3

Getting to Know the Android User Interface

WHAT YOU WILL LEARN IN THIS CHAPTER

- The various ViewGroups you can use to lay out your views
- How to adapt and manage changes in screen orientation
- How to create the UI programmatically
- How to listen for UI notifications

In Chapter 2, you learned about the `Activity` class and its life cycle. You learned that an activity is a means by which users interact with the application. However, an activity by itself does not have a presence on the screen. Instead, it has to draw the screen using *Views* and *ViewGroups*. In this chapter, you will learn the details about creating user interfaces in Android, and how users interact with them. In addition, you will learn how to handle changes in screen orientation on your Android devices.

UNDERSTANDING THE COMPONENTS OF A SCREEN

In Chapter 2, you learned that the basic unit of an Android application is an *activity*. An *activity* displays the user interface of your application, which may contain widgets such as buttons, labels, textboxes, and so on. Typically, you define your UI using an XML file (e.g., the `main.xml` file located in the `res/layout` folder of your project), which looks similar to the following:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
    android:orientation="vertical" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

</LinearLayout>
```

During runtime, you load the XML UI in the `onCreate()` method handler in your `Activity` class, using the `setContentView()` method of the `Activity` class:

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

During compilation, each element in the XML file 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.

NOTE *Although it is always easier to build your UI using an XML file, sometimes you need to build your UI dynamically during runtime (for example, when writing games). Hence, it is also possible to create your UI entirely using code. Later in this chapter you will see an example showing how this can be done.*

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 textboxes. A view derives from the base class `android.view.View`.

NOTE *Chapters 4 and 5 discuss the various common views in Android.*

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
- ScrollView

The following sections describe each of these ViewGroups in more detail. Note that in practice it is common to combine different types of layouts to create the UI you want.

LinearLayout

The `LinearLayout` arranges views in a single column or a single row. Child views can be arranged either vertically or horizontally. To see how `LinearLayout` works, 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:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <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 in Table 3-1.

TABLE 3-1: Common Attributes Used in Views and ViewGroups

ATTRIBUTE	DESCRIPTION
`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

continues

TABLE 3-1 *(continued)*

ATTRIBUTE	DESCRIPTION
`layout_gravity`	Specifies how child Views are positioned
`layout_weight`	Specifies how much of the extra space in the layout should be allocated to the View
`layout_x`	Specifies the x-coordinate of the View or ViewGroup
`layout_y`	Specifies the y-coordinate of the View or ViewGroup

NOTE *Some of these attributes are applicable only when a View is in a specific ViewGroup. For example, the* `layout_weight` *and* `layout_gravity` *attributes are applicable only when a View is in either a* `LinearLayout` *or a* `TableLayout`*.*

For example, 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 (in this case, the text contained within it). If you don't want the `<TextView>` view to occupy the entire row, you can set its `layout_width` attribute to `wrap_content`, like this:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
```

The preceding code will set the width of the view to be equal to the width of the text contained within it.

Consider the following layout:

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

<TextView
    android:layout_width="100dp"
    android:layout_height="wrap_content"
    android:text="@string/hello" />

<Button
    android:layout_width="160dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:onClick="onClick" />

</LinearLayout>
```

UNITS OF MEASUREMENT

When specifying the size of an element on an Android UI, you should be aware of the following units of measurement:

`dp` — Density-independent pixel. 1 `dp` is equivalent to one pixel on a 160 dpi screen. This is the recommended unit of measurement when specifying the dimension of views in your layout. The 160 dpi screen is the baseline density assumed by Android. You can specify either "`dp`" or "`dip`" when referring to a density-independent pixel.

`sp` — Scale-independent pixel. This is similar to `dp` and is recommended for specifying font sizes.

`pt` — Point. A point is defined to be 1/72 of an inch, based on the physical screen size.

`px` — Pixel. Corresponds to actual pixels on the screen. Using this unit is not recommended, as your UI may not render correctly on devices with a different screen resolution.

Here, you set the width of both the `TextView` and `Button` views to an absolute value. In this case, the width for the `TextView` is set to 100 density-independent pixels wide, and the `Button` to 160 density-independent pixels wide. Before you see how the views will look like on different screens with different pixel density, it is important to understand how Android recognizes screens of varying sizes and density.

Figure 3-1 shows the screen of the Nexus S. It has a 4-inch screen (diagonally), with a screen width of 2.04 inches. Its resolution is 480 (width) × 800 (height) pixels. With 480 pixels spread across a width of 2.04 inches, the result is a pixel density of about 235 dots per inch (dpi).

As you can see from the figure, the pixel density of a screen varies according to screen size and resolution.

FIGURE 3-1

Android defines and recognizes four screen densities:

- **Low density (*ldpi*)** — 120 dpi
- **Medium density (*mdpi*)** — 160 dpi
- **High density (*hdpi*)** — 240 dpi
- **Extra High density (*xhdpi*)** — 320 dpi

Your device will fall into one of the densities defined in the preceding list. For example, the Nexus S is regarded as a *hdpi* device, as its pixel density is closest to 240 dpi. The HTC Hero, however, has a

3.2-inch (diagonal) screen size and a resolution of 320 × 480. Its pixel density works out to be about 180 dpi. Therefore, it would be considered an *mdpi* device, as its pixel density is closest to 160 dpi.

To test how the views defined in the XML file will look when displayed on screens of different densities, create two AVDs with different screen resolutions and abstracted LCD densities. Figure 3-2 shows an AVD with 480 × 800 resolution and LCD density of 235, emulating the Nexus S.

Figure 3-3 shows another AVD with 320 × 480 resolution and LCD density of 180, emulating the HTC Hero.

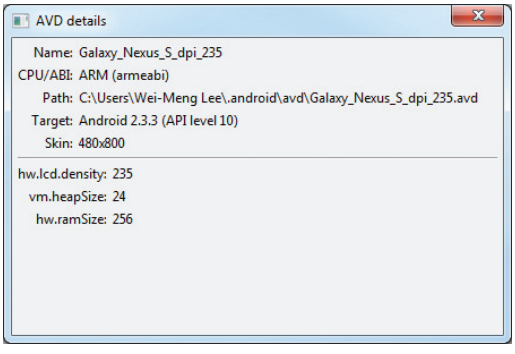


FIGURE 3-2

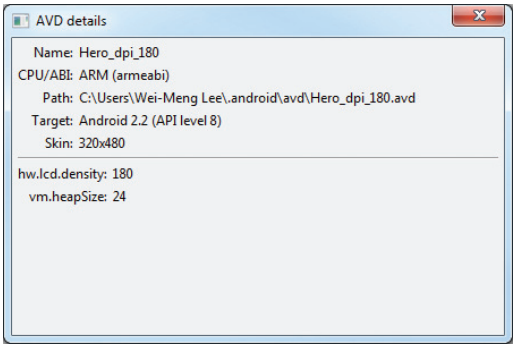


FIGURE 3-3

Figure 3-4 shows what the views look like when viewed on the emulator with a screen density of 235 dpi.

Figure 3-5 shows what the views look like when viewed on the emulator with a screen density of 180 dpi.

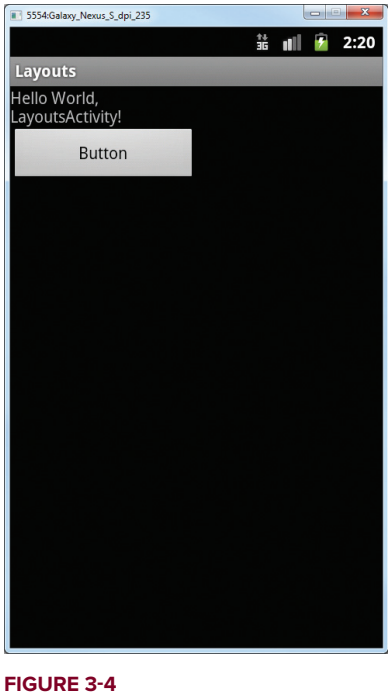


FIGURE 3-4

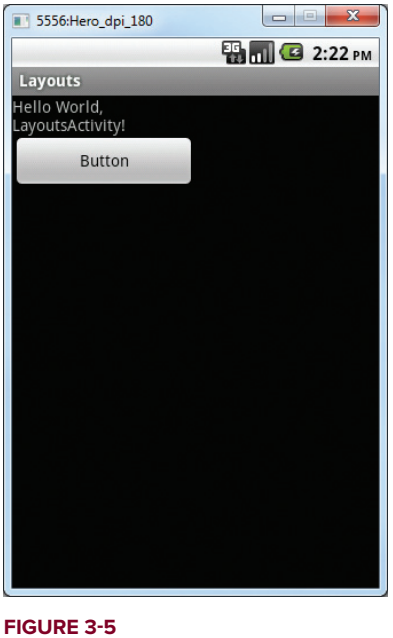


FIGURE 3-5

Using the *dp* unit ensures that your views are always displayed in the right proportion regardless of the screen density — Android automatically scales the size of the view depending on the density of the screen. Using the `Button` as an example, if it is displayed on a 180 dpi screen (a 180 dpi screen is treated just like a 160 dpi screen), its width would be 160 pixels. However, if it is displayed on a 235 dpi screen (which is treated as a 240 dpi screen), then its width would be 240 pixels.

HOW TO CONVERT DP TO PX

The formula for converting `dp` to `px` (pixels) is as follows:

Actual pixels = dp * (dpi / 160), where dpi is either 120, 160, 240, or 320.

Therefore, in the case of the `Button` on a 235 dpi screen, its actual width is 160 * (240/160) = 240 `px`. When run on the 180 dpi emulator (regarded as a 160 dpi device), its actual pixel is now 160 * (160/160) = 160 `px`. In this case, one `dp` is equivalent to one `px`.

To prove that this is indeed correct, you can use the `getWidth()` method of a `View` object to get its width in pixels:

```
public void onClick(View view) {
    Toast.makeText(this,
        String.valueOf(view.getWidth()),
        Toast.LENGTH_LONG).show();
}
```

What if instead of using `dp` you now specify the size using pixels (`px`)?

```
<TextView
    android:layout_width="100px"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
<Button
    android:layout_width="160px"
    android:layout_height="wrap_content"
    android:text="Button"
    android:onClick="onClick"/>
```

Figure 3-6 shows what the `Label` and `Button` look like on a 235 dpi screen. Figure 3-7 shows the same views on a 180 dpi screen. In this case, Android does not perform any conversion, as all the

sizes are specified in pixels. In general (with screen sizes being equal), if you use pixels for view sizes, the views will appear smaller on a device with a high dpi screen, compared to one with a lower dpi.

FIGURE 3-6

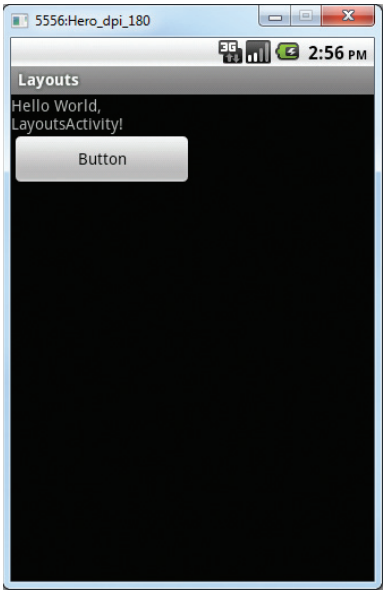


FIGURE 3-7

The preceding example also specifies that the orientation of the layout is vertical:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
```

The default orientation layout is horizontal, so if you omit the `android:orientation` attribute, the views will appear as shown in Figure 3-8.

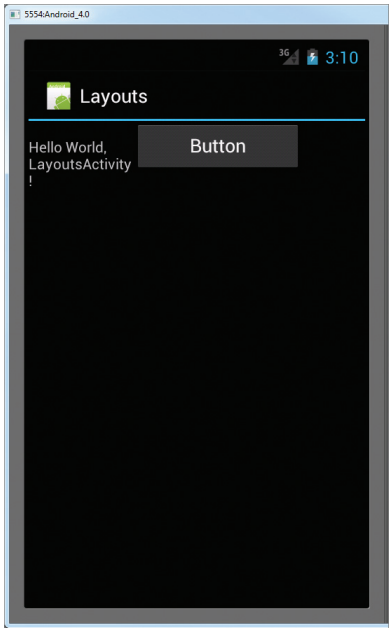


FIGURE 3-8

In `LinearLayout`, you can apply the `layout_weight` and `layout_gravity` attributes to views contained within it, as the following modifications to `main.xml` show:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button
    android:layout_width="160dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_gravity="left"
    android:layout_weight="1" />

<Button
    android:layout_width="160dp"
    android:layout_height="wrap_content"
    android:text="Button"
```

```
        android:layout_gravity="center"
        android:layout_weight="2" />

    <Button
        android:layout_width="160dp"
        android:layout_height="wrap_content"
        android:text="Button"
        android:layout_gravity="right"
        android:layout_weight="3" />

    </LinearLayout>
```

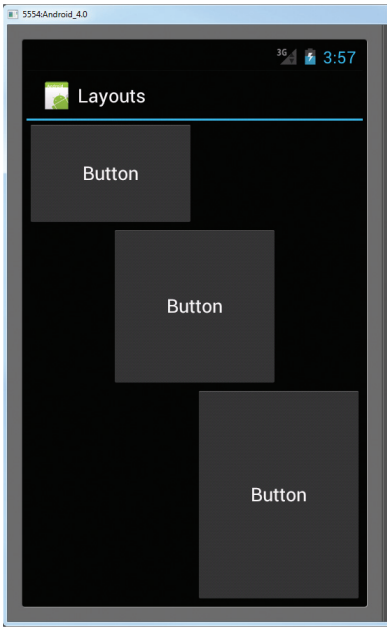


FIGURE 3-9

Figure 3-9 shows the positioning of the views as well as their heights. The `layout_gravity` attribute indicates the positions the views should gravitate towards, while the `layout_weight` attribute specifies the distribution of available space. In the preceding example, the three buttons occupy about 16.6% (1/(1+2+3) * 100), 33.3% (2/(1+2+3) * 100), and 50% (3/(1+2+3) * 100) of the available height, respectively.

If you change the orientation of the `LinearLayout` to horizontal, you need to change the width of each view to 0 `dp`, and the views will be displayed as shown in Figure 3-10:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >

<Button
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_gravity="left"
    android:layout_weight="1" />

<Button
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_gravity="center_horizontal"
    android:layout_weight="2" />

<Button
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_gravity="right"
    android:layout_weight="3" />

</LinearLayout>
```

FIGURE 3-10

AbsoluteLayout

The `AbsoluteLayout` enables you to specify the exact location of its children. Consider the following UI defined in `main.xml`:

```
<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" />
<Button
    android:layout_width="113dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_x="12px"
    android:layout_y="361px" />
</AbsoluteLayout>
```

Figure 3-11 shows the two `Button` views (tested on a 180 dpi AVD) located at their specified positions using the `android_layout_x` and `android_layout_y` attributes.

However, there is a problem with the `AbsoluteLayout` when the activity is viewed on a high-resolution screen (see Figure 3-12). For this reason, the `AbsoluteLayout` has been deprecated since Android 1.5 (although it is still supported in the current version). You should avoid using the `AbsoluteLayout` in your UI, as it is not guaranteed to be supported in future versions of Android. You should instead use the other layouts described in this chapter.

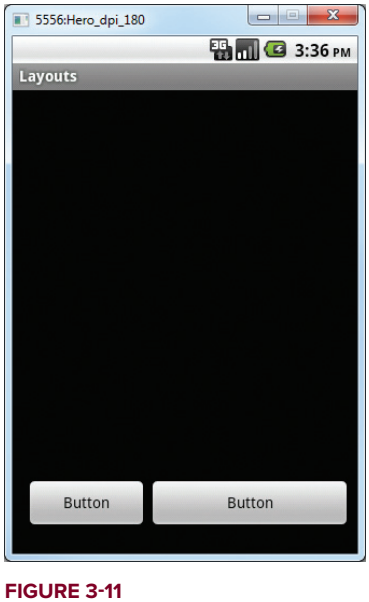


FIGURE 3-11

FIGURE 3-12

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:

```
<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 ="120dp"
            />
        <EditText
            android:id="@+id/txtUserName"
            android:width="200dp" />
    </TableRow>
    <TableRow>
        <TextView
            android:text="Password:"
            />
        <EditText
            android:id="@+id/txtPassword"
            android:password="true"
            />
    </TableRow>
    <TableRow>
        <TextView />
        <CheckBox android:id="@+id/chkRememberPassword"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Remember Password"
            />
    </TableRow>
    <TableRow>
        <Button
            android:id="@+id/buttonSignIn"
            android:text="Log In" />
    </TableRow>
</TableLayout>
```

Figure 3-13 shows what the preceding looks like when rendered on the Android emulator.

Note that in the preceding example, there are two columns and four rows in the `TableLayout`. The cell directly under the Password `TextView` is populated with a `<TextView/>` empty element. If you don't do this, the Remember Password checkbox will appear under the Password `TextView`, as shown in Figure 3-14.

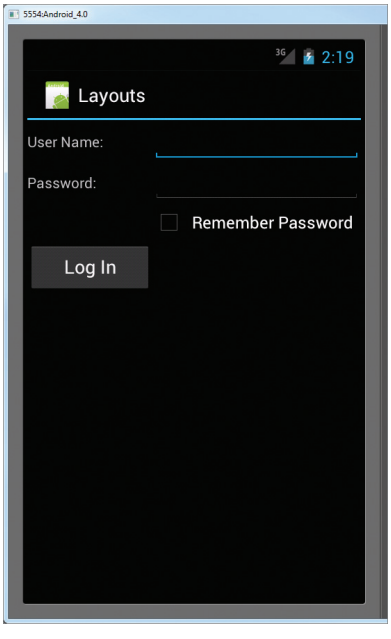


FIGURE 3-13

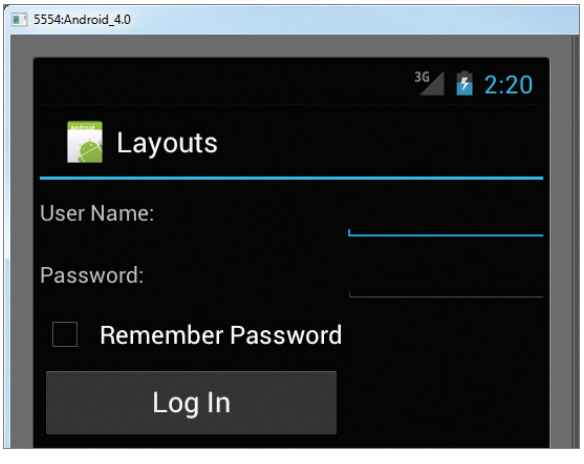


FIGURE 3-14

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" />

    <Button
        android:id="@+id/btnCancel"
        android:layout_width="124px"
        android:layout_height="wrap_content"
        android:text="Cancel"
        android:layout_below="@+id/txtComments"
        android:layout_alignLeft="@+id/txtComments" />
</RelativeLayout>
```

Notice that each view embedded within the `RelativeLayout` has attributes that enable it to align with another view. These attributes are as follows:

- `layout_alignParentTop`
- `layout_alignParentLeft`
- `layout_alignLeft`
- `layout_alignRight`
- `layout_below`
- `layout_centerHorizontal`

The value for each of these attributes is the ID for the view that you are referencing. The preceding XML UI creates the screen shown in Figure 3-15.

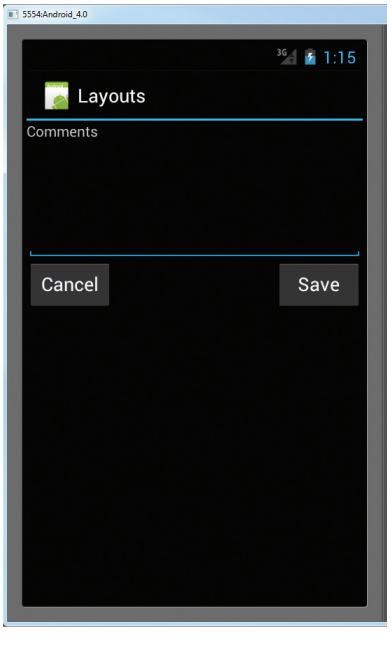


FIGURE 3-15

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. Consider the following content in `main.xml`:

```
<?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="Hello, Android!"
            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/droid"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />

        </FrameLayout>
    </RelativeLayout>
```

Here, you have a `FrameLayout` within a `RelativeLayout`. Within the `FrameLayout`, you embed an `ImageView`. The UI is shown in Figure 3-16.

FIGURE 3-16

NOTE *This example assumes that the* `res/drawable-mdpi` *folder has an image named* `droid.png`.

If you add another view (such as a `Button` view) within the `FrameLayout`, the view will overlap the previous view (see Figure 3-17):

```
<?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="Hello, Android!"
        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/droid"
            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>
```

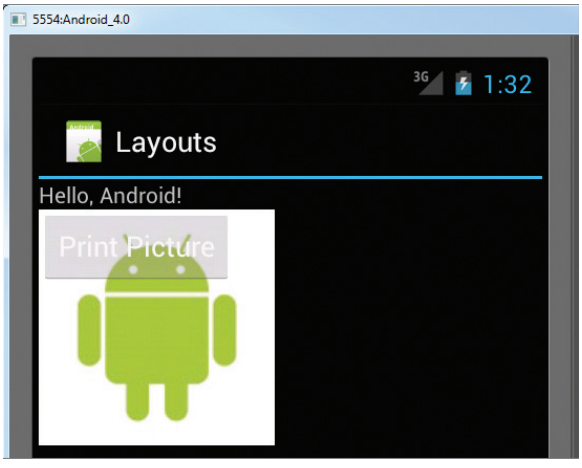


FIGURE 3-17

NOTE *You can add multiple views to a* `FrameLayout`*, but each will be stacked on top of the previous one. This is when you want to animate a series of images, with only one visible at a time.*

ScrollView

A `ScrollView` is a special type of `FrameLayout` in that it enables users to scroll through a list of views that occupy more space than the physical display. The `ScrollView` can contain only one child view or ViewGroup, which normally is a `LinearLayout`.

NOTE *Do not use a* `ListView` *(discussed in Chapter 4) together with the* `ScrollView`*. The* `ListView` *is designed for showing a list of related information and is optimized for dealing with large lists.*

The following `main.xml` content shows a `ScrollView` containing a `LinearLayout`, which in turn contains some `Button` and `EditText` views:

```
<ScrollView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android" >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >
        <Button
            android:id="@+id/button1"
```

```
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 1" />
        <Button
            android:id="@+id/button2"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 2" />
        <Button
            android:id="@+id/button3"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 3" />
        <EditText
            android:id="@+id/txt"
            android:layout_width="fill_parent"
            android:layout_height="600dp" />
        <Button
            android:id="@+id/button4"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 4" />
        <Button
            android:id="@+id/button5"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 5" />
    </LinearLayout>
</ScrollView>
```

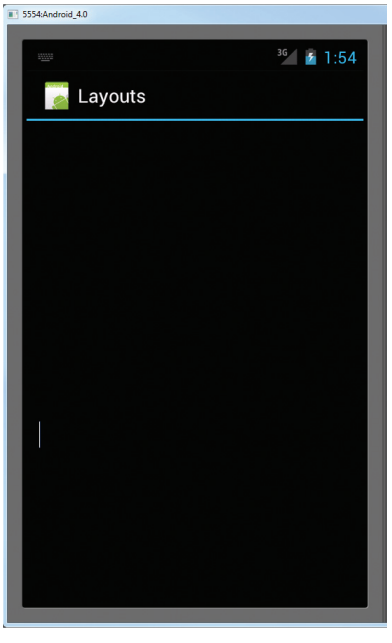


FIGURE 3-18

If you load the preceding code on the Android emulator, you will see something like Figure 3-18.

Because the `EditText` automatically gets the focus, it fills up the entire activity (as the height was set to 600dp). To prevent it from getting the focus, add the following two attributes to the `<LinearLayout>` element:

```
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:focusable="true"
        android:focusableInTouchMode="true" >
```

You will now be able to view the buttons and scroll through the list of views (see Figure 3-19).

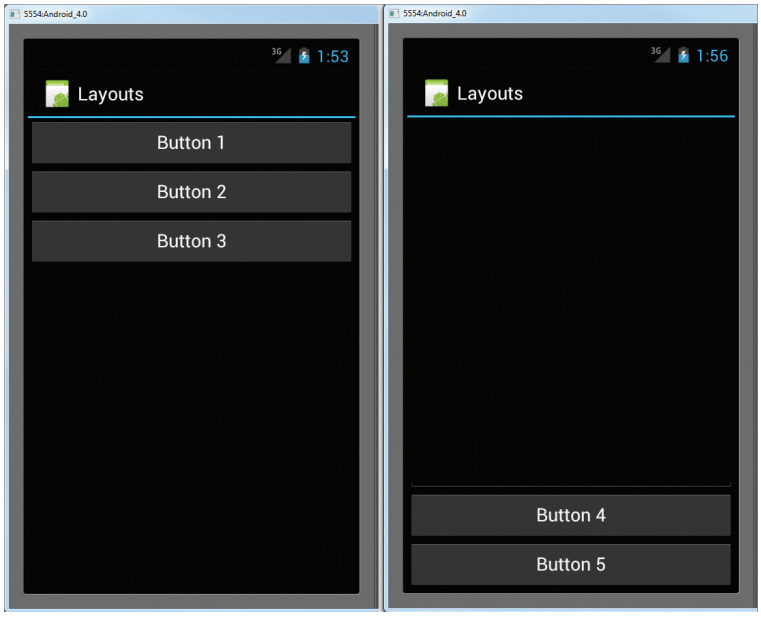


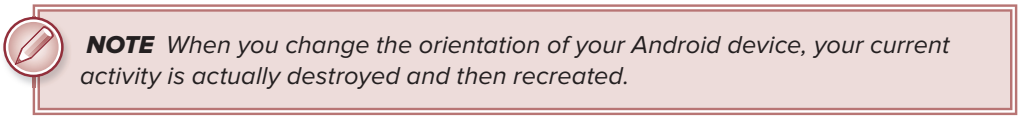
FIGURE 3-19

Sometimes you may want the `EditText` to automatically get the focus, but you do not want the soft input panel (keyboard) to appear automatically (which happens on a real device). To prevent the keyboard from appearing, add the following attribute to the `<activity>` element in the `AndroidManifest.xml` file:

```
<activity
    android:label="@string/app_name"
    android:name=".LayoutsActivity"
    android:windowSoftInputMode="stateHidden" >
    <intent-filter >
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

ADAPTING TO DISPLAY ORIENTATION

One of the key features of modern smartphones is their ability to switch screen orientation, and Android is no exception. Android supports two screen orientations: *portrait* and *landscape*. By default, when you change the display orientation of your Android device, the current activity that is displayed automatically redraws its content in the new orientation. This is because the `onCreate()` method of the activity is fired whenever there is a change in display orientation.



NOTE *When you change the orientation of your Android device, your current activity is actually destroyed and then recreated.*

However, when the views are redrawn, they may be drawn in their original locations (depending on the layout selected). Figure 3-20 shows one of the examples illustrated earlier displayed in both portrait and landscape mode.

