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.
Chapter 4
Designing Your User Interface with Views

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
<?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.
Using Basic Views

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
<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"

www.it-ebooks.info
```
CHAPTER 4  DESIGNING YOUR USER INTERFACE WITH VIEWS

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.

www.it-ebooks.info
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:

```xml
<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”):

```xml
<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:

```xml
<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).

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

![Figure 4-3](image.png)

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

```xml
<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:

```xml
<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:

```xml
?{package:}/[type:]name
```

The `RadioGroup` encloses two `RadioButton`s. 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 `RadioButton`s are automatically unselected:

```xml
<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.

The ToggleButton 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:

```xml
<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.

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.
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).
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:

```java
package net.learn2devlop.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.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) {
                DisplayToast("You have clicked the Autosave button");
            }
        });

        //---RadioGroup---
        RadioGroup radioGroup = (RadioGroup) findViewById(R.id.rgActions);
        radioGroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
            public void onCheckedChanged(RadioGroup group, int checkedId) {
                DisplayToast("You have checked the action button");
            }
        });

        //---RadioButton---
        RadioButton radioButton = (RadioButton) findViewById(R.id.rbRed);
        radioButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                DisplayToast("You have clicked the Red radio button");
            }
        });
    }
}
```

www.it-ebooks.info
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:

```java
btnOpen.setOnClickListener(new View.OnClickListener() {
    @Override
    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:

```java
CheckBox checkBox = (CheckBox) findViewById(R.id.chkAutosave);
checkBox.setOnClickListener(new View.OnClickListener() {
    @Override
    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:

```java
//---RadioButton---
RadioGroup radioGroup = (RadioGroup) findViewById(R.id.rdbGp1);
radioGroup.setOnCheckedChangeListener(new OnCheckedChangeListener() {
    @Override
    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 `RadioButton` 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:

```xml
<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`):

```java
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
<?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:

```java
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.idprogressbar);

        //---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---
            }
        });
    }
}
```
Using Basic Views

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.

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

    <!---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();

    </LinearLayout>
```

www.it-ebooks.info
2. Modify the BasicViews2Activity.java file by adding the following statements in bold:

```java
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 = 0;
    ProgressBar progressBar = (ProgressBar) findViewById(R.id.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);

        new Thread(new Runnable() {
            public void run() {
                //---do some work in background thread---
                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() {
                            progressBar.setVisibility(View.GONE);
                        }
                    });
                }
            }
        });
    }
}
```

www.it-ebooks.info
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%.

How It Works

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

```xml
<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:

```java
//---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:

```java
//---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
<?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:

```java
package net.learn2develop.BasicViews3;

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

www.it-ebooks.info
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.

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",

www.it-ebooks.info
"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:

```java
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:

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

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

```java
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**

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
   <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"/>

   ```

---

www.it-ebooks.info
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.

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

```java
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();
    }
}
```

FIGURE 4-12

www.it-ebooks.info
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).

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:

```java
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:

```java
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;
```

www.it-ebooks.info
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:");
    }
}
Using Picker Views

1. 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.

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);
```

www.it-ebooks.info
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
<?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).
3. Back in Eclipse, add the following statements in bold to the BasicViews4Activity.java file:

```java
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() + ",
        "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).
**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:

   ```java
   package net.learn2develop.BasicViews4;

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

   ![Figure 4-16](www.it-ebooks.info)
```java
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(
```
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() + "\n" + datePicker.getYear(), Toast.LENGTH_SHORT).show();
    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 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.

![DatePicker Example](image)

**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:

```java
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:

```java
//---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:

   ```java
   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.

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:

```java
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:

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

       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).
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:

```java
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:

```java
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.
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
<?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>
    </string-array>
</resources>
```
Modify the BasicViews5Activity.java file as shown in bold:

```java
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();
    }
}
```

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:

```java
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
<?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:

```java
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 =
```

CHAPTER 4  DESIGNING YOUR USER INTERFACE WITH VIEWS

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.

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:

```java
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**

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
<?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"
        
</LinearLayout>
```

www.it-ebooks.info
3. Add the following lines in bold to the strings.xml file located in the res/values folder:

```xml
<?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:

```java
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);
    }
```
Using List Views to Display Long Lists

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.

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](image)

**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**

*code file 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
   <?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
   <?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**.

   www.it-ebooks.info
6. Populate the `Fragment1.java` file as follows:

```java
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();
    }
}
```

www.it-ebooks.info
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).

