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).

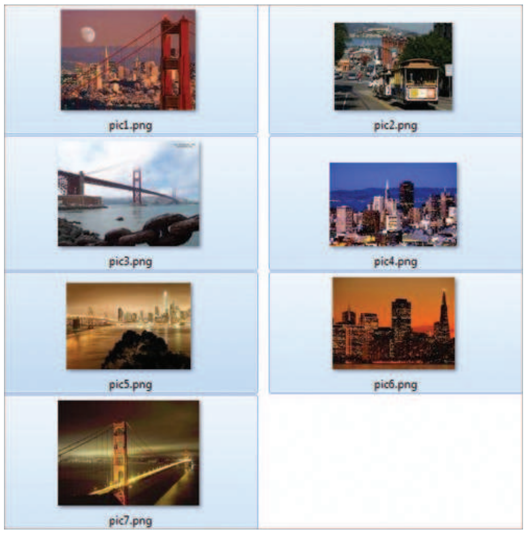


FIGURE 5-2

> ***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.

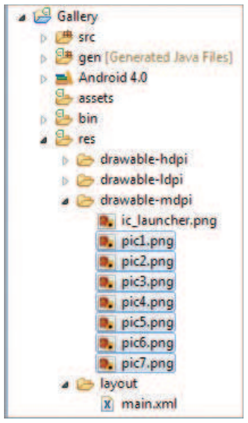


FIGURE 5-3

> ***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.

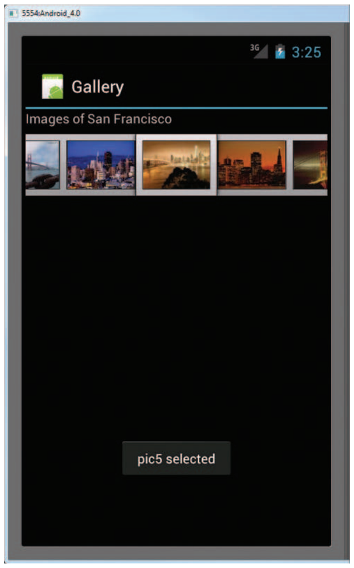


FIGURE 5-4

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" />
```

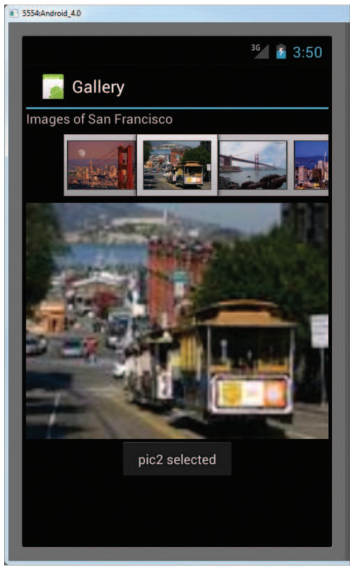


FIGURE 5-5

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
    };
```

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.