Note that in landscape mode, a lot of empty space on the right of the screen could be used. Furthermore, any additional views at the bottom of the screen would be hidden when the screen orientation is set to landscape.

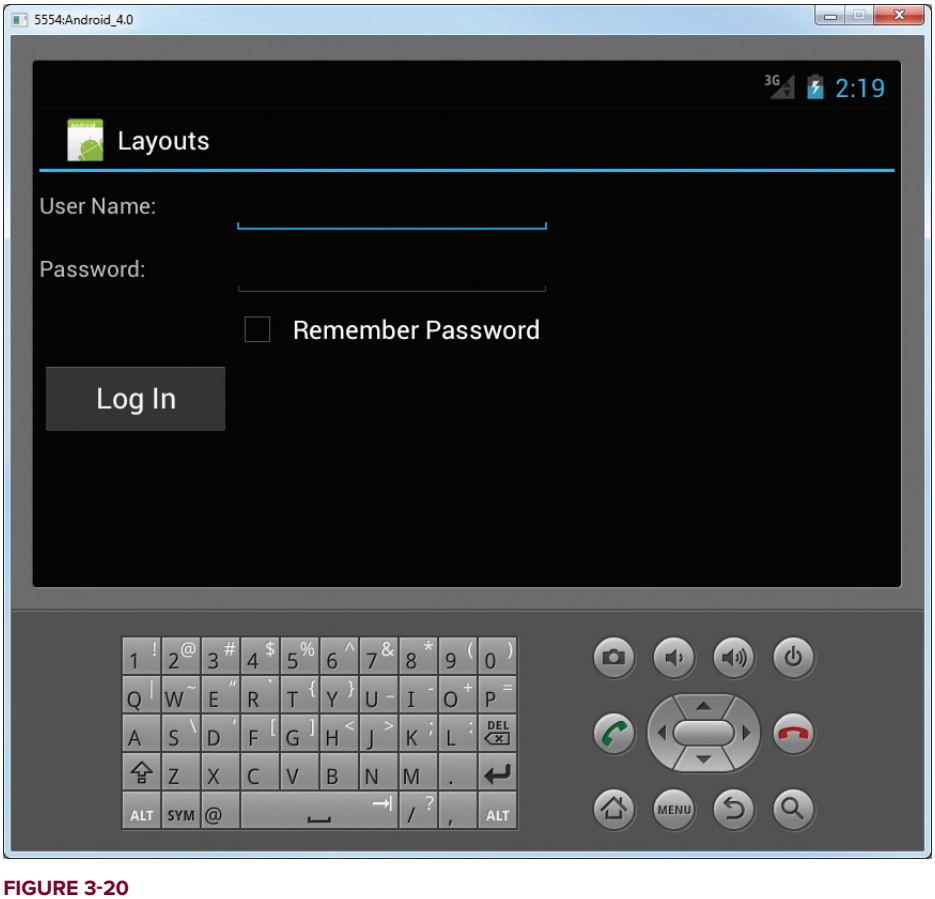


FIGURE 3-20

In general, you can employ two techniques to handle changes in screen orientation:

- **Anchoring** — The easiest way is to "anchor" your views to the four edges of the screen. When the screen orientation changes, the views can anchor neatly to the edges.

- **Resizing and repositioning** — Whereas anchoring and centralizing are simple techniques to ensure that views can handle changes in screen orientation, the ultimate technique is resizing each and every view according to the current screen orientation.

Anchoring Views

Anchoring can be easily achieved by using `RelativeLayout`. Consider the following `main.xml` file, which contains five `Button` views embedded within the `<RelativeLayout>` element:

```
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Right"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true" />
    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Right"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button5"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Middle"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Note the following attributes found in the various `Button` views:

- `layout_alignParentLeft` — Aligns the view to the left of the parent view
- `layout_alignParentRight` — Aligns the view to the right of the parent view
- `layout_alignParentTop` — Aligns the view to the top of the parent view
- `layout_alignParentBottom` — Aligns the view to the bottom of the parent view
- `layout_centerVertical` — Centers the view vertically within its parent view
- `layout_centerHorizontal` — Centers the view horizontally within its parent view

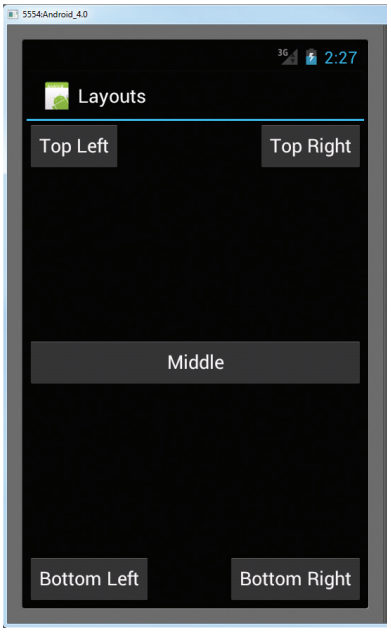


FIGURE 3-21

Figure 3-21 shows the activity when viewed in portrait mode.

When the screen orientation changes to landscape mode, the four buttons are aligned to the four edges of the screen, and the center button is centered in the middle of the screen with its width fully stretched (see Figure 3-22).

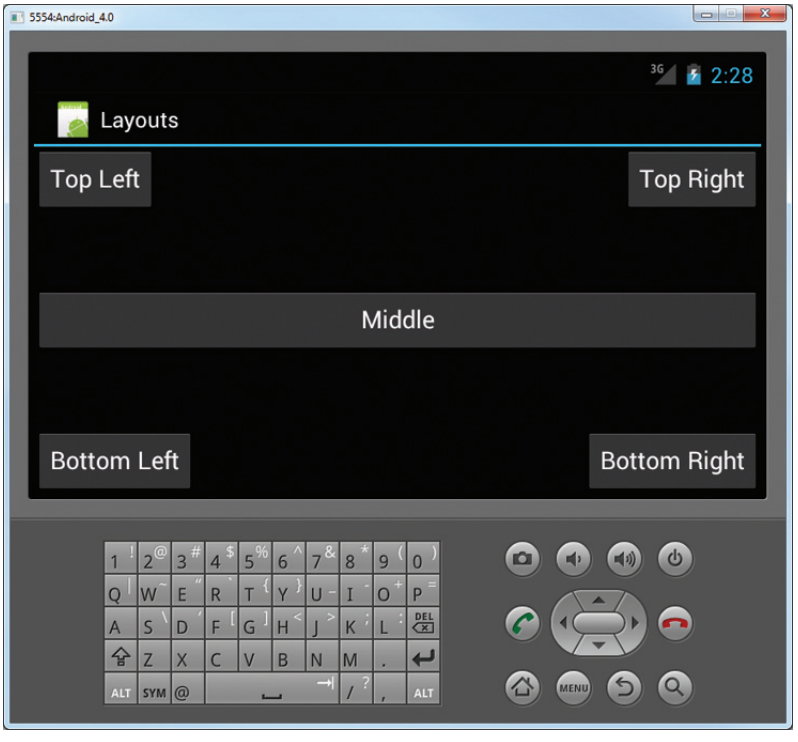


FIGURE 3-22

Resizing and Repositioning

Apart from anchoring your views to the four edges of the screen, an easier way to customize the UI based on screen orientation is to create a separate `res/layout` folder containing the XML files for the UI of each orientation. To support landscape mode, you can create a new folder in the `res` folder and name it as `layout-land` (representing landscape). Figure 3-23 shows the new folder containing the file `main.xml`.

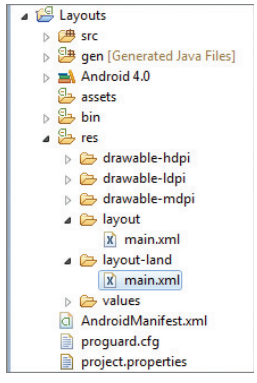


FIGURE 3-23

Basically, the `main.xml` file contained within the `layout` folder defines the UI for the activity in portrait mode, whereas the `main.xml` file in the `layout-land` folder defines the UI in landscape mode.

The following code shows the content of `main.xml` under the `layout` folder:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Right"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true" />
    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Right"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button5"
        android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Middle"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

The following shows the content of `main.xml` under the `layout-land` folder (the statements in bold are the additional views to display in landscape mode):

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Right"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true" />
    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Right"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button5"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Middle"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
    <Button
        android:id="@+id/button6"
        android:layout_width="180px"
```

```
            android:layout_height="wrap_content"
            android:text="Top Middle"
            android:layout_centerVertical="true"
            android:layout_centerHorizontal="true"
            android:layout_alignParentTop="true" />
        <Button
            android:id="@+id/button7"
            android:layout_width="180px"
            android:layout_height="wrap_content"
            android:text="Bottom Middle"
            android:layout_centerVertical="true"
            android:layout_centerHorizontal="true"
            android:layout_alignParentBottom="true" />
    </RelativeLayout>
```

When the activity is loaded in portrait mode, it will display five buttons, as shown in Figure 3-24.

When the activity is loaded in landscape mode, seven buttons are displayed (see Figure 3-25), demonstrating that different XML files are loaded when the device is in different orientations.

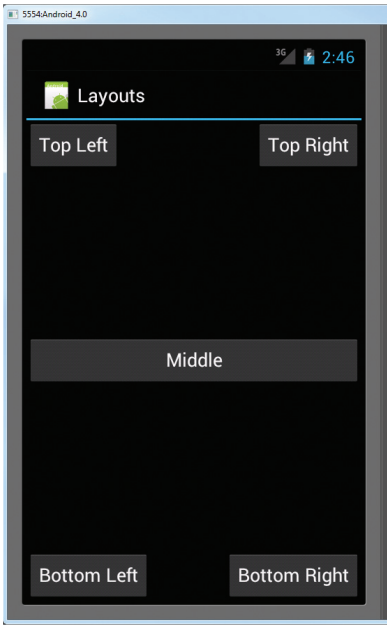


FIGURE 3-24

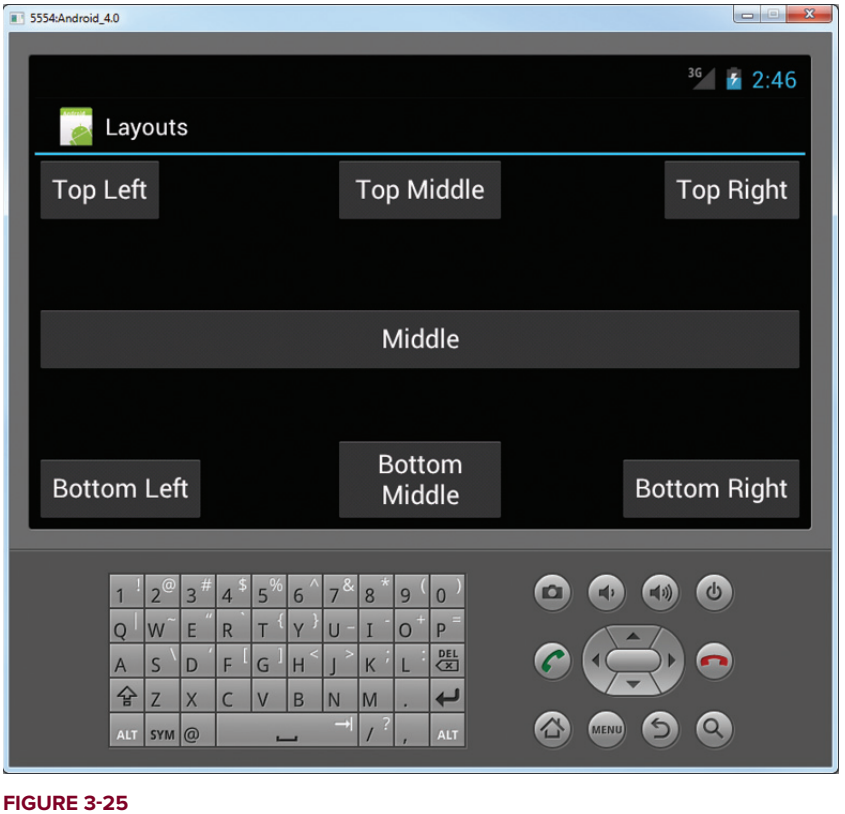


FIGURE 3-25

Using this method, when the orientation of the device changes, Android automatically loads the appropriate XML file for your activity depending on the current screen orientation.

MANAGING CHANGES TO SCREEN ORIENTATION

Now that you have looked at how to implement the two techniques for adapting to screen orientation changes, let's explore what happens to an activity's state when the device changes orientation.

The following Try It Out demonstrates the behavior of an activity when the device changes orientation.

TRY IT OUT Understanding Activity Behavior When Orientation Changes

codefile Orientations.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Orientations**.
2. Add the following statements in bold to the `main.xml` file:

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

<EditText
    android:id="@+id/txtField1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
</LinearLayout>
```

3. Add the following statements in bold to the `OrientationsActivity.java` file:

```
package net.learn2develop.Orientations;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class OrientationsActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Log.d("StateInfo", "onCreate");
}

@Override
public void onStart() {
    Log.d("StateInfo", "onStart");
    super.onStart();
}

@Override
public void onResume() {
    Log.d("StateInfo", "onResume");
    super.onResume();
}

@Override
public void onPause() {
    Log.d("StateInfo", "onPause");
    super.onPause();
}

@Override
public void onStop() {
    Log.d("StateInfo", "onStop");
    super.onStop();
}

@Override
public void onDestroy() {
    Log.d("StateInfo", "onDestroy");
    super.onDestroy();
}

@Override
public void onRestart() {
    Log.d("StateInfo", "onRestart");
    super.onRestart();
}
```

4. Press F11 to debug the application on the Android emulator.
5. Enter some text into the two `EditText` views (see Figure 3-26).

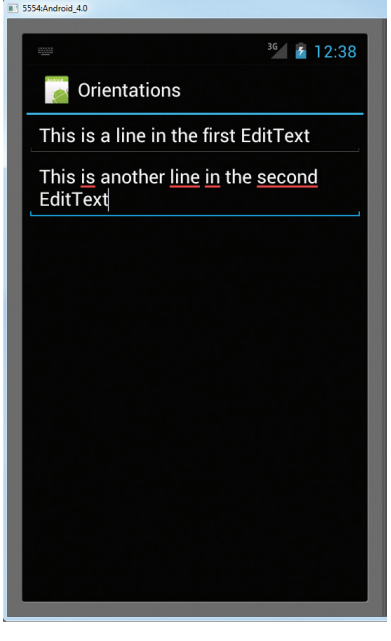


FIGURE 3-26

6. Change the orientation of the Android emulator by pressing Ctrl+F11. Figure 3-27 shows the emulator in landscape mode. Note that the text in the first `EditText` view is still visible, while the second `EditText` view is now empty.

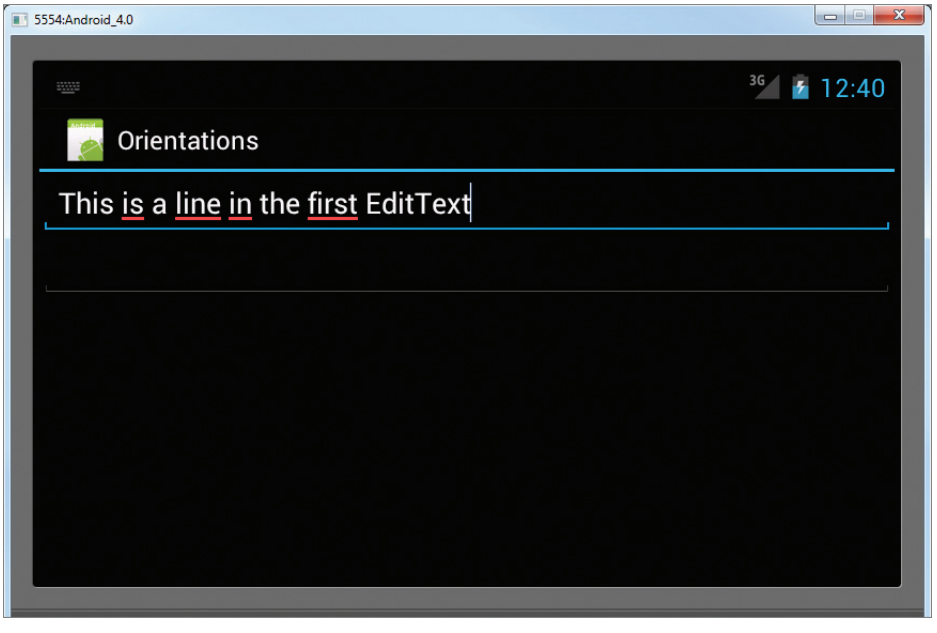


FIGURE 3-27

7. Observe the output in the LogCat window (you need to switch to the Debug perspective in Eclipse). You should see something like this:

```
12-15 12:27:20.747: D/StateInfo(557): onCreate
12-15 12:27:20.747: D/StateInfo(557): onStart
12-15 12:27:20.747: D/StateInfo(557): onResume
...
12-15 12:39:37.846: D/StateInfo(557): onPause
12-15 12:39:37.846: D/StateInfo(557): onStop
12-15 12:39:37.866: D/StateInfo(557): onDestroy
12-15 12:39:38.206: D/StateInfo(557): onCreate
12-15 12:39:38.216: D/StateInfo(557): onStart
12-15 12:39:38.257: D/StateInfo(557): onResume
```

How It Works

From the output shown in the LogCat window, it is apparent that when the device changes orientation, the activity is destroyed:

```
12-15 12:39:37.846: D/StateInfo(557): onPause
12-15 12:39:37.846: D/StateInfo(557): onStop
12-15 12:39:37.866: D/StateInfo(557): onDestroy
```

It is then recreated:

```
12-15 12:39:38.206: D/StateInfo(557): onCreate
12-15 12:39:38.216: D/StateInfo(557): onStart
12-15 12:39:38.257: D/StateInfo(557): onResume
```

It is important to understand this behavior because you need to ensure that you take the necessary steps to preserve the state of your activity before it changes orientation. For example, your activity may have variables that contain values needed for some calculations in the activity. For any activity, you should save whatever state you need to save in the `onPause()` method, which is fired every time the activity changes orientation. The following section demonstrates the different ways to save this state information.

Another important behavior to understand is that only views that are named (via the `android:id` attribute) in an activity will have their state persisted when the activity they are contained in is destroyed. For example, the user may change orientation while entering some text into an `EditText` view. When this happens, any text inside the `EditText` view will be persisted and restored automatically when the activity is recreated. Conversely, if you do not name the `EditText` view using the `android:id` attribute, the activity will not be able to persist the text currently contained within it.

Persisting State Information during Changes in Configuration

So far, you have learned that changing screen orientation destroys an activity and recreates it. Keep in mind that when an activity is recreated, its current state may be lost. When an activity is killed, it will fire one or both of the following two methods:

- `onPause()` — This method is always fired whenever an activity is killed or pushed into the background.
- `onSaveInstanceState()` — This method is also fired whenever an activity is about to be killed or put into the background (just like the `onPause()` method). However, unlike the `onPause()` method, the `onSaveInstanceState()` method is not fired when an activity is being unloaded from the stack (such as when the user pressed the back button), because there is no need to restore its state later.

In short, to preserve the state of an activity, you could always implement the `onPause()` method, and then use your own ways to preserve the state of your activity, such as using a database, internal or external file storage, and so on.

If you simply want to preserve the state of an activity so that it can be restored later when the activity is recreated (such as when the device changes orientation), a much simpler way is to implement the `onSaveInstanceState()` method, as it provides a `Bundle` object as an argument so that you can use it to save your activity's state. The following code shows that you can save the string `ID` into the `Bundle` object during the `onSaveInstanceState()` method:

```
@Override
public void onSaveInstanceState(Bundle outState) {
    //---save whatever you need to persist---
    outState.putString("ID", "1234567890");
    super.onSaveInstanceState(outState);
}
```

When an activity is recreated, the `onCreate()` method is first fired, followed by the `onRestoreInstanceState()` method, which enables you to retrieve the state that you saved previously in the `onSaveInstanceState()` method through the `Bundle` object in its argument:

```
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    //---retrieve the information persisted earlier---
    String ID = savedInstanceState.getString("ID");
}
```

Although you can use the `onSaveInstanceState()` method to save state information, note the limitation that you can only save your state information in a `Bundle` object. If you need to save more complex data structures, then this is not an adequate solution.

Another method that you can use is the `onRetainNonConfigurationInstance()` method. This method is fired when an activity is about to be destroyed due to a *configuration change* (such as a change in screen orientation, keyboard availability, etc.). You can save your current data by returning it in this method, like this:

```
@Override
public Object onRetainNonConfigurationInstance() {
    //---save whatever you want here; it takes in an Object type---
    return("Some text to preserve");
}
```

NOTE *When screen orientation changes, this change is part of what is known as a* configuration change. *A configuration change will cause your current activity to be destroyed.*

Note that this method returns an `Object` type, which allows you to return nearly any data type. To extract the saved data, you can extract it in the `onCreate()` method, using the `getLastNonConfigurationInstance()` method:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Log.d("StateInfo", "onCreate");
    String str = (String) getLastNonConfigurationInstance();
}
```

A good use for using the `onRetainNonConfigurationInstance()` and `getLastNonConfigurationInstance()` methods is when you need to persist some data momentarily, such as when you have downloaded data from a web service and the

user changes the screen orientation. In this scenario, saving the data using the preceding two methods is much more efficient than downloading the data again.

Detecting Orientation Changes

Sometimes you need to know the device's current orientation during runtime. To determine that, you can use the `WindowManager` class. The following code snippet demonstrates how you can programmatically detect the current orientation of your activity:

```
import android.view.Display;
import android.view.WindowManager;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //---get the current display info---
        WindowManager wm = getWindowManager();
        Display d = wm.getDefaultDisplay();

        if (d.getWidth() > d.getHeight()) {
            //---landscape mode---
            Log.d("Orientation", "Landscape mode");
        }
        else {
            //---portrait mode---
            Log.d("Orientation", "Portrait mode");
        }
    }
```

The `getDefaultDisplay()` method returns a `Display` object representing the screen of the device. You can then get its width and height and deduce the current orientation.

Controlling the Orientation of the Activity

Occasionally, you might want to ensure that your application is displayed in only a certain orientation. For example, you may be writing a game that should be viewed only in landscape mode. In this case, you can programmatically force a change in orientation using the `setRequestOrientation()` method of the `Activity` class:

```
import android.content.pm.ActivityInfo;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //---change to landscape mode---
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
    }
```

To change to portrait mode, use the `ActivityInfo.SCREEN_ORIENTATION_PORTRAIT` constant.

Besides using the `setRequestOrientation()` method, you can also use the `android:screenOrientation` attribute on the `<activity>` element in `AndroidManifest.xml` as follows to constrain the activity to a certain orientation:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.Orientations"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".OrientationsActivity"
            android:screenOrientation="landscape" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

The preceding example constrains the activity to a certain orientation (landscape in this case) and prevents the activity from being destroyed; that is, the activity will not be destroyed and the `onCreate()` method will not be fired again when the orientation of the device changes.

Following are two other values that you can specify in the `android:screenOrientation` attribute:

- `portrait` — Portrait mode
- `sensor` — Based on the accelerometer (default)

UTILIZING THE ACTION BAR

Besides fragments, another newer feature introduced in Android 3 and 4 is the Action Bar. In place of the traditional title bar located at the top of the device's screen, the Action Bar displays the application icon together with the activity title. Optionally, on the right side of the Action Bar are *action items*. Figure 3-28 shows the built-in Email application displaying the application icon, the activity title, and some action items in the Action Bar. The next section discusses action items in more detail.

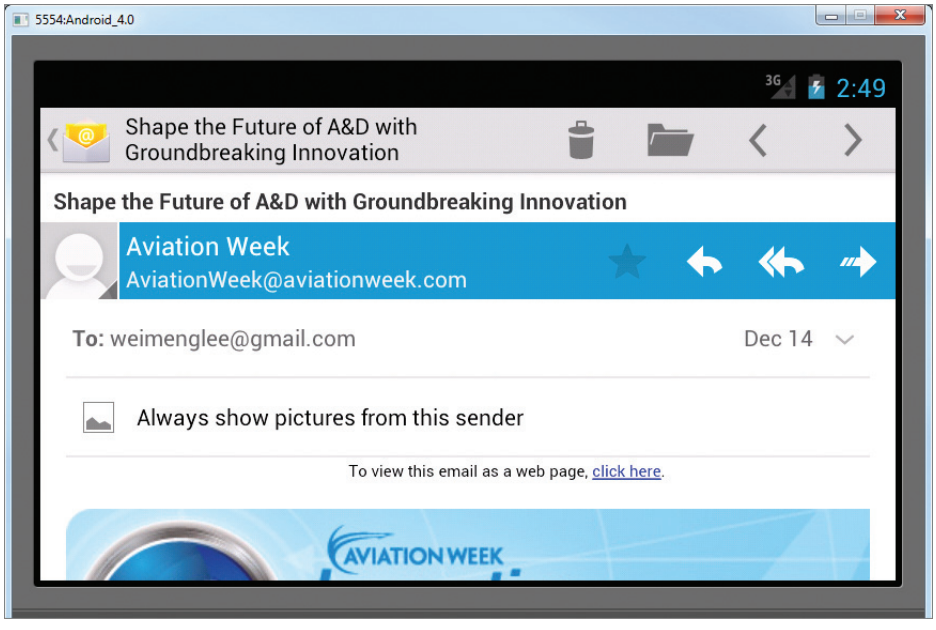


FIGURE 3-28

The following Try It Out shows how you can programmatically hide or display the Action Bar.

TRY IT OUT Showing and Hiding the Action Bar

1. Using Eclipse, create a new Android project and name it **MyActionBar**.
2. Press F11 to debug the application on the Android emulator. You should see the application and its Action Bar located at the top of the screen (containing the application icon and the application name "MyActionBar"; see Figure 3-29).

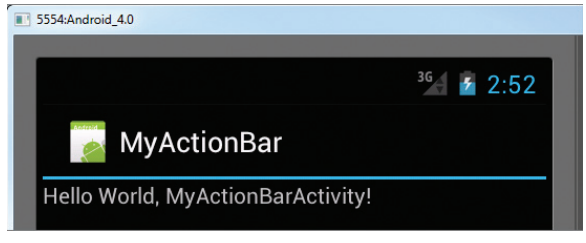


FIGURE 3-29

3. To hide the Action Bar, add the following line in bold to the `AndroidManifest.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.MyActionBar"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
```

```
        <activity
            android:label="@string/app_name"
            android:name=".MyActionBarActivity"
            android:theme="@android:style/Theme.Holo.NoActionBar" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

4. Select the project name in Eclipse and then press F11 to debug the application on the Android emulator again. This time, the Action Bar is not displayed (see Figure 3-30).

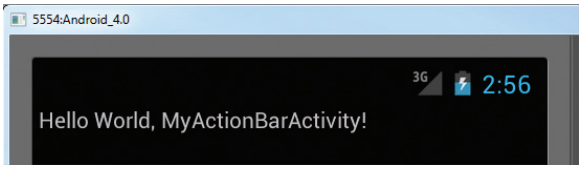


FIGURE 3-30

5. You can also programmatically remove the Action Bar using the `ActionBar` class. To do so, you first need to remove the `android:theme` attribute you added in the previous step. This is important, otherwise the next step will cause the application to raise an exception.

