CHAPTER 12

Programming in Android with the Eclipse IDE

This chapter is all about how to program with the Eclipse IDE, which is a great tool to use when creating an Android application. I already talked in Chapter 1 about how to set up the Eclipse IDE with the Java JDK and the Android SDK.

This chapter is for those Android developers who have never programmed before. You will discover how Java, Android's main programming language, works, and there is an application that you can write at the end. The purpose of this chapter is to make you as an Android developer understand the programming commitment that must be made in order to put out an application that will succeed in the crowded market. I hope that it helps you figure out how long it will take to write your application so that you can create a marketing plan around its creation.

If you want to learn about Android programming in greater detail, I highly recommend reading *Android Apps for Absolute Beginners* (Apress, 2010), by Wallace Jackson, a programming master whose book helped me out quite a bit.

Setting Up the Eclipse IDE to Work with the Android SDK

Go ahead and open up the Eclipse IDE. If you like, you can start a new workspace, or simply use the one that you may have been working with before. Once you select your workspace, you should see a screen like Figure 12-1.

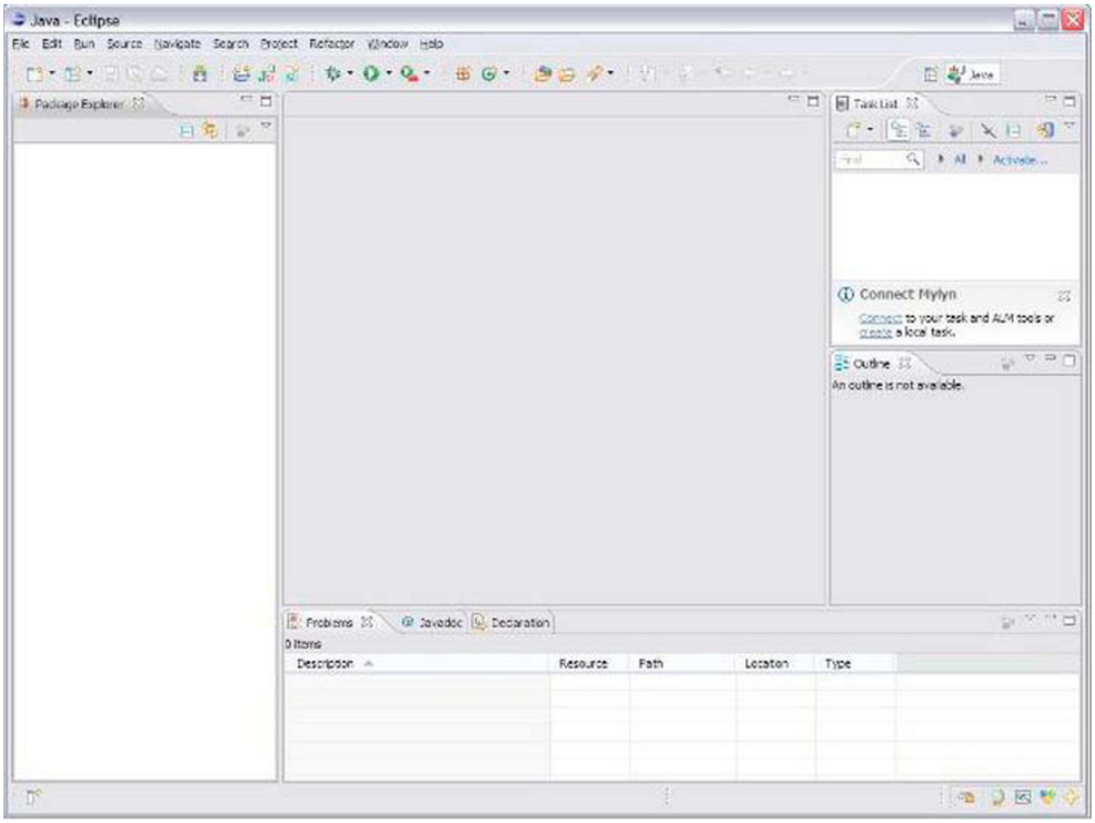
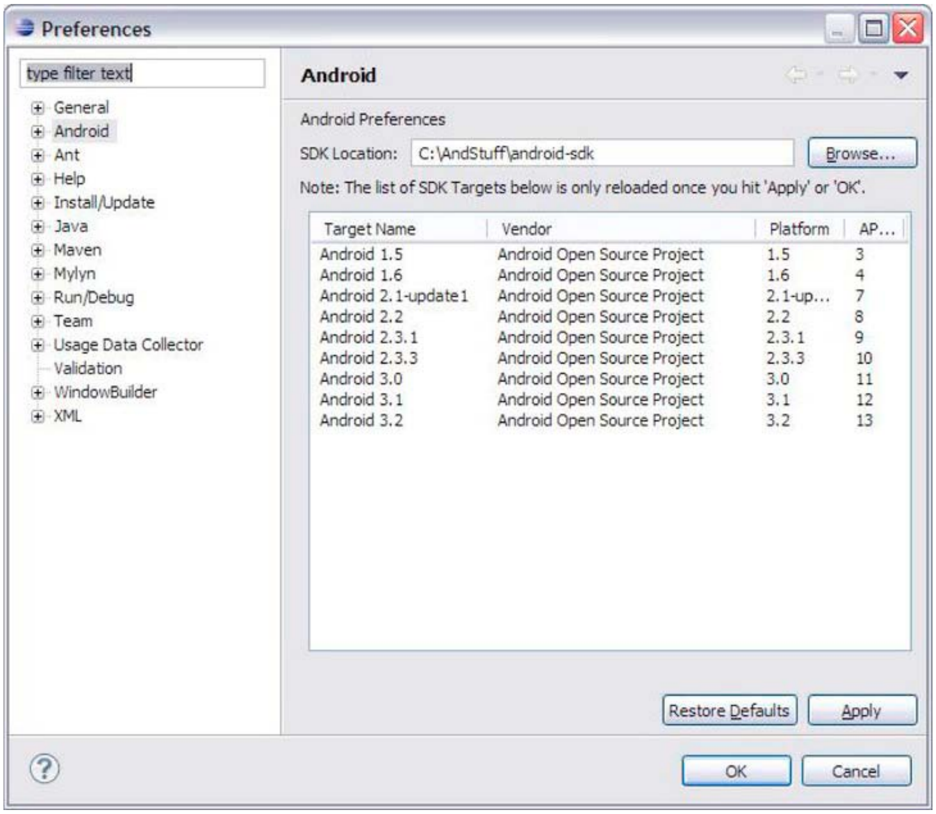

Java - Eclipse
File Edit Run Source Navigate Search Project Refactor Window Help
Package Explorer
Task List
Find
All
Activate...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
An outline is not available.
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type
Java

Figure 12-1. *The Eclipse IDE workspace*

The panel on the left is known as Package Explorer, which is a place for the folder hierarchy. The Task List and Outline panels on the right will also help you with your application. The panel at the bottom right is very helpful as well, as it will give you important information during various stages of your programming. The center panel (blank in Figure 12-1) is the editor, which is where you will be inputting your code.

If you are using a new workspace, then you are going to need to set up your project to work with the Android SDK. You will need to select Windows and then Preferences. You will see a window like Figure 12-2.



Android Preferences

SDK Location: C:\AndStuff\android-sdk

Note: The list of SDK Targets below is only reloaded once you hit 'Apply' or 'OK'.

Target Name	Vendor	Platform	AP...
Android 1.5	Android Open Source Project	1.5	3
Android 1.6	Android Open Source Project	1.6	4
Android 2.1-update1	Android Open Source Project	2.1-up...	7
Android 2.2	Android Open Source Project	2.2	8
Android 2.3.1	Android Open Source Project	2.3.1	9
Android 2.3.3	Android Open Source Project	2.3.3	10
Android 3.0	Android Open Source Project	3.0	11
Android 3.1	Android Open Source Project	3.1	12
Android 3.2	Android Open Source Project	3.2	13

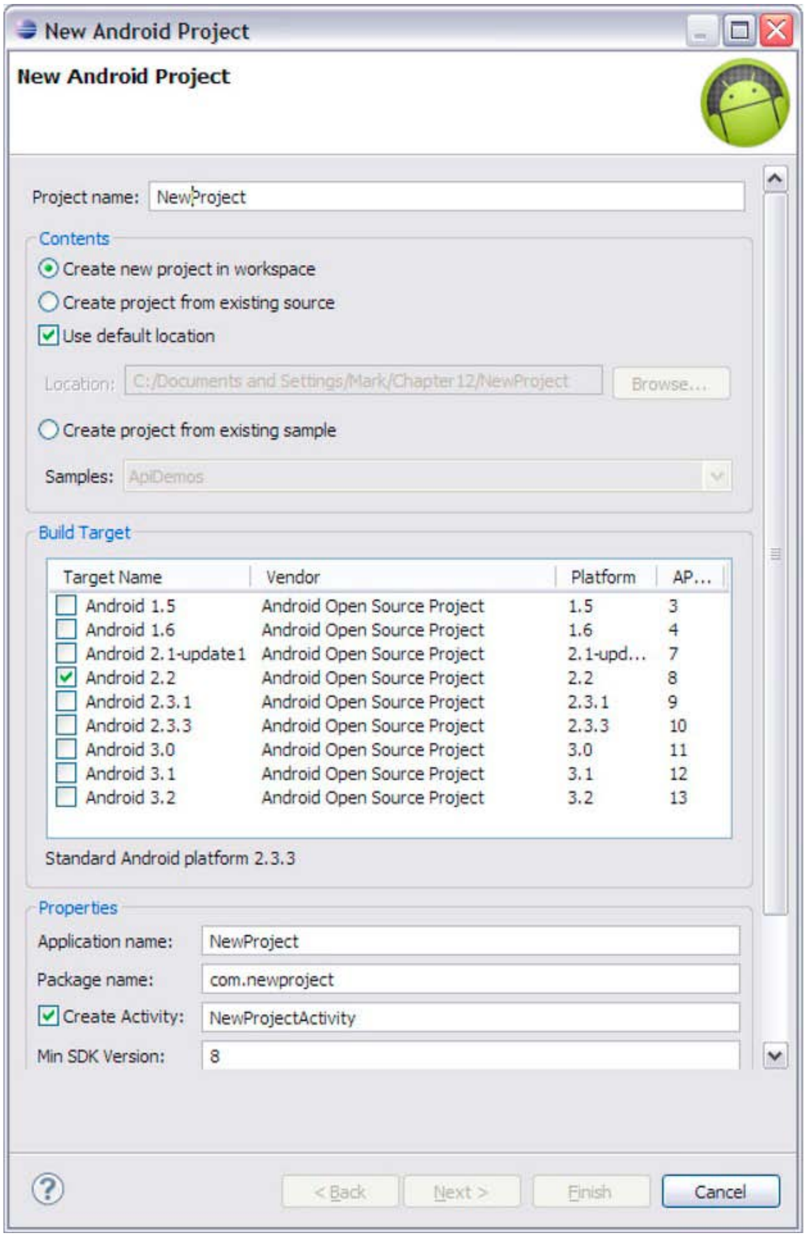
Figure 12-2. *What you will see when you set up your preferences in the Eclipse IDE*

In the Preferences window, you will see a list on the left. Select the Android option, as shown in Figure 12-2.