**How It Works**

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

```xml
<?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>
```

www.it-ebooks.info
To create a list fragment, the Java class for the fragment must extend the `ListFragment` base class:

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

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

```java
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:

```java
@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:

```java
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
<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"/>
</LinearLayout>
```
Understanding Specialized Fragments

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**

*codfile 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:

```java
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) {
        www.it-ebooks.info
```
4. Populate the `DialogFragmentExampleActivity.java` file as shown here in bold:

```java
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.

How It Works

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

```java
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:

```java
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():

```java
@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();
                }
            });
}
```

www.it-ebooks.info
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:

```java
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:

```java
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).

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

```xml
<?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" />
    </PreferenceCategory>

    <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 text here" />
    </PreferenceScreen>
</PreferenceScreen>
```
4. Add a Java Class file to the package and name it Fragment1.

5. Populate the Fragment1.java file as follows:

```java
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:

```java
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).

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.
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:

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

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

```java
@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:

```java
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.

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

<table>
<thead>
<tr>
<th>TOPIC</th>
<th>KEY CONCEPTS</th>
</tr>
</thead>
</table>
| **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"  
/>` |
<table>
<thead>
<tr>
<th>TOPIC</th>
<th>KEY CONCEPTS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>android:layout_height=&quot;wrap_content&quot;</td>
</tr>
<tr>
<td></td>
<td>android:text=&quot;Option 1&quot; /&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;RadioButton android:id=&quot;@+id/rdb2&quot;</td>
</tr>
<tr>
<td></td>
<td>android:layout_width=&quot;fill_parent&quot;</td>
</tr>
<tr>
<td></td>
<td>android:layout_height=&quot;wrap_content&quot;</td>
</tr>
<tr>
<td></td>
<td>android:text=&quot;Option 2&quot; /&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/RadioGroup&gt;</td>
</tr>
<tr>
<td>ToggleButton</td>
<td>&lt;ToggleButton android:id=&quot;@+id/toggle1&quot;</td>
</tr>
<tr>
<td></td>
<td>android:layout_width=&quot;wrap_content&quot;</td>
</tr>
<tr>
<td>ProgressBar</td>
<td>&lt;ProgressBar android:id=&quot;@+id/progressbar&quot;</td>
</tr>
<tr>
<td></td>
<td>android:layout_width=&quot;wrap_content&quot;</td>
</tr>
<tr>
<td></td>
<td>android:layout_height=&quot;wrap_content&quot; /&gt;</td>
</tr>
<tr>
<td>AutoCompleteTextBox</td>
<td>&lt;AutoCompleteTextView android:id=&quot;@+id/txtCountries&quot;</td>
</tr>
<tr>
<td></td>
<td>android:layout_width=&quot;fill_parent&quot;</td>
</tr>
<tr>
<td></td>
<td>android:layout_height=&quot;wrap_content&quot; /&gt;</td>
</tr>
<tr>
<td>TimePicker</td>
<td>&lt;TimePicker android:id=&quot;@+id/timePicker&quot;</td>
</tr>
<tr>
<td></td>
<td>android:layout_width=&quot;wrap_content&quot;</td>
</tr>
<tr>
<td></td>
<td>android:layout_height=&quot;wrap_content&quot; /&gt;</td>
</tr>
<tr>
<td>DatePicker</td>
<td>&lt;DatePicker android:id=&quot;@+id/datePicker&quot;</td>
</tr>
<tr>
<td></td>
<td>android:layout_width=&quot;wrap_content&quot;</td>
</tr>
<tr>
<td></td>
<td>android:layout_height=&quot;wrap_content&quot; /&gt;</td>
</tr>
<tr>
<td>Spinner</td>
<td>&lt;Spinner android:id=&quot;@+id/spinner1&quot;</td>
</tr>
<tr>
<td></td>
<td>android:layout_width=&quot;wrap_content&quot;</td>
</tr>
<tr>
<td></td>
<td>android:layout_height=&quot;wrap_content&quot;</td>
</tr>
<tr>
<td></td>
<td>android:drawSelectorOnTop=&quot;true&quot; /&gt;</td>
</tr>
<tr>
<td>Specialized fragment types</td>
<td>ListFragment, DialogFragment, and PreferenceFragment</td>
</tr>
</tbody>
</table>