6. Modify the `MyActionBarActivity.java` file as follows:

```
package net.learn2develop.MyActionBar;

import android.app.ActionBar;
import android.app.Activity;
import android.os.Bundle;

public class MyActionBarActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        ActionBar actionBar = getActionBar();
        actionBar.hide();
        //actionBar.show(); //---show it again---
    }
}
```

7. Press F11 to debug the application on the emulator again. The Action Bar remains hidden.

How It Works

The `android:theme` attribute enables you to turn off the display of the Action Bar for your activity. Setting this attribute to "`@android:style/Theme.Holo.NoActionBar`" hides the Action Bar. Alternatively, you can programmatically get a reference to the Action Bar during runtime by using the `getActionBar()` method. Calling the `hide()` method hides the Action Bar, and calling the `show()` method displays it.

Note that if you use the `android:theme` attribute to turn off the Action Bar, calling the `getActionBar()` method returns a `null` during runtime. Hence, it is always better to turn the Action Bar on/off programmatically using the `ActionBar` class.

Adding Action Items to the Action Bar

Besides displaying the application icon and the activity title on the left of the Action Bar, you can also display additional items on the Action Bar. These additional items are called *action items*. Action items are shortcuts to some of the commonly performed operations in your application. For example, you might be building an RSS reader application, in which case some of the action items might be "Refresh feed," "Delete feed" and "Add new feed."

The following Try It Out shows how you can add action items to the Action Bar.

TRY IT OUT Adding Action Items

1. Using the MyActionBar project created in the previous section, add the following code in bold to the `MyActionBarActivity.java` file:

```
package net.learn2develop.MyActionBar;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class MyActionBarActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //ActionBar actionBar = getActionBar();
        //actionBar.hide();
        //actionBar.show(); //---show it again---
    }

    @Override
    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.setIcon(R.drawable.ic_launcher);
            mnu1.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        }
        MenuItem mnu2 = menu.add(0, 1, 1, "Item 2");
        {
            mnu2.setIcon(R.drawable.ic_launcher);
            mnu2.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        }
        MenuItem mnu3 = menu.add(0, 2, 2, "Item 3");
        {
            mnu3.setIcon(R.drawable.ic_launcher);
            mnu3.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        }
        MenuItem mnu4 = menu.add(0, 3, 3, "Item 4");
        {
            mnu4.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        }
        MenuItem mnu5 = menu.add(0, 4, 4, "Item 5");
        {
            mnu5.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        }
    }

    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:
            Toast.makeText(this, "You clicked on Item 2",
                Toast.LENGTH_LONG).show();
            return true;
        case 2:
            Toast.makeText(this, "You clicked on Item 3",
                Toast.LENGTH_LONG).show();
            return true;
        case 3:
            Toast.makeText(this, "You clicked on Item 4",
                Toast.LENGTH_LONG).show();
            return true;
        case 4:
            Toast.makeText(this, "You clicked on Item 5",
                Toast.LENGTH_LONG).show();
            return true;
        }
        return false;
    }

}
```

2. Press F11 to debug the application on the Android emulator. Observe the icons on the right side of the Action Bar (see Figure 3-31). If you click the MENU button on the emulator, you will see the rest of the menu items (see Figure 3-32). This is known as the *overflow menu*. On devices that do not have the MENU button, an overflow menu is represented by an icon with an arrow. Figure 3-33 shows the same application running on the Asus Eee Pad Transformer (Android 3.2.1). Clicking the overflow menu displays the rest of the menu items.

3. Clicking each menu item will cause the `Toast` class to display the name of the menu item selected (see Figure 3-34).

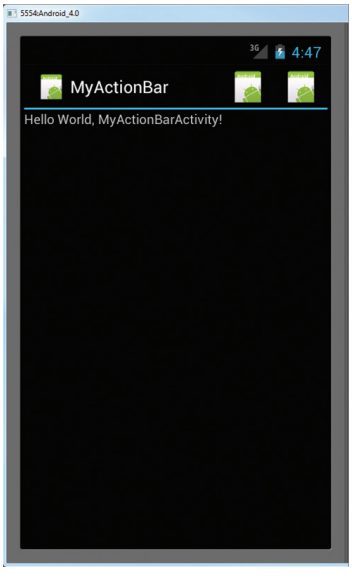


FIGURE 3-31

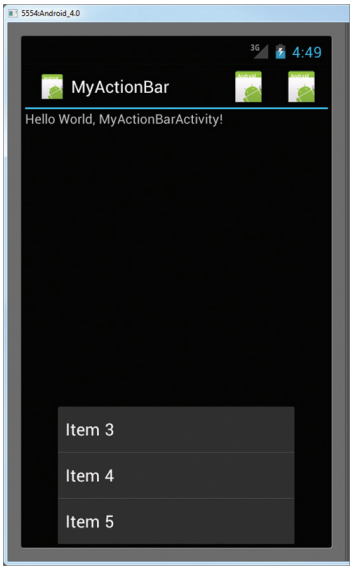


FIGURE 3-32

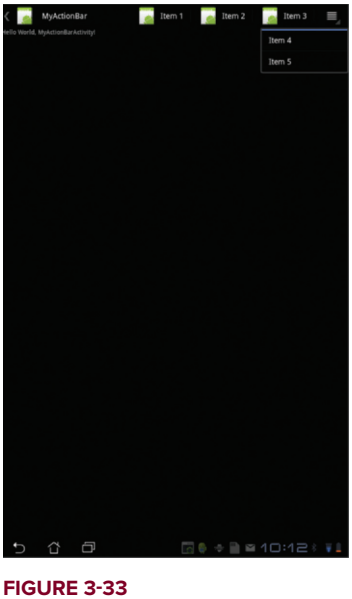


FIGURE 3-33

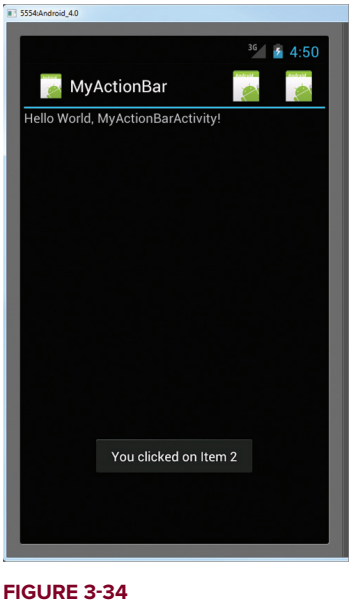


FIGURE 3-34

4. Press Control-F11 to change the display orientation of the emulator to landscape mode. You will now see four action items on the Action Bar, as shown in Figure 3-35, three with icons and one with text.

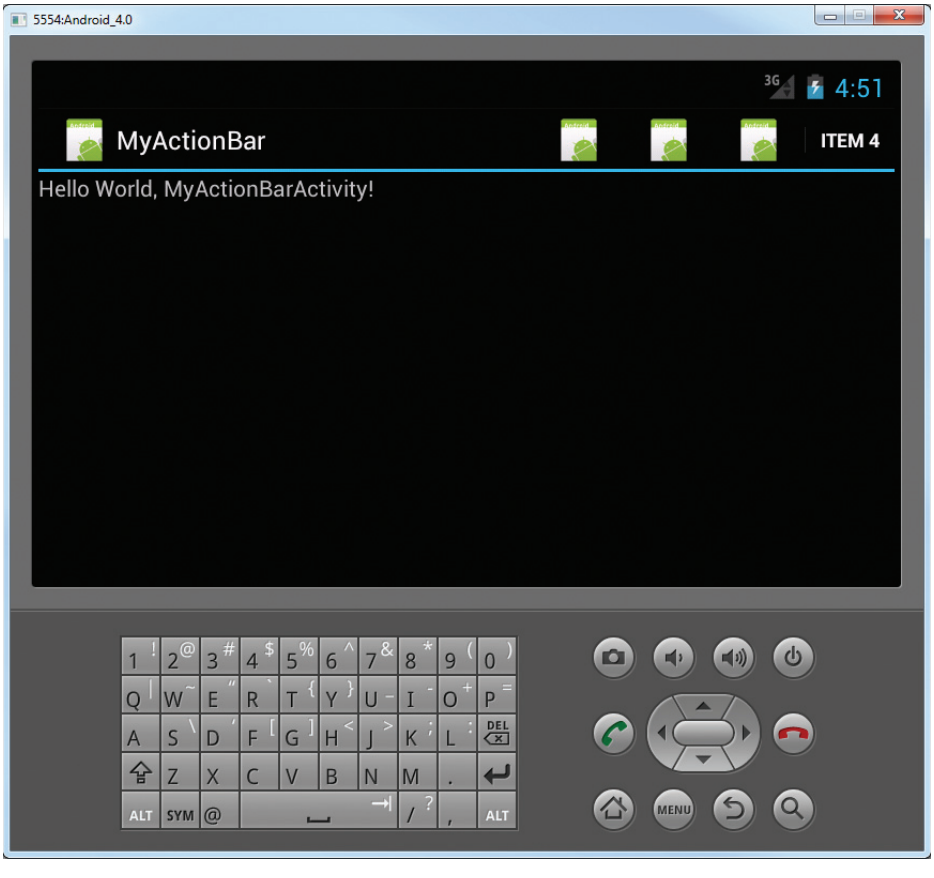


FIGURE 3-35

How It Works

The Action Bar populates its action items by calling the `onCreateOptionsMenu()` method of an activity:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    CreateMenu(menu);
    return true;
}
```

In the preceding example, you called the `CreateMenu()` method to display a list of menu items:

```
private void CreateMenu(Menu menu)
{
    MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");
```

```
        {
            mnu1.setIcon(R.drawable.ic_launcher);
            mnu1.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        }
        MenuItem mnu2 = menu.add(0, 1, 1, "Item 2");
        {
            mnu2.setIcon(R.drawable.ic_launcher);
            mnu2.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        }
        MenuItem mnu3 = menu.add(0, 2, 2, "Item 3");
        {
            mnu3.setIcon(R.drawable.ic_launcher);
            mnu3.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        }
        MenuItem mnu4 = menu.add(0, 3, 3, "Item 4");
        {
            mnu4.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        }
        MenuItem mnu5 = menu.add(0, 4, 4, "Item 5");
        {
            mnu5.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        }
    }
```

To make the menu item appear as an action item, you call its `setShowAsAction()` method using the `SHOW_AS_ACTION_IF_ROOM` constant. This tells the Android device to display the menu item as an action item if there is room for it.

When a menu item is selected by the user, the `onOptionsItemSelected()` method is called:

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
     return MenuChoice(item);
}
```

Here, you called the self-defined `MenuChoice()` method to check which menu item was clicked, and then printed out a message:

```
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:
        Toast.makeText(this, "You clicked on Item 2",
            Toast.LENGTH_LONG).show();
        return true;
    case 2:
        Toast.makeText(this, "You clicked on Item 3",
            Toast.LENGTH_LONG).show();
        return true;
```

```
            case 3:
                Toast.makeText(this, "You clicked on Item 4",
                    Toast.LENGTH_LONG).show();
                return true;
            case 4:
                Toast.makeText(this, "You clicked on Item 5",
                    Toast.LENGTH_LONG).show();
                return true;
            }
            return false;
        }
```

Customizing the Action Items and Application Icon

In the previous example, the menu items are displayed without the text. If you want to display the text for the action item together with the icon, you could use the "|" operator together with the `MenuItem.SHOW_AS_ACTION_WITH_TEXT` constant:

```
MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");
{
    mnu1.setIcon(R.drawable.ic_launcher);
    mnu1.setShowAsAction(
        MenuItem.SHOW_AS_ACTION_IF_ROOM |
        MenuItem.SHOW_AS_ACTION_WITH_TEXT);
}
```

This causes the icon to be displayed together with the text of the menu item (see Figure 3-36).

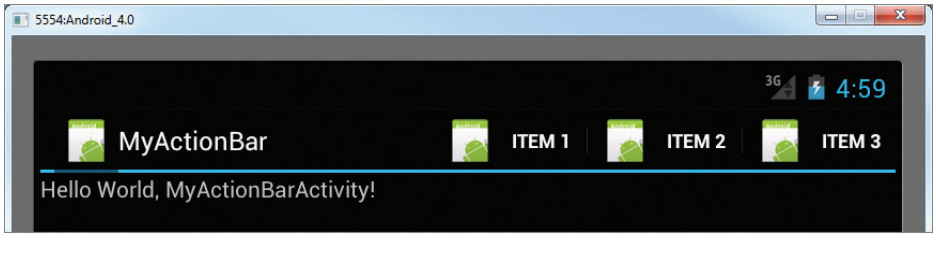


FIGURE 3-36

Besides clicking the action items, users can also click the application icon on the Action Bar. When the application icon is clicked, the `onOptionsItemSelected()` method is called. To identify the application icon being called, you check the item ID against the `android.R.id.home` constant:

```
private boolean MenuChoice(MenuItem item)
{
    switch (item.getItemId()) {
    case  android.R.id.home:
        Toast.makeText(this,
            "You clicked on the Application icon",
            Toast.LENGTH_LONG).show();
```

```
            return true;

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

To make the application icon clickable, you need to call the `setDisplayHomeAsUpEnabled()` method:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    ActionBar actionBar = getActionBar();
    actionBar.setDisplayHomeAsUpEnabled(true);
    //actionBar.hide();
    //actionBar.show(); //---show it again---
}
```

Figure 3-37 shows the arrow button displayed next to the application icon.

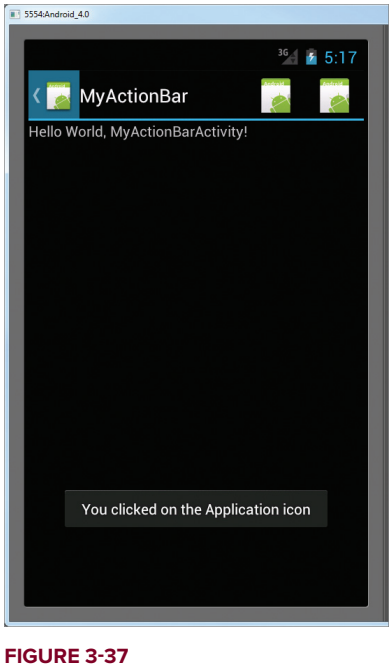


FIGURE 3-37

The application icon is often used by applications to enable them to return to the main activity of the application. For example, your application may have several activities, and you can use the application icon as a shortcut for users to return directly to the main activity of your application. To do this, it is always a good practice to create an `Intent` object and set it using the `Intent.FLAG_ACTIVITY_CLEAR_TOP` flag:

```
case  android.R.id.home:
    Toast.makeText(this,
        "You clicked on the Application icon",
        Toast.LENGTH_LONG).show();

    Intent i = new Intent(this, MyActionBarActivity.class);
    i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(i);

    return true;
```

The `Intent.FLAG_ACTIVITY_CLEAR_TOP` flag ensures that the series of activities in the back stack is cleared when the user clicks the application icon on the Action Bar. This way, if the user clicks the back button, the other activities in the application do not appear again.

CREATING THE USER INTERFACE PROGRAMMATICALLY

So far, all the UIs you have seen in this chapter are created using XML. As mentioned earlier, besides using XML you can also create the UI using code. This approach is useful if your UI needs to be dynamically generated during runtime. For example, suppose you are building a cinema ticket reservation system and your application will display the seats of each cinema using buttons. In this case, you would need to dynamically generate the UI based on the cinema selected by the user.

The following Try It Out demonstrates the code needed to dynamically build the UI in your activity.

TRY IT OUT Creating the UI via Code

codefile UICode.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **UICode**.
2. In the `UICodeActivity.java` file, add the following statements in bold:

```
package net.learn2develop.UICode;

import android.app.Activity;
import android.os.Bundle;
import android.view.ViewGroup.LayoutParams;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;

public class UICodeActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.main);
        //---param for views---
        LayoutParams params =
            new LinearLayout.LayoutParams(
                LayoutParams.FILL_PARENT,
                LayoutParams.WRAP_CONTENT);

        //---create a layout---
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);

        //---create a textview---
        TextView tv = new TextView(this);
        tv.setText("This is a TextView");
        tv.setLayoutParams(params);

        //---create a button---
        Button btn = new Button(this);
        btn.setText("This is a Button");
        btn.setLayoutParams(params);

        //---adds the textview---
        layout.addView(tv);

        //---adds the button---
        layout.addView(btn);

        //---create a layout param for the layout---
        LinearLayout.LayoutParams layoutParam =
            new LinearLayout.LayoutParams(
                    LayoutParams.FILL_PARENT,
                    LayoutParams.WRAP_CONTENT );

        this.addContentView(layout, layoutParam);
    }
}
```

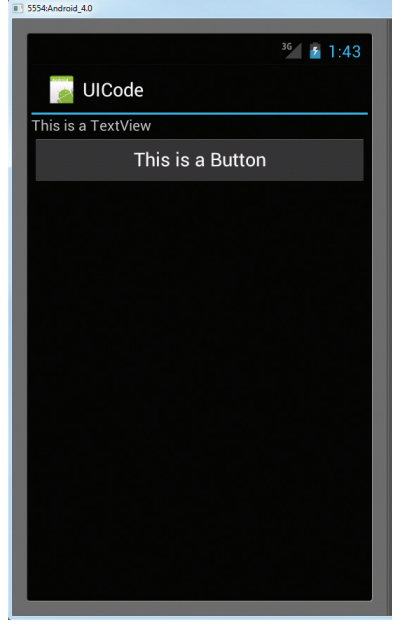


FIGURE 3-38

3. Press F11 to debug the application on the Android emulator. Figure 3-38 shows the activity created.

How It Works

In this example, you first commented out the `setContentView()` statement so that it does not load the UI from the `main.xml` file.

You then created a `LayoutParams` object to specify the layout parameter that can be used by other views (which you will create next):

```
        //---param for views---
        LayoutParams params =
            new LinearLayout.LayoutParams(
                LayoutParams.FILL_PARENT,
                LayoutParams.WRAP_CONTENT);
```

You also created a `LinearLayout` object to contain all the views in your activity:

```
//---create a layout---
LinearLayout layout = new LinearLayout(this);
layout.setOrientation(LinearLayout.VERTICAL);
```

Next, you created a `TextView` and a `Button` view:

```
//---create a textview---
TextView tv = new TextView(this);
tv.setText("This is a TextView");
tv.setLayoutParams(params);

//---create a button---
Button btn = new Button(this);
btn.setText("This is a Button");
btn.setLayoutParams(params);
```

You then added them to the `LinearLayout` object:

```
//---adds the textview---
layout.addView(tv);

//---adds the button---
layout.addView(btn);
```

You also created a `LayoutParams` object to be used by the `LinearLayout` object:

```
//---create a layout param for the layout---
LinearLayout.LayoutParams layoutParam =
    new LinearLayout.LayoutParams(
            LayoutParams.FILL_PARENT,
            LayoutParams.WRAP_CONTENT );
```

Finally, you added the `LinearLayout` object to the activity:

```
this.addContentView(layout, layoutParam);
```

As you can see, using code to create the UI is quite a laborious affair. Hence, dynamically generate your UI using code only when necessary.

LISTENING FOR UI NOTIFICATIONS

Users interact with your UI at two levels: the activity level and the view level. At the activity level, the `Activity` class exposes methods that you can override. Some common methods that you can override in your activities include the following:

- `onKeyDown` — Called when a key was pressed and not handled by any of the views contained within the activity

- `onKeyUp` — Called when a key was released and not handled by any of the views contained within the activity
- `onMenuItemSelected` — Called when a panel's menu item has been selected by the user (covered in Chapter 5)
- `onMenuOpened` — Called when a panel's menu is opened by the user (covered in Chapter 5)

Overriding Methods Defined in an Activity

To demonstrate how activities interact with the user, the following example overrides some of the methods defined in the activity's base class.

TRY IT OUT Overriding Activity Methods

codefile UIActivity.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **UIActivity**.
2. Add the following statements in bold to `main.xml` (replacing the `TextView`):

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

    <TextView
        android:layout_width="214dp"
        android:layout_height="wrap_content"
        android:text="Your Name" />
    <EditText
        android:id="@+id/txt1"
        android:layout_width="214dp"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/btn1"
        android:layout_width="106dp"
        android:layout_height="wrap_content"
        android:text="OK" />
    <Button
        android:id="@+id/btn2"
        android:layout_width="106dp"
        android:layout_height="wrap_content"
        android:text="Cancel" />

</LinearLayout>
```

3. Add the following statements in bold to the `UIActivityActivity.java` file:

```
package net.learn2develop.UIActivity;

import android.app.Activity;
import android.os.Bundle;
```

```
import android.view.KeyEvent;
import android.widget.Toast;

public class UIActivityActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        switch (keyCode)
        {
            case KeyEvent.KEYCODE_DPAD_CENTER:
                Toast.makeText(getBaseContext(),
                        "Center was clicked",
                        Toast.LENGTH_LONG).show();
                break;
            case KeyEvent.KEYCODE_DPAD_LEFT:
                Toast.makeText(getBaseContext(),
                        "Left arrow was clicked",
                        Toast.LENGTH_LONG).show();
                break;
            case KeyEvent.KEYCODE_DPAD_RIGHT:
                Toast.makeText(getBaseContext(),
                        "Right arrow was clicked",
                        Toast.LENGTH_LONG).show();
                break;
            case KeyEvent.KEYCODE_DPAD_UP:
                Toast.makeText(getBaseContext(),
                        "Up arrow was clicked",
                        Toast.LENGTH_LONG).show();

                break;
            case KeyEvent.KEYCODE_DPAD_DOWN:
                Toast.makeText(getBaseContext(),
                        "Down arrow was clicked",
                        Toast.LENGTH_LONG).show();
                break;
        }
        return false;
    }
```

4. Press F11 to debug the application on the Android emulator.
5. When the activity is loaded, type some text into the `EditText`. Next, click the down arrow key on the directional pad. Observe the message shown on the screen, as shown in Figure 3-39.

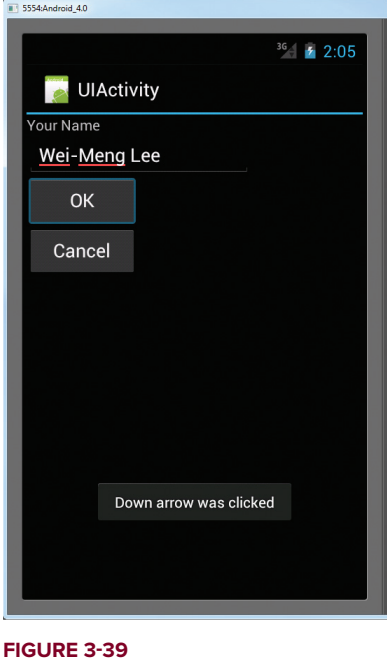


FIGURE 3-39

How It Works

When the activity is loaded, the cursor will be blinking in the `EditText` view, as it has the focus.

In the `MainActivitiy` class, you override the `onKeyDown()` method of the base `Activity` class:

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event)
{
    switch (keyCode)
    {
        case KeyEvent.KEYCODE_DPAD_CENTER:
            //...
            break;
        case KeyEvent.KEYCODE_DPAD_LEFT:
            //...
            break;
        case KeyEvent.KEYCODE_DPAD_RIGHT:
            //...
            break;
        case KeyEvent.KEYCODE_DPAD_UP:
            //...
            break;
        case KeyEvent.KEYCODE_DPAD_DOWN:
            //...
            break;
    }
    return false;
}
```

In Android, whenever you press any keys on your device, the view that currently has the focus will try to handle the event generated. In this case, when the `EditText` has the focus and you press a key, the `EditText` view handles the event and displays the character you have just pressed in the view. However, if you press the up or down directional arrow key, the `EditText` view does not handle this, and instead passes the event to the activity. In this case, the `onKeyDown()` method is called. In this example, you checked the key that was pressed and displayed a message indicating the key pressed. Observe that the focus is now also transferred to the next view, which is the OK button.

Interestingly, if the `EditText` view already has some text in it and the cursor is at the end of the text, then clicking the left arrow key does not fire the `onKeyDown()` method; it simply moves the cursor one character to the left. This is because the `EditText` view has already handled the event. If you press the right arrow key instead (when the cursor is at the end of the text), then the `onKeyDown()` method will be called (because now the `EditText` view will not be handling the event). The same applies when the cursor is at the beginning of the `EditText` view. Clicking the left arrow will fire the `onKeyDown()` method, whereas clicking the right arrow will simply move the cursor one character to the right.

With the OK button in focus, press the center button in the directional pad. Note that the message "Center was clicked" is not displayed. This is because the `Button` view itself is handling the click event. Hence, the event is not caught by the `onKeyDown()` method.

