UNIT – III

Android User Interface Design

Introducing Android Views and Layouts

Before we go any further,we need to define a few terms.This gives you a better understanding of certain capabilities provided by the Android SDK before they are fully introduced.

Introducing the Android View

The Android SDK has a Java packaged named android.view.This package contains a number of interfaces and classes related to drawing on the screen. However, when we refer to the View object,we actually refer to only one of the classes within this package: the android.view.View class.
The View class is the basic user interface building block within Android. It represents a rectangular portion of the screen.The View class serves as the base class for nearly all the user interface controls and layouts within the Android SDK.

Introducing the Android Control

The Android SDK contains a Java package named android.widget.When we refer to controls,we are typically referring to a class within this package.The Android SDK includes classes to draw most common objects, including ImageView, FrameLayout, EditText, and Button classes.As mentioned previously, all controls are typically derived from the View class.

TextView

- ✓ The TextView control is a child control within other screen elements and controls. As with most of the user interface elements, it is derived from View and is within the android.widget package.
- ✓ Because it is a View, all the standard attributes such as width, height, padding, and visibility can be applied to the object. However, as a text-displaying control, you can apply many other TextView-specific attributes to control behavior and how the text is viewed in a variety of situations.
- ✓ <TextView> is the XML layout file tag used to display text on the screen.You can set the android:text property of the TextView to be either a raw text string in the layout file or a reference to a string resource.
- ✓ Here are examples of both methods you can use to set the android:text attribute of a TextView.The first method sets the text attribute to a raw string; the second methoduses a string resource called sample_text, which must be defined in the strings.xml resourcefile.

```
<TextView
android:id ”@+id/TextView01”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”Some sample text here” />
<TextView
android:id ”@+id/TextView02”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”@string/sample_text” />
```

To display this TextView on the screen, all your Activity needs to do is call the
setContentView() method :- with the layout resource identifier in which you defined the preceding XML shown.
setText() method :- change the text displayed programmatically by calling the on the TextView object.
getText() :- Retrieving the text is done with the method.

Configuring Layout and Sizing

The width of a TextView can be controlled in terms of the ems measurement rather than in pixels.An em is a term used in typography that is defined in terms of the point size of a particular font.
Through the **ems attribute, you can set the width of the TextView**.Additionally, you can use the maxEms and minEms attributes to set the maximum width and minimum width, respectively, of the TextView in terms of ems.
The height of a TextView can be set in terms of lines of text rather than pixels.Again, this is useful for controlling how much text can be viewed regardless of the font size.
The lines attribute sets the number of lines that the TextView can display.You can also use maxLines and minLines to control the maximum height and minimum height, respectively, that the Textview displays.
Here is an example that combines these two types of sizing attributes.This TextView is two lines of text high and 12 ems of text wide.The layout width and height are specified to the size of the TextView and are required attributes in the XML schema:

```
<TextView
android:id ”@+id/TextView04”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:lines ”2”
android:ems ”12”
android:text ”@string/autolink_test” />
```

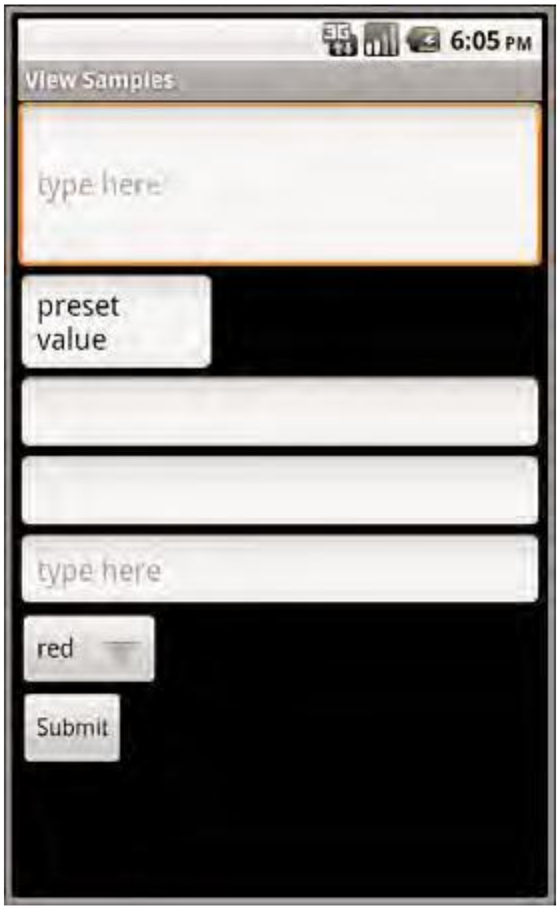
EditText

The Android SDK provides a convenient control called EditText to handle text input from a user.The EditText class is derived from TextView. In fact, most of its functionality is contained within TextView but enabled when created as an EditText.

Define an EditText control in an XML layout file:

```
<EditText
android:id ”@+id/EditText01”
android:layout_height ”wrap_content”
android:hint ”type here”
android:lines ”4”
android:layout_width ”fill_parent” />
```

hint :- attribute puts some text in the edit box that goes away when the user starts entering text.
Lines :- which defines how many lines tall the input box is. If this is not set, the entry field grows as the user enters text.