You will see a tab marked "SDK location." Click Browse and find the directory where you put the Android SDK. You should see the area under the "SDK location" tab fill with information, as in Figure 12-2. Click Apply, and you can have a build target for when you create an Android project.

Starting a New Android Project

Now it is time to start a new Android application. From the toolbar, select File ➤ New ➤ Android Project. You should see a menu like in Figure 12-3. The versions of Android may be different, depending on the versions that you set up earlier.



Project name: NewProject

Contents

- [✓] Create new project in workspace
- Create project from existing source
- [✓] Use default location

Location: C:/Documents and Settings/Mark/Chapter12/NewProject   Browse...

- Create project from existing sample

Samples: ApiDemos

Build Target

	Target Name	Vendor	Platform	AP...
	Android 1.5	Android Open Source Project	1.5	3
	Android 1.6	Android Open Source Project	1.6	4
	Android 2.1-update1	Android Open Source Project	2.1-upd...	7
✓	Android 2.2	Android Open Source Project	2.2	8
	Android 2.3.1	Android Open Source Project	2.3.1	9
	Android 2.3.3	Android Open Source Project	2.3.3	10
	Android 3.0	Android Open Source Project	3.0	11
	Android 3.1	Android Open Source Project	3.1	12
	Android 3.2	Android Open Source Project	3.2	13

Standard Android platform 2.3.3

Properties

Application name: NewProject

Package name: com.newproject

[✓] Create Activity: NewProjectActivity

Min SDK Version: 8

***Figure 12-3.** Starting a new project for Android on Eclipse will reveal this window.*

Figure 12-3 has to be filled out in a certain way, and I will go through the steps.

We'll start with the obvious, the project name. Whatever you type into the "Project name" field will be the name of the file that it will be in your workspace file directory. Spaces are not supported in Java names, so if the name of your project is a few words, you have to run them together. For example, Brand New Project would have to become BrandNewProject. The project name does not have to be the name of your application, as you learned in Chapter 9 when you created an APK file.

As for the Contents section, you have many options. If you select "Create new project in workspace," this creates an original Android project. If your project is similar to one that already exists, such as an application that you created before, you can select your project from some other source in your directory. You can also select from a default location, which automatically selects from the workspace that is currently being used.

You can even "borrow" from some existing sample. For example, you can take from some of the SDK samples, provided they give you license. I found that I was able to incorporate a sample file known as Notepad into my Android application, and all that was required was that I include a copy of the license in the Android code. If you would like to use the code in these samples, check to see if they will give you license, and what steps you will need to follow to get it. These samples are found in the Android SDK file under the appropriately named folder `samples`.

These samples are separated into files for different versions of Android, so you will have to use the proper version for the proper sample. These samples range from simple games like Lunar Lander to interesting applications like Notepad.

In the Build Target section, select the version of Android you want to use for your Android project. You have hopefully loaded it up properly. Remember that if you want more people to download your application, you are going to have to use a lower version of Android. I recommend version 1.6, which will enable all the Boost Mobile users as well as the more advanced Android crowd.

In the Properties section, you enter the application name, which Eclipse will use to set up the framework for your application. It is precisely what shows in the application's title bar as the app runs. So whatever you want on your title bar, put here; this *can* have spaces and punctuation.

The package name is what needs to be used for your Java package. It is the container that holds all of the Java code that the application uses. Java packages are separated by periods, to indicate the hierarchy. A Java package starts with the highest-level domain name of the organization, with all subdomains listed in reverse order. This could be anything; in this example I chose `new.project`.

You can leave the Create Activity box selected. This Java activity class is a collection of code that controls your user interface.

The Min SDK Version field is where you declare the version of Android yet again; here it has to match. I believe it defaults to the same build target:

- 3 = Android 1.5
- 4 = Android 1.6
- 5 = Android 2.0
- 6 = Android 2.0.1
- 7 = Android 2.1 update 1
- 8 = Android 2.2
- 9 = Android 2.3.1
- 10 = Android 2.3.3
- 11 = Android 3.0
- 12 = Android 3.1

- 13 = Android 3.2

There might be other versions out by the time you are reading this. For now, click Finish and that is all.

What You Will Notice About Your Android Project Structure

One of the things that you will notice about Eclipse is how it does a lot of autogeneration. Package Explorer generates some interesting default folders, as shown on the left side of Figure 12-4.

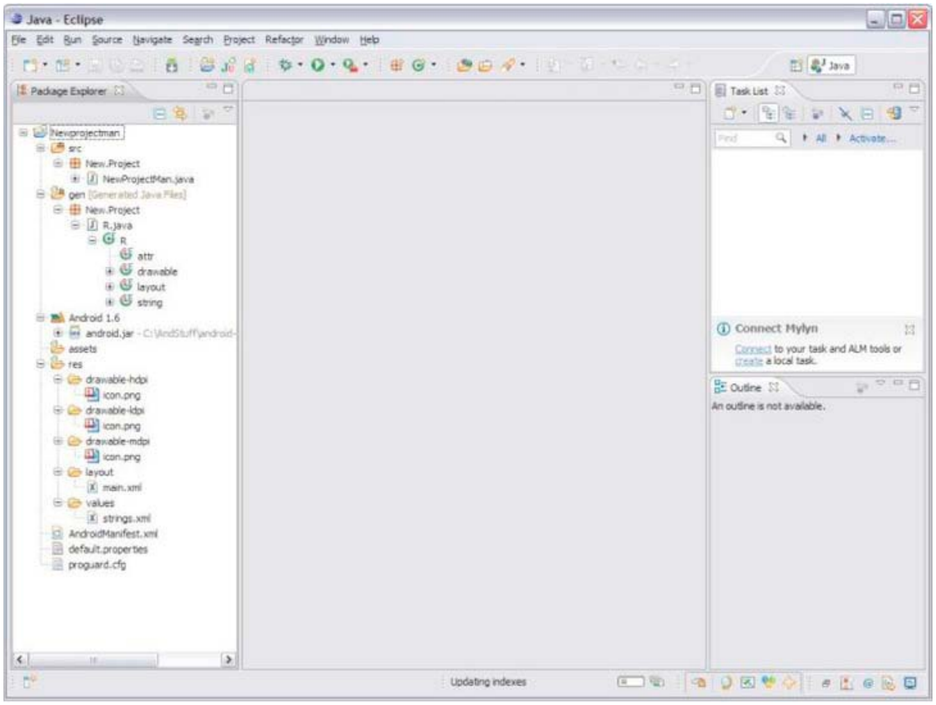


Figure 12-4. *After you start up a new Android project, you will see a new file directory in Package Explorer.*

You should know that these are not the only types of folders that one can have on an Android project. I guarantee you that you will want to make new ones. The following list briefly describes what these folders and files are for:

- `src`: This is the source folder. You will quickly notice that whatever you choose as the package name will be in the file. This contains the Java code that drives an application.

- `gen`: This contains Java files generated by ADT. The `gen` folder contains items generated from the `res` directory. Generally, it is recommended not to mess with this manually, as it might not compile by itself.

- `android.jar`: As you can tell, this matches your build target. It is an item in Eclipse presented through ADT.
- `assets`: This folder is by default empty, but it is made to store raw asset files. Raw assets (for example, a SQLite database) could be required to get your application to work.
- `res`: This stands for *resources*, and there are a lot of important files here. There are many subdirectories, and most of them are in XML files, including the following:
    - `anim/` for animations
    - `color/` for a list of colors
    - `drawable/` for bitmap files (I will explain the `hdpi`, `mdpi`, and `ldpi` files later)
    - `layout/` for a user interface layout
    - `menu/` for application menus
    - `raw/` for arbitrary files in their raw form
    - `values/` for simple values, including resource arrays, colors, dimensions, strings, and styles
- `AndroidManifest.xml`: This is the first file that an Android application will seek out. It is always located in the directory structure so it is easy to find. Here are some of the things that it contains:
    - References to the Java code that you will write for your application so the code can be both found and run.
    - Definitions of permissions for application security and for talking with other Android applications.
    - Definitions of the components of the Android application, including when they can be launched.
    - Declaration of the minimum level of Android operating system version support.
- `default.properties`: This is a file used with both Eclipse and ADT, and it contains the project settings. Like the `gen` file, it should not be tampered with manually.
- `proguard.cfg`: This is automatically generated in the root directory of the project. According to the Android Developers site, the ProGuard tool "shrinks, optimizes, and obfuscates your code by removing unused code and renaming classes, fields, and methods with semantically obscure names."

Let's look at what is in the files. At any given time, we can edit what is in the folders in three ways:

- Double-clicking the file
- Right-clicking the file and clicking Open
- Selecting the file and clicking F3

This will open up the file in the editor, and you can make changes accordingly.

Uploading the Icon and Background for Your Android Application

By now, you should have decided what type of icon you want, as well as the background for your application. These are important elements of your style, which I discussed in Chapter 3.

You will notice that the `drawable` folder in the `res` folder of Package Explorer has three folders: `hdpi`, `mdpi`, and `ldpi`. You will also notice that each of them has a file of the same name, `icon.png`. `icon.png` is set up for different types of bitmap images. These are the required measurements for the icon and background, in pixels:

- *hdpi* is for high-density DPI screen images. The icon must be 72×72 pixels, and the background must be 800×480 pixels.
- *mdpi* is for medium-density DPI screen images. The icon must be 48×48 pixels, and the background must be 320×480 pixels.
- *ldpi* is for low-density DPI screen images. The icon must be 32×32 pixels, and the background 320×420 pixels.