Note also that the `onKeyDown()` method returns a `boolean` result. You should return `true` when you want to tell the system that you are done with the event and that the system should not proceed further with it. For example, consider the case when you return `true` after each key has been matched:

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event)
{
    switch (keyCode)
    {
        case KeyEvent.KEYCODE_DPAD_CENTER:
            Toast.makeText(getBaseContext(),
                    "Center was clicked",
                    Toast.LENGTH_LONG).show();
            //break;
            return true;
        case KeyEvent.KEYCODE_DPAD_LEFT:
            Toast.makeText(getBaseContext(),
                    "Left arrow was clicked",
                    Toast.LENGTH_LONG).show();
            //break;
            return true;
        case KeyEvent.KEYCODE_DPAD_RIGHT:
            Toast.makeText(getBaseContext(),
                    "Right arrow was clicked",
                    Toast.LENGTH_LONG).show();
            //break;
            return true;
        case KeyEvent.KEYCODE_DPAD_UP:
            Toast.makeText(getBaseContext(),
                    "Up arrow was clicked",
                    Toast.LENGTH_LONG).show();

            //break;
            return true;
        case KeyEvent.KEYCODE_DPAD_DOWN:
            Toast.makeText(getBaseContext(),
                    "Down arrow was clicked",
                    Toast.LENGTH_LONG).show();
            //break;
            return true;
    }
    return false;
}
```

If you test this, you will see that now you cannot navigate between the views using the arrow keys.

Registering Events for Views

Views can fire events when users interact with them. For example, when a user touches a `Button` view, you need to service the event so that the appropriate action can be performed. To do so, you need to explicitly register events for views.

Using the same example discussed in the previous section, recall that the activity has two `Button` views; therefore, you can register the button click events using an anonymous class, as shown here:

```
package net.learn2develop.UIActivity;

import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class UIActivityActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //---the two buttons are wired to the same event handler---
        Button btn1 = (Button)findViewById(R.id.btn1);
        btn1.setOnClickListener(btnListener);

        Button btn2 = (Button)findViewById(R.id.btn2);
        btn2.setOnClickListener(btnListener);
    }

    //---create an anonymous class to act as a button click listener---
    private OnClickListener btnListener = new OnClickListener()
    {
        public void onClick(View v)
        {
            Toast.makeText(getBaseContext(),
                    ((Button) v).getText() + " was clicked",
                    Toast.LENGTH_LONG).show();
        }
    };

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        //...
    }
}
```

If you now click either the OK button or the Cancel button, the appropriate message will be displayed (see Figure 3-40), proving that the event is wired up properly.

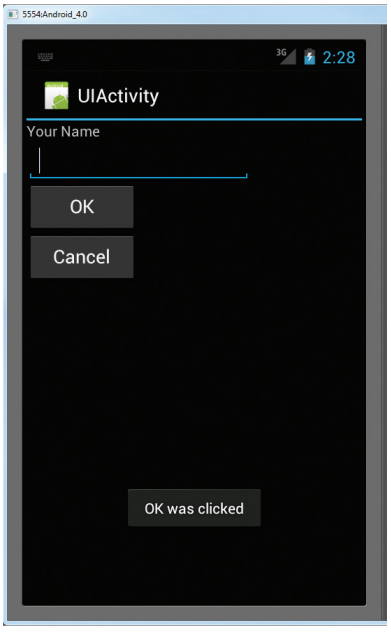


FIGURE 3-40

Besides defining an anonymous class for the event handler, you can also define an anonymous inner class to handle an event. The following example shows how you can handle the `onFocusChange()` method for the `EditText` view:

```
import android.widget.EditText;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //---the two buttons are wired to the same event handler---
        Button btn1 = (Button)findViewById(R.id.btn1);
        btn1.setOnClickListener(btnListener);

        Button btn2 = (Button)findViewById(R.id.btn2);
        btn2.setOnClickListener(btnListener);

        //---create an anonymous inner class to act as an onfocus listener---
        EditText txt1 = (EditText)findViewById(R.id.txt1);
        txt1.setOnFocusChangeListener(new View.OnFocusChangeListener()
        {
            @Override
            public void onFocusChange(View v, boolean hasFocus) {
                Toast.makeText(getBaseContext(),
```

```
                    ((EditText) v).getId() + " has focus - " + hasFocus,
                    Toast.LENGTH_LONG).show();
            }
        });
    }
```

NOTE *For this example, you should ensure that the* `onKeyDown()` *method returns a* `false`*, instead of* `true` *as you have tried in the previous section.*

As shown in Figure 3-41, when the `EditText` view receives the focus, a message is printed on the screen.

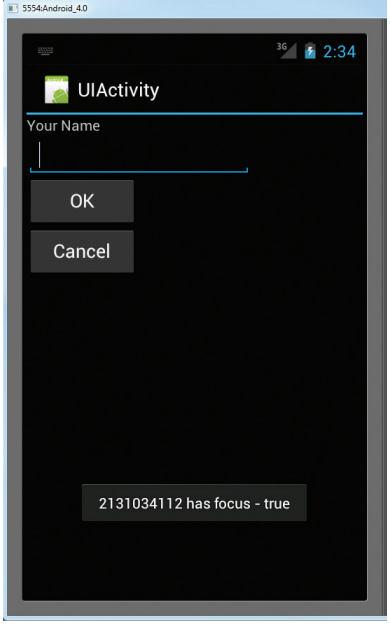


FIGURE 3-41

Using the anonymous inner class, the click event handler for the two `Button`s can also be rewritten as follows:

```
//---the two buttons are wired to the same event handler---
Button btn1 = (Button)findViewById(R.id.btn1);
//btn1.setOnClickListener(btnListener);
btn1.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        //---do something---
    }
});

Button btn2 = (Button)findViewById(R.id.btn2);
```

```
//btn2.setOnClickListener(btnListener);
btn2.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        //---do something---
    }
});
```

Which method should you use to handle events? An anonymous class is useful if you have multiple views handled by one event handler. The anonymous inner class method (the latter method described) is useful if you have an event handler for a single view.

SUMMARY

In this chapter, you have learned how user interfaces are created in Android. You have also learned about the different layouts that you can use to position the views in your Android UI. Because Android devices support more than one screen orientation, you need to take special care to ensure that your UI can adapt to changes in screen orientation.

EXERCISES

1. What is the difference between the `dp` unit and the `px` unit? Which one should you use to specify the dimension of a view?
2. Why is the `AbsoluteLayout` not recommended for use?
3. What is the difference between the `onPause()` method and the `onSaveInstanceState()` method?
4. Name the three methods you can override to save an activity's state. In what instances should you use the various methods?
5. How do you add action items to the Action Bar?

Answers to the exercises can be found in Appendix C.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
`LinearLayout`	Arranges views in a single column or single row
`AbsoluteLayout`	Enables you to specify the exact location of its children
`TableLayout`	Groups views into rows and columns
`RelativeLayout`	Enables you to specify how child views are positioned relative to each other
`FrameLayout`	A placeholder on screen that you can use to display a single view
`ScrollView`	A special type of `FrameLayout` in that it enables users to scroll through a list of views that occupy more space than the physical display allows
Unit of Measure	Use `dp` for specifying the dimension of views and `sp` for font size.
Two ways to adapt to changes in orientation	Anchoring, and resizing and repositioning
Using different XML files for different orientations	Use the `layout` folder for portrait UI, and `layout-land` for landscape UI.
Three ways to persist activity state	Use the `onPause()` method. Use the `onSaveInstanceState()` method. Use the `onRetainNonConfigurationInstance()` method.
Getting the dimension of the current device	Use the `WindowManager` class's `getDefaultDisplay()` method.
Constraining the activity's orientation	Use the `setRequestOrientation()` method, or the `android:screenOrientation` attribute in the `AndroidManifest.xml` file.
Action Bar	Replaces the traditional title bar for older versions of Android
Action items	Action items are displayed on the right of the Action Bar. They are created just like options menus.
Application icon	Usually used to return to the "home" activity of an application. It is advisable to use the `Intent` object with the `Intent.FLAG_ACTIVITY_CLEAR_TOP` flag.

4

Designing Your User Interface With Views

WHAT YOU WILL LEARN IN THIS CHAPTER

- How to use the basic views in Android to design your user interface
- How to use the picker views to display lists of items
- How to use the list views to display lists of items
- How to use specialized fragments

In the previous chapter, you learned about the various layouts that you can use to position your views in an activity. You also learned about the techniques you can use to adapt to different screen resolutions and sizes. In this chapter, you will take a look at the various views that you can use to design the user interface for your applications.

In particular, you will learn about the following ViewGroups:

- 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
- **Specialized fragments** — Special fragments that perform specific functions

Subsequent chapters cover the other views not covered in this chapter, such as the analog and digital clock views and other views for displaying graphics, and so on.

USING BASIC VIEWS

To get started, this section explores some of 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. The following sections explore all these views in more detail.

TextView View

When you create a new Android project, Eclipse always creates the `main.xml` file (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:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <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. If you need to allow users to edit the text displayed, you should use the subclass of `TextView`, `EditText`, which is discussed in the next section.

NOTE *In some other platforms, the* `TextView` *is commonly known as the* label view. *Its sole purpose is to display text on the screen.*

Button, ImageButton, EditText, CheckBox, ToggleButton, RadioButton, and RadioGroup Views

Besides the `TextView` view, which you will likely use the most often, there are some other basic views that you will find yourself frequently using:

- `Button` — Represents a push-button widget
- `ImageButton` — Similar to the `Button` view, except that it also displays an image
- `EditText` — A subclass of the `TextView` view that allows users to edit its text content
- `CheckBox` — A special type of button that has two states: checked or unchecked
- `RadioGroup` and `RadioButton` — The `RadioButton` has two states: either checked or unchecked.. A `RadioGroup` is used to group together one or more `RadioButton` views, thereby allowing only one `RadioButton` to be checked within the `RadioGroup`.
- `ToggleButton` — Displays checked/unchecked states using a light indicator

The following Try It Out provides details about how these views work.

TRY IT OUT Using the Basic Views

codefile BasicViews1.zip available for download at Wrox.com

1. Using Eclipse, create an Android project and name it **BasicViews1.**
2. Modify the `main.xml` file located in the `res/layout` folder by adding the following elements shown in bold:

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

<Button android:id="@+id/btnSave"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="save" />

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

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

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

```
    android:layout_height="wrap_content" />

<CheckBox android:id="@+id/chkAutosave"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Autosave" />

<CheckBox android:id="@+id/star"
    style="?android:attr/starStyle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

<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 android:id="@+id/toggle1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

</LinearLayout>
```

3. To see the views in action, debug the project in Eclipse by selecting the project name and pressing F11. Figure 4-1 shows the various views displayed in the Android emulator.

4. Click the various views and note how they vary in their look and feel. Figure 4-2 shows the following changes to the view:

   - The first `CheckBox` view (Autosave) is checked.
   - The second `CheckBox` View (star) is selected.
   - The second `RadioButton` (Option 2) is selected.
   - The `ToggleButton` is turned on.

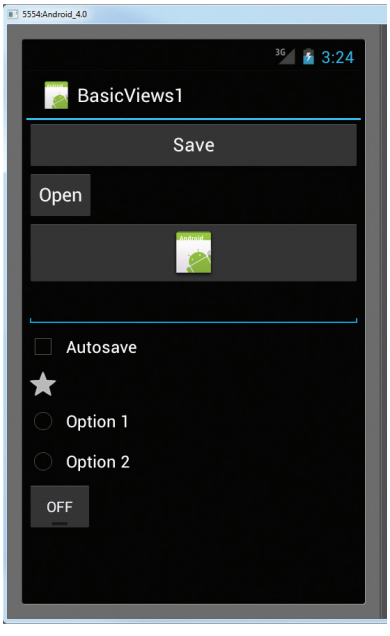


FIGURE 4-1

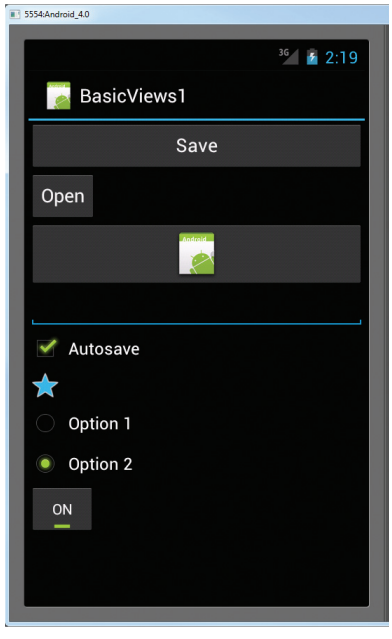


FIGURE 4-2

How It Works

So far, all the views are relatively straightforward — they are listed using the `<LinearLayout>` element, so they are stacked on top of each other when they are displayed in the activity.

For the first `Button`, the `layout_width` attribute is set to `fill_parent` so that its width occupies the entire width of the screen:

```
<Button android:id="@+id/btnSave"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="save" />
```

For the second Button, the `layout_width` attribute is set to `wrap_content` so that its width will be the width of its content — specifically, the text that it is displaying (i.e.,"Open"):

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

The `ImageButton` displays a button with an image. The image is set through the `src` attribute. In this case, you simply used the image for the application icon:

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

The `EditText` view displays a rectangular region where the user can enter some text. You set the `layout_height` to `wrap_content` so that if the user enters a long string of text, its height will automatically be adjusted to fit the content (see Figure 4-3).

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

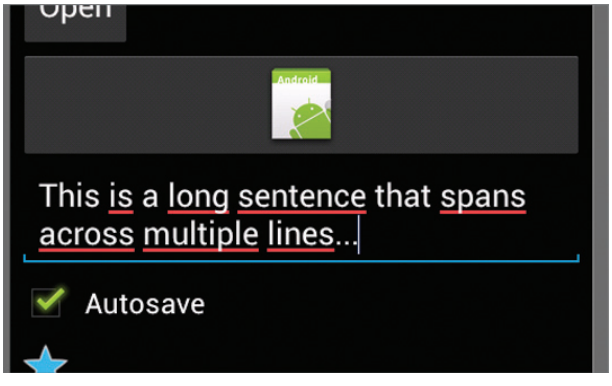


FIGURE 4-3

The `CheckBox` displays a checkbox that users can tap to check or uncheck:

```
<CheckBox android:id="@+id/chkAutosave"
   android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="Autosave" />
```

If you do not like the default look of the `CheckBox`, you can apply a style attribute to it to display it as another image, such as a star:

```
<CheckBox android:id="@+id/star"
    style="?android:attr/starStyle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

The format for the value of the `style` attribute is as follows:

```
?[package:][type:]name
```

The `RadioGroup` encloses two `RadioButtons`. This is important because radio buttons are usually used to present multiple options to the user for selection. When a `RadioButton` in a `RadioGroup` is selected, all other `RadioButtons` are automatically unselected:

```
<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>
```

Notice that the `RadioButtons` are listed vertically, one on top of another. If you want to list them horizontally, you need to change the `orientation` attribute to `horizontal`. You would also need to ensure that the `layout_width` attribute of the `RadioButtons` are set to `wrap_content`:

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

Figure 4-4 shows the `RadioButtons` displayed horizontally.

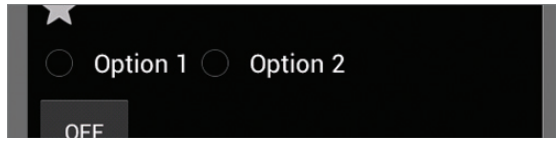


FIGURE 4-4

The `ToogleButton` displays a rectangular button that users can toggle on and off by clicking:

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

One thing that has been consistent throughout this example is that each view has the `id` attribute set to a particular value, such as in the case of the `Button`:

```
<Button android:id="@+id/btnSave"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/save" />
```

The `id` attribute is an identifier for a view so that it may later be retrieved using the `View.findViewById()` or `Activity.findViewById()` methods.

The various views that you have just seen were tested on an Android emulator emulating an Android 4.0 smartphone. What will they look like when run on older versions of Android devices? What about Android tablets?

Figure 4-5 shows what your activity will look like if you change the `android:minSdkVersion` attribute in the `AndroidManifest.xml` file to 10 and run it on the Google Nexus S running Android 2.3.6:

```
<uses-sdk android:minSdkVersion="10" />
```

Figure 4-6 shows what your activity will look like if you change the `android:minSdkVersion` attribute in the `AndroidManifest.xml` file to 13 and run it on the Asus Eee Pad Transformer running Android 3.2.1.

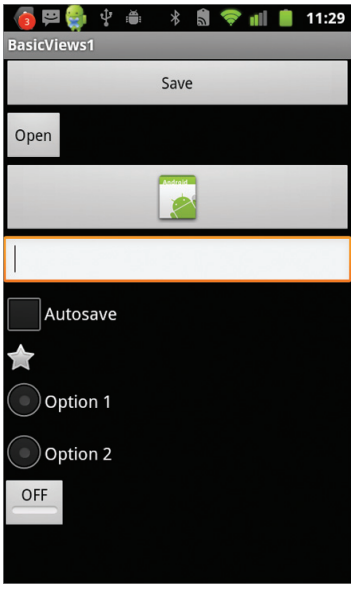


FIGURE 4-5

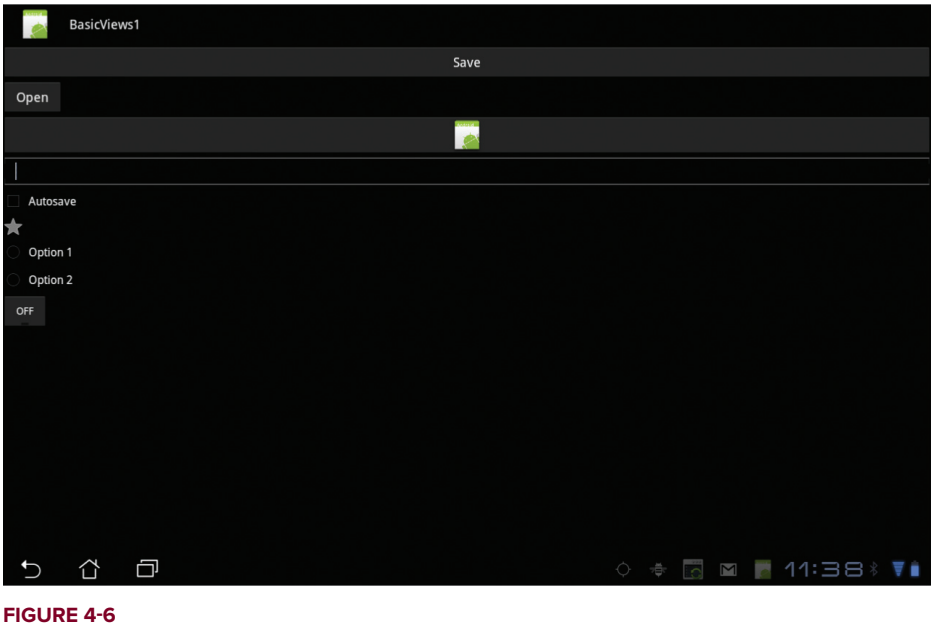


FIGURE 4-6

If you now run it on the Asus Eee Pad Transformer running Android 3.2.1 with the `android:minSdkVersion` attribute set to 8 or smaller, you will see the additional button that appears in Figure 4-7.

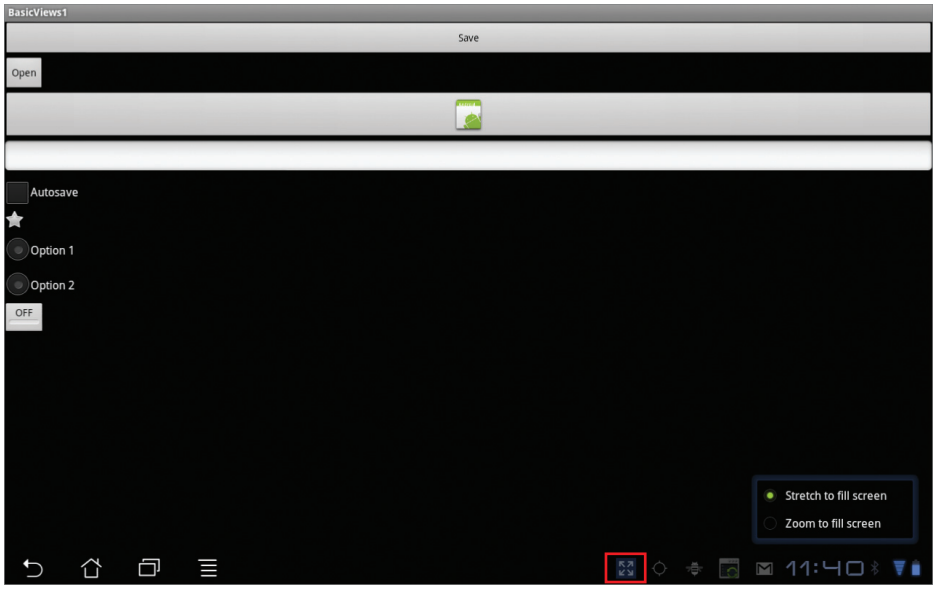


FIGURE 4-7

Tapping on the button will reveal the option to stretch the activity to fill the entire screen (default) or zoom the activity to fill the screen (see Figure 4-8).

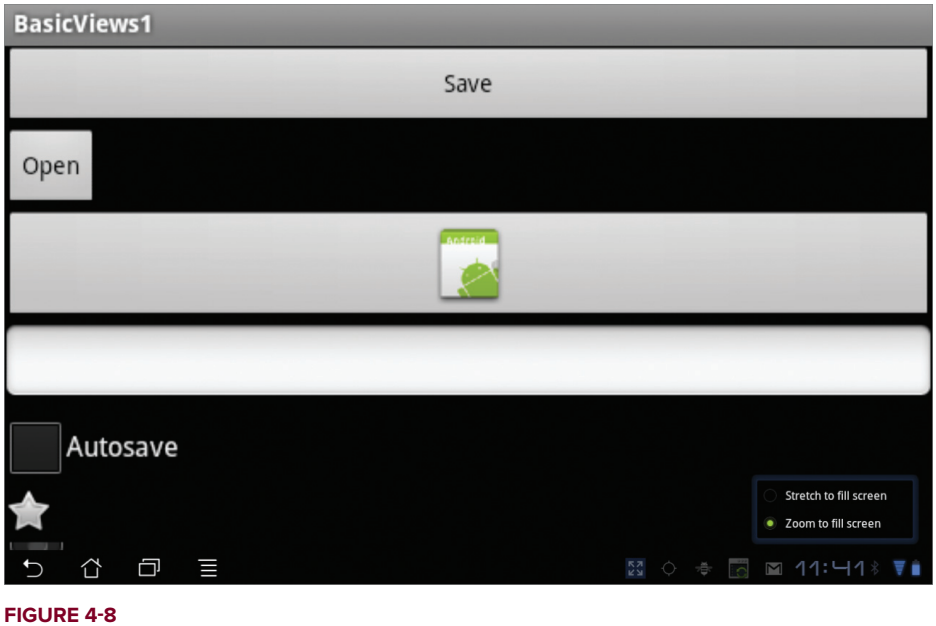


FIGURE 4-8

In short, applications with the minimum SDK version set to 8 or lower can be displayed at the screen ratios with which they were originally designed, or they can automatically stretch to fill the screen (default behavior).

Now that you have seen what the various views for an activity look like, the following Try It Out demonstrates how you can programmatically control them.

TRY IT OUT Handling View Events

1. Using the BasicViews1 project created in the previous Try It Out, modify the `BasicViews1Activity.java` file by adding the following statements in bold:

```
package net.learn2develop.BasicViews1;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.RadioGroup.OnCheckedChangeListener;
import android.widget.Toast;
import android.widget.ToggleButton;

public class BasicViews1Activity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //---Button view---
        Button btnOpen = (Button) findViewById(R.id.btnOpen);
        btnOpen.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                DisplayToast("You have clicked the Open button");
            }
        });

        //---Button view---
        Button btnSave = (Button) findViewById(R.id.btnSave);
        btnSave.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View v) {
                DisplayToast("You have clicked the Save button");
            }
        });

        //---CheckBox---
        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---
        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!");
                }
            }
        });

        //---ToggleButton---
        ToggleButton toggleButton =
                (ToggleButton) findViewById(R.id.toggle1);
        toggleButton.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View v) {
                if (((ToggleButton)v).isChecked())
                    DisplayToast("Toggle button is On");
                else
                    DisplayToast("Toggle button is Off");
            }
        });
    }

    private void DisplayToast(String msg)
    {
        Toast.makeText(getBaseContext(), msg,
                Toast.LENGTH_SHORT).show();
    }

}
```

2. Press F11 to debug the project on the Android emulator.
3. Click on the various views and observe the message displayed in the Toast window.

How It Works

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 `findViewById()` method (belonging to the `Activity` base class), supplying it with the ID of the view:

```
        //---Button view---
        Button btnOpen = (Button) findViewById(R.id.btnOpen);
```

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

```
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.

To determine the state of the `CheckBox`, 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");
    }
});
```

For the `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:

```
//---RadioButton---
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.

The `ToggleButton` works just like the `CheckBox`.

So far, to handle the events on the views, you first had to get a reference to the view and then register a callback to handle the event. There is another way to handle view events. Using the `Button` as an example, you can add an attribute called `onClick` to it:

```
<Button android:id="@+id/btnSave"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/save"
    android:onClick="btnSaved_clicked"/>
```

The `onClick` attribute specifies the click event of the button. The value of this attribute is the name of the event handler. Therefore, to handle the click event of the button, you simply need to create a method called `btnSaved_clicked`, as shown in the following example (note that the method must have a single parameter of type `View`):

```
public class BasicViews1Activity extends Activity {

    public void btnSaved_clicked (View view) {
        DisplayToast("You have clicked the Save button1");
    }

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //...
    }

    private void DisplayToast(String msg)
    {
        Toast.makeText(getBaseContext(), msg,
                Toast.LENGTH_SHORT).show();
    }

}
```

If you compare this approach to the earlier ones used, this is much simpler. Which method you use is really up to you, but this book mostly uses the latter approach.

ProgressBar View

The `ProgressBar` view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background. For example, you might be downloading some data from the web and need to update the user about the status of the download. In this case, the `ProgressBar` view is a good choice for this task. The following activity demonstrates how to use this view.

TRY IT OUT Using the ProgressBar View

codefile BasicViews2.zip available for download at Wrox.com

1. Using Eclipse, create an Android project and name it **BasicViews2**.
2. Modify the `main.xml` file located in the `res/layout` folder by adding the following code in bold:

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

<ProgressBar android:id="@+id/progressbar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

</LinearLayout>
```

3. In the `BasicViews2Activity.java` file, add the following statements in bold:

```
package net.learn2develop.BasicViews2;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.widget.ProgressBar;

public class BasicViews2Activity extends Activity {
    static int progress;
    ProgressBar progressBar;
    int progressStatus = 0;
    Handler handler = new Handler();

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        progress = 0;
        progressBar = (ProgressBar) findViewById(R.id.progressbar);

        //---do some work in background thread---
        new Thread(new Runnable()
        {
            public void run()
            {
                //---do some work here---
                while (progressStatus < 10)
                {
                    progressStatus = doSomeWork();
                }

                //---hides the progress bar---
```

```
                handler.post(new Runnable()
                {
                    public void run()
                    {
                        //---0 - VISIBLE; 4 - INVISIBLE; 8 - GONE---
                        progressBar.setVisibility(View.GONE);
                    }
                });
            }

            //---do some long running work here---
            private int doSomeWork()
            {
                try {
                    //---simulate doing some work---
                    Thread.sleep(500);
                } catch (InterruptedException e)
                {
                    e.printStackTrace();
                }
                return ++progress;
            }
        }).start();
    }
}
```

4. Press F11 to debug the project on the Android emulator. Figure 4-9 shows the `ProgressBar` animating. After about five seconds, it will disappear.

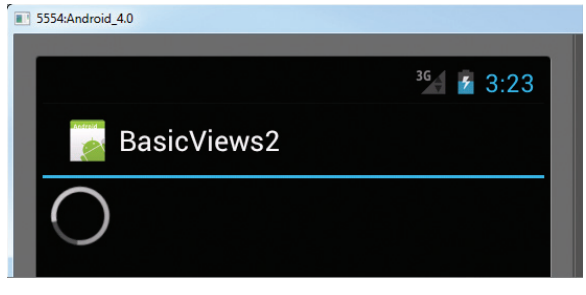


FIGURE 4-9

How It Works

The default mode of the `ProgressBar` view is indeterminate — that is, it shows a cyclic animation. This mode is useful for tasks that do not have specific completion times, such as when you are sending some data to a web service and waiting for the server to respond. If you simply put the `<ProgressBar>` element in your `main.xml` file, it will display a spinning icon continuously. It is your responsibility to stop it when your background task has completed.