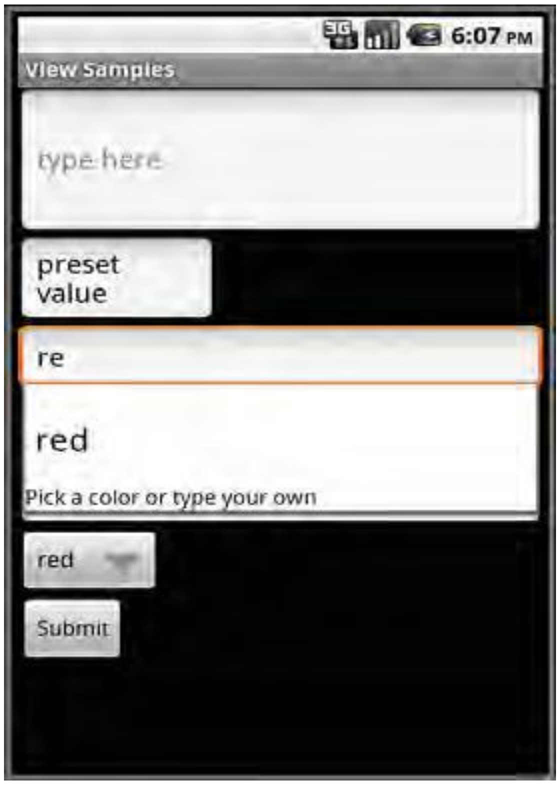


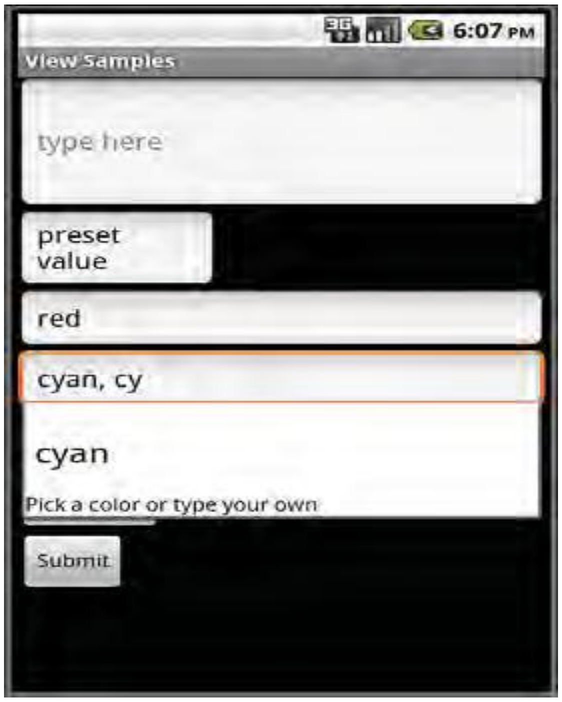
The EditText object is essentially an editable TextView.This means that you can read
text from it in the same way as you did with TextView

getText() :- read text from it
setText() :- set initial text to draw in the text entry area

Helping the User with Auto Completion

1) AutoCompleteTextView :- There are two forms of auto-complete. One is the more standard style of filling in the entire text entry based on what the user types. If the user begins typing a string that matches a word in a developer-provided list, the user can choose to complete the word with just a tap
2) MultiAutoCompleteTextView :- allows the user to enter a list of items, each of which has autocomplete Functionality These items must be separated in some way by providing a Tokenizer. A common Tokenizer implementation is provided for comma-separated lists and is used by specifying the MultiAutoCompleteTextView.CommaTokenizer object.
   This can be helpful for lists of specifying common tags and the like.





Using **AutoCompleteTextView** (left) and **MultiAutoCompleteTextView** (right).

Both of the auto-complete text editors use an adapter to get the list of text that they use to provide completions to the user.This example shows how to provide an AutoCompleteTextView for the user

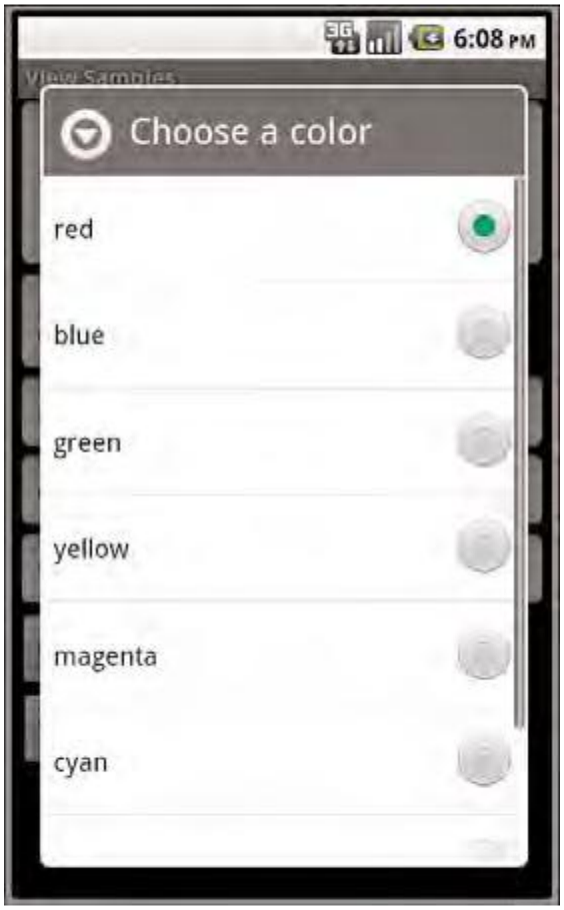
```
final String[] COLORS {
“red”, “green”, “orange”, “blue”, “purple”,
“black”, “yellow”, “cyan”, “magenta” };

ArrayAdapter<String> adapter
new ArrayAdapter<String>(this,
android.R.layout.simple_dropdown_item_1line, COLORS);
AutoCompleteTextView text (AutoCompleteTextView)
findViewById(R.id.AutoCompleteTextView01);
text.setAdapter(adapter);
```

Spinner Controls

Sometimes you want to limit the choices available for users to type. For instance, if users are going to enter the name of a state, you might as well limit them to only the valid states because this is a known set.
Although you could do this by letting them type something and then blocking invalid entries, you can also provide similar functionality with a Spinner control.As with the auto-complete method, the possible choices for a spinner can come from an Adapter.
You can also set the available choices in the layout definition by using the entries attribute with an array resource (specifically a string-array that is referenced as something such as @array/state-list).The Spinner control isn’t actually an EditText,
although it is frequently used in a similar fashion. Here is an example of the XML layout definition for a Spinner control for choosing a color:

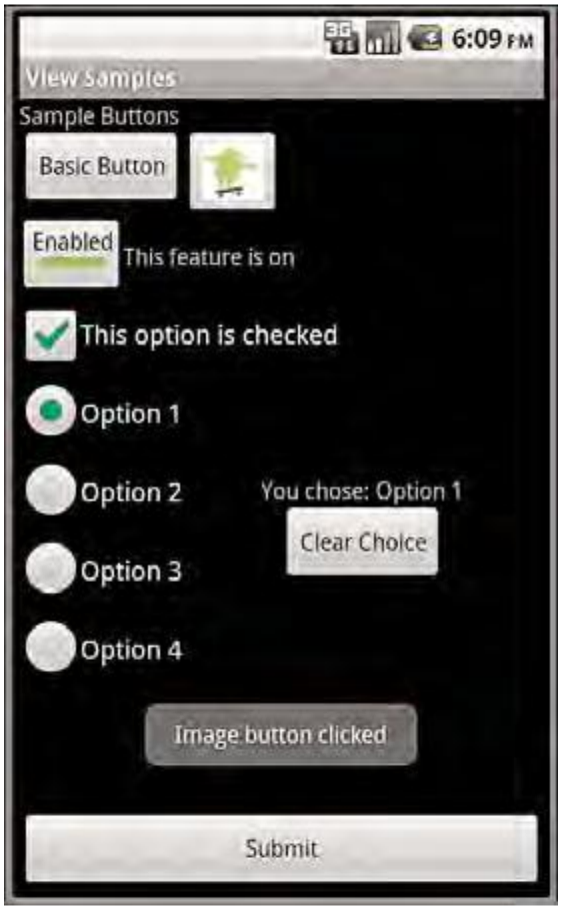
```
<Spinner
android:id ”@+id/Spinner01”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:entries ”@array/colors”
android:prompt ”@string/spin_prompt” />
```



When the user selects it, a pop-up shows the prompt text followed by a list of the possible choices.This list allows only a single item to be selected at a time, and when one is selected, the pop-up goes away
There are a couple of things to notice here.
First, the entries attribute is set to the values that shows by assigning it to an array resource, referred to here as @array/colors.
Second, the prompt attribute is defined to a string resource. Unlike some other string attributes, this one is required to be a string resource.The prompt displays when the popup comes up and can be used to tell the user what kinds of values that can be selected from.