It is important that your icon and background conform to these dimensions, as Android is set up for them for very specific instances. For example, shifting your application from portrait to landscape mode will look much better if your background is presented correctly.

How to Replace the Default icon.png File

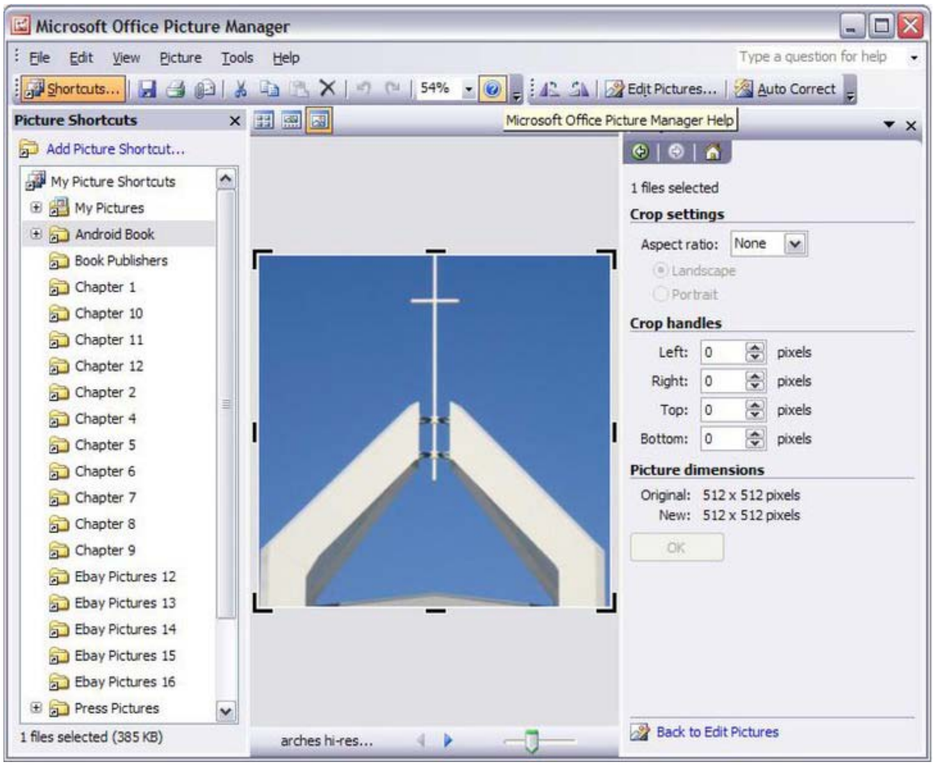
Note that each of the `drawable` folders in Figure 12-4 has a specific image file known as `icon.png`, which is the bitmap for the icon. What you want to do is replace that default icon, which is nothing more than the Android logo in its default state, as shown in Figure 12-5.

Figure 12-5. *The default Android icon, which automatically appears when you create an Android application*

Here are some steps you can follow to change the default icon to one that you have designed:

1. You should get the graphics that you want for your icon, and save it in PNG format if it is not already. You can easily convert your image from its current file type by opening up Microsoft Windows' default Paint program and saving the image in PNG format.
2. You will need to crop the PNG image so that it's square—preferably 200×200 pixels (it can be larger, provided the length equals the width). This can be

accomplished in Microsoft Office Picture Manager by selecting the image, selecting Picture from the top menu, and then selecting Crop. The Crop tool can make your image square; you can see the exact numbers of pixels in your image on the right, in the "Picture dimensions" section (see Figure 12-6).



1 files selected

Crop settings

Aspect ratio: None

Landscape

Portrait

Crop handles

Left: 0 pixels

Right: 0 pixels

Top: 0 pixels

Bottom: 0 pixels

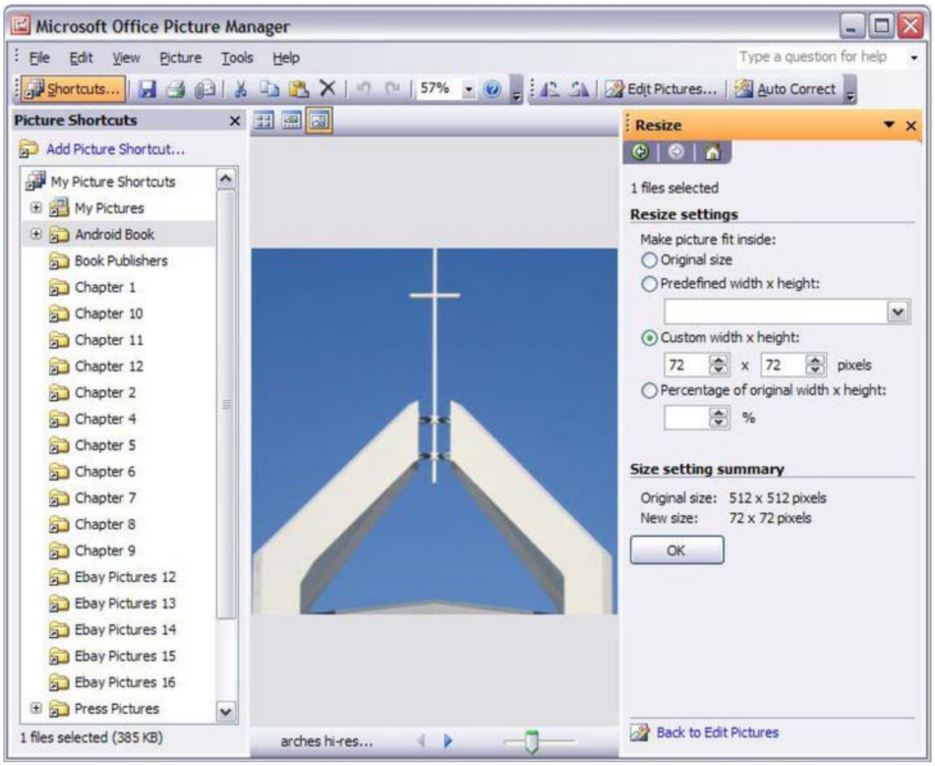
Picture dimensions

Original: 512 x 512 pixels

New: 512 x 512 pixels

Figure 12-6. *Using Microsoft Office Picture Manager is one way of turning your graphic for your icon into a perfect square.*

3. You can use the Resize tool (also a selectable from the picture menu bar) to bring your image to the appropriate size (see Figure 12-7). Keep in mind that Picture Manager's resizing tool can distort things, so you might want to try Photoshop or other tools that can resize an image.



Resize

1 files selected

Resize settings

Make picture fit inside:

Original size

Predefined width x height:

[✓ Custom width x height:] 72 x 72 pixels

Percentage of original width x height: %

Size setting summary

Original size: 512 x 512 pixels

New size: 72 x 72 pixels

OK

Figure 12-7. *The Resize tool can reduce your icon to its proper size.*

4. Resize your icon to 72×72 for the `hdpi` version and click OK. Click Save As from the top menu. Go to the directory where you saved your workspace, and click the `res` folder and then `drawable-hdpi`. Save the file as `icon.png`. It will ask you if you want to overwrite, so click Yes. See Figure 12-8.

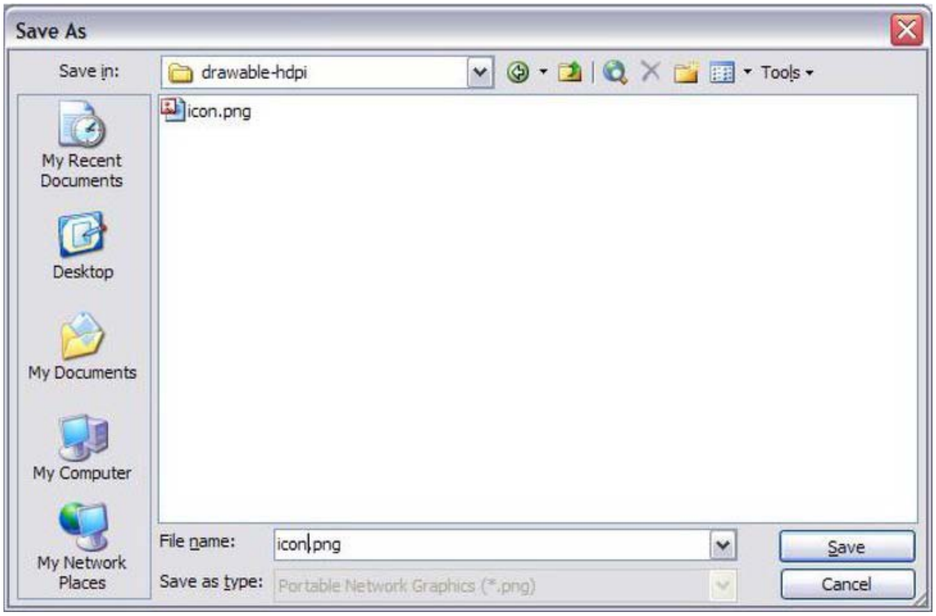


Figure 12-8. *Saving your icon as an icon.png file (in the proper format) will ensure that it will show up in your application.*

5. Go back to your original image (preferably 200×200), resize it to 48×48 for the `mdpi` file, and Click OK. Click Save As from the top menu.
6. Go to the directory where you saved your workspace, and click the `res` folder and then `drawable-mdpi`. Save the file as `icon.png`. It will ask you if you want to overwrite, so click Yes.
7. Repeat Steps 5 and 6 for your icon, this time for `ldpi`; the dimensions need to be 32×32.

You now have your icon for all three different screen resolutions.

How to Set a Background

If you do not set a background for your application, it will appear as a black, blank screen, as shown in Figure 12-9. Since that is a little boring, you should make one of your own, one that is consistent with your style.