The code that you have added in the Java file shows how you can spin off a background thread to simulate performing some long-running tasks. To do so, you use the `Thread` class together with a `Runnable` object. The `run()` method starts the execution of the thread, which in this case calls the `doSomeWork()` method to simulate doing some work. When the simulated work is done (after about five seconds), you use a `Handler` object to send a message to the thread to dismiss the `ProgressBar`:

```
//---do some work in background thread---
new Thread(new Runnable()
{
    public void run()
    {
        //---do some work here---
        while (progressStatus < 10)
        {
            progressStatus = doSomeWork();
```

```
            }

            //---hides the progress bar---
            handler.post(new Runnable()
            {
                public void run()
                {
                    //---0 - VISIBLE; 4 - INVISIBLE; 8 - GONE---
                    progressBar.setVisibility(View.GONE);
                }
            });
        }

        //---do some long running work here---
        private int doSomeWork()
        {
            try {
                //---simulate doing some work---
                Thread.sleep(500);
            } catch (InterruptedException e)
            {
                e.printStackTrace();
            }
            return ++progress;
        }
    }).start();
```

When the task is completed, you hide the `ProgressBar` by setting its `Visibility` property to `View.GONE` (value `8`). The difference between the `INVISIBLE` and `GONE` constants is that the `INVISIBLE` constant simply hides the `ProgressBar` (the region occupied by the `ProgressBar` is still taking up space in the activity); whereas the `GONE` constant removes the `ProgressBar` view from the activity and does not take up any space on it.

The next Try It Out shows how you can change the look of the `ProgressBar`.

TRY IT OUT Customizing the ProgressBar View

1. Using the BasicViews2 project created in the previous Try It Out, modify the `main.xml` file as shown here:

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

<ProgressBar android:id="@+id/progressbar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@android:style/Widget.ProgressBar.Horizontal" />

</LinearLayout>
```

2. Modify the `BasicViews2Activity.java` file by adding the following statements in bold:

```
package net.learn2develop.BasicViews2;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.widget.ProgressBar;

public class BasicViews2Activity extends Activity {
    static int progress;
    ProgressBar progressBar;
    int progressStatus = 0;
    Handler handler = new Handler();

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        progress = 0;
        progressBar = (ProgressBar) findViewById(R.id.progressbar);
        progressBar.setMax(200);

        //---do some work in background thread---
        new Thread(new Runnable()
        {
            public void run()
            {
                //---do some work here---
                while (progressStatus < 100)
                {
                    progressStatus = doSomeWork();

                    //---Update the progress bar---
                    handler.post(new Runnable()
                    {
                        public void run() {
                            progressBar.setProgress(progressStatus);
                        }
                    });
                }

                //---hides the progress bar---
                handler.post(new Runnable()
                {
                    public void run()
                    {
                        //---0 - VISIBLE; 4 - INVISIBLE; 8 - GONE---
                        progressBar.setVisibility(View.GONE);
                    }
                });
```

```
            }

            //---do some long running work here---
            private int doSomeWork()
            {
                try {
                    //---simulate doing some work---
                    Thread.sleep(500);
                } catch (InterruptedException e)
                {
                    e.printStackTrace();
                }
                return ++progress;
            }
        }).start();
    }
}
```

3. Press F11 to debug the project on the Android emulator.

4. Figure 4-10 shows the `ProgressBar` displaying the progress. The `ProgressBar` disappears when the progress reaches 50%.

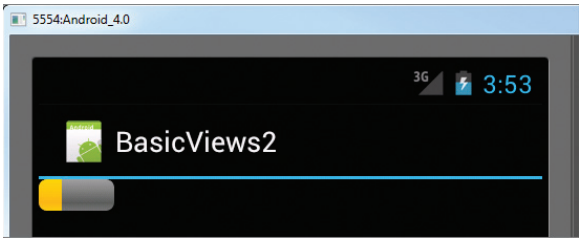


FIGURE 4-10

How It Works

To make the `ProgressBar` display horizontally, simply set its `style` attribute to `@android:style/Widget.ProgressBar.Horizontal`:

```
<ProgressBar android:id="@+id/progressbar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@android:style/Widget.ProgressBar.Horizontal" />
```

To display the progress, call its `setProgress()` method, passing in an integer indicating its progress:

```
//---Update the progress bar---
handler.post(new Runnable()
{
    public void run() {
        progressBar.setProgress(progressStatus);
    }
});
```

In this example, you set the range of the `ProgressBar` from 0 to 200 (via the `setMax()` method). Hence, the `ProgressBar` will stop and then disappear when it is halfway through (since you only continue to call the `doSomeWork()` method as long as the `progressStatus` is less than 100). To ensure that the `ProgressBar` disappears only when the progress reaches 100%, either set the maximum value to 100 or modify the `while` loop to stop when the `progressStatus` reaches 200, like this:

```
//---do some work here---
while (progressStatus < 200)
```

Besides the horizontal style for the `ProgressBar` that you have used for this example, you can also use the following constants:

- `Widget.ProgressBar.Horizontal`
- `Widget.ProgressBar.Small`
- `Widget.ProgressBar.Large`
- `Widget.ProgressBar.Inverse`
- `Widget.ProgressBar.Small.Inverse`
- `Widget.ProgressBar.Large.Inverse`

AutoCompleteTextView View

The `AutoCompleteTextView` is a view that is similar to `EditText` (in fact it is a subclass of `EditText`), except that it shows a list of completion suggestions automatically while the user is typing. The following Try It Out shows how to use the `AutoCompleteTextView` to automatically help users complete the text entry.

TRY IT OUT Using the AutoCompleteTextView

codefile BasicViews3.zip available for download at Wrox.com

1. Using Eclipse, create an Android project and name it **BasicViews3**.
2. Modify the `main.xml` file located in the `res/layout` folder as shown here in bold:

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

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Name of President" />

<AutoCompleteTextView android:id="@+id/txtCountries"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

</LinearLayout>
```

3. Add the following statements in bold to the `BasicViews3Activity.java` file:

```
package net.learn2develop.BasicViews3;

import android.app.Activity;
import android.os.Bundle;
```

```
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;

public class BasicViews3Activity extends Activity {
    String[] presidents = {
            "Dwight D. Eisenhower",
            "John F. Kennedy",
            "Lyndon B. Johnson",
            "Richard Nixon",
            "Gerald Ford",
            "Jimmy Carter",
            "Ronald Reagan",
            "George H. W. Bush",
            "Bill Clinton",
            "George W. Bush",
            "Barack Obama"
        };

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_dropdown_item_1line, presidents);

        AutoCompleteTextView textView = (AutoCompleteTextView)
            findViewById(R.id.txtCountries);

        textView.setThreshold(3);
        textView.setAdapter(adapter);
    }
}
```

4. Press F11 to debug the application on the Android emulator. As shown in Figure 4-11, a list of matching names appears as you type into the `AutoCompleteTextView`.

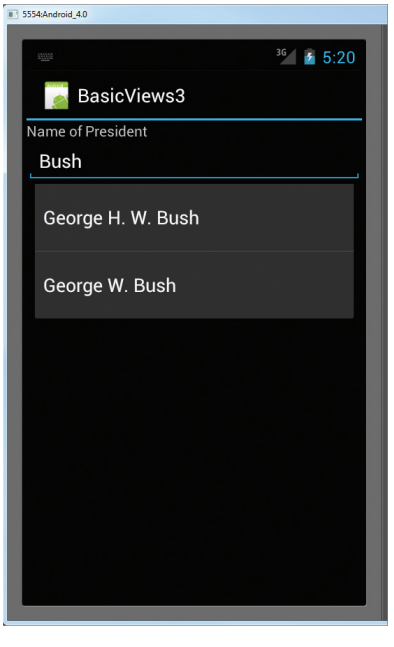


FIGURE 4-11

How It Works

In the `BasicViews3Activity` class, you first created a `String` array containing a list of presidents' names:

```
String[] presidents = {
        "Dwight D. Eisenhower",
        "John F. Kennedy",
        "Lyndon B. Johnson",
        "Richard Nixon",
        "Gerald Ford",
        "Jimmy Carter",
        "Ronald Reagan",
        "George H. W. Bush",
        "Bill Clinton",
```

```
        "George W. Bush",
        "Barack Obama"
    };
```

The `ArrayAdapter` object manages the array of strings that will be displayed by the `AutoCompleteTextView`. In the preceding example, you set the `AutoCompleteTextView` to display in the `simple_dropdown_item_1line` mode:

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_dropdown_item_1line, presidents);
```

The `setThreshold()` method sets the minimum number of characters the user must type before the suggestions appear as a drop-down menu:

```
textView.setThreshold(3);
```

The list of suggestions to display for the `AutoCompleteTextView` is obtained from the `ArrayAdapter` object:

```
textView.setAdapter(adapter);
```

USING PICKER VIEWS

Selecting the date and time is one of the common tasks you need to perform in a mobile application. Android supports this functionality through the `TimePicker` and `DatePicker` views. The following sections demonstrate how to use 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. The following Try It Out shows you how to use it.

TRY IT OUT Using the TimePicker View

codefile BasicViews4.zip available for download at Wrox.com

1. Using Eclipse, create an Android project and name it **BasicViews4**.
2. Modify the `main.xml` file located in the `res/layout` folder by adding the following lines in bold:

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

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

```
    android:layout_height="wrap_content" />

<Button android:id="@+id/btnSet"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="I am all set!"
    android:onClick="onClick" />

</LinearLayout>
```

3. Select the project name in Eclipse and press F11 to debug the application on the Android emulator. Figure 4-12 shows the `TimePicker` in action. Besides clicking the plus (+) and minus (-) buttons, you can use the numeric keypad on the device to change the hour and minute, and click the AM button to toggle between AM and PM.

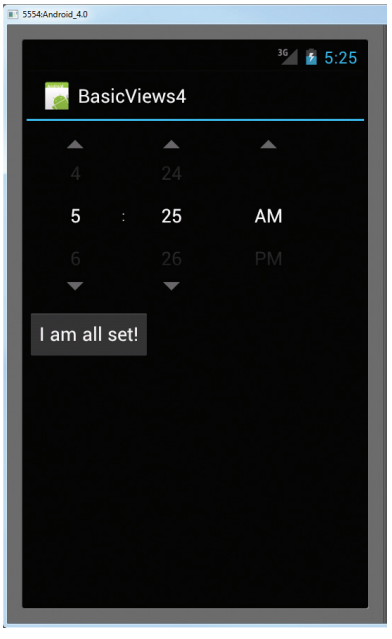


FIGURE 4-12

4. Back in Eclipse, add the following statements in bold to the `BasicViews4Activity.java` file:

```
package net.learn2develop.BasicViews4;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TimePicker;
import android.widget.Toast;

public class BasicViews4Activity extends Activity {
    TimePicker timePicker;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        timePicker = (TimePicker) findViewById(R.id.timePicker);
        timePicker.setIs24HourView(true);
    }

    public void onClick(View view) {
        Toast.makeText(getBaseContext(),
                "Time selected:" +
                timePicker.getCurrentHour() +
                ":" + timePicker.getCurrentMinute(),
                Toast.LENGTH_SHORT).show();
    }

}
```

5. Press F11 to debug the application on the Android emulator. This time, the `TimePicker` will be displayed in the 24-hour format. Clicking the `Button` will display the time that you have set in the `TimePicker` (see Figure 4-13).

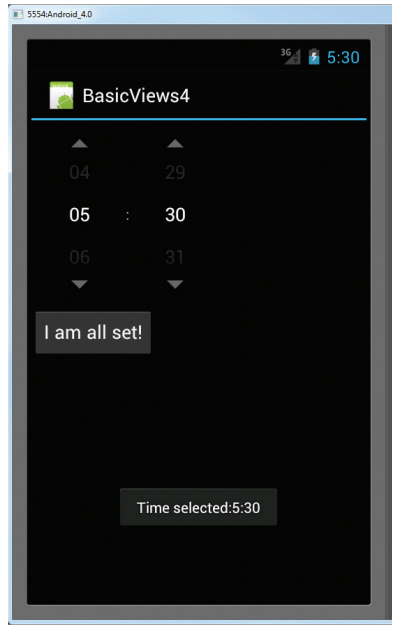


FIGURE 4-13

How It Works

The `TimePicker` displays a standard UI to enable users to set a time. By default, it displays the time in the AM/PM format. If you wish to display the time in the 24-hour format, you can use the `setIs24HourView()` method.

To programmatically get the time set by the user, use the `getCurrentHour()` and `getCurrentMinute()` methods:

```
Toast.makeText(getBaseContext(),
        "Time selected:" +
        timePicker.getCurrentHour() +
        ":" + timePicker.getCurrentMinute(),
        Toast.LENGTH_SHORT).show();
```

NOTE *The `getCurrentHour()` method always returns the hour in 24-hour format (i.e., a value from 0 to 23).*

Although you can display the `TimePicker` in an activity, it's better to display it in a dialog window; that way, once the time is set, it disappears and doesn't take up any space in an activity. The following Try It Out demonstrates how to do just that.

TRY IT OUT Using a Dialog to Display the TimePicker View

1. Using the BasicViews4 project created in the previous Try It Out, modify the `BasicViews4Activity.java` file as shown here:

```
package net.learn2develop.BasicViews4;

import java.text.SimpleDateFormat;
import java.util.Date;

import android.app.Activity;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.os.Bundle;
import android.view.View;
import android.widget.TimePicker;
```

```
import android.widget.Toast;

public class BasicViews4Activity extends Activity {
    TimePicker timePicker;

    int hour, minute;
    static final int TIME_DIALOG_ID = 0;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        timePicker = (TimePicker) findViewById(R.id.timePicker);
        timePicker.setIs24HourView(true);

        showDialog(TIME_DIALOG_ID);
    }

    @Override
    protected Dialog onCreateDialog(int id)
    {
        switch (id) {
        case TIME_DIALOG_ID:
            return new TimePickerDialog(
                    this, mTimeSetListener, hour, minute, false);
        }
        return null;
    }

    private TimePickerDialog.OnTimeSetListener mTimeSetListener =
    new TimePickerDialog.OnTimeSetListener()
    {
        public void onTimeSet(
                TimePicker view, int hourOfDay, int minuteOfHour)
        {
            hour = hourOfDay;
            minute = minuteOfHour;

            SimpleDateFormat timeFormat = new SimpleDateFormat("hh:mm aa");
            Date date = new Date(0,0,0, hour, minute);
            String strDate = timeFormat.format(date);

            Toast.makeText(getBaseContext(),
                    "You have selected " + strDate,
                    Toast.LENGTH_SHORT).show();
        }
    };

    public void onClick(View view) {
        Toast.makeText(getBaseContext(),
                "Time selected:" +
```

```
                                timePicker.getCurrentHour() +
                                ":" + timePicker.getCurrentMinute(),
                                Toast.LENGTH_SHORT).show();
        }

}
```

2. Press F11 to debug the application on the Android emulator. When the activity is loaded, you can see the `TimePicker` displayed in a dialog window (see Figure 4-14). Set a time and then click the Set button. You will see the Toast window displaying the time that you just set.

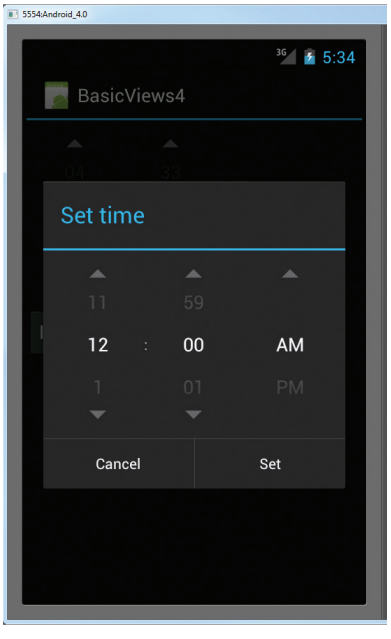


FIGURE 4-14

How It Works

To display a dialog window, you use the `showDialog()` method, passing it an ID to identify the source of the dialog:

```
showDialog(TIME_DIALOG_ID);
```

When the `showDialog()` method is called, the `onCreateDialog()` method will be called:

```
@Override
protected Dialog onCreateDialog(int id)
{
    switch (id) {
    case TIME_DIALOG_ID:
        return new TimePickerDialog(
                this, mTimeSetListener, hour, minute, false);
    }
    return null;
}
```

Here, you create a new instance of the `TimePickerDialog` class, passing it the current context, the callback, the initial hour and minute, as well as whether the `TimePicker` should be displayed in 24-hour format.

When the user clicks the Set button in the `TimePicker` dialog window, the `onTimeSet()` method is called:

```
private TimePickerDialog.OnTimeSetListener mTimeSetListener =
new TimePickerDialog.OnTimeSetListener()
{
    public void onTimeSet(
            TimePicker view, int hourOfDay, int minuteOfHour)
    {
        hour = hourOfDay;
        minute = minuteOfHour;

        SimpleDateFormat timeFormat = new SimpleDateFormat("hh:mm aa");
        Date date = new Date(0,0,0, hour, minute);
```

```
            String strDate = timeFormat.format(date);

            Toast.makeText(getBaseContext(),
                    "You have selected " + strDate,
                    Toast.LENGTH_SHORT).show();
        }
    };
```

Here, the `onTimeSet()` method contains the hour and minute set by the user via the `hourOfDay` and `minuteOfHour` arguments, respectively.

DatePicker View

Another view that is similar to the `TimePicker` is the `DatePicker`. Using the `DatePicker`, you can enable users to select a particular date on the activity. The following Try It Out shows you how to use the `DatePicker`.

TRY IT OUT Using the DatePicker View

1. Using the BasicViews4 project created earlier, modify the `main.xml` file as shown here:

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

<Button android:id="@+id/btnSet"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="I am all set!"
    android:onClick="onClick" />

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

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

</LinearLayout>
```

2. Press F11 to debug the application on the Android emulator. Figure 4-15 shows the `DatePicker` view (you have to change the emulator's orientation to landscape by pressing Ctrl-F11; portrait mode is too narrow to display the `DatePicker`).

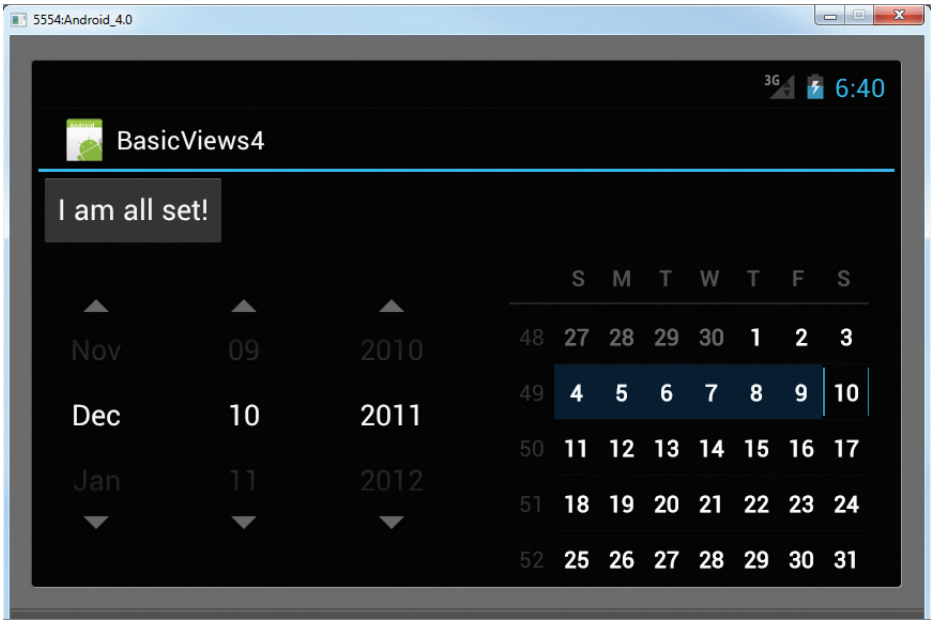


FIGURE 4-15

3. Back in Eclipse, add the following statements in bold to the `BasicViews4Activity.java` file:

```
package net.learn2develop.BasicViews4;

import java.text.SimpleDateFormat;
import java.util.Date;

import android.app.Activity;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.os.Bundle;
import android.view.View;
import android.widget.DatePicker;
import android.widget.TimePicker;
import android.widget.Toast;

public class BasicViews4Activity extends Activity {
    TimePicker timePicker;
    DatePicker datePicker;

    int hour, minute;
    static final int TIME_DIALOG_ID = 0;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

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

        timePicker = (TimePicker) findViewById(R.id.timePicker);
        timePicker.setIs24HourView(true);

        // showDialog(TIME_DIALOG_ID);
        datePicker = (DatePicker) findViewById(R.id.datePicker);
    }

    @Override
    protected Dialog onCreateDialog(int id)
    {
        switch (id) {
        case TIME_DIALOG_ID:
            return new TimePickerDialog(
                    this, mTimeSetListener, hour, minute, false);
        }
        return null;
    }

    private TimePickerDialog.OnTimeSetListener mTimeSetListener =
    new TimePickerDialog.OnTimeSetListener()
    {
        public void onTimeSet(
                TimePicker view, int hourOfDay, int minuteOfHour)
        {
            hour = hourOfDay;
            minute = minuteOfHour;

            SimpleDateFormat timeFormat = new SimpleDateFormat("hh:mm aa");
            Date date = new Date(0,0,0, hour, minute);
            String strDate = timeFormat.format(date);

            Toast.makeText(getBaseContext(),
                    "You have selected " + strDate,
                    Toast.LENGTH_SHORT).show();
        }
    };

    public void onClick(View view) {
        Toast.makeText(getBaseContext(),
                "Date selected:" + (datePicker.getMonth() + 1) +
                "/" + datePicker.getDayOfMonth() +
                "/" + datePicker.getYear() + "\n" +
                "Time selected:" + timePicker.getCurrentHour() +
                ":" + timePicker.getCurrentMinute(),
                Toast.LENGTH_SHORT).show();
    }

}
```

4. Press F11 to debug the application on the Android emulator. Once the date is set, clicking the `Button` will display the date set (see Figure 4-16).

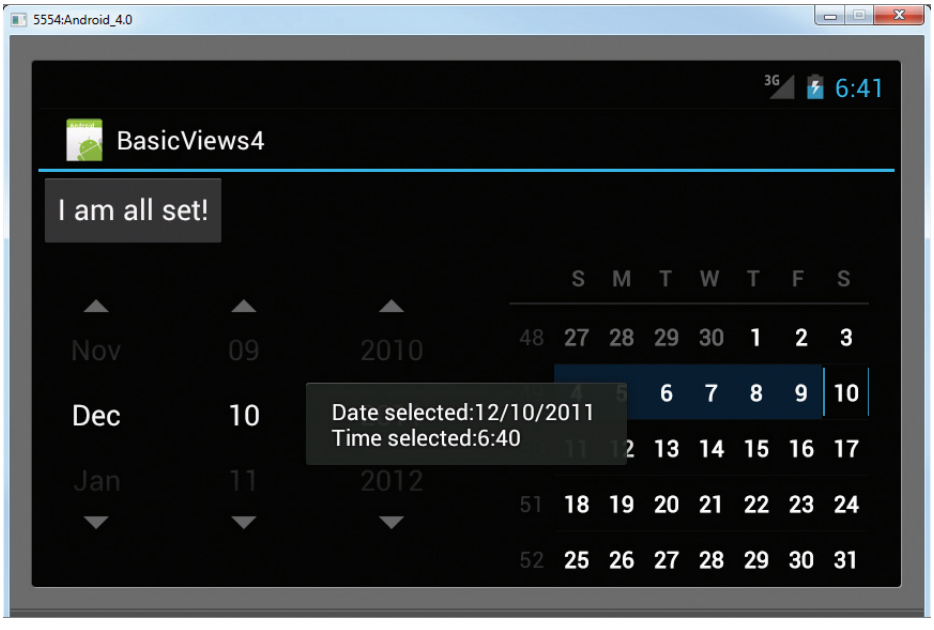


FIGURE 4-16

How It Works

Like the `TimePicker`, you call the `getMonth()`, `getDayOfMonth()`, and `getYear()` methods to get the month, day, and year, respectively:

```
"Date selected:" + (datePicker.getMonth() + 1) +
"/" + datePicker.getDayOfMonth() +
"/" + datePicker.getYear() + "\n" +
```

Note that the `getMonth()` method returns 0 for January, 1 for February, and so on. Hence, you need to increment the result of this method by one to get the corresponding month number.

Like the `TimePicker`, you can also display the `DatePicker` in a dialog window. The following Try It Out shows you how.

TRY IT OUT Using a Dialog to Display the DatePicker View

1. Using the BasicViews4 project created earlier, add the following statements in bold to the `BasicViews4Activity.java` file:

```
package net.learn2develop.BasicViews4;

import java.text.SimpleDateFormat;
import java.util.Calendar;
```

```
import java.util.Date;

import android.app.Activity;
import android.app.DatePickerDialog;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.os.Bundle;
import android.view.View;
import android.widget.DatePicker;
import android.widget.TimePicker;
import android.widget.Toast;

public class BasicViews4Activity extends Activity {
    TimePicker timePicker;
    DatePicker datePicker;

    int hour, minute;
    int yr, month, day;

    static final int TIME_DIALOG_ID = 0;
    static final int DATE_DIALOG_ID = 1;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        timePicker = (TimePicker) findViewById(R.id.timePicker);
        timePicker.setIs24HourView(true);

        // showDialog(TIME_DIALOG_ID);
        datePicker = (DatePicker) findViewById(R.id.datePicker);

        //---get the current date---
        Calendar today = Calendar.getInstance();
        yr = today.get(Calendar.YEAR);
        month = today.get(Calendar.MONTH);
        day = today.get(Calendar.DAY_OF_MONTH);

        showDialog(DATE_DIALOG_ID);
    }

    @Override
    protected Dialog onCreateDialog(int id)
    {
        switch (id) {
        case TIME_DIALOG_ID:
            return new TimePickerDialog(
                    this, mTimeSetListener, hour, minute, false);
        case DATE_DIALOG_ID:
            return new DatePickerDialog(
```

```
                    this, mDateSetListener, yr, month, day);
        }
        return null;
    }

    private DatePickerDialog.OnDateSetListener mDateSetListener =
    new DatePickerDialog.OnDateSetListener()
    {
        public void onDateSet(
                DatePicker view, int year, int monthOfYear, int dayOfMonth)
        {
            yr = year;
            month = monthOfYear;
            day = dayOfMonth;
            Toast.makeText(getBaseContext(),
                    "You have selected : " + (month + 1) +
                    "/" + day + "/" + year,
                    Toast.LENGTH_SHORT).show();
        }
    };

    private TimePickerDialog.OnTimeSetListener mTimeSetListener =
    new TimePickerDialog.OnTimeSetListener()
    {
        public void onTimeSet(
                TimePicker view, int hourOfDay, int minuteOfHour)
        {
            hour = hourOfDay;
            minute = minuteOfHour;

            SimpleDateFormat timeFormat = new SimpleDateFormat("hh:mm aa");
            Date date = new Date(0,0,0, hour, minute);
            String strDate = timeFormat.format(date);

            Toast.makeText(getBaseContext(),
                    "You have selected " + strDate,
                    Toast.LENGTH_SHORT).show();
        }
    };

    public void onClick(View view) {
        Toast.makeText(getBaseContext(),
                "Date selected:" + (datePicker.getMonth() + 1) +
                "/" + datePicker.getDayOfMonth() +
                "/" + datePicker.getYear() + "\n" +
                "Time selected:" + timePicker.getCurrentHour() +
                ":" + timePicker.getCurrentMinute(),
                Toast.LENGTH_SHORT).show();
    }

}
```

2. Press F11 to debug the application on the Android emulator. When the activity is loaded, you can see the `DatePicker` displayed in a dialog window (see Figure 4-17). Select a date and then click the Set button. The Toast window will display the date you have just set.