Button

The android.widget.Button class provides a basic button implementation in the Android SDK.Within the XML layout resources, buttons are specified using the Button element.
The primary attribute for a basic button is the text field.This is the label that appears on the middle of the button’s face.You often use basic Button controls for buttons with text such as “Ok,”“Cancel,” or “Submit.”



Various types of button controls

The following XML layout resource file shows a typical Button control definition:

```
<Button
android:id ”@+id/basic_button”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”Basic Button” />
```

A button won’t do anything, other than animate, without some code to handle the click event. Here is an example of some code that handles a click for a basic button and displays a Toast message on the screen:

```
setContentView(R.layout.buttons);
final Button basic_button (Button) findViewById(R.id.basic_button);
basic_button.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
Toast.makeText(ButtonsActivity.this,
“Button clicked”, Toast.LENGTH_SHORT).show();
}
});
```

To handle the click event for when a button is pressed,we first get a reference to the
Button by its resource identifier. Next, the setOnClickListener() method is called. It
requires a valid instance of the class View.OnClickListener.A simple way to provide this is to define the instance right in the method call.This requires implementing the
onClick() method.Within the onClick() method, you are free to carry out whatever
actions you need.

A button with its primary label as an image is an **ImageButton**.An ImageButton is, for most purposes, almost exactly like a basic button. Click actions are handled in the same way.The primary difference is that you can set its src attribute to be an image.

example of an ImageButton definition in an XML layout resource file:

```
<ImageButton
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:id ”@+id/image_button”
android:src ”@drawable/droid” />
```

Check Boxes

The check box button is often used in lists of items where the user can select multiple
items.The Android check box contains a text attribute that appears to the side of the
check box.This is used in a similar way to the label of a basic button. an XML layout resource definition for a CheckBox control:

```
<CheckBox
android:id ”@+id/checkbox”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”Check me?” />
```

The following example shows how to check for the state of the button programmatically and change the text label to reflect the change:

```
final CheckBox check_button (CheckBox) findViewById(R.id.checkbox);
check_button.setOnClickListener(new View.OnClickListener() {
public void onClick (View v) {
TextView tv (TextView)findViewById(R.id.checkbox);
tv.setText(check_button.isChecked() ?
“This option is checked” :
“This option is not checked”);
}
}) ;
```

A check box automatically shows the check as enabled or disabled.This enables us to deal with behavior in our application rather than worrying about how the button should behave.The layout shows that the text starts out one way but, after the user presses the button, the text changes to one of two different things depending
on the checked state
A Toggle Button is similar to a check box in behavior but is usually used to show or
alter the on or off state of something. Like the CheckBox, it has a state (checked or not).
Also like the check box, the act of changing what displays on the button is handled for us.
Unlike the CheckBox, it does not show text next to it. Instead, it has two text fields.
The first attribute is textOn, which is the text that displays on the button when its checked state is on.The second attribute is textOff, which is the text that displays on the button when its checked state is off.The default text for these is “ON” and “OFF,” respectively layout code shows a definition for a toggle button that shows “Enabled”
or “Disabled” based on the state of the button:

```
<ToggleButton
android:id ”@+id/toggle_button”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”Toggle”
android:textOff ”Disabled”
android:textOn ”Enabled” />
```

Using RadioGroups and RadioButtons

use radio buttons when a user should be allowed to only select one item from a small group of items. For instance, a question asking for gender can give three options:
male, female, and unspecified. Only one of these options should be checked at a time.
The RadioButton objects are similar to CheckBox objects.They have a text label next to them, set via the text attribute, and they have a state (checked or unchecked).
group RadioButton objects inside a RadioGroup that handles enforcing their combined states so that only one RadioButton can be checked at a time. If the user selects a RadioButton that is already checked, it does not become unchecked. However, you can provide the user with an action to clear the state of the entire RadioGroup so that none of the buttons are checked.

XML layout resource

```
<RadioGroup
android:id ”@+id/RadioGroup01”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”>
<RadioButton
android:id ”@+id/RadioButton01”
android:layout_width ”wrap_content”
```

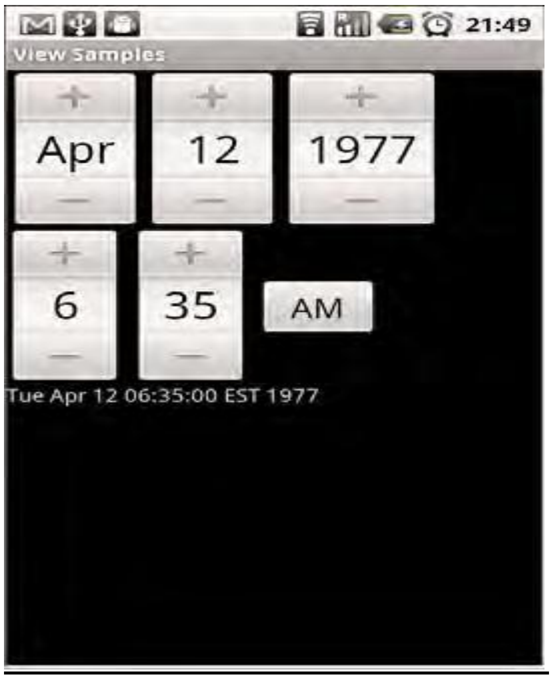
```
android:layout_height ”wrap_content”
android:text ”Option 1”></RadioButton>
<RadioButton
android:id ”@+id/RadioButton02”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”Option 2”></RadioButton>
<RadioButton
android:id ”@+id/RadioButton03”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”Option 3”></RadioButton>
<RadioButton
android:id ”@+id/RadioButton04”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”Option 4”></RadioButton>
</RadioGroup>
```

<u>actions on these RadioButton objects through the RadioGroup</u>

```
final RadioGroup group (RadioGroup)findViewById(R.id.RadioGroup01);
final TextView tv (TextView)
findViewById(R.id.TextView01);
group.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
public void onCheckedChanged(
RadioGroup group, int checkedId) {
if (checkedId ! -1) {
RadioButton rb (RadioButton)
findViewById(checkedId);
if (rb ! null) {
tv.setText(“You chose: “ + rb.getText());
}
} else {
tv.setText(“Choose 1”);
}
}
});
```

Date and Times from users

The Android SDK provides a couple controls for getting date and time input from the user.The first is the DatePicker control (Figure 7.8, top). It can be used to get a month, day, and year from the user.



