your application can listen for selected items. Although the parameters are the same, your application receives a call as the highlighted item changes. This can be in response to the user scrolling with the arrow keys and not selecting an item for action.

Let us sum up

In this module you learn about layout and its use, we have seen that the layout can be created using XML as well as programmatically. You have learned about different types of built-in layouts and few data driven controls such as List View, Grid View and Gallery view.

Further Reading

- https://developer.android.com/reference/android/widget/FrameLayout
- https://developer.android.com/reference/android/widget/TableLayout

- https://developer.android.com/reference/android/widget/RelativeLayout
- https://developer.android.com/reference/android/widget/GridLayout
- https://developer.android.com/reference/android/widget/ListView

Activity

- Design Interface using different Layouts for collecting personal information such as Id, Name, Date of Birth, Gender, Address etc.

Acknowledgement: “The content in this module is modifications based on work created and shared by the Android Open-Source Project and used according to terms described in the Creative Commons 2.5 Attribution License.”