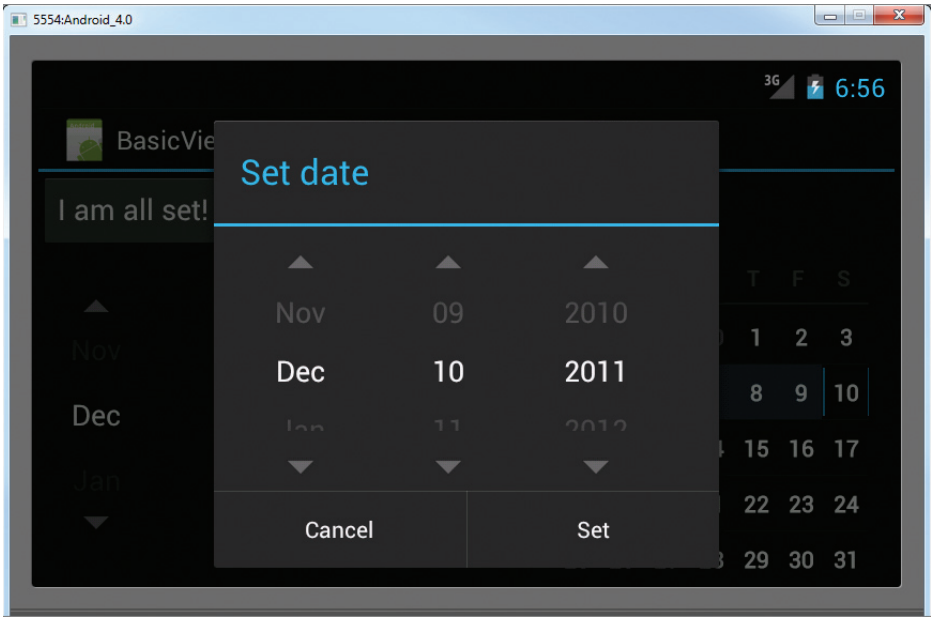


FIGURE 4-17

How It Works

The `DatePicker` works exactly like the `TimePicker`. When a date is set, it fires the `onDateSet()` method, where you can obtain the date set by the user:

```
public void onDateSet(
        DatePicker view, int year, int monthOfYear, int dayOfMonth)
{
    yr = year;
    month = monthOfYear;
    day = dayOfMonth;
    Toast.makeText(getBaseContext(),
            "You have selected : " + (month + 1) +
            "/" + day + "/" + year,
            Toast.LENGTH_SHORT).show();
}
```

Note that you have to initialize the three variables — `yr`, `month`, and `day` — before showing the dialog:

```
//---get the current date---
Calendar today = Calendar.getInstance();
yr = today.get(Calendar.YEAR);
```

```
month = today.get(Calendar.MONTH);
day = today.get(Calendar.DAY_OF_MONTH);

showDialog(DATE_DIALOG_ID);
```

If you don't, you will get an illegal argument exception error ("`current should be >= start and <= end`") during runtime when you create an instance of the `DatePickerDialog` class.

USING LIST VIEWS TO DISPLAY LONG LISTS

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. The following Try It Outs show them in action.

ListView View

The `ListView` displays a list of items in a vertically scrolling list. The following Try It Out demonstrates how to display a list of items using the `ListView`.

TRY IT OUT Displaying a Long List of Items Using the ListView

1. Using Eclipse, create an Android project and name it **BasicViews5**.

codefile BasicViews5.zip available for download at Wrox.com

2. Modify the `BasicViews5Activity.java` file by inserting the statements shown here in bold:

```
package net.learn2develop.BasicViews5;

import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

public class BasicViews5Activity extends ListActivity  {
    String[] presidents = {
            "Dwight D. Eisenhower",
            "John F. Kennedy",
            "Lyndon B. Johnson",
            "Richard Nixon",
            "Gerald Ford",
            "Jimmy Carter",
            "Ronald Reagan",
            "George H. W. Bush",
            "Bill Clinton",
            "George W. Bush",
```

```
            "Barack Obama"
        };

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //---no need to call this---
        //setContentView(R.layout.main);

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

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

}
```

3. Press F11 to debug the application on the Android emulator. Figure 4-18 shows the activity displaying the list of presidents' names.

4. Click on an item. A message containing the item selected will be displayed.

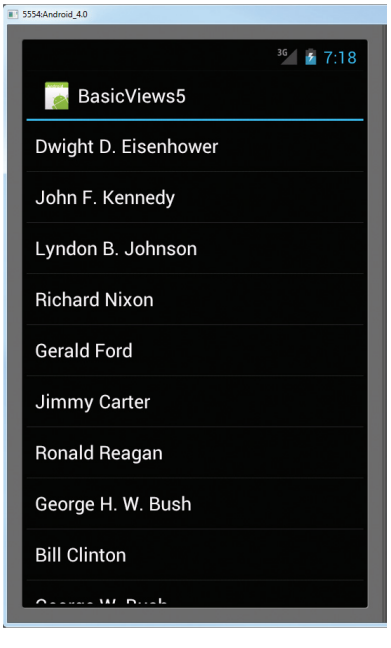


FIGURE 4-18

How It Works

The first thing to notice in this example is that the `BasicViews5Activity` class extends the `ListActivity` class. The `ListActivity` class extends the `Activity` class and it displays a list of items by binding to a data source. Also note that there is no need to modify the `main.xml` file to include the `ListView`; the `ListActivity` class itself contains a `ListView`. Hence, in the `onCreate()` method, you don't need to call the `setContentView()` method to load the UI from the `main.xml` file:

```
        //---no need to call this---
        //setContentView(R.layout.main);
```

In the `onCreate()` method, you use the `setListAdapter()` method to programmatically fill the entire screen of the activity with a `ListView`. 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:

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

Here, you simply display the name of the president selected using the `Toast` class.

Customizing the ListView

The `ListView` is a versatile view that you can further customize. The following Try It Out shows how you can allow multiple items in the `ListView` to be selected and how you can enable filtering support.

TRY IT OUT Enabling Filtering and Multi-Item Support in the ListView

1. Using the BasicViews5 project created in the previous section, add the following statements in bold to the `BasicViews5Activity.java` file:

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    //---no need to call this---
    //setContentView(R.layout.main);

    ListView lstView = getListView();
    //lstView.setChoiceMode(ListView.CHOICE_MODE_NONE);
    //lstView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
    lstView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
    lstView.setTextFilterEnabled(true);

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

2. Press F11 to debug the application on the Android emulator. You can now click on each item to display the check icon next to it (see Figure 4-19).

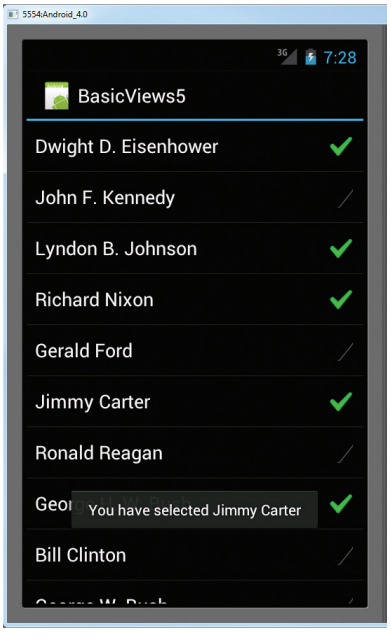


FIGURE 4-19

How It Works

To programmatically get a reference to the `ListView` object, you use the `getListView()` method, which fetches the `ListActivity`'s list view. You need to do this so that you can programmatically modify the behavior of the `ListView`. In this case, you used the `setChoiceMode()` method to specify how the `ListView` should handle a user's click. For this example, you set it to `ListView.CHOICE_MODE_MULTIPLE`, which means that the user can select multiple items:

```
ListView lstView = getListView();
//lstView.setChoiceMode(ListView.CHOICE_MODE_NONE);
//lstView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
lstView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
```

A very cool feature of the `ListView` is its support for filtering. When you enable filtering through the `setTextFilterEnabled()` method, users will be able to type on the keypad and the `ListView` will automatically filter the items to match what was typed:

```
lstView.setTextFilterEnabled(true);
```

Figure 4-20 shows the list filtering in action. Here, all items in the list that contain the word "john" will appear in the result list.

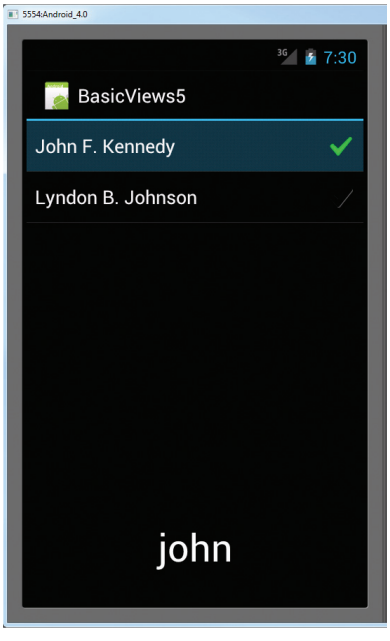


FIGURE 4-20

While the previous example shows that the list of presidents' names is stored in an array, in a real-life application it is recommended that you either retrieve them from a database or at least store them in the `strings.xml` file. The following Try It Out shows you how.

TRY IT OUT Storing Items in the strings.xml File

1. Using the BasicViews5 project created earlier, add the following lines in bold to the `strings.xml` file located in the `res/values` folder:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, BasicViews5Activity!</string>
    <string name="app_name">BasicViews5</string>
    <string-array name="presidents_array">
        <item>Dwight D. Eisenhower</item>
        <item>John F. Kennedy</item>
        <item>Lyndon B. Johnson</item>
        <item>Richard Nixon</item>
        <item>Gerald Ford</item>
        <item>Jimmy Carter</item>
        <item>Ronald Reagan</item>
        <item>George H. W. Bush</item>
        <item>Bill Clinton</item>
        <item>George W. Bush</item>
```

```
        <item>Barack Obama</item>
    </string-array>
</resources>
```

2. Modify the `BasicViews5Activity.java` file as shown in bold:

```
public class BasicViews5Activity extends ListActivity  {
    String[] presidents;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        //---no need to call this---
        //setContentView(R.layout.main);

        ListView lstView = getListView();
        //lstView.setChoiceMode(ListView.CHOICE_MODE_NONE);
        //lstView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
        lstView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
        lstView.setTextFilterEnabled(true);

        presidents =
                getResources().getStringArray(R.array.presidents_array);

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

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

}
```

3. Press F11 to debug the application on the Android emulator. You should see the same list of names that appeared in the previous Try It Out.

How It Works

With the names now stored in the `strings.xml` file, you can retrieve it programmatically in the `BasicViews5Activity.java` file using the `getResources()` method:

```
        presidents =
                getResources().getStringArray(R.array.presidents_array);
```

In general, you can programmatically retrieve resources bundled with your application using the `getResources()` method.

This example demonstrated how to make items in a `ListView` selectable. At the end of the selection process, how do you know which item or items are selected? The following Try It Out shows you how.

TRY IT OUT Checking Which Items Are Selected

1. Using the BasicViews5 project again, add the following lines in bold to the `main.xml` file:

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

<Button
    android:id="@+id/btn"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Show selected items"
    android:onClick="onClick"/>

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

</LinearLayout>
```

2. Add the following lines in bold to the `BasicViews5Activity.java` file:

```
package net.learn2develop.BasicViews5;

import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

public class BasicViews5Activity extends ListActivity  {

    String[] presidents;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        ListView lstView = getListView();
        //lstView.setChoiceMode(ListView.CHOICE_MODE_NONE);
        //lstView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
        lstView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
        lstView.setTextFilterEnabled(true);

        presidents =
```

```
                getResources().getStringArray(R.array.presidents_array);

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

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

    public void onClick(View view) {
        ListView lstView = getListView();

        String itemsSelected = "Selected items: \n";
        for (int i=0; i<lstView.getCount(); i++) {
            if (lstView.isItemChecked(i)) {
                itemsSelected += lstView.getItemAtPosition(i) + "\n";
            }
        }
        Toast.makeText(this, itemsSelected, Toast.LENGTH_LONG).show();
    }

}
```

3. Press F11 to debug the application on the Android emulator. Click on a few items and then click the Show selected items button (see Figure 4-21). The list of names selected will be displayed.

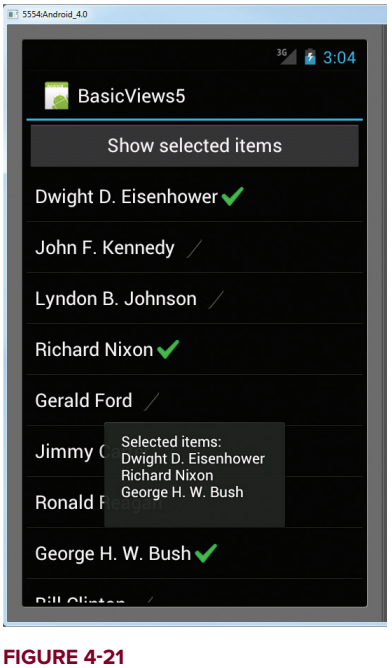


FIGURE 4-21

How It Works

In the previous section's exercise, you saw how to populate a `ListView` that occupies the entire activity — in that example, there is no need to add a `<ListView>` element to the `main.xml` file. In this example, you saw how a `ListView` can partially fill up an activity. To do that, you needed to add a `<ListView>` element with the `id` attribute set to `@+id/android:list`:

```
<ListView
    android:id="@+id/android:list"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

You then needed to load the content of the activity using the `setContentView()` method (previously commented out):

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

To find out which items in the `ListView` have been checked, you use the `isItemChecked()` method:

```
ListView lstView = getListView();
String itemsSelected = "Selected items: \n";
for (int i=0; i<lstView.getCount(); i++) {
    if (lstView.isItemChecked(i)) {
        itemsSelected += lstView.getItemAtPosition(i) + "\n";
    }
}
Toast.makeText(this, itemsSelected, Toast.LENGTH_LONG).show();
```

The `getItemAtPosition()` method returns the name of the item at the specified position.

NOTE *So far, all the examples show how to use the* `ListView` *inside a* `ListActivity`*. This is not absolutely necessary — you can also use the* `ListView` *inside an* `Activity`*. In this case, to programmatically refer to the* `ListView`*, you use the* `findViewByID()` *method instead of the* `getListView()` *method; and the* `id` *attribute of the* `<ListView>` *element can use the format of* `@+id/<view_name>`*.*

Using the Spinner View

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 view like 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.

The following Try It Out shows how you can use the `SpinnerView` in your activity.

TRY IT OUT Using the SpinnerView to Display One Item at a Time

codefile BasicViews6.zip available for download at Wrox.com

1. Using Eclipse, create an Android project and name it **BasicViews6**.
2. Modify the `main.xml` file located in the `res/layout` folder as shown here:

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

<Spinner
    android:id="@+id/spinner1"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:drawSelectorOnTop="true" />

</LinearLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, BasicViews6Activity!</string>
    <string name="app_name">BasicViews6</string>
    <string-array name="presidents_array">
        <item>Dwight D. Eisenhower</item>
        <item>John F. Kennedy</item>
        <item>Lyndon B. Johnson</item>
        <item>Richard Nixon</item>
        <item>Gerald Ford</item>
        <item>Jimmy Carter</item>
        <item>Ronald Reagan</item>
        <item>George H. W. Bush</item>
        <item>Bill Clinton</item>
        <item>George W. Bush</item>
        <item>Barack Obama</item>
    </string-array>
</resources>
```

4. Add the following statements in bold to the `BasicViews6Activity.java` file:

```
package net.learn2develop.BasicViews6;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;

public class BasicViews6Activity extends Activity {
    String[] presidents;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        presidents =
                getResources().getStringArray(R.array.presidents_array);
```

```
        Spinner s1 = (Spinner) findViewById(R.id.spinner1);

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_spinner_item, presidents);

        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) { }
        });
    }
}
```

5. Press F11 to debug the application on the Android emulator. Click on the `SpinnerView` and you will see a pop-up displaying the list of presidents' names (see Figure 4-22). Clicking an item will display a message showing you the item selected.

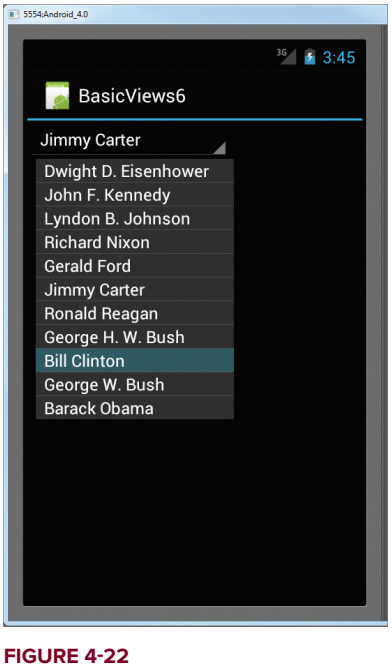


FIGURE 4-22

How It Works

The preceding example works very much like the `ListView`. One additional method you need to implement is the `onNothingSelected()` method. This method is fired when the user presses the back button, which dismisses the list of items displayed. In this case, nothing is selected so you do not need to do anything.

Instead of displaying the items in the `ArrayAdapter` as a simple list, you can also display them using radio buttons. To do so, modify the second parameter in the constructor of the `ArrayAdapter` class:

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_single_choice, presidents);
```

This causes the items to be displayed as a list of radio buttons (see Figure 4-23).

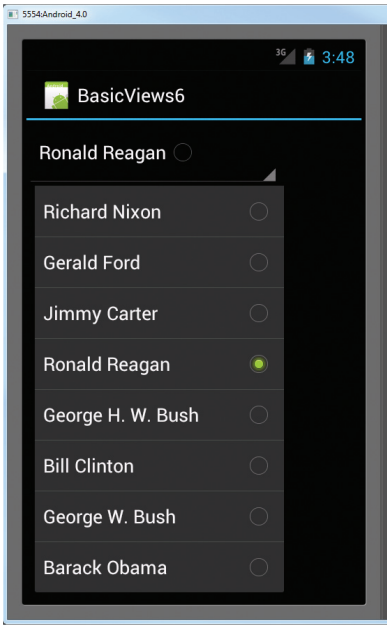


FIGURE 4-23

UNDERSTANDING SPECIALIZED FRAGMENTS

In Chapter 2, you learned about the fragment feature that is available beginning with Android 3. Using fragments, you can customize the user interface of your Android application by dynamically rearranging fragments to fit within an activity. This enables you to build applications that run on devices with different screen sizes.

As you have learned, fragments are really "mini-activities" that have their own life cycles. To create a fragment, you need a class that extends the `Fragment` base class. Besides the `Fragment` base class, you can also extend from some other subclasses of the `Fragment` base class to create more specialized fragments. The following sections discuss the three subclasses of `Fragment`: `ListFragment`, `DialogFragment`, and `PreferenceFragment`.

Using a ListFragment

A list fragment is a fragment that contains a `ListView`, displaying a list of items from a data source such as an array or a `Cursor`. A list fragment is very useful, as you may often have one fragment that contains a list of items (such as a list of RSS postings), and another fragment that displays

details about the selected posting. To create a list fragment, you need to extend the `ListFragment` base class.

The following Try It Out shows you how to get started with a list fragment.

TRY IT OUT Creating and Using a List Fragment

codefile ListFragmentExample.zip available for download at Wrox.com

1. Using Eclipse, create an Android project and name it **ListFragmentExample**.
2. Modify the `main.xml` file as shown in bold:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >

<fragment
    android:name="net.learn2develop.ListFragmentExample.Fragment1"
    android:id="@+id/fragment1"
    android:layout_weight="0.5"
    android:layout_width="0dp"
    android:layout_height="200dp" />

<fragment
    android:name="net.learn2develop.ListFragmentExample.Fragment1"
    android:id="@+id/fragment2"
    android:layout_weight="0.5"
    android:layout_width="0dp"
    android:layout_height="300dp" />

</LinearLayout>
```

3. Add an XML file to the `res/layout` folder and name it `fragment1.xml`.
4. Populate the `fragment1.xml` as follows:

```
<?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">
    <ListView
        android:id="@id/android:list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:drawSelectorOnTop="false"/>
</LinearLayout>
```

5. Add a Java Class file to the package and name it `Fragment1`.

6. Populate the `Fragment1.java` file as follows:

```
package net.learn2develop.ListFragmentExample;

import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

public class Fragment1 extends ListFragment {
    String[] presidents = {
        "Dwight D. Eisenhower",
        "John F. Kennedy",
        "Lyndon B. Johnson",
        "Richard Nixon",
        "Gerald Ford",
        "Jimmy Carter",
        "Ronald Reagan",
        "George H. W. Bush",
        "Bill Clinton",
        "George W. Bush",
        "Barack Obama"
    };

    @Override
    public View onCreateView(LayoutInflater inflater,
    ViewGroup container, Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment1, container, false);
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setListAdapter(new ArrayAdapter<String>(getActivity(),
            android.R.layout.simple_list_item_1, presidents));
    }

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

}
```

7. Press F11 to debug the application on the Android emulator. Figure 4-24 shows the two list fragments displaying the two lists of presidents' names.

8. Click on any of the items in the two `ListView` views, and a message is displayed (see Figure 4-25).

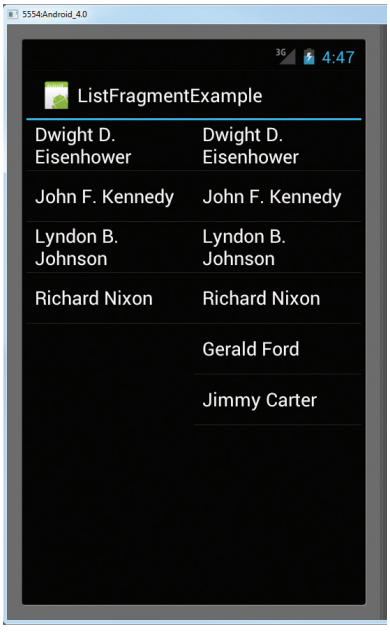


FIGURE 4-24

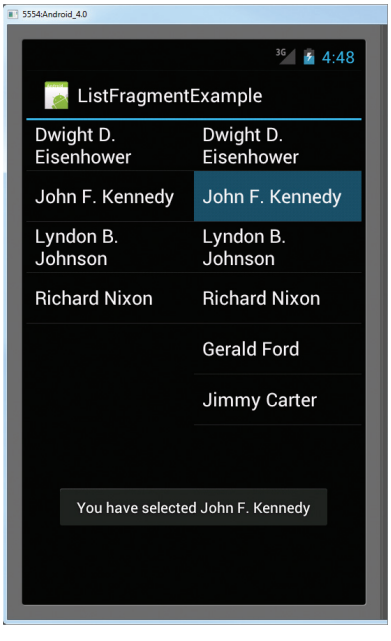


FIGURE 4-25

How It Works

First, you created the XML file for the fragment by adding a `ListView` element to it:

```
<?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">
    <ListView
        android:id="@id/android:list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:drawSelectorOnTop="false"/>
</LinearLayout>
```

To create a list fragment, the Java class for the fragment must extend the `ListFragment` base class:

```
public class Fragment1 extends ListFragment {
}
```

You then declared an array to contain the list of presidents' names in your activity:

```
String[] presidents = {
    "Dwight D. Eisenhower",
    "John F. Kennedy",
    "Lyndon B. Johnson",
    "Richard Nixon",
    "Gerald Ford",
    "Jimmy Carter",
    "Ronald Reagan",
    "George H. W. Bush",
    "Bill Clinton",
    "George W. Bush",
    "Barack Obama"
};
```

In the `onCreate()` event, you use the `setListAdapter()` method to programmatically fill the `ListView` with the content of the array. The `ArrayAdapter` object manages the array of strings that will be displayed by the `ListView`. In this example, you set the `ListView` to display in the `simple_list_item_1` mode:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setListAdapter(new ArrayAdapter<String>(getActivity(),
        android.R.layout.simple_list_item_1, presidents));
}
```

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

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

Finally, you added two fragments to the activity. Note the height of each fragment:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >

<fragment
    android:name="net.learn2develop.ListFragmentExample.Fragment1"
    android:id="@+id/fragment1"
    android:layout_weight="0.5"
```

```
        android:layout_width="0dp"
        android:layout_height="200dp" />

    <fragment
        android:name="net.learn2develop.ListFragmentExample.Fragment1"
        android:id="@+id/fragment2"
        android:layout_weight="0.5"
        android:layout_width="0dp"
        android:layout_height="300dp" />

    </LinearLayout>
```

Using a DialogFragment

Another type of fragment that you can create is a dialog fragment. A dialog fragment floats on top of an activity and is displayed modally. Dialog fragments are useful for cases in which you need to obtain the user's response before continuing with execution. To create a dialog fragment, you need to extend the `DialogFragment` base class.

The following Try It Out shows how to create a dialog fragment.

TRY IT OUT Creating and Using a Dialog Fragment

codefile DialogFragmentExample.zip available for download at Wrox.com

1. Using Eclipse, create an Android project and name it **DialogFragmentExample**.
2. Add a Java Class file under the package and name it **Fragment1**.
3. Populate the `Fragment1.java` file as follows:

```
package net.learn2develop.DialogFragmentExample;

import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;

public class Fragment1 extends DialogFragment {

    static Fragment1 newInstance(String title) {
        Fragment1 fragment = new Fragment1();
        Bundle args = new Bundle();
        args.putString("title", title);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
```

```
        String title = getArguments().getString("title");
        return new AlertDialog.Builder(getActivity())
        .setIcon(R.drawable.ic_launcher)
        .setTitle(title)
        .setPositiveButton("OK",
                new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog,
                    int whichButton) {
                ((DialogFragmentExampleActivity)
                        getActivity()).doPositiveClick();
            }
        })
        .setNegativeButton("Cancel",
                new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog,
                    int whichButton) {
                ((DialogFragmentExampleActivity)
                        getActivity()).doNegativeClick();
            }
        }).create();
    }

}
```

4. Populate the `DialogFragmentExampleActivity.java` file as shown here in bold:

```
package net.learn2develop.DialogFragmentExample;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class DialogFragmentExampleActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Fragment1 dialogFragment = Fragment1.newInstance(
                "Are you sure you want to do this?");
            dialogFragment.show(getFragmentManager(), "dialog");
    }

    public void doPositiveClick() {
        //---perform steps when user clicks on OK---
        Log.d("DialogFragmentExample", "User clicks on OK");
    }

    public void doNegativeClick() {
        //---perform steps when user clicks on Cancel---
        Log.d("DialogFragmentExample", "User clicks on Cancel");
    }

}
```

5. Press F11 to debug the application on the Android emulator. Figure 4-26 shows the fragment displayed as an alert dialog. Click either the OK button or the Cancel button and observe the message displayed.