Date and time controls

XML layout resource definition for a DatePicker follows:

```
<DatePicker
android:id ”@+id/DatePicker01”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content” />
```

<u>onDateChanged()call when the date changes</u>

```
final DatePicker date (DatePicker)findViewById(R.id.DatePicker01);
date.init(date.getYear(), date.getMonth(), date.getDayOfMonth(),
new DatePicker.OnDateChangedListener() {
public void onDateChanged(DatePicker view, int year,
int monthOfYear, int dayOfMonth) {
Date dt new Date(year-1900, monthOfYear, dayOfMonth, time.getCurrentHour(),
time.getCurrentMinute());
text.setText(dt.toString());
}
});
```

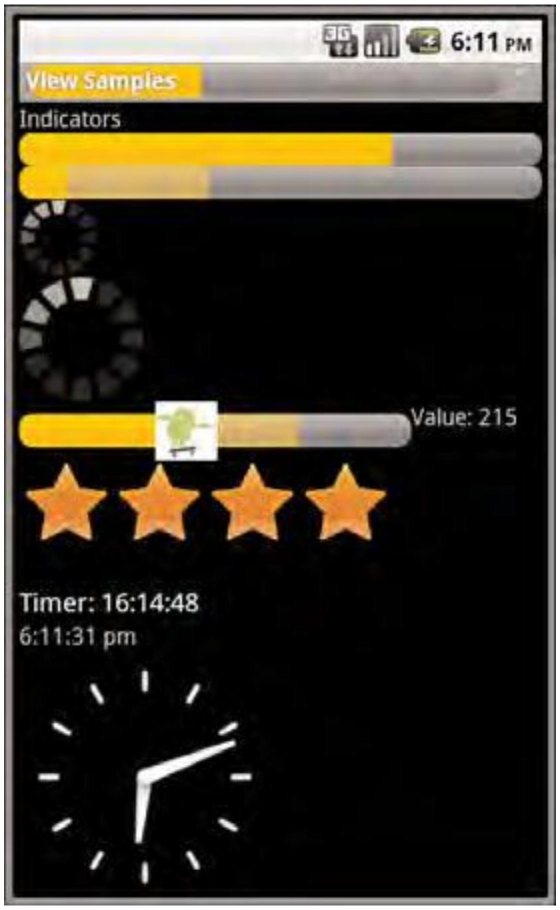
The preceding code sets the DatePicker.OnDateChangedListener by a call to the DatePicker.init() method.A DatePicker control is initialized with the current date.A
TextView is set with the date value that the user entered into the DatePicker control.
The value of 1900 is subtracted from the year parameter to make it compatible with the java.util.Date class.

A <u>**TimePicker**</u> control (also shown in Figure 7.8, bottom) is similar to the DatePicker control. It also doesn't have any unique attributes. However, to register for a method call when the values change, you call the more traditional method of

```
TimePicker.setOnTimeChangedListener().
time.setOnTimeChangedListener(new TimePicker.OnTimeChangedListener() {
public void onTimeChanged(TimePicker view,
int hourOfDay, int minute) {
Date dt new Date(date.getYear()-1900, date.getMonth(),
date.getDayOfMonth(), hourOfDay, minute);
text.setText(dt.toString());
}
});
```

ProgressBar

The standard progress bar is a circular indicator that only animates. It does not show
how complete an action is. It can, however, show that something is taking place.This is useful when an action is indeterminate in length.There are three sizes of this type of
progress indicator



Various types of progress and rating indicators.

The second type is a horizontal progress bar that shows the completeness of an action.
(For example, you can see how much of a file is downloading.) This horizontal progress bar can also have a secondary progress indicator on it.This can be used, for instance, to show the completion of a downloading media file while that file plays.
This is an XML layout resource definition for a basic indeterminate progress bar:

```
<ProgressBar
android:id ”@+id/progress_bar”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content” />
```

Time Passage with the Chronometer

Sometimes you want to show time passing instead of incremental progress. In this case, you can use the Chronometer control as a timer. This might be
useful if it’s the user who is taking time doing some task or in a game where some action needs to be timed.The Chronometer control can be formatted with text, as shown in this

XML layout resource definition:

```
<Chronometer
android:id ”@+id/Chronometer01”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:format ”Timer: %s” />
```

You can use the Chronometer object’s format attribute to put text around the time that displays.A Chronometer won’t show the passage of time until its start() method is
called. To stop it, simply call its stop() method. Finally, you can change the time from
which the timer is counting.That is, you can set it to count from a particular time in the past instead of from the time it’s started.You call the setBase() method to do this
In this next example code, the timer is retrieved from the View by its resource identifier. We then check its base value and set it to 0. Finally,we start the timer counting up from there.

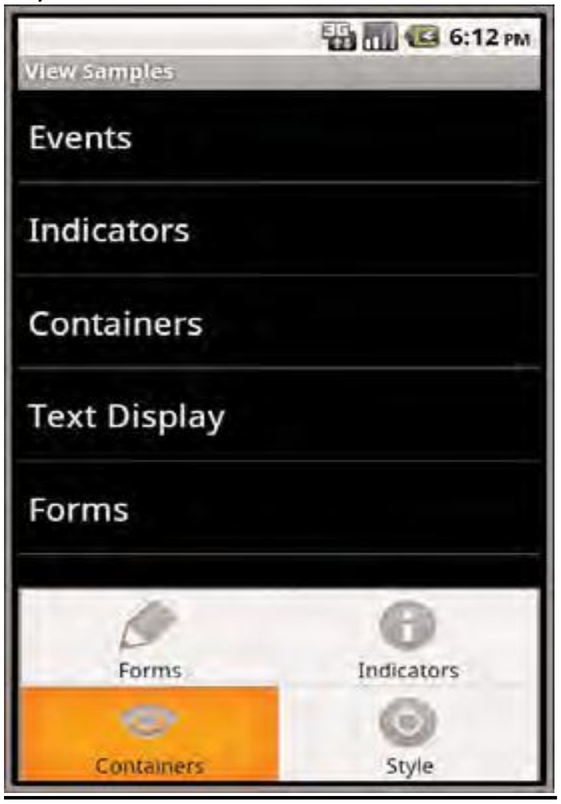
```
final Chronometer timer
(Chronometer)findViewById(R.id.Chronometer01);
long base timer.getBase();
Log.d(ViewsMenu.debugTag, “base “+ base);
timer.setBase(0);
timer.start();
```

Menu

two special application menus for use within your Android applications:
the options menu and the context menu.