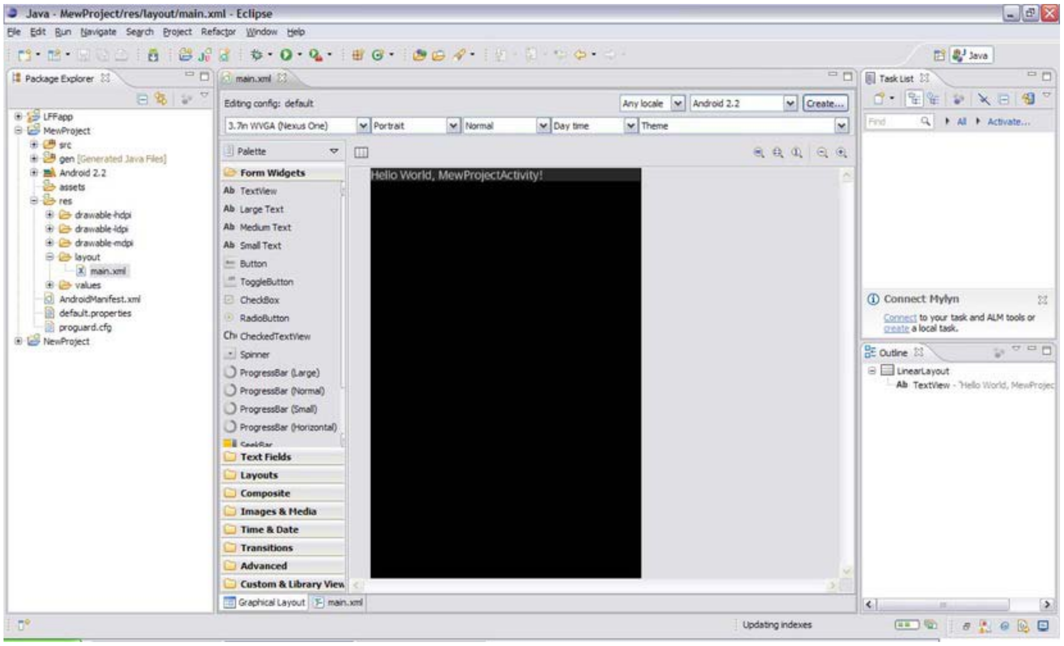


Figure 12-9. *What your application will look like in Graphical Layout mode, sans background*

Similar to the steps for the icon, you will need to create a `background.png` file for the `hdpi`, `mdpi`, and `ldpi` files. The one for `hdpi` has to be 800×480 pixels, the one for `mdpi` has to be 320×480 pixels, and the one for `ldpi` must be 320×420 pixels. Save them under the same name (`background.png`) in the same manner that you saved the `icon.png` file.

Sadly, Android does not automatically set the background, but you can use a simple line of code to do it yourself in `main.xml`:

```
android:background="@drawable/background
```

Perfect. It has gone from ugly black to customized! If you want to see what it looks like, feel free to jump ahead and check out Figure 12-17.

Understanding Java SE

The Android development environment is essentially a combination of Java and XML. The Android SDK includes a runtime that translates the Java and XML code into a language that the operating system and the individual device can understand.

XML stands for eXtensible Markup Language, and it is very similar to the HTML (HyperText Markup Language) that is used for web site design. In fact, HTML is a subset or implementation of XML. XML is designed to structure data for items that require a predefined data structure, and to define constructs so that the user does not need to create them in more complex Java code. If you have done any work in HTML, you will see a familiar sight in XML, known as *tags*. These tags are bracketed by < and > characters.

Android works with Java Standard Edition, or Java SE. Java SE was created by Oracle, and is much more powerful than Java Micro Edition (Java ME), which is on most mobile phones. What you need to know about Java is that it is an object-oriented programming (OOP) language. What you need to know about OOP is that it uses modular, self-contained constructs known as *objects*. Objects, like all programming constructs, are merely abstractions intended to help programmers model certain aspects of the real world in terms of logic and math.

Objects

If you need an example of an object in programming, just look around you for physical objects around the room. Right now, I am sitting in a coffee shop and I am trying to figure out the objects as a computer program would see them. A computer doesn't necessarily know the importance of any given object, unless it is properly defined. To define these objects around me, I am going to pretend that I am receiving a call from my cell phone, and the person on the other end wants me to describe the room around me. How I define it is really up to me, as long as it makes sense to the person on the other end.

I see a chair, and it is made of wood, with four legs, and let's say that it is 3 feet (36 inches) tall. In front of me is a table, and it is made of metal. Let's say the table is 37 inches tall, and only has one leg (due to a wide base on the bottom).

Here is my formula for a chair:

```
class Chair {
int legs = 4;
int height = 36;
String material = "Wood";
}
```

Here is a formula for a table:

```
class Table {
int legs = 1;
int height = 37;
String material = "Metal";
}
```

In the cases above, the blueprint is set with the single word `class`, and then the object. Within the curly brackets ({ and }) are the variables that hold the states of the objects. *States* are the characteristics of the objects. In terms of grammatical parts of speech, think of the objects as the nouns and the characteristics as the adjectives. Whole number data is given the declaration of `int`, while text uses `String`. Defaults are set with an equal sign, followed by their value.

Now, as it so happens, I decided to use the same descriptors in both the chair and the table to define them. You might also notice that I could have gone into more detail when describing these particular objects. Let's go back to my friend on the phone, who wants to know more about the objects around me in the coffee shop. Let's say he asks, "What are the shapes of the tabletops? Are the chairs pushed in or pulled out?"

I can simply alter my formulas to the following new equations. Here is my formula for a chair:

```
class Chair {
int legs = 4;
int height = 36;
String material = "Wood";
String position = "Pushed in";
}
```

Here is a formula for a table:

```
class Table {
int legs = 1;
int height = 37;
String material = "Metal";
String shape = "Rectangular";
}
```

Notice that I put a `position` variable on my chair, as the chair can be "pushed in" or "pulled out." The table is stable, and doesn't need this variable. Let's just say that all the chairs are one shape, and I don't need a shape for them. This is not the state of all the tables in our coffee shop, which is why I chose `Rectangular` for this.

You can imagine what my friend on the phone would think if I altered the information in the variables. If I wanted to heighten my table by 3 inches, I could set its `int height` to `40`. I could make my chair look like an octopus if I set `int legs = 8`. I could even have fun and make my table and chair made of marshmallows with `String material = "Marshmallow."`

Yes, I am being quite a code magician. Of course, objects are really boring if just left to themselves. An object's fields, or variables, hold its state. However, it needs some actions associated with it, which is why we use *methods*: programming routines that operate on the object's internal states. These methods are the verbs to the objects' nouns.

Methods

Everything that we have created about our table and chairs are simply default labels. A method will define how these objects act on the variables to define their current operational state. In the case of the chairs, I want them to move back and forth, so people can sit on them.

It's time to create a method, using the `void` keyword. The method `void` means that it doesn't return anything, but if you want it to return something, such as an object, you put that afterward.

```
void moveChair (String newPosition) {
```

You will note the method name with lowercase letters followed by uppercase. This is known as *camel case*, and it is a normal method-naming convention that begins with a lowercase letter and then goes to uppercase letters to begin words embedded in the method name, like this: thisIsCamelCase. Also note the first (opening) curly bracket (`{`). We have to follow that up by a closing curly bracket (`}`), and something in between the two. So let's add this:

```
void moveChair (String newPosition) {
position = newPosition;
}
```

This basically tells us that we are setting the table's position to that which was passed into the `moveChair` method.

Let's say I call this in the program, in order to move the chair back. It is as simple as this:

```
moveChair(back);
```

Now, our objects are defined within the class, but they cannot do anything until the user creates an instance of the objects, or *instantiates* them. Let's say I want to define the two chairs at my table; I could use this formula here:

```
Public void onCreate (Bundle savedInstanceState) {
        Super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

Chair chairOne = new Chair();
        Chair chairTwo = new Chair();

chairOne.moveChair(forward);

        chairTwo.moveChair(back);
```

Were this formula put into a program, we would start with two standard chairs. The new keyword creates that object. We have defined the object `Chair`, given names to our objects, and set default variable values.

We can use the method thanks to the following code construct:

```
objectName.methodName(variable);
```

In the case of our chairs, I have moved one forward, and one back.

Inheritance

Now let us talk about the concept of *inheritance*. Java supports the development of different types of objects that are more specific in their construction. These more specific objects would be subclassed from the main object. A class that is used for inheritance by a subclass is known as a *superclass*.

For example, I could create a superclass of objects for our chairs and tables known as Furniture. Each of these chairs and tables would be subclasses, and will inherit whatever the superclass has—for example, number of legs, material, height—but I could add specific things for each individual subclass. For example, the position for the chairs and the shape for the tables.

You can create a subclass from a superclass by using the keyword `extends`, like so:

```
Class Chair extends Furniture {insert new fields and methods here}
```

Interfaces

Certain classes conform to a certain pattern in many Java applications, because the rest of the application should know what to expect of those classes when they are instantiated as objects.

A public interface that the classes present to the application makes their use more predictable and allows the user to use them in places where any class of that pattern is suitable. In the words of programmer Wallace Jackson, who I previously mentioned, “The public interface is a label that tells the application what this class can do, without the application needing to test its capabilities.” Implementing an interface is as easy as using an `implements` command.

The Package Declaration

Recall when I discussed creating an Android application the concept of a *package name*. I am going to go into detail about what it is. This is the first line of code in any Android and Java application. This is written with a keyword and declaration method like this:

```
package application.activity
```

The package concept is like the folder hierarchy that you use on your own computer, and it organizes its code by functionality. The way Android does it is that it organizes its classes into logical packages, which get imported throughout the program.

Programs need `import` statements, which use code from elsewhere. The way I understand `import` statements is like this: if you are writing a book, you will probably need other books as resources to help you. If we look at your Android program as the book you are writing, then the `import` statements are the other books that you need to make your book complete. Android doesn't need a whole library, just a few necessary volumes to do certain things.

In the example that we used with the two chairs, we used the code of `Public void onCreate (Bundle savedInstanceState)`.

This cannot be used unless we import Bundle. This is done as follows:

```
import android.os.Bundle;
```

An `import` statement is usually written in the following format:

```
import platform.functionality.classname;
```

In the case of `import.android.os.Bundle`, it's using the Android platform, with functionality of the `os` with `Bundle`, allowing the program to create bundles of variables for convenience and organization.