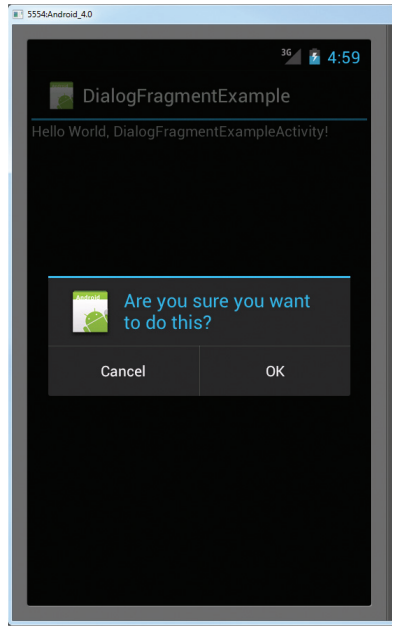


FIGURE 4-26

How It Works

To create a dialog fragment, first your Java class must extend the `DialogFragment` base class:

```
public class Fragment1 extends DialogFragment {
}
```

In this example, you created an alert dialog, which is a dialog window that displays a message with optional buttons. Within the `Fragment1` class, you defined the `newInstance()` method:

```
static Fragment1 newInstance(String title) {
    Fragment1 fragment = new Fragment1();
    Bundle args = new Bundle();
    args.putString("title", title);
    fragment.setArguments(args);
    return fragment;
}
```

The `newInstance()` method allows a new instance of the fragment to be created, and at the same time it accepts an argument specifying the string (`title`) to display in the alert dialog. The `title` is then stored in a `Bundle` object for use later.

Next, you defined the `onCreateDialog()` method, which is called after `onCreate()` and before `onCreateView()`:

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    String title = getArguments().getString("title");
    return new AlertDialog.Builder(getActivity())
    .setIcon(R.drawable.ic_launcher)
    .setTitle(title)
    .setPositiveButton("OK",
            new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,
                int whichButton) {
            ((DialogFragmentExampleActivity)
                    getActivity()).doPositiveClick();
        }
    })
    .setNegativeButton("Cancel",
            new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,
                int whichButton) {
            ((DialogFragmentExampleActivity)
                    getActivity()).doNegativeClick();
```

```
            }
        }).create();
    }
```

Here, you created an alert dialog with two buttons: OK and Cancel. The string to be displayed in it is obtained from the `title` argument saved in the `Bundle` object.

To display the dialog fragment, you created an instance of it and then called its `show()` method:

```
        Fragment1 dialogFragment = Fragment1.newInstance(
                "Are you sure you want to do this?");
        dialogFragment.show(getFragmentManager(), "dialog");
```

You also needed to implement two methods, `doPositiveClick()` and `doNegativeClick()`, to handle the user clicking the OK or Cancel buttons, respectively:

```
public void doPositiveClick() {
    //---perform steps when user clicks on OK---
    Log.d("DialogFragmentExample", "User clicks on OK");
}

public void doNegativeClick() {
    //---perform steps when user clicks on Cancel---
    Log.d("DialogFragmentExample", "User clicks on Cancel");
}
```

Using a PreferenceFragment

Your Android applications will typically provide preferences that allow users to personalize the application for their own use. For example, you may allow users to save the login credentials that they use to access their web resources, or save information such as how often the feeds must be refreshed (such as in an RSS reader application), and so on. In Android, you can use the `PreferenceActivity` base class to display an activity for the user to edit the preferences. In Android 3.0 and later, you can use the `PreferenceFragment` class to do the same thing.

The following Try It Out shows you how to create and use a preference fragment in Android 3 and 4.

TRY IT OUT Creating and Using a Preference Fragment

codefile PreferenceFragmentExample.zip available for download at Wrox.com

1. Using Eclipse, create an Android project and name it **PreferenceFragmentExample**.
2. Create a new `xml` folder under the `res` folder and then add a new Android XML file to it. Name the XML file `preferences.xml` (see Figure 4-27).

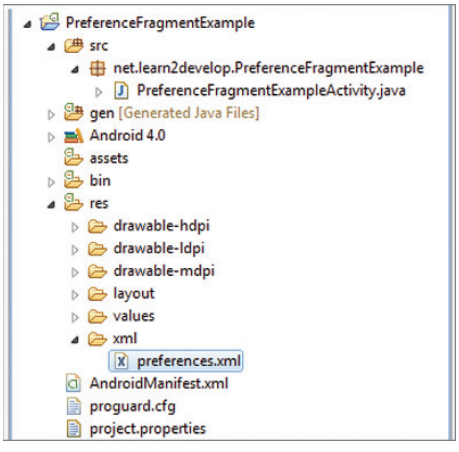


FIGURE 4-27

3. Populate the `preferences.xml` file as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">

    <PreferenceCategory android:title="Category 1">
        <CheckBoxPreference
            android:title="Checkbox"
            android:defaultValue="false"
            android:summary="True of False"
            android:key="checkboxPref" />
        </PreferenceCategory>

    <PreferenceCategory android:title="Category 2">
        <EditTextPreference
            android:name="EditText"
            android:summary="Enter a string"
            android:defaultValue="[Enter a string here]"
            android:title="Edit Text"
            android:key="editTextPref" />
        <RingtonePreference
            android:name="Ringtone Preference"
            android:summary="Select a ringtone"
            android:title="Ringtones"
            android:key="ringtonePref" />
        <PreferenceScreen
            android:title="Second Preference Screen"
            android:summary=
                "Click here to go to the second Preference Screen"
            android:key="secondPrefScreenPref">
            <EditTextPreference
                android:name="EditText"
```

```
            android:summary="Enter a string"
            android:title="Edit Text (second Screen)"
            android:key="secondEditTextPref" />
        </PreferenceScreen>
    </PreferenceCategory>

</PreferenceScreen>
```

4. Add a Java Class file to the package and name it `Fragment1`.

5. Populate the `Fragment1.java` file as follows:

```
package net.learn2develop.PreferenceFragmentExample;

import android.os.Bundle;
import android.preference.PreferenceFragment;

public class Fragment1 extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        //---load the preferences from an XML file---
        addPreferencesFromResource(R.xml.preferences);
    }
}
```

6. Modify the `PreferenceFragmentExampleActivity.java` file as shown in bold:

```
package net.learn2develop.PreferenceFragmentExample;

import android.app.Activity;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.os.Bundle;

public class PreferenceFragmentExampleActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction fragmentTransaction =
            fragmentManager.beginTransaction();
        Fragment1 fragment1 = new Fragment1();
        fragmentTransaction.replace(android.R.id.content, fragment1);
        fragmentTransaction.addToBackStack(null);
        fragmentTransaction.commit();
    }
}
```

7. Press F11 to debug the application on the Android emulator. Figure 4-28 shows the preference fragment displaying the list of preferences that the user can modify.

8. When the Edit Text preference is clicked, a pop-up will be displayed (see Figure 4-29).

9. Clicking the Second Preference Screen item will cause a second preference screen to be displayed (see Figure 4-30).

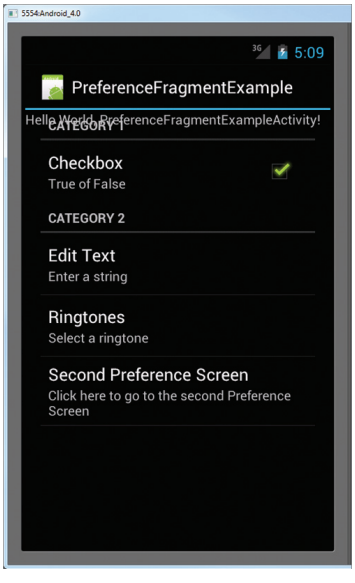


FIGURE 4-28

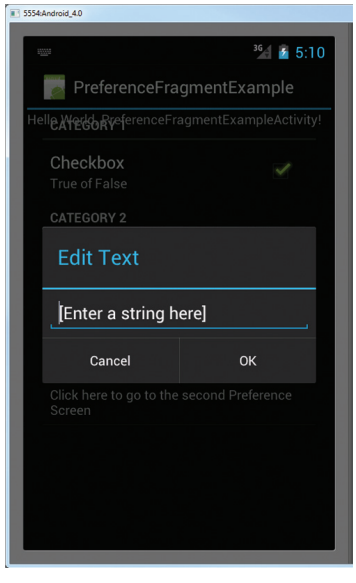


FIGURE 4-29

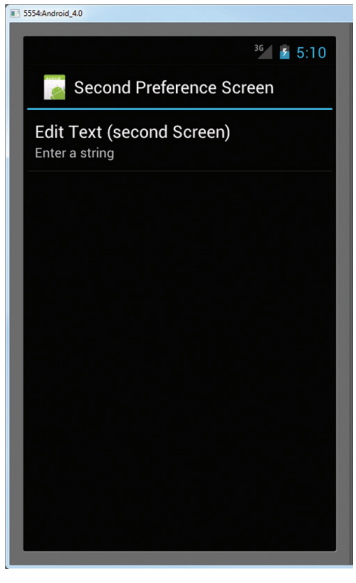


FIGURE 4-30

10. To cause the preference fragment to go away, click the back button on the emulator.

11. If you look at the File Explorer (available in the DDMS perspective), you will be able to locate the preferences file located in the `/data/data/net.learn2develop.PreferenceFragmentExample/shared_prefs/` folder (see Figure 4-31). All changes made by the user are persisted in this file.

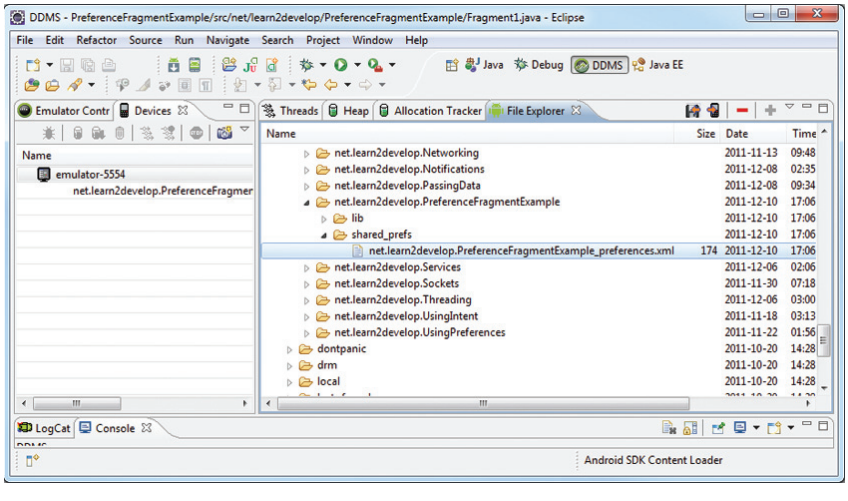


FIGURE 4-31

NOTE *Chapter 6 describes how to retrieve the values saved in a preference file.*

How It Works

To create a list of preferences in your Android application, you first needed to create the `preferences.xml` file and populate it with the various XML elements. This XML file defines the various items that you want to persist in your application.

To create the preference fragment, you needed to extend the `PreferenceFragment` base class:

```
public class Fragment1 extends PreferenceFragment {
}
```

To load the preferences file in the preference fragment, you use the `addPreferencesFromResource()` method:

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

    //---load the preferences from an XML file---
    addPreferencesFromResource(R.xml.preferences);
}
```

To display the preference fragment in your activity, you can make use of the `FragmentManager` and the `FragmentTransaction` classes:

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction =
    fragmentManager.beginTransaction();
Fragment1 fragment1 = new Fragment1();
fragmentTransaction.replace(android.R.id.content, fragment1);
fragmentTransaction.addToBackStack(null);
fragmentTransaction.commit();
```

You needed to add the preference fragment to the back stack using the `addToBackStack()` method so that the user can dismiss the fragment by clicking the back button.

SUMMARY

This chapter provided a brief look at some of the commonly used views in an Android application. While it is not possible to exhaustively examine each view in detail, the views you learned about here should provide a good foundation for designing your Android application's user interface, regardless of its requirements.

EXERCISES

1. How do you programmatically determine whether a `RadioButton` is checked?
2. How do you access the string resource stored in the `strings.xml` file?
3. Write the code snippet to obtain the current date.
4. Name the three specialized fragments you can use in your Android application and describe their uses.

Answers to the exercises can be found in Appendix C.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS

TextView	`<TextView`
	`android:layout_width="fill_parent"`
	`android:layout_height="wrap_content"`
	`android:text="@string/hello"`
	`/>`

Button	`<Button android:id="@+id/btnSave"`
	`android:layout_width="fill_parent"`
	`android:layout_height="wrap_content"`
	`android:text="Save" />`

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

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

CheckBox	`<CheckBox android:id="@+id/chkAutosave"`
	`android:layout_width="fill_parent"`
	`android:layout_height="wrap_content"`
	`android:text="Autosave" />`

RadioGroup and RadioButton	`<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"`

TOPIC	KEY CONCEPTS
	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	<ToggleButton android:id="@+id/toggle1" android:layout_width="wrap_content"
ProgressBar	<ProgressBar android:id="@+id/progressbar" android:layout_width="wrap_content" android:layout_height="wrap_content" />
AutoCompleteTextBox	<AutoCompleteTextView android:id="@+id/txtCountries" android:layout_width="fill_parent" android:layout_height="wrap_content" />
TimePicker	<TimePicker android:id="@+id/timePicker" android:layout_width="wrap_content" android:layout_height="wrap_content" />
DatePicker	<DatePicker android:id="@+id/datePicker" android:layout_width="wrap_content" android:layout_height="wrap_content" />
Spinner	<Spinner android:id="@+id/spinner1" android:layout_width="wrap_content" android:layout_height="wrap_content" android:drawSelectorOnTop="true" />
Specialized fragment types	ListFragment, DialogFragment, and PreferenceFragment

5

Displaying Pictures and Menus with Views

WHAT YOU WILL LEARN IN THIS CHAPTER

- How to use the Gallery, ImageSwitcher, GridView, and ImageView views to display images
- How to display options menus and context menus
- How to display time using the AnalogClock and DigitalClock views
- How to display web content using the WebView view

In the previous chapter, you learned about the various views that you can use to build the user interface of your Android application. In this chapter, you continue your exploration of the other views that you can use to create robust and compelling applications.

In particular, you will learn how to work with views that enable you to display images. In addition, you will learn how to create option and context menus in your Android application. This chapter ends with a discussion of some helpful views that enable users to display the current time and web content.

USING IMAGE VIEWS TO DISPLAY PICTURES

So far, all the views you have seen until this point are used to display text information. For displaying images, you can use the `ImageView`, `Gallery`, `ImageSwitcher`, and `GridView` views.

The following sections discuss each view in detail.

Gallery and ImageView Views

The `Gallery` is a view that shows items (such as images) in a center-locked, horizontal scrolling list. Figure 5-1 shows how the `Gallery` view looks when it is displaying some images.

FIGURE 5-1

The following Try It Out shows you how to use the `Gallery` view to display a set of images.

TRY IT OUT Using the Gallery View

codefile Gallery.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Gallery**.
2. Modify the `main.xml` file as shown in bold:

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

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Images of San Francisco" />

<Gallery
    android:id="@+id/gallery1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

<ImageView
    android:id="@+id/image1"
    android:layout_width="320dp"
    android:layout_height="250dp"
    android:scaleType="fitXY" />

</LinearLayout>
```

3. Right-click on the `res/values` folder and select New ➪ File. Name the file **attrs.xml**.
4. Populate the `attrs.xml` file as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="Gallery1">
        <attr name="android:galleryItemBackground" />
    </declare-styleable>
</resources>
```

5. Prepare a series of images and name them pic1.png, pic2.png, and so on for each subsequent image (see Figure 5-2).

FIGURE 5-2

NOTE *You can download the series of images from this book's support website at* `www.wrox.com`.

6. Drag and drop all the images into the `res/drawable-mdpi` folder (see Figure 5-3). When a dialog is displayed, check the Copy files option and click OK.

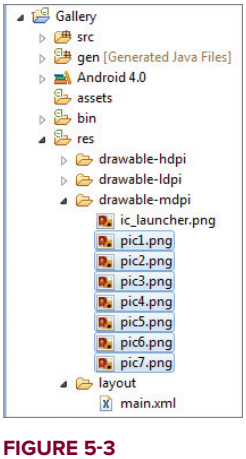


FIGURE 5-3

NOTE *This example assumes that this project will be tested on an AVD with medium DPI screen resolution. For a real-life project, you need to ensure that each* `drawable` *folder has a set of images (of different resolutions).*

7. Add the following statements in bold to the `GalleryActivity.java` file:

```
package net.learn2develop.Gallery;

import android.app.Activity;
import android.content.Context;
import android.content.res.TypedArray;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.BaseAdapter;
import android.widget.Gallery;
import android.widget.ImageView;
import android.widget.Toast;

public class GalleryActivity extends Activity {
    //---the images to display---
    Integer[] imageIDs = {
            R.drawable.pic1,
            R.drawable.pic2,
            R.drawable.pic3,
            R.drawable.pic4,
            R.drawable.pic5,
            R.drawable.pic6,
            R.drawable.pic7
    };

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Gallery gallery = (Gallery) findViewById(R.id.gallery1);

        gallery.setAdapter(new ImageAdapter(this));
        gallery.setOnItemClickListener(new OnItemClickListener()
        {
            public void onItemClick(AdapterView parent, View v,
            int position, long id)
            {
                Toast.makeText(getBaseContext(),
                        "pic" + (position + 1) + " selected",
                        Toast.LENGTH_SHORT).show();
            }
        });
    }

    public class ImageAdapter extends BaseAdapter
    {
        Context context;
```

```
        int itemBackground;

        public ImageAdapter(Context c)
        {
            context = c;
            //---setting the style---
            TypedArray a = obtainStyledAttributes(
                                R.styleable.Gallery1);
            itemBackground = a.getResourceId(
            R.styleable.Gallery1_android_galleryItemBackground,
            0);
            a.recycle();
        }

        //---returns the number of images---
        public int getCount() {
            return imageIDs.length;
        }

        //---returns the item---
        public Object getItem(int position) {
            return position;
        }

         //---returns the ID of an item---
        public long getItemId(int position) {
            return position;
        }

        //---returns an ImageView view---
        public View getView(int position, View convertView,
        ViewGroup parent) {
            ImageView imageView;
            if (convertView == null) {
                imageView = new ImageView(context);
                imageView.setImageResource(imageIDs[position]);
                imageView.setScaleType(
                    ImageView.ScaleType.FIT_XY);
                imageView.setLayoutParams(
                    new Gallery.LayoutParams(150, 120));
            } else {
                imageView = (ImageView) convertView;
            }
            imageView.setBackgroundResource(itemBackground);
            return imageView;
        }
    }

}
```

8. Press F11 to debug the application on the Android emulator. Figure 5-4 shows the `Gallery` view displaying the series of images. You can swipe the images to view the entire series. Observe that as you click on an image, the `Toast` class displays its name.

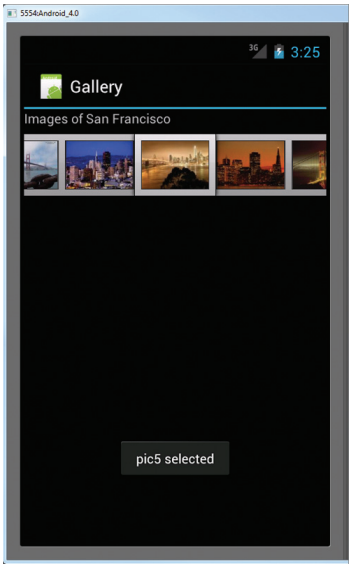


FIGURE 5-4

9. To display the selected image in the `ImageView`, add the following statements in bold to the `GalleryActivity.java` file:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Gallery gallery = (Gallery) findViewById(R.id.gallery1);

    gallery.setAdapter(new ImageAdapter(this));
    gallery.setOnItemClickListener(new OnItemClickListener()
    {
        public void onItemClick(AdapterView parent, View v,
        int position, long id)
        {
            Toast.makeText(getBaseContext(),
                    "pic" + (position + 1) + " selected",
                    Toast.LENGTH_SHORT).show();

            //---display the images selected---
            ImageView imageView =
                (ImageView) findViewById(R.id.image1);
            imageView.setImageResource(imageIDs[position]);
        }
    });
}
```

10. Press F11 to debug the application again. This time, the image selected will be displayed in the `ImageView` (see Figure 5-5).

How It Works

You first added the `Gallery` and `ImageView` views to `main.xml`:

```
<Gallery
    android:id="@+id/gallery1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

<ImageView
    android:id="@+id/image1"
    android:layout_width="320dp"
    android:layout_height="250dp"
    android:scaleType="fitXY" />
```

As mentioned earlier, the `Gallery` view is used to display a series of images in a horizontal scrolling list. The `ImageView` is used to display the image selected by the user.

The list of images to be displayed is stored in the `imageIDs` array:

```
//---the images to display---
Integer[] imageIDs = {
        R.drawable.pic1,
        R.drawable.pic2,
        R.drawable.pic3,
        R.drawable.pic4,
        R.drawable.pic5,
        R.drawable.pic6,
        R.drawable.pic7
};
```

FIGURE 5-5

You create the `ImageAdapter` class (which extends the `BaseAdapter` class) so that it can bind to the `Gallery` view with a series of `ImageView` views. The `BaseAdapter` class acts as a bridge between an `AdapterView` and the data source that feeds data into it. Examples of `AdapterViews` are as follows:

- `ListView`
- `GridView`
- `Spinner`
- `Gallery`

There are several subclasses of the `BaseAdapter` class in Android:

- `ListAdapter`
- `ArrayAdapter`
- `CursorAdapter`
- `SpinnerAdapter`

For the `ImageAdapter` class, you implemented the following methods in bold:

```
public class ImageAdapter extends BaseAdapter {
    public ImageAdapter(Context c) { ... }

    //---returns the number of images---
    public int getCount() { ... }

    //---returns the item---
    public Object getItem(int position) { ... }

     //---returns the ID of an item---
    public long getItemId(int position) { ... }

    //---returns an ImageView view---
    public View getView(int position, View convertView,
    ViewGroup parent) { ... }
}
```

In particular, the `getView()` method returns a `View` at the specified position. In this case, you returned an `ImageView` object.

When an image in the `Gallery` view is selected (i.e., clicked), the selected image's position (0 for the first image, 1 for the second image, and so on) is displayed and the image is displayed in the `ImageView`:

```
Gallery gallery = (Gallery) findViewById(R.id.gallery1);

gallery.setAdapter(new ImageAdapter(this));
gallery.setOnItemClickListener(new OnItemClickListener()
{
    public void onItemClick(AdapterView<?> parent, View v,
    int position, long id)
    {
        Toast.makeText(getBaseContext(),
                "pic" + (position + 1) + " selected",
                Toast.LENGTH_SHORT).show();

        //---display the images selected---
        ImageView imageView =
            (ImageView) findViewById(R.id.image1);
        imageView.setImageResource(imageIDs[position]);
    }
});
```

ImageSwitcher

The previous section demonstrated how to use the `Gallery` view together with an `ImageView` to display a series of thumbnail images so that when one is selected, it is displayed in the `ImageView`. However, sometimes you don't want an image to appear abruptly when the user selects it in the `Gallery` view — you might, for example, want to apply some animation to the image when it

transitions from one image to another. In this case, you need to use the `ImageSwitcher` together with the `Gallery` view. The following Try It Out shows you how.

TRY IT OUT Using the ImageSwitcher View

codefile ImageSwitcher.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **ImageSwitcher**.
2. Modify the `main.xml` file by adding the following statements in bold:

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

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Images of San Francisco" />

<Gallery
    android:id="@+id/gallery1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

<ImageSwitcher
    android:id="@+id/switcher1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentBottom="true" />

</LinearLayout>
```

3. Right-click on the `res/values` folder and select New ➪ File. Name the file **attrs.xml**.
4. Populate the `attrs.xml` file as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="Gallery1">
        <attr name="android:galleryItemBackground" />
    </declare-styleable>
</resources>
```

5. Drag and drop a series of images into the `res/drawable-mdpi` folder (refer to the previous example for the images). When a dialog is displayed, check the Copy files option and click OK.

6. Add the following bold statements to the `ImageSwitcherActivity.java` file:

```
package net.learn2develop.ImageSwitcher;

import android.app.Activity;
import android.content.Context;
import android.content.res.TypedArray;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.view.ViewGroup.LayoutParams;
import android.view.animation.AnimationUtils;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.BaseAdapter;
import android.widget.Gallery;
import android.widget.ImageSwitcher;
import android.widget.ImageView;
import android.widget.ViewSwitcher.ViewFactory;

public class ImageSwitcherActivity extends Activity implements ViewFactory {
    //---the images to display---
    Integer[] imageIDs = {
            R.drawable.pic1,
            R.drawable.pic2,
            R.drawable.pic3,
            R.drawable.pic4,
            R.drawable.pic5,
            R.drawable.pic6,
            R.drawable.pic7
    };

    private ImageSwitcher imageSwitcher;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        imageSwitcher = (ImageSwitcher) findViewById(R.id.switcher1);
        imageSwitcher.setFactory(this);
        imageSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
                android.R.anim.fade_in));
        imageSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
                android.R.anim.fade_out));

        Gallery gallery = (Gallery) findViewById(R.id.gallery1);
        gallery.setAdapter(new ImageAdapter(this));
        gallery.setOnItemClickListener(new OnItemClickListener()
        {
            public void onItemClick(AdapterView<?> parent,
            View v, int position, long id)
            {
                imageSwitcher.setImageResource(imageIDs[position]);
```

```
        }
    });
}

public View makeView()
{
    ImageView imageView = new ImageView(this);
    imageView.setBackgroundColor(0xFF000000);
    imageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
    imageView.setLayoutParams(new
            ImageSwitcher.LayoutParams(
                    LayoutParams.FILL_PARENT,
                    LayoutParams.FILL_PARENT));
    return imageView;
}

public class ImageAdapter extends BaseAdapter
{
    private Context context;
    private int itemBackground;

    public ImageAdapter(Context c)
    {
        context = c;

        //---setting the style---
        TypedArray a = obtainStyledAttributes(R.styleable.Gallery1);
        itemBackground = a.getResourceId(
                R.styleable.Gallery1_android_galleryItemBackground, 0);
        a.recycle();
    }

    //---returns the number of images---
    public int getCount()
    {
        return imageIDs.length;
    }

    //---returns the item---
    public Object getItem(int position)
    {
        return position;
    }

    //---returns the ID of an item---
    public long getItemId(int position)
    {
        return position;
    }

    //---returns an ImageView view---
    public View getView(int position, View convertView, ViewGroup parent)
    {
        ImageView imageView = new ImageView(context);

        imageView.setImageResource(imageIDs[position]);
```

```
            imageView.setScaleType(ImageView.ScaleType.FIT_XY);
            imageView.setLayoutParams(new Gallery.LayoutParams(150, 120));
            imageView.setBackgroundResource(itemBackground);

            return imageView;
        }
    }

}
```

7. Press F11 to debug the application on the Android emulator. Figure 5-6 shows the `Gallery` and `ImageSwitcher` views, with both the collection of images as well as the image selected.

FIGURE 5-6

How It Works

The first thing to note in this example is that the `ImageSwitcherActivity` not only extends `Activity`, but also implements `ViewFactory`. To use the `ImageSwitcher` view, you need to implement the `ViewFactory` interface, which creates the views for use with the `ImageSwitcher` view. For this, you need to implement the `makeView()` method:

```
public View makeView()
{
    ImageView imageView = new ImageView(this);
    imageView.setBackgroundColor(0xFF000000);
    imageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
    imageView.setLayoutParams(new
            ImageSwitcher.LayoutParams(
                    LayoutParams.FILL_PARENT,
                    LayoutParams.FILL_PARENT));
    return imageView;
}
```

This method creates a new `View` to be added in the `ImageSwitcher` view, which in this case is an `ImageView`.

Like the `Gallery` example in the previous section, you also implemented an `ImageAdapter` class so that it can bind to the `Gallery` view with a series of `ImageView` views.

In the `onCreate()` method, you get a reference to the `ImageSwitcher` view and set the animation, specifying how images should "fade" in and out of the view. Finally, when an image is selected from the `Gallery` view, the image is displayed in the `ImageSwitcher` view:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    imageSwitcher = (ImageSwitcher) findViewById(R.id.switcher1);
    imageSwitcher.setFactory(this);
    imageSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
```

```
            android.R.anim.fade_in));
        imageSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
            android.R.anim.fade_out));

        Gallery gallery = (Gallery) findViewById(R.id.gallery1);
        gallery.setAdapter(new ImageAdapter(this));
        gallery.setOnItemClickListener(new OnItemClickListener()
        {
            public void onItemClick(AdapterView<?> parent,
            View v, int position, long id)
            {
                imageSwitcher.setImageResource(imageIDs[position]);
            }
        });
    }
```

In this example, when an image is selected in the `Gallery` view, it appears by "fading" in. When the next image is selected, the current image fades out. If you want the image to slide in from the left and slide out to the right when another image is selected, try the following animation:

```
imageSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
        android.R.anim.slide_in_left));
imageSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
        android.R.anim.slide_out_right));
```

GridView

The `GridView` shows items in a two-dimensional scrolling grid. You can use the `GridView` together with an `ImageView` to display a series of images. The following Try It Out demonstrates how.