I) Options Menus

The Android SDK provides a method for users to bring up a menu by pressing the menu key from within the application. You can use options menus within your application to bring up help, to navigate, to provide additional controls, or to configure options.The OptionsMenu control can contain icons, submenus, and keyboard shortcuts.



An options menu.

For an options menu to show when a user presses the Menu button on their device, you need to override the implementation of onCreateOptionsMenu() in your Activity.

Here is a sample implementation that gives the user three menu items to choose from:

```
public boolean onCreateOptionsMenu( android.view.Menu menu) {
super.onCreateOptionsMenu(menu);
menu.add(“Forms”)
.setIcon(android.R.drawable.ic_menu_edit)
.setIntent(new Intent(this, FormsActivity.class));
menu.add(“Indicators”)
.setIntent(new Intent(this, IndicatorsActivity.class))
.setIcon(android.R.drawable.ic_menu_info_details);
menu.add(“Containers”)
.setIcon(android.R.drawable.ic_menu_view)
.setIntent(new Intent(this, ContainersActivity.class));
return true;
}
```

ii) **ContextMenu**

The ContextMenu is a subtype of Menu that you can configure to display when a long press is performed on a View.As the name implies, the ContextMenu provides for contextual menus to display to the user for performing additional actions on selected items.
ContextMenu objects are slightly more complex than OptionsMenu objects.You need to implement the onCreateContextMenu() method of your Activity for one to display.
However, before that is called, you must call the registerForContextMenu() method and pass in the View for which you want to have a context menu.This means each View on your screen can have a different context menu, which is appropriate as the menus are designed to be highly contextual.

Recall that any View control can register to trigger a call to the onCreateContextMenu() method when the user performs a long press.That means we have to check which View control it was for which the user tried to get a context menu. Next,we inflate the appropriate menu from a menu resource that we defined with XML. Because we can't define header information in the menu resource file,we set a stock Android SDK resource to it and add a title. Here is the menu resource that is inflated:

```
<menu
xmlns:android ”http://schemas.android.com/apk/res/android”>
<item
android:id ”@+id/start_timer”
android:title ”Start” />
<item
android:id ”@+id/stop_timer”
android:title ”Stop” />
<item
android:id ”@+id/reset_timer”
android:title ”Reset” />
</menu>
```

Now we need to handle the ContextMenu clicks by implementing the onContextItemSelected() method in our Activity.

Here's an example:

```
public boolean onContextItemSelected(MenuItem item) {
super.onContextItemSelected(item);
boolean result false;
Chronometer timer (Chronometer)findViewById(R.id.Chronometer01);
switch (item.getItemId()){
case R.id.stop_timer:
timer.stop();
result true;
break;
case R.id.start_timer:
timer.start();
result true;
break;
case R.id.reset_timer:
timer.setBase(SystemClock.elapsedRealtime());
result true;
break;
}
return result;
}
```

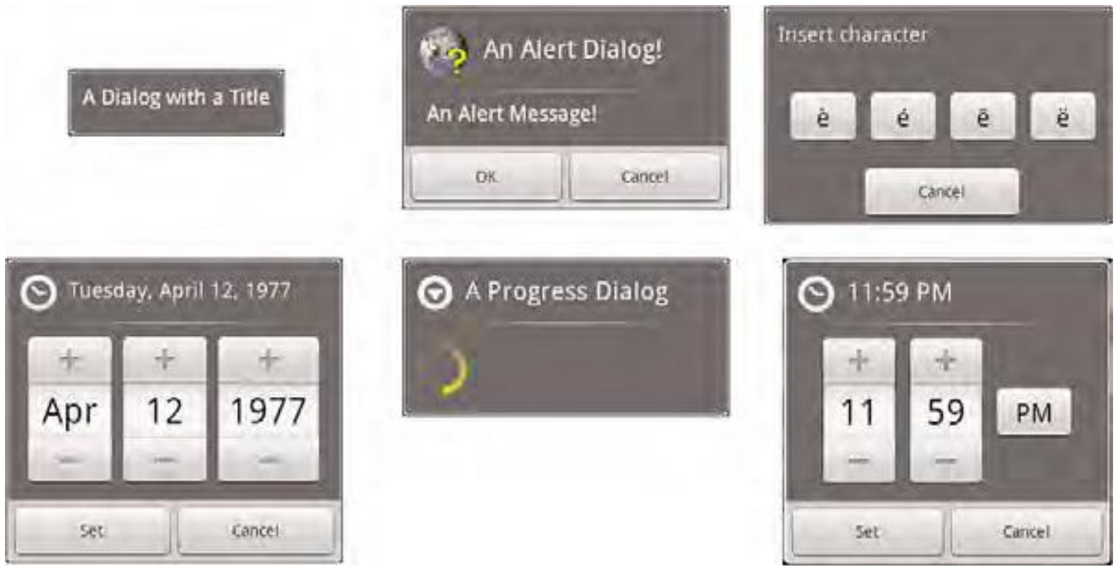
Dialogs

An Activity can use dialogs to organize information and react to user-driven events. For example, an activity might display a dialog informing the user of a problem or ask the user to confirm an action such as deleting a data record

Exploring the Different Types of Dialogs

There are a number of different dialog types available within the Android SDK. Each has a special function that most users should be somewhat familiar with.The dialog types available include

1) **Dialog:** The basic class for all Dialog types. A basic Dialog is shown in the top left in below figure.
2) **AlertDialog:** A Dialog with one, two, or three Button controls. An AlertDialog is shown in the top center in below figure.
3) **CharacterPickerDialog:** A Dialog for choosing an accented character associated with a base character.A CharacterPickerDialog is shown in the top right in below figure.
4) **DatePickerDialog:** A Dialog with a DatePicker control. A DatePickerDialog is shown in the bottom left of in below figure.
5) **ProgressDialog:** A Dialog with a determinate or indeterminate ProgressBar control.An indeterminate ProgressDialog is shown in the bottom center of in following figure.
1) **TimePickerDialog:** A Dialog with a TimePicker control. A TimePickerDialog is shown in the bottom right in below figure



The different dialog types available in Android.