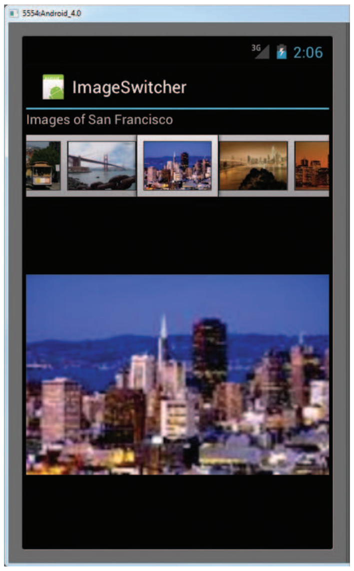


FIGURE 5-6

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;
        }
    }

}
```

5. Press F11 to debug the application on the Android emulator. Figure 5-7 shows the `GridView` displaying all the images.

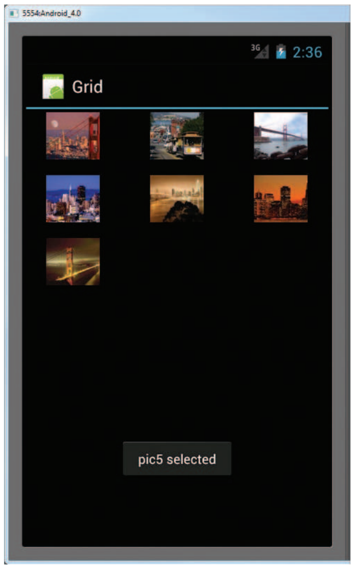


FIGURE 5-7

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.

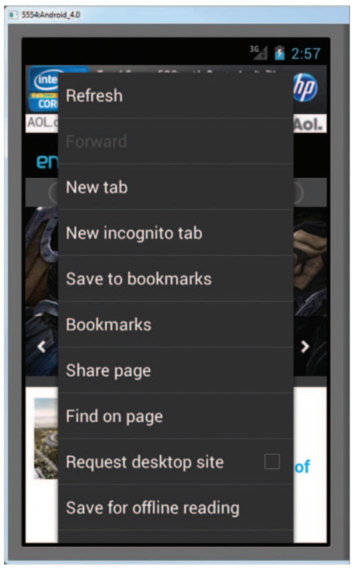


FIGURE 5-8

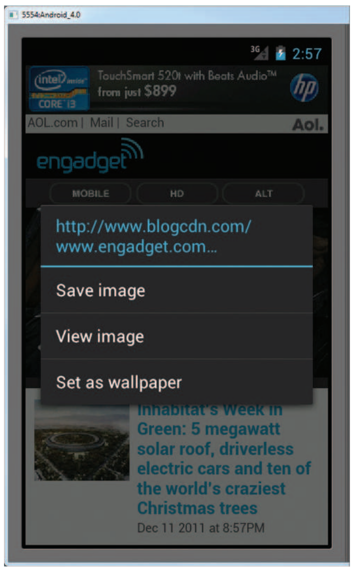


FIGURE 5-9

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" />
```

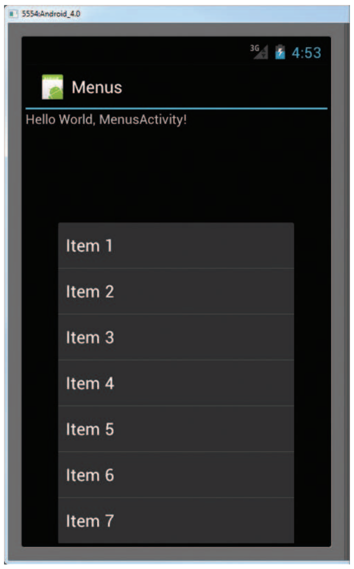


FIGURE 5-10

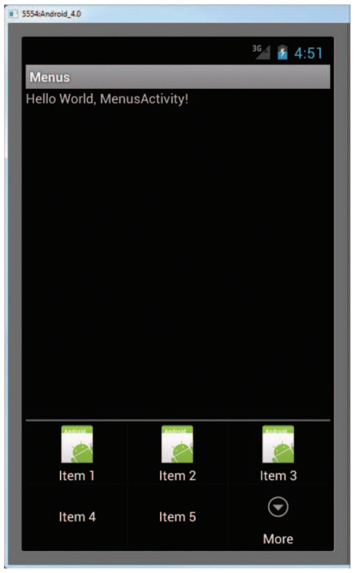


FIGURE 5-11

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.

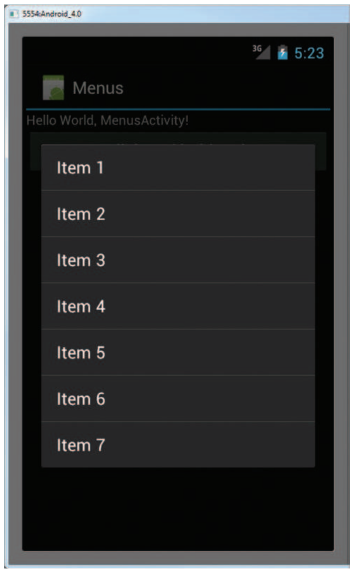


FIGURE 5-12

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:

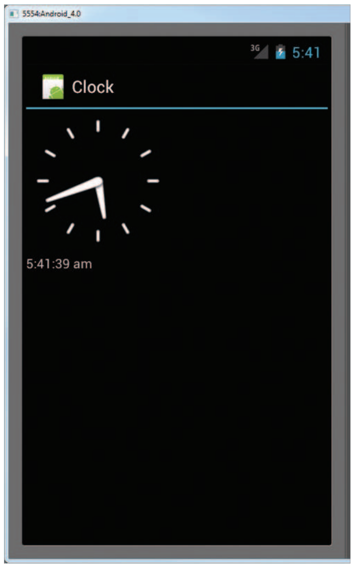
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<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 shows the `AnalogClock` and `DigitalClock` views in action.



5:41:39 am

FIGURE 5-13

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.

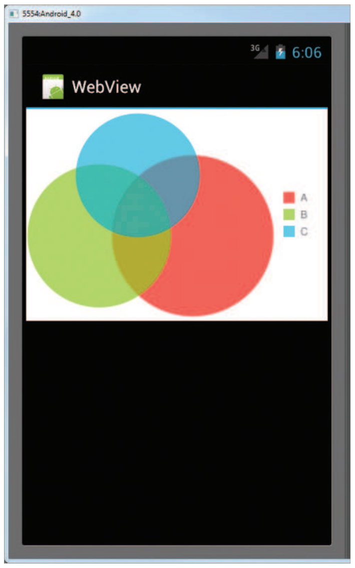


FIGURE 5-14

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.

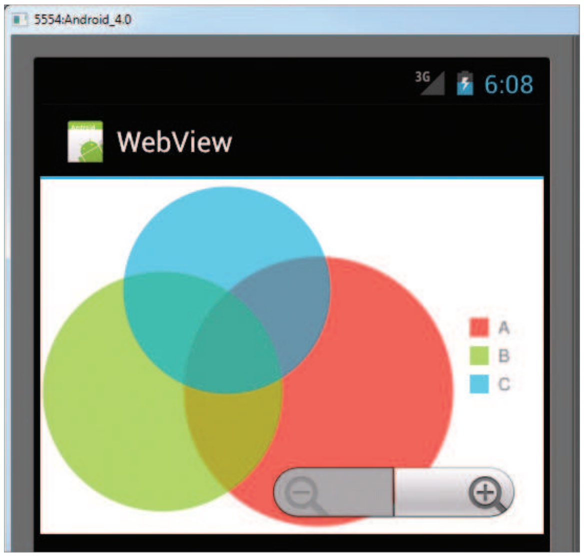


FIGURE 5-15

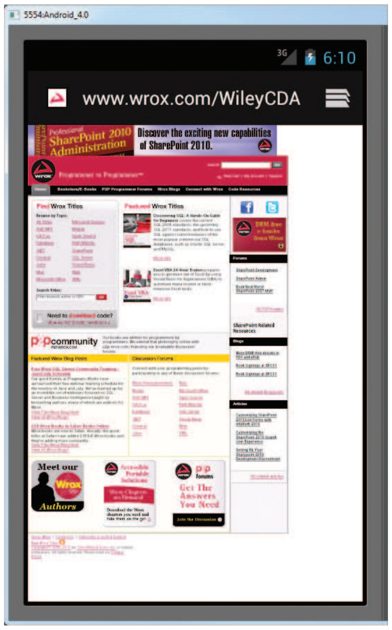


FIGURE 5-16

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`.