TRY IT OUT Using the GridView View

codefile Grid.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Grid**.
2. Drag and drop a series of images into the `res/drawable-mdpi` folder (see the previous example for the images). When a dialog is displayed, check the Copy files option and click OK.
3. Populate the `main.xml` file with the following content:

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

<GridView
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:columnWidth="90dp"
    android:stretchMode="columnWidth"
    android:gravity="center" />

</LinearLayout>
```

4. Add the following statements in bold to the `GridActivity.java` file:

```
package net.learn2develop.Grid;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;
import android.widget.Toast;

public class GridActivity extends Activity {
    //---the images to display---
    Integer[] imageIDs = {
            R.drawable.pic1,
            R.drawable.pic2,
            R.drawable.pic3,
            R.drawable.pic4,
            R.drawable.pic5,
            R.drawable.pic6,
            R.drawable.pic7
    };

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        GridView gridView = (GridView) findViewById(R.id.gridview);
        gridView.setAdapter(new ImageAdapter(this));

        gridView.setOnItemClickListener(new OnItemClickListener()
        {
            public void onItemClick(AdapterView parent,
            View v, int position, long id)
            {
                Toast.makeText(getBaseContext(),
                        "pic" + (position + 1) + " selected",
                        Toast.LENGTH_SHORT).show();
            }
```

```
        });

    }

    public class ImageAdapter extends BaseAdapter
    {
        private Context context;

        public ImageAdapter(Context c)
        {
            context = c;
        }

        //---returns the number of images---
        public int getCount() {
            return imageIDs.length;
        }

        //---returns the item---
        public Object getItem(int position) {
            return position;
        }

        //---returns the ID of an item---
        public long getItemId(int position) {
            return position;
        }

        //---returns an ImageView view---
        public View getView(int position, View convertView,
        ViewGroup parent)
        {
            ImageView imageView;
            if (convertView == null) {
                imageView = new ImageView(context);
                imageView.setLayoutParams(new
                    GridView.LayoutParams(85, 85));
                imageView.setScaleType(
                    ImageView.ScaleType.CENTER_CROP);
                imageView.setPadding(5, 5, 5, 5);
            } else {
                imageView = (ImageView) convertView;
            }
            imageView.setImageResource(imageIDs[position]);
            return imageView;
        }
    }

}
```

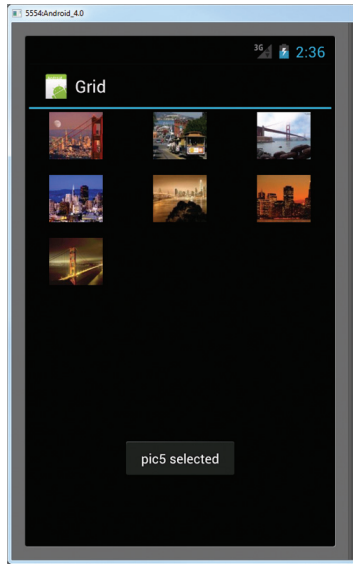


FIGURE 5-7

5. Press F11 to debug the application on the Android emulator. Figure 5-7 shows the `GridView` displaying all the images.

How It Works

Like the `Gallery` and `ImageSwitcher` example, you implemented the `ImageAdapter` class and then bound it to the `GridView`:

```
GridView gridView = (GridView) findViewById(R.id.gridview);
gridView.setAdapter(new ImageAdapter(this));

gridView.setOnItemClickListener(new OnItemClickListener()
{
    public void onItemClick(AdapterView parent,
    View v, int position, long id)
    {
        Toast.makeText(getBaseContext(),
                "pic" + (position + 1) + " selected",
                Toast.LENGTH_SHORT).show();
    }
});
```

When an image is selected, you display a `Toast` message indicating the selected image.

Within the `getView()` method you can specify the size of the images and how images are spaced in the `GridView` by setting the padding for each image:

```
//---returns an ImageView view---
public View getView(int position, View convertView,
ViewGroup parent)
{
    ImageView imageView;
    if (convertView == null) {
        imageView = new ImageView(context);
        imageView.setLayoutParams(new
            GridView.LayoutParams(85, 85));
        imageView.setScaleType(
            ImageView.ScaleType.CENTER_CROP);
        imageView.setPadding(5, 5, 5, 5);
    } else {
        imageView = (ImageView) convertView;
    }
    imageView.setImageResource(imageIDs[position]);
    return imageView;
}
}
```

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 button.

- **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.

Figure 5-8 shows an example of an options menu in the Browser application. The options menu is displayed whenever the user presses the MENU button. The menu items displayed vary according to the current activity that is running.

Figure 5-9 shows a context menu that is displayed when the user taps and holds on an image displayed on the page. The menu items displayed vary according to the component or view currently selected. In general, to activate the context menu, the user selects an item on the screen and taps and holds it.

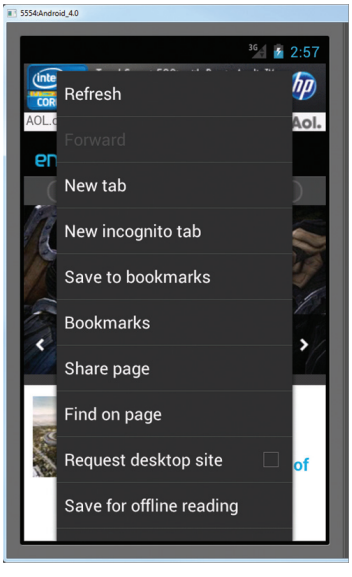


FIGURE 5-8

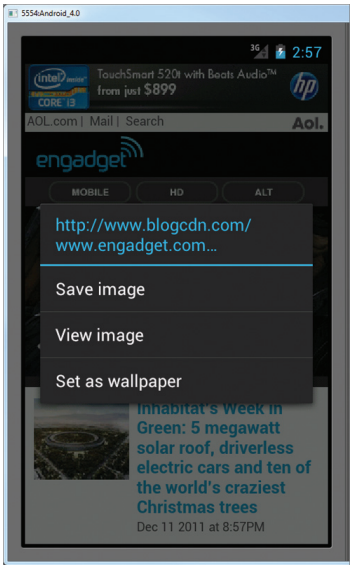


FIGURE 5-9

Creating the Helper Methods

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.

TRY IT OUT Creating the Menu Helper Methods

codefile Menus.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Menus**.
2. In the `MenusActivity.java` file, add the following statements in bold:

```
package net.learn2develop.Menus;

import android.app.Activity;
```

```
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class MenusActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    private void CreateMenu(Menu menu)
    {
        MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");
        {
            mnu1.setAlphabeticShortcut('a');
            mnu1.setIcon(R.drawable.ic_launcher);
        }
        MenuItem mnu2 = menu.add(0, 1, 1, "Item 2");
        {
            mnu2.setAlphabeticShortcut('b');
            mnu2.setIcon(R.drawable.ic_launcher);
        }
        MenuItem mnu3 = menu.add(0, 2, 2, "Item 3");
        {
            mnu3.setAlphabeticShortcut('c');
            mnu3.setIcon(R.drawable.ic_launcher);
        }
        MenuItem mnu4 = menu.add(0, 3, 3, "Item 4");
        {
            mnu4.setAlphabeticShortcut('d');
        }
        menu.add(0, 4, 4, "Item 5");
        menu.add(0, 5, 5, "Item 6");
        menu.add(0, 6, 6, "Item 7");
    }

    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:
            Toast.makeText(this, "You clicked on Item 2",
                Toast.LENGTH_LONG).show();
            return true;
        case 2:
            Toast.makeText(this, "You clicked on Item 3",
                Toast.LENGTH_LONG).show();
            return true;
        case 3:
```

```
                Toast.makeText(this, "You clicked on Item 4",
                    Toast.LENGTH_LONG).show();
                return true;
            case 4:
                Toast.makeText(this, "You clicked on Item 5",
                    Toast.LENGTH_LONG).show();
                return true;
            case 5:
                Toast.makeText(this, "You clicked on Item 6",
                    Toast.LENGTH_LONG).show();
                return true;
            case 6:
                Toast.makeText(this, "You clicked on Item 7",
                    Toast.LENGTH_LONG).show();
                return true;
            }
            return false;
        }

    }
```

How It Works

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()` method of the `Menu` object:

```
MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");
{
    mnu1.setAlphabeticShortcut('a');
    mnu1.setIcon(R.drawable.ic_launcher);
}
```

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` — A unique item ID
- `order` — The order in which the item should be displayed
- `title` — The text to display for the menu item

You can use the `setAlphabeticShortcut()` method to assign a shortcut key to the menu item so that users can select an item by pressing a key on the keyboard. 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 selected. It then displays a `Toast` message to let the user know which menu item was selected.

Options Menu

You are now ready to modify the application to display the options menu when the user presses the MENU key on the Android device.

TRY IT OUT Displaying an Options Menu

1. Using the same project created in the previous section, add the following statements in bold to the `MenusActivity.java` file:

```
package net.learn2develop.Menus;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class MenusActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    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)
    {
        //...
    }

    private boolean MenuChoice(MenuItem item)
    {
        //...
    }
}
```

2. Press F11 to debug the application on the Android emulator. Figure 5-10 shows the options menu that pops up when you click the MENU button. To select a menu item, either click on an

individual item or use its shortcut key (A to D; applicable only to the first four items). Note that menu items 1 to 3 did not display the icons even though the code explicitly did so.

3. If you now change the minimum SDK attribute of the `AndroidManifest.xml` file to a value of 10 or less and then rerun the application on the emulator, the icons will be displayed as shown in Figure 5-11. Note that any menu items after the fifth item are encapsulated in the item named More. Clicking on More will reveal the rest of the menu items.

```
<uses-sdk android:minSdkVersion="10" />
```

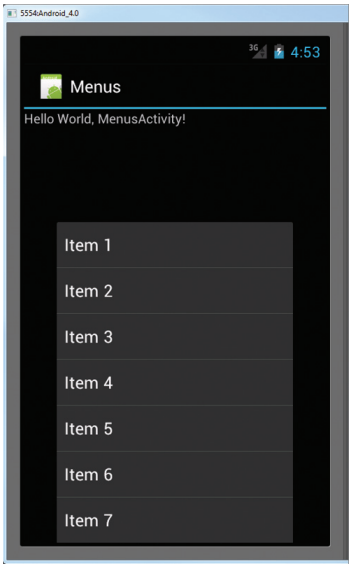


FIGURE 5-10

FIGURE 5-11

How It Works

To display the options menu for your activity, you need to implement two methods in your activity: `onCreateOptionsMenu()` and `onOptionsItemSelected()`. The `onCreateOptionsMenu()` method is called when the MENU button is pressed. In this case, 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 perform whatever action is appropriate).

Take note of the look and feel of the options menu in different versions of Android. Starting with Honeycomb, the options menu items do not have icons and display all menu items in a scrollable list. For versions of Android before Honeycomb, no more than five menu items are displayed; any additional menu items are part of a "More" menu item that represents the rest of the menu items.

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 taps and holds an item. For example, if the user taps on a `Button` view and holds 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. The following Try It Out shows how you can associate a context menu with a Button view.

TRY IT OUT Displaying a Context Menu

codefile Menus.zip available for download at Wrox.com

1. Using the same project from the previous example, add the following statements to the `main.xml` file:

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

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Click and hold on it" />

</LinearLayout>
```

2. Add the following statements in bold to the `MenusActivity.java` file:

```
package net.learn2develop.Menus;

import android.app.Activity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
```

```
import android.widget.Toast;

public class MenusActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Button btn = (Button) findViewById(R.id.button1);
        btn.setOnCreateContextMenuListener(this);
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View view,
    ContextMenuInfo menuInfo)
    {
         super.onCreateContextMenu(menu, view, menuInfo);
         CreateMenu(menu);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        //...
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
         return MenuChoice(item);
    }

    private void CreateMenu(Menu menu)
    {
        //...
    }

    private boolean MenuChoice(MenuItem item)
    {
        //...
    }
}
```

NOTE *If you changed the minimum SDK attribute of the* `AndroidManifest.xml` *file to a value of 10 earlier, be sure to change it back to 14 before you debug your application in the next step.*

3. Press F11 to debug the application on the Android emulator. Figure 5-12 shows the context menu that is displayed when you click and hold the `Button` view.

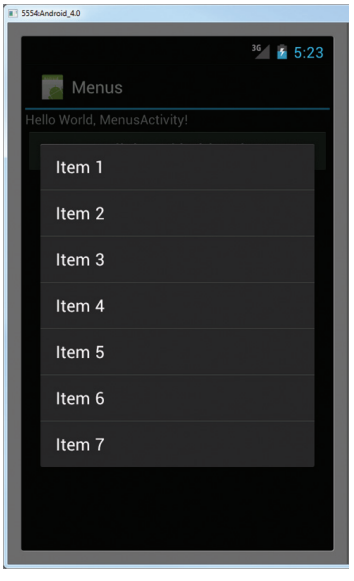


FIGURE 5-12

How It Works

In the preceding example, you call the `setOnCreateContextMenuListener()` method of the `Button` view to associate it with a context menu.

When the user taps and holds 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.

Notice that the shortcut keys for the menu items do not work. To enable the shortcuts keys, you need to call the `setQuertyMode()` method of the `Menu` object, like this:

```
private void CreateMenu(Menu menu)
{
    menu.setQwertyMode(true);
    MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");
    {
        mnu1.setAlphabeticShortcut('a');
        mnu1.setIcon(R.drawable.ic_launcher);
    }
    //...
}
```

SOME ADDITIONAL VIEWS

Besides the standard views that you have seen up to this point, the Android SDK provides some additional views that make your applications much more interesting. In this section, you will learn more about the following views: `AnalogClock`, `DigitalClock`, and `WebView`.

AnalogClock and DigitalClock Views

The `AnalogClock` view displays an analog clock with two hands — one for minutes and one for hours. Its counterpart, the `DigitalClock` view, displays the time digitally. Both views display the system time only, and do not allow you to display a particular time (such as the current time in another time zone). Hence, if you want to display the time for a particular region, you have to build your own custom views.

NOTE *Creating your own custom views in Android is beyond the scope of this book. However, if you are interested in this area, take a look at Google's Android documentation on this topic at* `http://developer.android.com/guide/topics/ui/custom-components.html`.

Using the `AnalogClock` and `DigitalClock` views are straightforward; simply declare them in your XML file (such as `main.xml`), like this:

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

<AnalogClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

<DigitalClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

</LinearLayout>
```

FIGURE 5-13

Figure 5-13 shows the `AnalogClock` and `DigitalClock` views in action.

WebView

The `WebView` enables you to embed a web browser in your activity. This is very useful if your application needs to embed some web content, such as maps from some other providers, and so on. The following Try It Out shows how you can programmatically load the content of a web page and display it in your activity.

TRY IT OUT **Using the WebView View**

codefile WebView.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **WebView**.
2. Add the following statements to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<WebView android:id="@+id/webview1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

</LinearLayout>
```

3. In the `WebViewActivity.java` file, add the following statements in bold:

```
package net.learn2develop.WebView;

import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;

public class WebViewActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        WebView wv = (WebView) findViewById(R.id.webview1);

        WebSettings webSettings = wv.getSettings();
        webSettings.setBuiltInZoomControls(true);
        wv.loadUrl(
            "http://chart.apis.google.com/chart" +
            "?chs=300x225" +
            "&cht=v" +
            "&chco=FF6342,ADDE63,63C6DE" +
            "&chd=t:100,80,60,30,30,30,10" +
            "&chdl=A|B|C");
    }
}
```

4. In the `AndroidManifest.xml` file, add the following permission:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.WebView"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />
    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
```

```
            android:label="@string/app_name"
            android:name=".WebViewActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

5. Press F11 to debug the application on the Android emulator. Figure 5-14 shows the content of the `WebView`.

FIGURE 5-14

How It Works

To use the `WebView` to load a web page, you use the `loadUrl()` method and pass it a URL, like this:

```
wv.loadUrl(
    "http://chart.apis.google.com/chart" +
    "?chs=300x225" +
    "&cht=v" +
    "&chco=FF6342,ADDE63,63C6DE" +
    "&chd=t:100,80,60,30,30,30,10" +
    "&chdl=A|B|C");
```

To display the built-in zoom controls, you need to first get the `WebSettings` property from the `WebView` and then call its `setBuiltInZoomControls()` method:

```
WebSettings webSettings = wv.getSettings();
webSettings.setBuiltInZoomControls(true);
```

Figure 5-15 shows the built-in zoom controls that appear when you use the mouse to click and drag the content of the `WebView` on the Android emulator.

NOTE *While most Android devices support multi-touch screens, the built-in zoom controls are useful for zooming your web content when testing your application on the Android emulator.*

Sometimes when you load a page that redirects you (for example, loading `www.wrox.com` redirects you to `www.wrox.com/wileyCDA`), `WebView` will cause your application to launch the device's Browser application to load the desired page. In Figure 5-16, note the URL bar at the top of the screen.

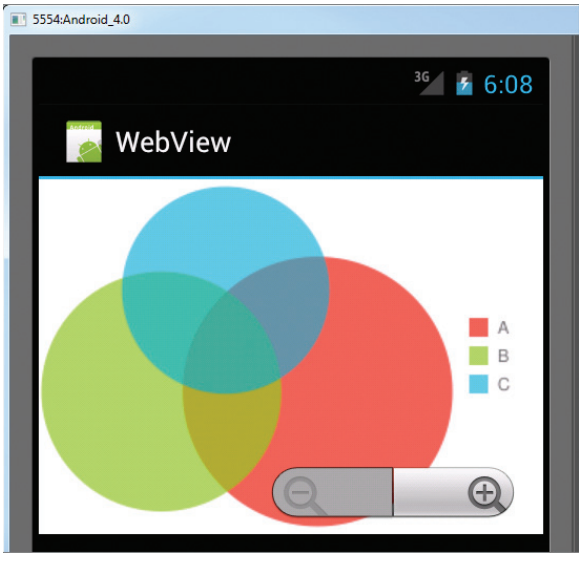


FIGURE 5-15

FIGURE 5-16

To prevent this from happening, you need to implement the `WebViewClient` class and override the `shouldOverrideUrlLoading()` method, as shown in the following example:

```
package net.learn2develop.WebView;

import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class WebViewActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        WebView wv = (WebView) findViewById(R.id.webview1);

        WebSettings webSettings = wv.getSettings();
        webSettings.setBuiltInZoomControls(true);
        wv.setWebViewClient(new Callback());
        wv.loadUrl("http://www.wrox.com");
```

```
        }

        private class Callback extends WebViewClient {
            @Override
            public boolean shouldOverrideUrlLoading(
            WebView view, String url) {
                return(false);
            }
        }

    }
```

Figure 5-17 shows the Wrox.com home page now loading correctly in the `WebView`.

FIGURE 5-17

You can also dynamically formulate an HTML string and load it into the `WebView`, using the `loadDataWithBaseURL()` method:

```
WebView wv = (WebView) findViewById(R.id.webview1);
final String mimeType = "text/html";
final String encoding = "UTF-8";
String html = "<H1>A simple HTML page</H1><body>" +
    "<p>The quick brown fox jumps over the lazy dog</p>" +
    "</body>";
wv.loadDataWithBaseURL("", html, mimeType, encoding, "");
```

Figure 5-18 shows the content displayed by the `WebView`.

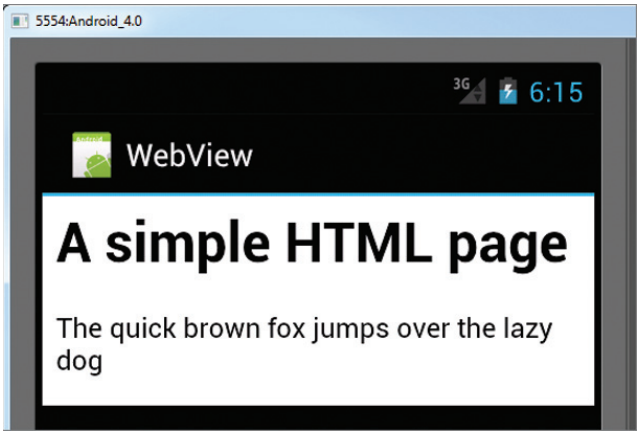


FIGURE 5-18

Alternatively, if you have an HTML file located in the `assets` folder of the project (see Figure 5-19), you can load it into the `WebView` using the `loadUrl()` method:

```
WebView wv = (WebView) findViewById(R.id.webview1);
wv.loadUrl("file:///android_asset/Index.html");
```

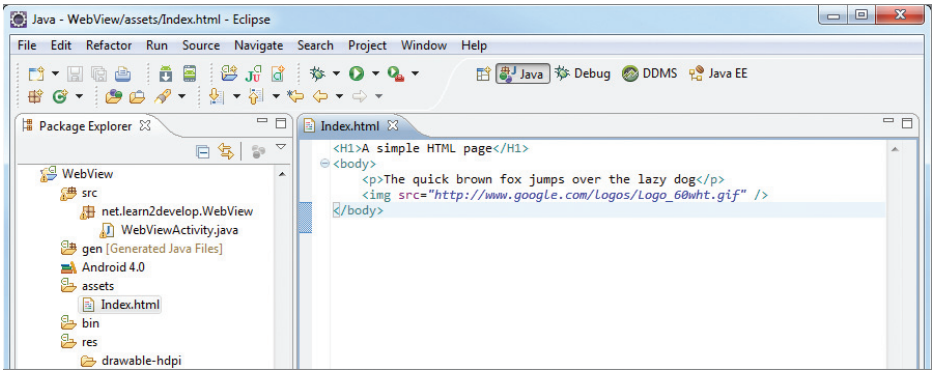


FIGURE 5-19

Figure 5-20 shows the content of the `WebView`.

FIGURE 5-20

SUMMARY

In this chapter, you have taken a look at the various views that enable you to display images: `Gallery`, `ImageView`, `ImageSwitcher`, and `GridView`. In addition, you learned about the difference between options menus and context menus, and how to display them in your application. Finally, you learned about the `AnalogClock` and `DigitalClock` views, which display the current time graphically, as well as the `WebView`, which displays the content of a web page.

EXERCISES

1. What is the purpose of the `ImageSwitcher`?
2. Name the two methods you need to override when implementing an options menu in your activity.
3. Name the two methods you need to override when implementing a context menu in your activity.
4. How do you prevent the `WebView` from invoking the device's web browser when a redirection occurs in the `WebView`?

Answers to the exercises can be found in Appendix C.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Using the `Gallery` view	Displays a series of images in a horizontal scrolling list

`Gallery`	`<Gallery`
	`android:id="@+id/gallery1"`
	`android:layout_width="fill_parent"`
	`android:layout_height="wrap_content" />`

`ImageView`	`<ImageView`
	`android:id="@+id/image1"`
	`android:layout_width="320px"`
	`android:layout_height="250px"`
	`android:scaleType="fitXY" />`

Using the `ImageSwitcher` view	Performs animation when switching between images

`ImageSwitcher`	`<ImageSwitcher`
	`android:id="@+id/switcher1"`
	`android:layout_width="fill_parent"`
	`android:layout_height="fill_parent"`
	`android:layout_alignParentLeft="true"`
	`android:layout_alignParentRight="true"`
	`android:layout_alignParentBottom="true" />`

Using the `GridView`	Shows items in a two-dimensional scrolling grid

`GridView`	`<GridView`
	`android:id="@+id/gridview"`
	`android:layout_width="fill_parent"`
	`android:layout_height="fill_parent"`
	`android:numColumns="auto_fit"`
	`android:verticalSpacing="10dp"`
	`android:horizontalSpacing="10dp"`
	`android:columnWidth="90dp"`
	`android:stretchMode="columnWidth"`
	`android:gravity="center" />`

`AnalogClock`	`<AnalogClock`
	`android:layout_width="wrap_content"`
	`android:layout_height="wrap_content" />`

`DigitalClock`	`<DigitalClock`
	`android:layout_width="wrap_content"`
	`android:layout_height="wrap_content" />`

`WebView`	`<WebView android:id="@+id/webview1"`
	`android:layout_width="wrap_content"`
	`android:layout_height="wrap_content" />`

6

Data Persistence

WHAT YOU WILL LEARN IN THIS CHAPTER

- How to save simple data using the SharedPreferences object
- Enabling users to modify preferences using a PreferenceActivity class
- How to write and read files in internal and external storage
- Creating and using a SQLite database

In this chapter, you will learn how to persist data in your Android applications. Persisting data is an important topic in application development, as users typically expect to reuse data in the future. 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

The techniques discussed in this chapter enable applications to create and access their own private data. In the next chapter you'll learn how you can share data across applications.

SAVING AND LOADING USER PREFERENCES

Android provides the `SharedPreferences` object to help you save simple application data. For example, your application may have an option that enables users to specify the font size of the text displayed in your application. In this case, your application needs to remember the size set by the user so that the next time he or she uses the application again, it can set the size appropriately. In order to do so, you have several options. You can save the data to a file, but you have to perform some file management routines, such as writing the data to