Tracing the Lifecycle of a Dialog

Each Dialog must be defined within the Activity in which it is used.A Dialog may be launched once, or used repeatedly. Understanding how an Activity manages the Dialog lifecycle is important to implementing a Dialog correctly. Let's look at the key methods that an Activity must use to manage a Dialog:

1) The showDialog() method is used to display a Dialog.
2) The dismissDialog() method is used to stop showing a Dialog.The Dialog is kept around in the Activity's Dialog pool. If the Dialog is shown again using showDialog(), the cached version is displayed once more.
3) The removeDialog() method is used to remove a Dialog from the Activity objects Dialog pool.The Dialog is no longer kept around for future use. If you call showDialog() again, the Dialog must be re-created.

Adding the Dialog to an Activity involves several steps:

1) Define a unique identifier for the Dialog within the Activity.
2) Implement the onCreateDialog() method of the Activity to return a Dialog of the appropriate type, when supplied the unique identifier
3) Implement the onPrepareDialog() method of the Activity to initialize the Dialog as appropriate.
4) Launch the Dialog using the showDialog() method with the unique identifier.

Defining a Dialog

A Dialog used by an Activity must be defined in advance. Each Dialog has a special

identifier (an integer).When the showDialog() method is called, you pass in this identifier. At this point, the onCreateDialog() method is called and must return a Dialog of the appropriate type.
It is up to the developer to override the onCreateDialog() method of the Activity
and return the appropriate Dialog for a given identifier. If an Activity has multiple
Dialog windows, the onCreateDialog() method generally contains a switch statement to return the appropriate Dialog based on the incoming parameter—the Dialog identifier.

Initializing a Dialog

Because a Dialog is often kept around by the Activity in its Dialog pool, it might be
important to re-initialize a Dialog each time it is shown, instead of just when it is created the first time. For this purpose, you can override the onPrepareDialog() method of the Activity.
Although the onCreateDialog() method may only be called once for initial Dialog
creation, the onPrepareDialog() method is called each time the showDialog() method is called, giving the Activity a chance to modify the Dialog before it is shown to the user.

Launching a Dialog

You can display any Dialog defined within an Activity by calling its showDialog()
method and passing it a valid Dialog identifier—one that will be recognized by the
onCreateDialog() method.

Dismissing a Dialog

Most types of dialogs have automatic dismissal circumstances. However, if you want to force a Dialog to be dismissed, simply call the dismissDialog() method and pass in the Dialog identifier.

Removing a Dialog from Use

Dismissing a Dialog does not destroy it. If the Dialog is shown again, its cached contents are redisplayed. If you want to force an Activity to remove a Dialog from its pool and not use it again, you can call the removeDialog() method, passing in the valid Dialog identifier.

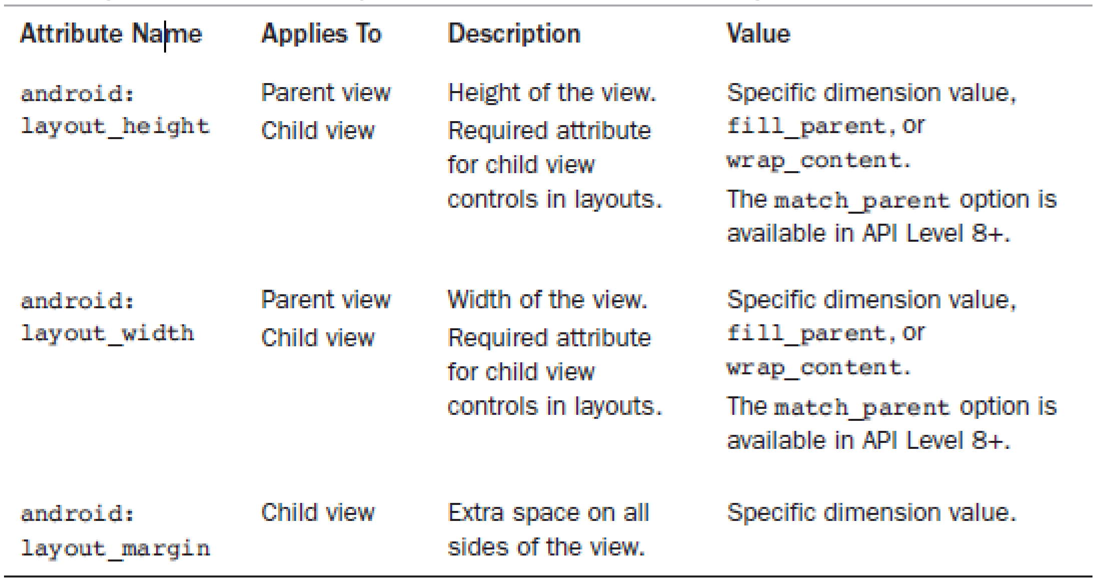
3.2) Layout Classes

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

a) FrameLayout
b) LinearLayout
c) RelativeLayout
d) 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.



Attribute Name	Applies To	Description	Value
android: layout_height	Parent view Child view	Height of the view. Required attribute for child view controls in layouts.	Specific dimension value, fill_parent, or wrap_content. The match_parent option is available in API Level 8+.
android: layout_width	Parent view Child view	Width of the view. Required attribute for child view controls in layouts.	Specific dimension value, fill_parent, or wrap_content. The match_parent option is available in API Level 8+.
android: layout_margin	Child view	Extra space on all sides of the view.	Specific dimension value.

an XML layout resource example of a LinearLayout set to the size of the screen, containing one TextView that is set to its full height and the width of the LinearLayout (and therefore the screen):

```
<LinearLayout xmlns:android
“http://schemas.android.com/apk/res/android”
android:layout_width ”fill_parent”
android:layout_height ”fill_parent”>
<TextView
android:id ”@+id/TextView01”
android:layout_height ”fill_parent”
android:layout_width ”fill_parent” />
</LinearLayout>
```

an example of a Button object with some margins set via XML used in a layout resource file:

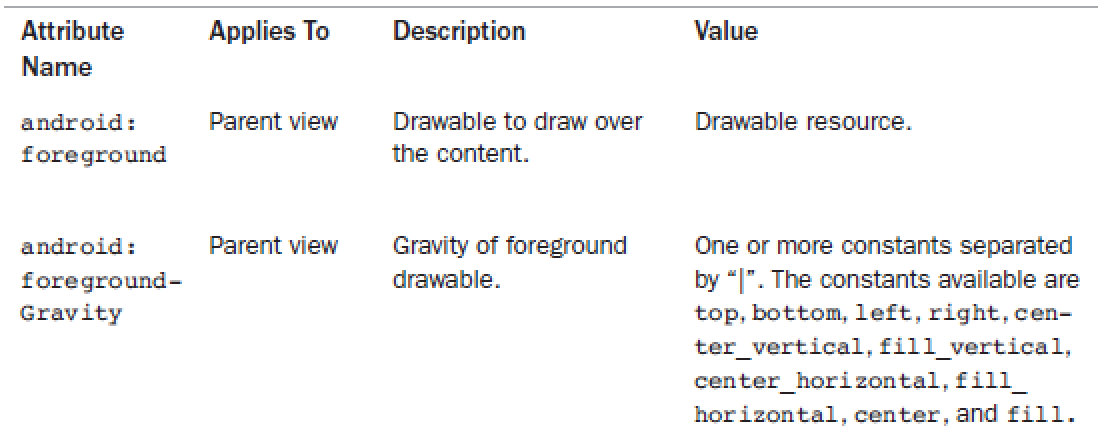
```
<Button
android:id ”@+id/Button01”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”Press Me”
```

android:layout_marginRight ”20px”
android:layout_marginTop ”60px” />

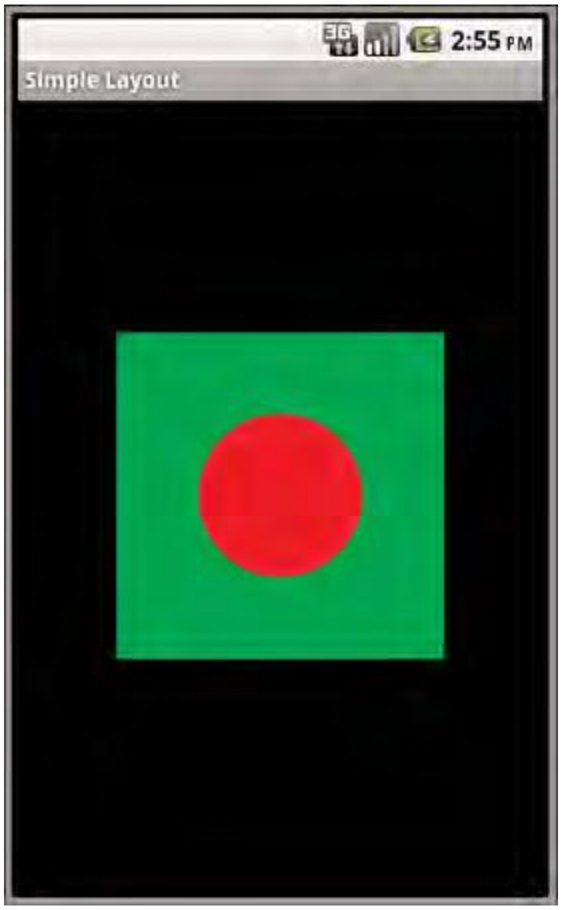
layout elements can cover any rectangular space on the screen; it doesn’t
need to be the entire screen. Layouts can be nested within one another.

Frame layout

A FrameLayout view is designed to display a stack of child View items.You can add multiple views to this layout, but each View is drawn from the top-left corner of the layout. find the layout attributes available for FrameLayout child View objects in
android.control.FrameLayout.LayoutParams



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`.



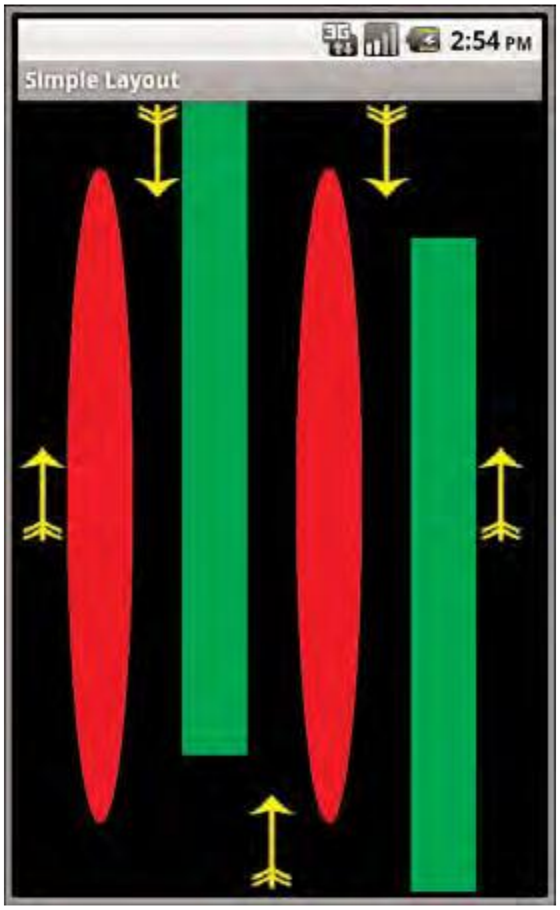
an XML layout resource with a FrameLayout and two child View objects, both ImageView objects.The green rectangle is drawn first and the red oval is drawn on top of it.The green rectangle is larger, so it defines the bounds of the
FrameLayout:

```
<FrameLayout xmlns:android
http://schemas.android.com/apk/res/android
android:id ”@+id/FrameLayout01”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:layout_gravity ”center”>
<ImageView
android:id ”@+id/ImageView01”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:src ”@drawable/green_rect”
android:minHeight ”200px”
android:minWidth ”200px” />
<ImageView
```