You can find a whole list of packages on the Android Developers web site at `http://developer.android.com/reference/packages.html`. Just so you know, the package isn't the highest level of organization as far as Java is concerned. There is a platform or application programming interface (API). This is a collection of all the core packages for a given language or the packages of a specialized product, like Android.

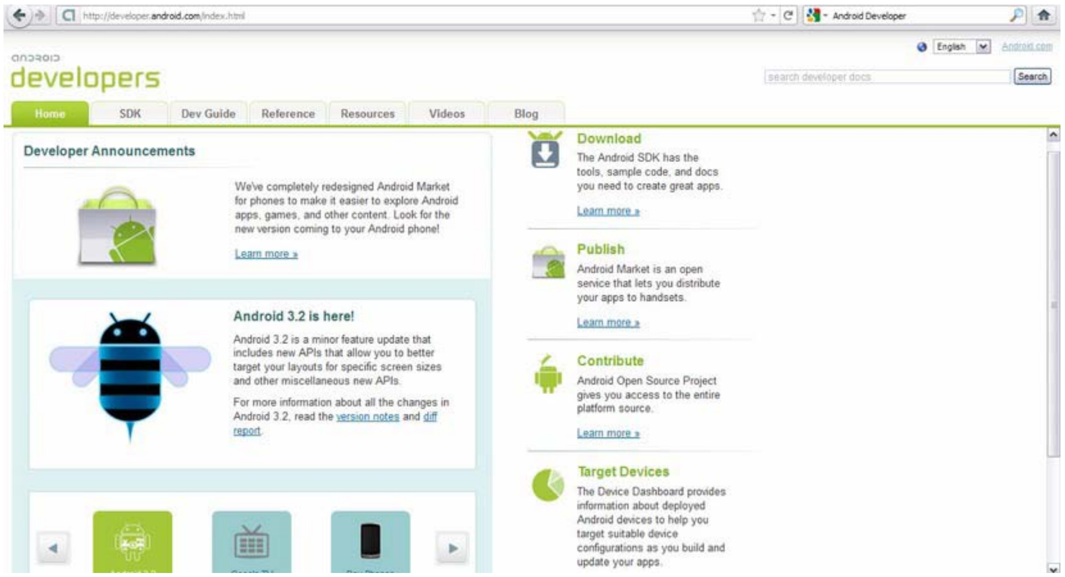
Getting Your Program to Do What You Want

One of the keys to programming any application in Android is simply figuring out what it is that you want to do, and how to tell Android to do it with code. Fortunately, there have been many who have gone before you, and sometimes it is all about using the same programming code that others have used.

There are places where you can go online to receive help, as there is a community of Android developers who are happy to share their knowledge with you.

The Android Developers Web Site

Believe it or not, the Android development team actually wants you to know how to program on it. It lays out a lot of things that you need to do in order to program on it. The home page can be seen in Figure 12-10; you will remember being on the SDK tab when you had to download the Android SDK.



Developer Announcements

We've completely redesigned Android Market for phones to make it easier to explore Android apps, games, and other content. Look for the new version coming to your Android phone!

Learn more »

Android 3.2 is here!

Android 3.2 is a minor feature update that includes new APIs that allow you to better target your layouts for specific screen sizes and other miscellaneous new APIs.

For more information about all the changes in Android 3.2, read the version notes and diff report.

Download

The Android SDK has the tools, sample code, and docs you need to create great apps.

Learn more »

Publish

Android Market is an open service that lets you distribute your apps to handsets.

Learn more »

Contribute

Android Open Source Project gives you access to the entire platform source.

Learn more »

Target Devices

The Device Dashboard provides information about deployed Android devices to help you target suitable device configurations as you build and update your apps.

Figure 12-10. *The Android Developers site home screen, at http://developer.android.com*

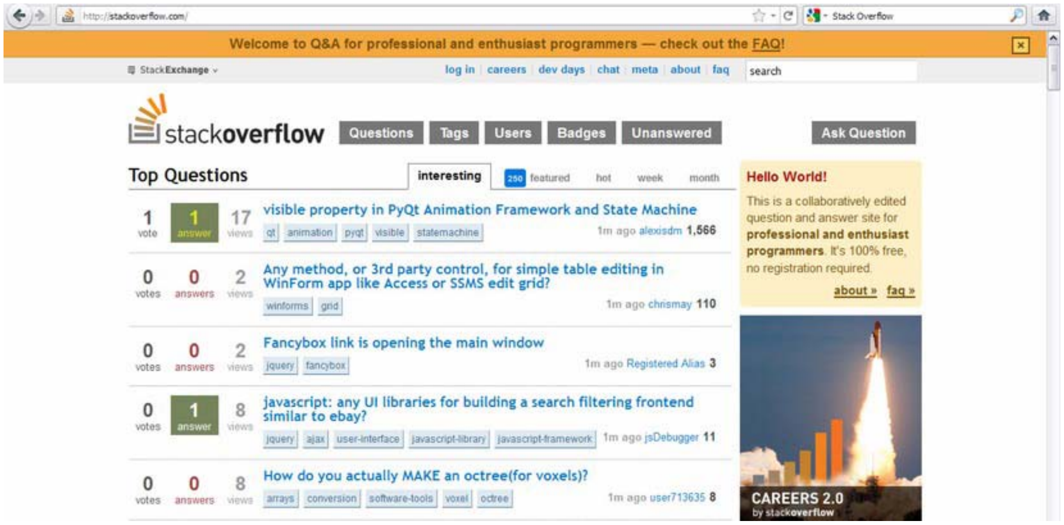
The third tab is Dev Guide, and it will help you to understand some of the basics of Android programming. The References tab is also handy because it has a package index and a class index, and the Resources tab is good for articles, tutorials, sample code, and more. The Video and Blog tabs keep a developer in the know as far as new things on Android.

Most of the Android Developers site will seem very confusing at first, but I guarantee that certain things will become clearer when you use it. Note its handy search engine; you can use it to figure out how to make your program do what you want it to do.

Stack Overflow

As you get more adept at programming in Android, you will run into more complex problems with your application. You are going to need answers.

Stack Overflow was created by Jeff Atwood and Joel Spolsky in 2008 for questions and answers about computer programming. You can ask questions and get answers, and usually someone before you has asked the same question. You can see its home page in Figure 12-11, but I guarantee that this has changed by now.



Welcome to Q&A for professional and enthusiast programmers — check out the FAQ!

Hello World!

This is a collaboratively edited question and answer site for **professional and enthusiast programmers**. It's 100% free, no registration required.

1 vote | 1 answer | 17 views — visible property in PyQt Animation Framework and State Machine — qt animation pyqt visible statemachine — 1m ago alexisdm 1,566

0 votes | 0 answers | 2 views — Any method, or 3rd party control, for simple table editing in WinForm app like Access or SSMS edit grid? — winforms grid — 1m ago chrismay 110

0 votes | 0 answers | 2 views — Fancybox link is opening the main window — jquery fancybox — 1m ago Registered Alias 3

0 votes | 1 answer | 8 views — javascript: any UI libraries for building a search filtering frontend similar to ebay? — jquery ajax user-interface javascript-library javascript-framework — 1m ago jsDebugger 11

0 votes | 0 answers | 8 views — How do you actually MAKE an octree(for voxels)? — arrays conversion software-tools voxel octree — 1m ago user713635 8

Figure 12-11. *The Stack Overflow site, at http://stackoverflow.com/*

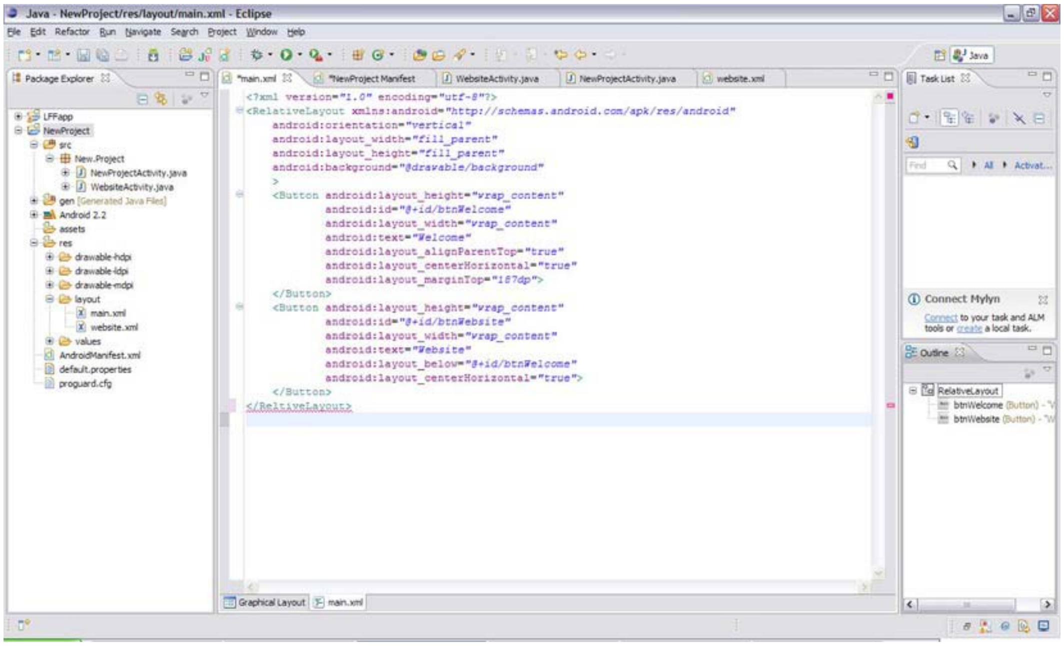
Other Ways to Find Programming Answers

Of course there are more places to find answers online than just Stack Overflow and the Android Developers site. However, when I have had questions about programming on Android and searched for the answers on Google, these two sites invariably came up in the top ten.

You will find that just doing a search for anything on Google is advantageous. Sometimes you just need to find the right combination of words, such as, "How do I link to Facebook on my Android app?"

Handling Errors in Eclipse

As you are programming your application, you will notice the red lines that come up. There will also be red squares to their left, and red marks on the folders in Package Explorer, as in Figure 12-12. These appear because of errors in your code.