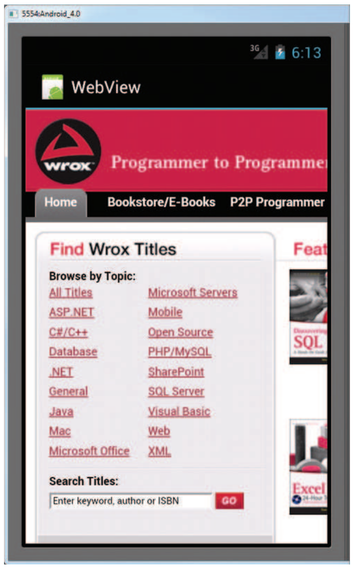
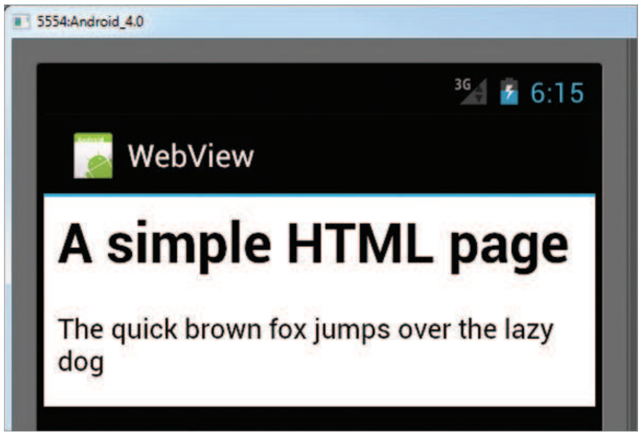


FIGURE 5-17

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`.



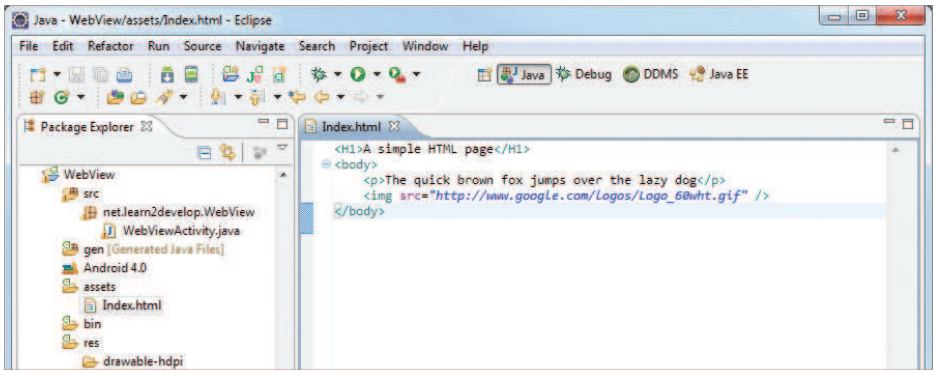
A simple HTML page

The quick brown fox jumps over the lazy dog

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:

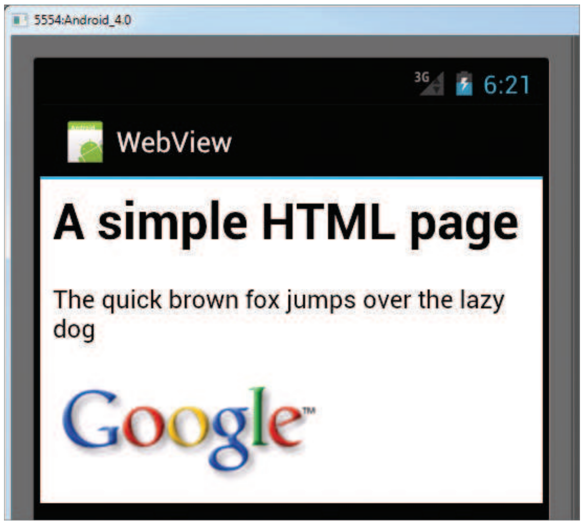
```
WebView wv = (WebView) findViewById(R.id.webview1);
wv.loadUrl("file:///android_asset/Index.html");
```


Java - WebView/assets/Index.html - Eclipse
File Edit Refactor Run Source Navigate Search Project Window Help
Java
Debug
DDMS
Java EE
Package Explorer
WebView
src
net.learn2develop.WebView
WebViewActivity.java
gen [Generated Java Files]
Android 4.0
assets
Index.html
bin
res
drawable-hdpi
Index.html

```
<H1>A simple HTML page</H1>
<body>
    <p>The quick brown fox jumps over the lazy dog</p>
    <img src="http://www.google.com/logos/Logo_60wht.gif" />
</body>
```

FIGURE 5-19

Figure 5-20 shows the content of the `WebView`.



A simple HTML page

The quick brown fox jumps over the lazy dog

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" />`