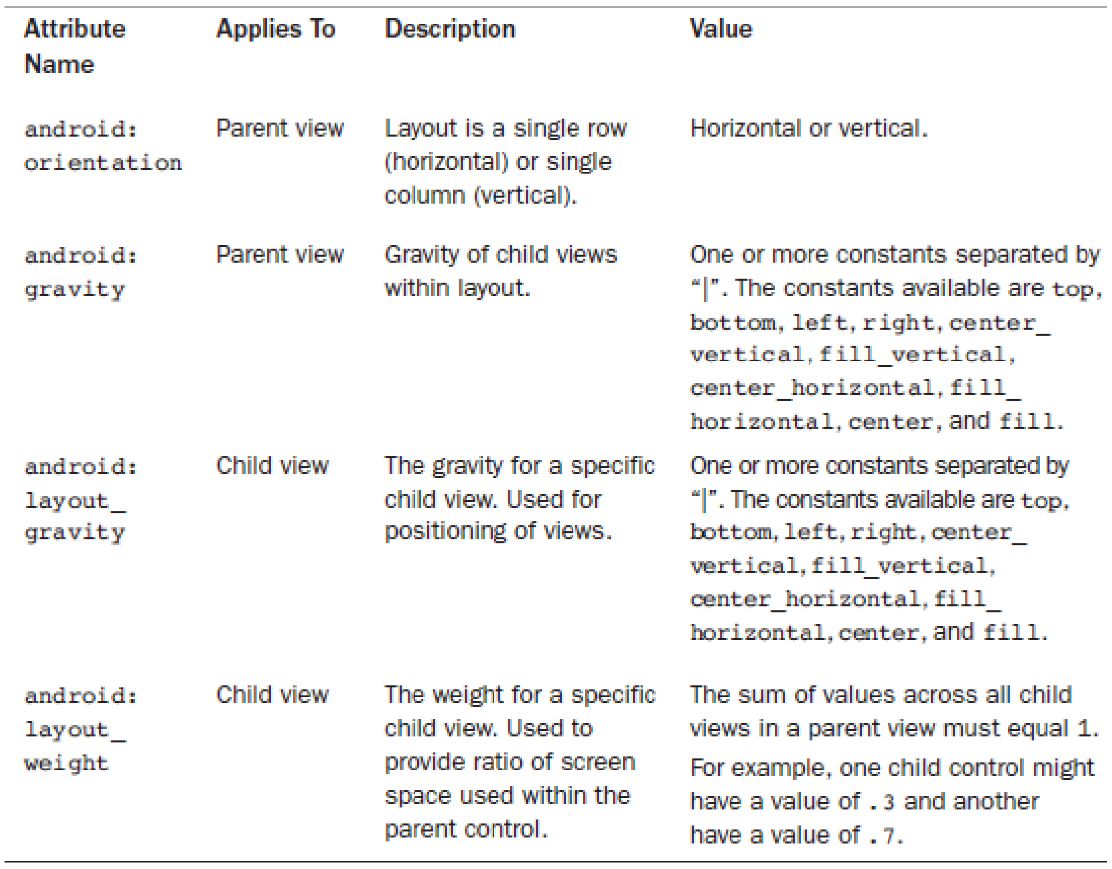
```
android:id ”@+id/ImageView02”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:src ”@drawable/red_oval”
android:minHeight ”100px”
android:minWidth ”100px”
android:layout_gravity ”center” />
</FrameLayout>
```

LinearLayout

A LinearLayout view organizes its child View objects in a single row 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.Table 8.3 describes some of the important attributes specific to LinearLayout views.



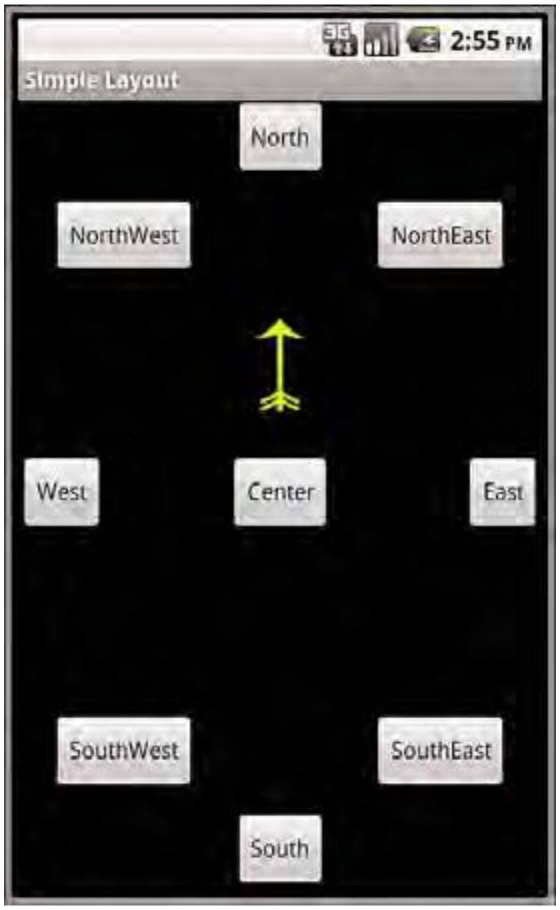
Attribute Name	Applies To	Description	Value
`android:orientation`	Parent view	Layout is a single row (horizontal) or single column (vertical).	Horizontal or vertical.
`android:gravity`	Parent view	Gravity of child views within layout.	One or more constants separated by "|". The constants available are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

<u>**RelativeLayout**</u>

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.



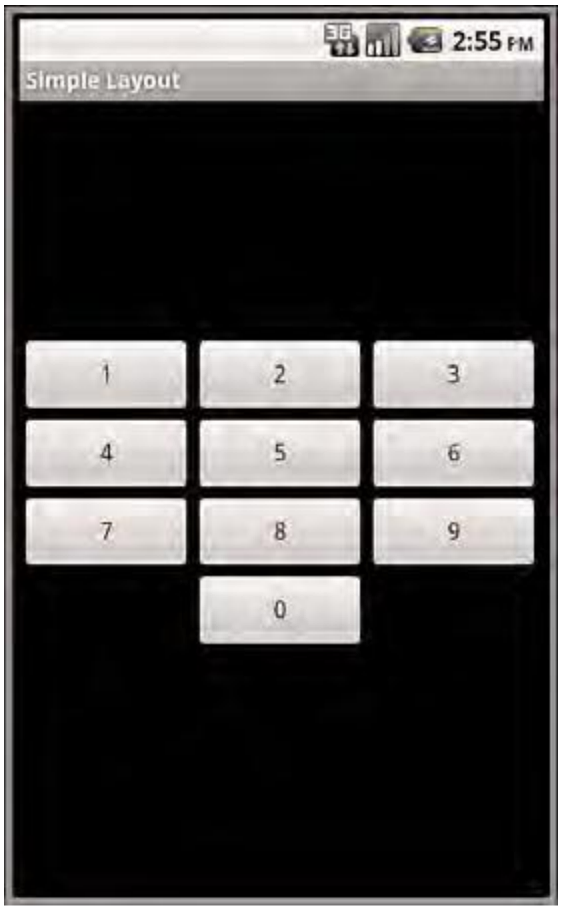
Attribute Name	Applies To	Description	Value
`android:gravity`	Parent view	Gravity of child views within layout.	One or more constants separated by "|". The constants available are `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, and `fill`.
`android:layout_centerInParent`	Child view	Centers child view horizontally and vertically within parent view.	True or false.

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

```
<?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 Figure 8.7.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 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.

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`”
`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`

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>
```

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.

ArrayAdapter

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);
```

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” />
```