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/background"
    >
    <Button android:layout_height="wrap_content"
            android:id="@+id/btnWelcome"
            android:layout_width="wrap_content"
            android:text="Welcome"
            android:layout_alignParentTop="true"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="187dp">
    </Button>
    <Button android:layout_height="wrap_content"
            android:id="@+id/btnWebsite"
            android:layout_width="wrap_content"
            android:text="Website"
            android:layout_below="@+id/btnWelcome"
            android:layout_centerHorizontal="true">
    </Button>
</ReltiveLayout>
```

Figure 12-12. *What happens when you get an error on Eclipse*

The Eclipse IDE makes it painfully obvious when your program has errors because your program will not run with them. Your only course of action is to correct them.

Notice that my error in Figure 12-12 is simple spelling. If only all errors were this simple to fix! If you hover your cursor over these errors in the code, Eclipse will tell you what it believes the problem is. It will even tell you how it believes that you can solve the problem. Oftentimes, the error occurs because an `import` statement is needed, and that is easy to detect when you click the error. Eclipse's suggestions are often quite helpful.

Errors that occur when your program calls upon something that isn't there, however, don't correct themselves automatically. For example, if you set a background, but forget to upload your background to the `drawable` file, there will be an error. You might find that when you upload the file, you still have the error. You should probably click Refresh or Save All in cases like this. You'll find that the red *X*s should be gone if you have done it right.

Running Applications on Eclipse

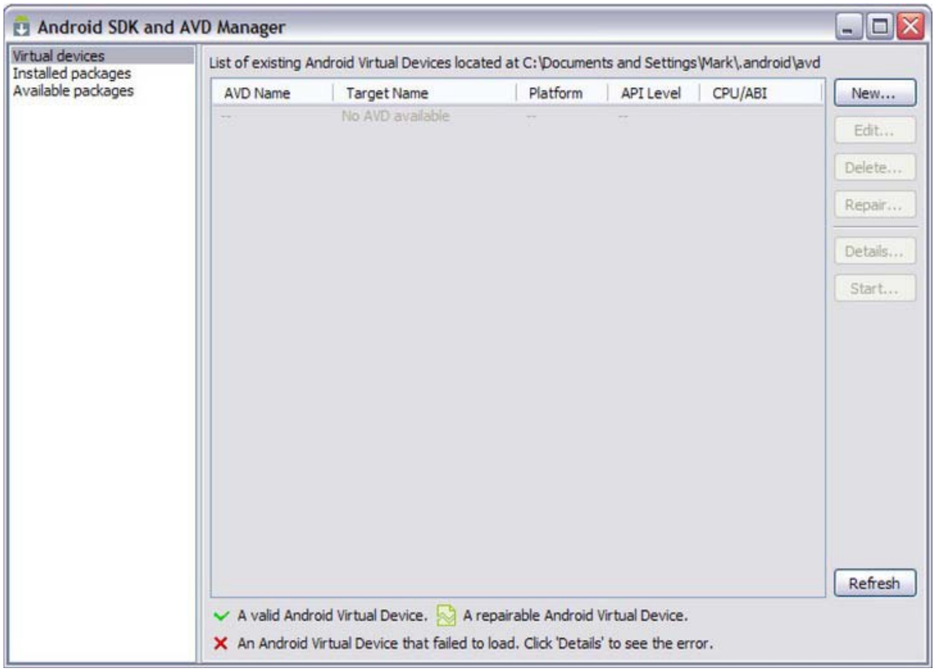
Eventually, you have to test your program out in the real world. I highly suggest you do it on an Android device, but you can use the emulator on Eclipse itself.

Creating an Emulator

Here is how you set an emulator up within the Eclipse workspace:

1. Start up Eclipse, and open it to your workspace.

2. Go to the top menu, click Window, and then Android SDK and AVD Manager. You should see a window like you see in Figure 12-13.



List of existing Android Virtual Devices located at C:\Documents and Settings\Mark\.android\avd

AVD Name	Target Name	Platform	API Level	CPU/ABI
--	No AVD available	--	--	

✓ A valid Android Virtual Device. A repairable Android Virtual Device.
✗ An Android Virtual Device that failed to load. Click 'Details' to see the error.

Figure 12-13. *Android SDK and AVD Manager, where you can set up an emulator*

3. If it is not already selected, click the Virtual Devices column on the left.
4. On the menu on the right, select New. You should see a window like in Figure 12-14.

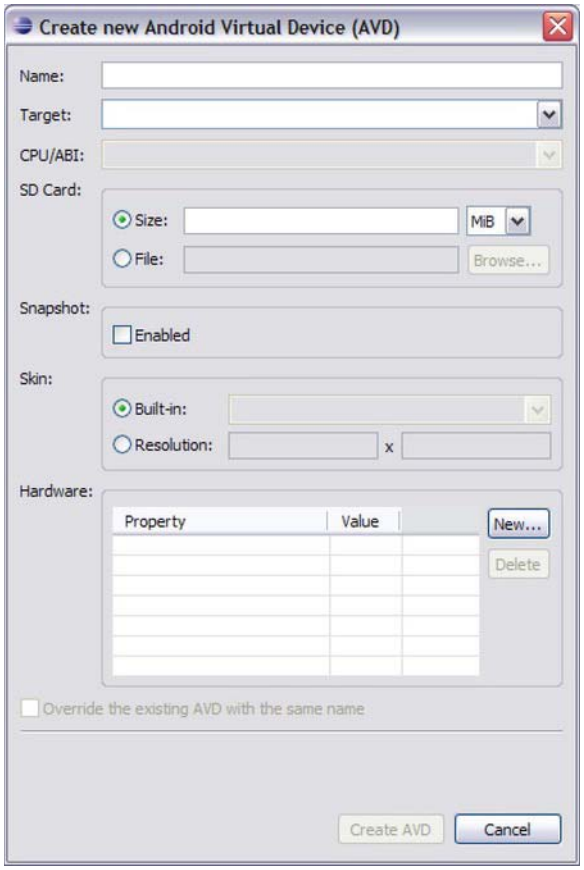
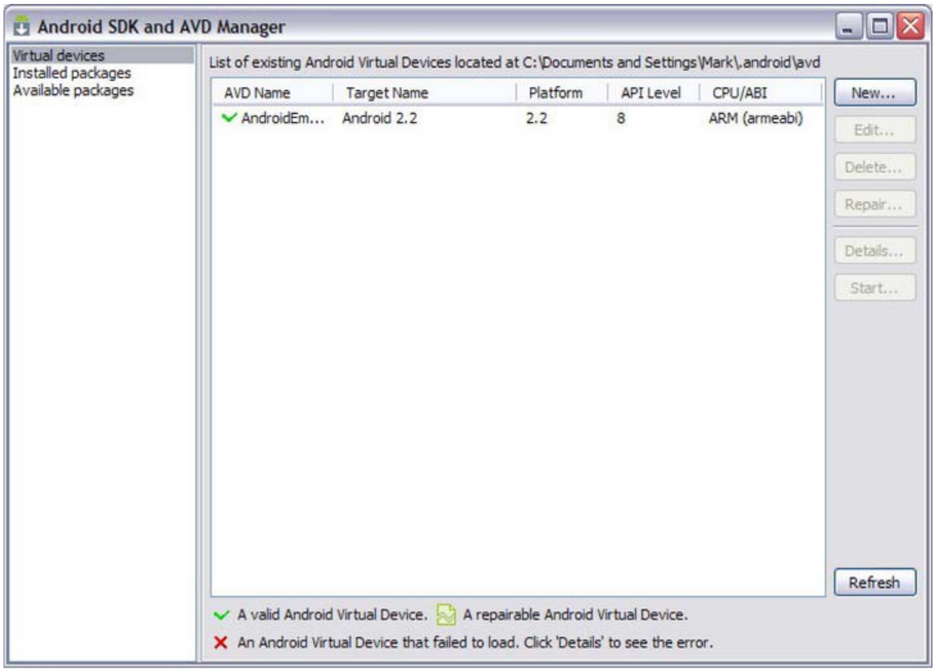


Figure 12-14. *How to create an emulator of your own in Android*

5. Now it is time to fill in the appropriate information for the Android Virtual Device (AVD). For Name, you can pick anything you want as long as you remember it is an Android emulator. It doesn't accept spaces, so keep that in mind.

6. For Target, select the version of Android that you are working with. The SD Card section can be left blank, you don't need to enable Snapshot, you can leave the skin at its default (HVGA), and you don't need to select anything in the Hardware section.

7. Click Create AVD, and you'll have your emulator. You should see a window like in Figure 12-15.



List of existing Android Virtual Devices located at C:\Documents and Settings\Mark\.android\avd

AVD Name	Target Name	Platform	API Level	CPU/ABI
✓ AndroidEm...	Android 2.2	2.2	8	ARM (armeabi)

✓ A valid Android Virtual Device. A repairable Android Virtual Device.
✗ An Android Virtual Device that failed to load. Click 'Details' to see the error.

Figure 12-15. *Your Android emulator is ready to go!*

Running Your Application on Your Emulator

Once you have your program ready and it is error free, your program can be run in several ways:

- You can right-click the top folder in Package Explorer, and select Run As ➤ Android Application.
- Select the application in Package Explorer and click Run on the top menu.
- Select the green arrow at the top and select Run As.
- Select your application and click Ctrl+F11.

If you have chosen to use the emulator on Eclipse, prepare for a long wait. The screen on the left will show all manner of Android logos before it looks like Figure 12-16.. You might have to unlock the emulator like an Android phone, but use your cursor and mouse instead of a finger and touchscreen. That will make sense once it is up and running. You should see your application and interact with it like you would a web site on your computer.

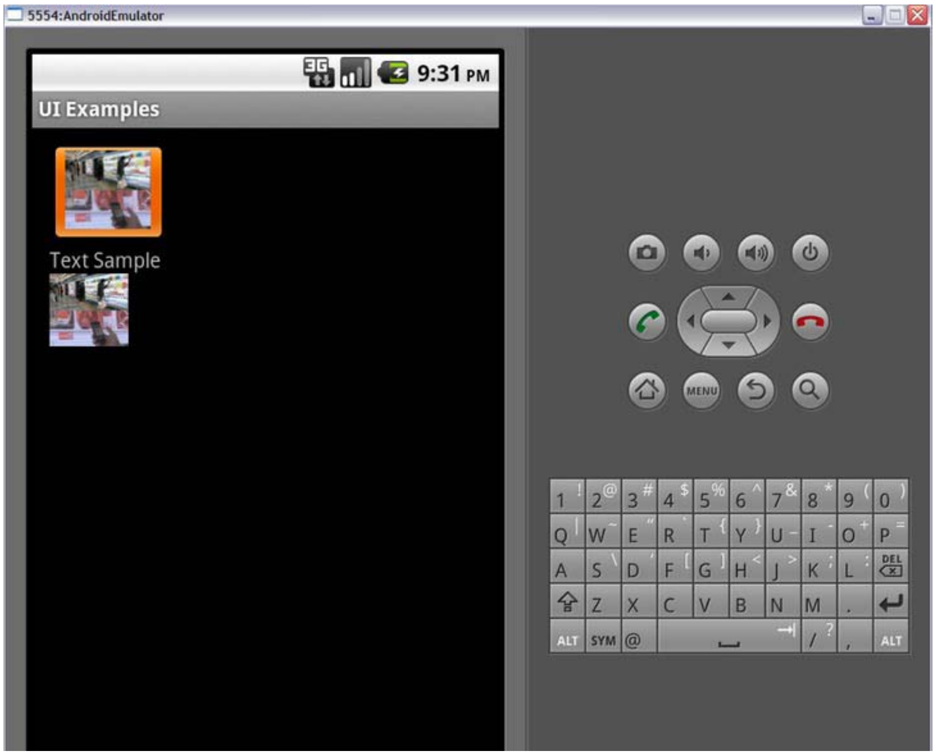


Figure 12-16. The Android emulator without a device

I have a five-year-old laptop that I use to write Android programs, and I find that the Eclipse emulator won't do much before it crashes. For this reason and more, I highly recommend using your own Android device. Your application will show up on your Android phone or tablet almost instantly. In fact, the icon for your application will even show up on your home screen menu. The best part is you can use your Android application after you have disconnected your Android device. You will have to have your computer set up to recognize your Android device (I discuss this in Chapter 9 in the "How to Obtain a Screenshot of Your Android Device" section).

Example of an Android Application

The Android Developers web site has a tutorial known as "Hello, World," which allows you to create a simple app that displays text that says anything you want it to say (the tutorial uses "Hello, World," hence its name). You can find this tutorial at `http://developer.android.com/resources/tutorials/hello-world.html`, but I thought I would set up a program to do something a little more complex in order to demonstrate some of the concepts mentioned earlier.

This program, which I just call New Project, creates two buttons in the center of the screen. One button, labeled "Welcome," creates a dialog that appears in the form of a small, pop-up window. The other button, marked "Website," opens the application to a web site of your own picking.

Go ahead and set up a new program, as shown previously in Figure 12-3. Feel free to use the same names if you like. Once the default folders in Package Explorer appear, open up the `main.xml` file in the `res/layout` folder (I explained three ways of doing this earlier). You will see some automated code in the editor. You won't be needing the TextView section, so feel free to highlight and delete the lines of code starting with `<TextView` to anything before `</LinearLayout>`. What you should have leftover is this:

```
<?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"
    >
</LinearLayout>
```

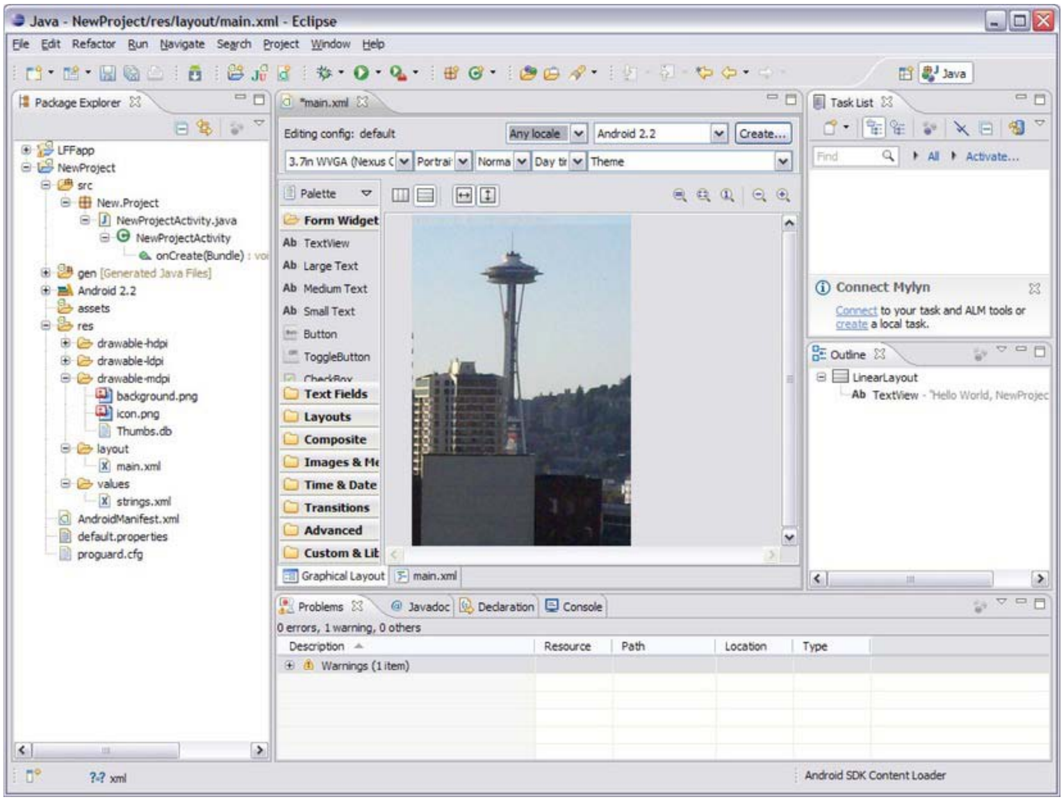
The first line is the default XML declaration, which is made to let Eclipse and Android know precisely what type of file you're using. The next four lines determine the linear layout, which is designed to contain and arrange UI elements placed inside it on the screen horizontally or vertically.

The next three lines configure the settings of the view so Android knows how to view the screen. Note the invoking of android at the beginning of each of them. The line `android:orientation="vertical"` lets Android know that the view is to be in portrait (not landscape) format. The line `android:layout_width="fill_parent"` lets the view fill up horizontal space until it reaches its "parent." As for `android:layout_height="fill_parent"`, this is designed to make the layout as tall as possible for the parent.

You may notice in the editor window two tabs in the lower-left corner. One is marked "Graphical Layout" and another "main.xml." Go ahead and select the Graphical Layout option, and you will see exactly what the main screen of your Android application looks like. It is blank for now, but we can fix that. Just add a simple line of code in the LinearLayout section right before the last bracket, like this:

```
<?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"
    android:background="@drawable/background"
    >
```

I used a picture that I took in Seattle, but you can use anything you want. Just make certain that it is uploaded and sized correctly, as I described previously in the section about how to upload an icon and background. The graphical layout should look something like Figure 12-17.



Editing config: default   Any locale   Android 2.2   Create...
3.7in WVGA (Nexus One)   Portrait   Normal   Day time   Theme

Connect to your task and ALM tools or create a local task.

0 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				

Figure 12-17. A graphical layout of your project, complete with a background

In Figure 12-17, you can see an area marked Palette, which is full of file folders. If you click one of these, you can drag and drop some interesting things on your application. For this exercise, go ahead and drag and drop two buttons on your screen, as shown in Figure 12-18.

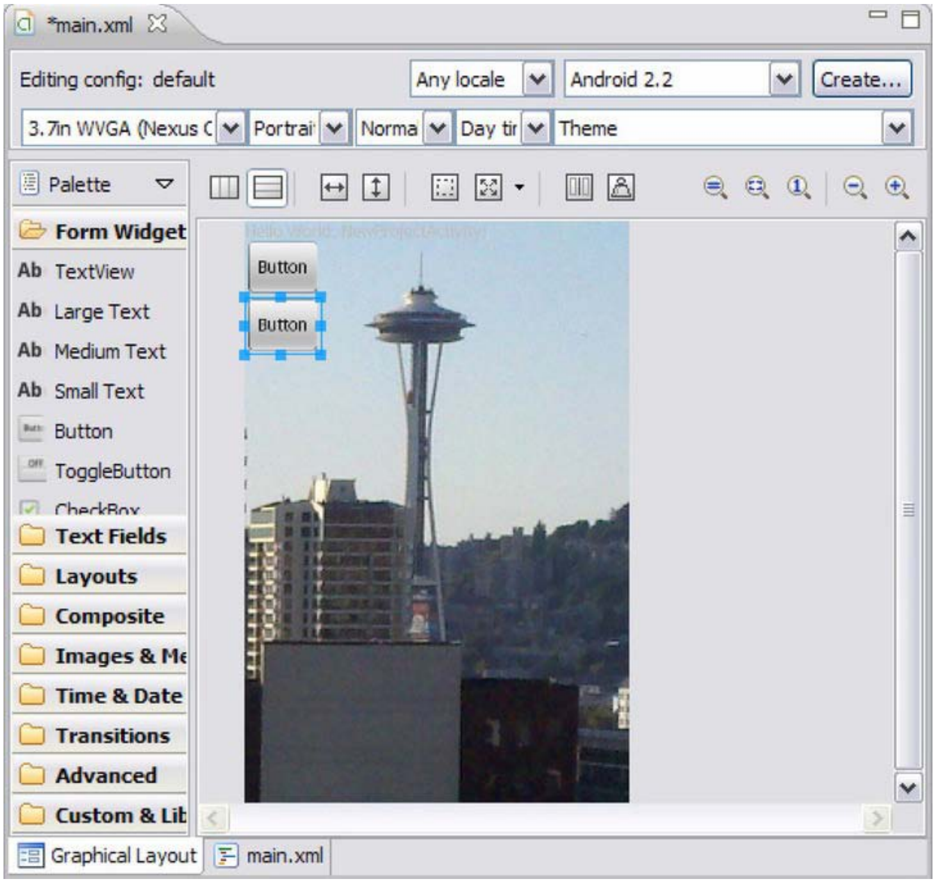
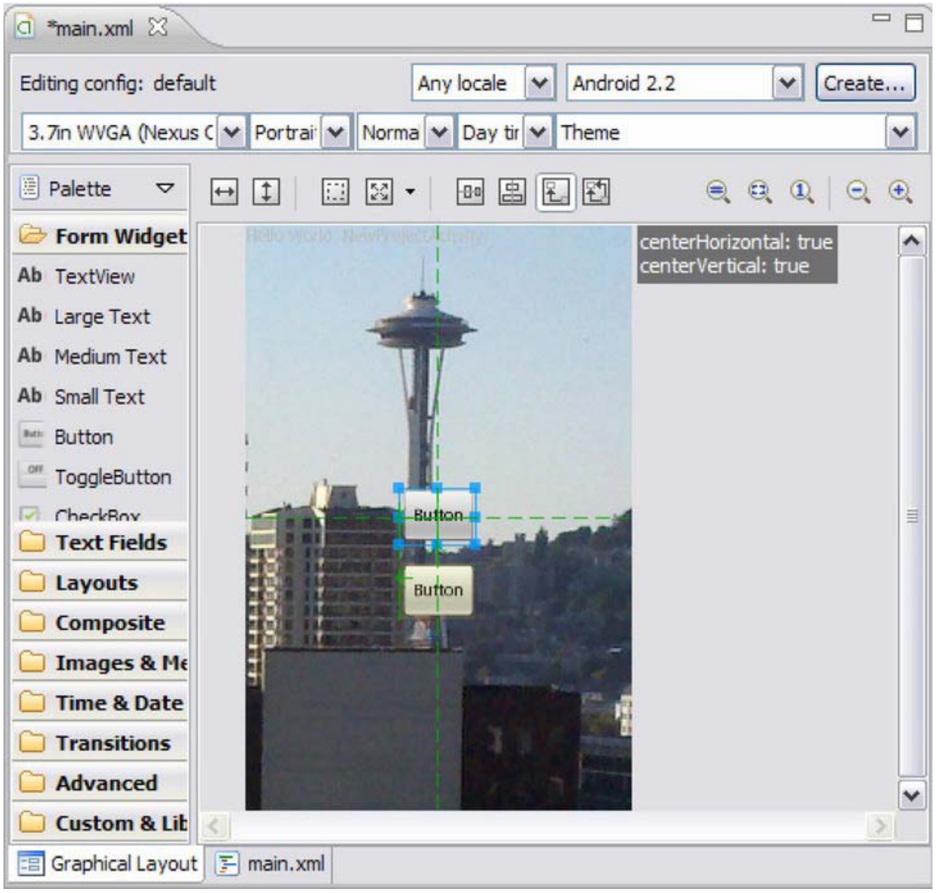


Figure 12-18. *Two buttons have now been added to your Android application.*

Notice from Figure 12-18 that you can only put the buttons in two positions, which certainly is boring. `LinearLayout` is really only designed for simple horizontal or vertical orientation. If you want to change that into something a little more customized, go ahead and click the `main.xml` tab at the bottom-left corner of the editor and alter the code from `LinearLayout` to `RelativeLayout`.

`RelativeLayout` is a subclass of the `ViewGroup` class, which allows the user to define how the UI elements are to be placed on the screen relative to each other. There are other subclasses of the `ViewGroup` as well, including `AbsoluteLayout`, `FrameLayout`, and `SlidingDrawer`, but I won't go into them at this time. You can have the fun of researching and trying them out for yourself.

Go back to the Graphical Layout tab, and you can drag and drop your two buttons into place. Put your buttons in the center of the screen, like in Figure 12-19.



centerHorizontal: true
centerVertical: true

Figure 12-19. *After switching to RelativeLayout and moving your buttons, your application might look like this.*

Notice that if you shift back to the `main.xml` text, the two `Button` objects will be automatically coded for you.

From there, you can play with the size of the buttons by going to the Outline section of Eclipse (on the right). By right-clicking the two buttons, you can manipulate their definitions, including their place on the page, what their text reads, and much more. You might want to take some time to play with the characteristics here, just so you can learn how much control you have as a programmer.

You can also just type over the code in the editor itself. For example, you can simply change the text in quotation marks on two of the `android:text` lines from `Button` to `Welcome` and `Website`, respectively.

Also, it is good to get into the habit of identifying your buttons with something more than just “Button 1” and “Button 2,” as this can lead to confusion about what button does what. It is better to name them by their functionality. Notice that I changed the sections marked `android:id=` to `"@+id/btnWelcome"` and `"@+id/btnWebsite"`, respectively.

Your code for main.xml for NewActivity should look like the following. Don't worry if it doesn't look precisely like this, as it will depend on where you decide your two buttons will appear on your application.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/background"
    >
<Button android:layout_height="wrap_content"
                android:id="@+id/btnWelcome"
                android:layout_width="wrap_content"
                android:text="Welcome"
                android:layout_alignParentTop="true"
                android:layout_centerHorizontal="true"
                android:layout_marginTop="187dp">
    </Button>
    <Button android:layout_height="wrap_content"
                android:id="@+id/btnWebsite"
                android:layout_width="wrap_content"
                android:text="Website"
                android:layout_below="@+id/btnWelcome"
                android:layout_centerHorizontal="true">
    </Button>
</RelativeLayout>
```

As far as a graphical view, it should resemble Figure 12-20.

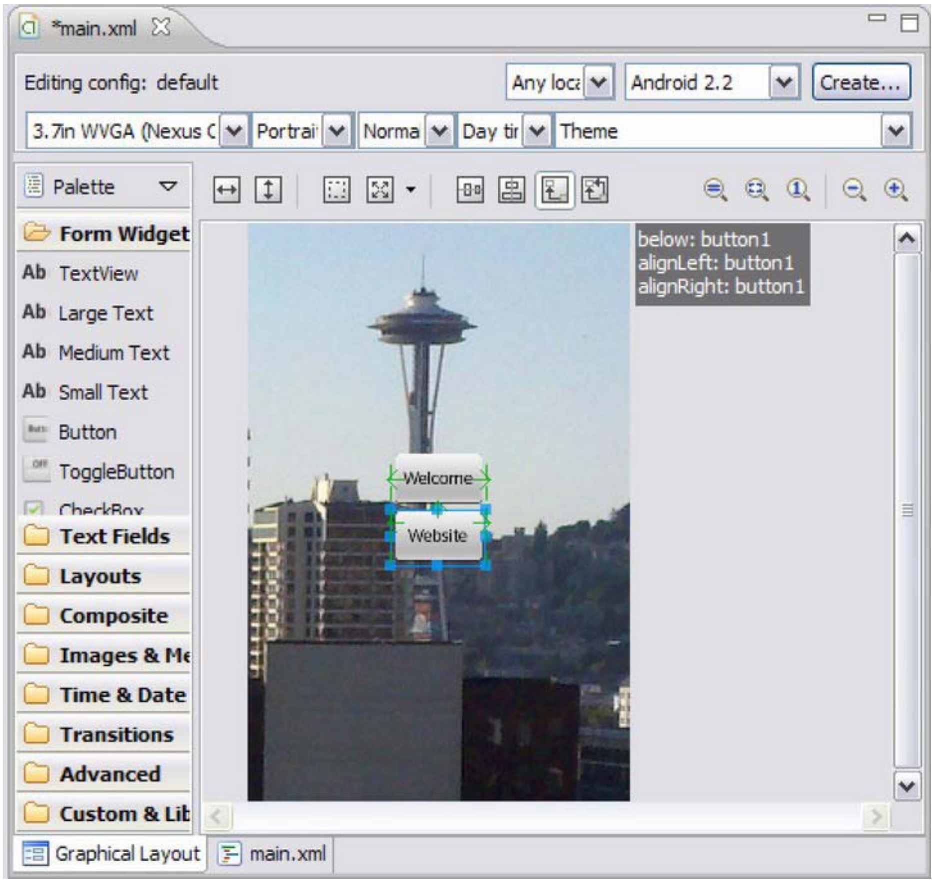


Figure 12-20. What your sample program should look like in graphical layout, provided the main.xml file is programmed with the above code

Now that we got our main.xml file taken care of, go ahead click src/NewProject/NewProjectActivity and open the NewActivity.java. You will see some already programmed code like this:

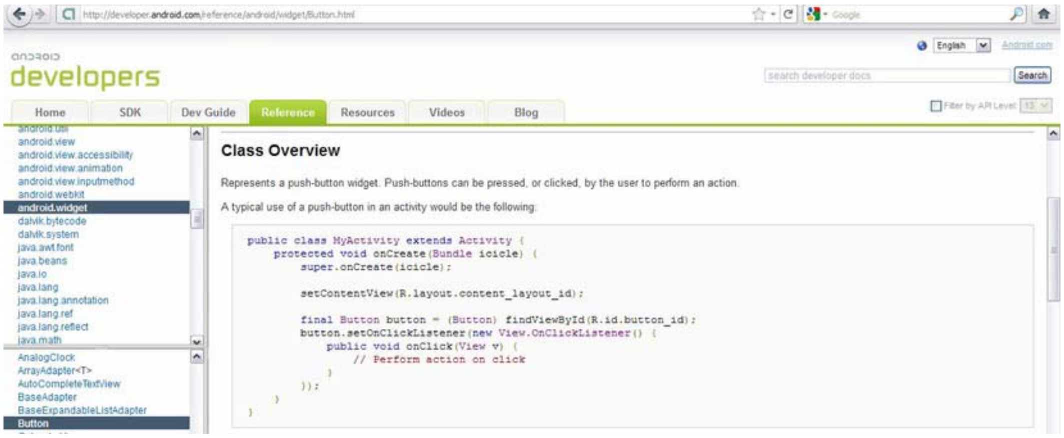
```
package New.Project;

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

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

As you can see, the program is already using the concepts of `packages` and `import` statements that I discussed previously. It is also using the principles of inheritance, as `NewProjectActivity` extends `Activity`. You will recognize `public void onCreate` from earlier as well. Those three lines of code after the `@Override` line set your content view to the `main.xml` screen layout XML definition.

Note the comments in between the `/**` and `*/`. Anything entered in between those two symbols will not be read by Android. This is a useful way of describing to others what a certain line or lines of code can do. Comments also come in handy for times when you need to troubleshoot problems with your code. Putting these symbols around areas of code can help you isolate where the problem is.

Let's get back to our program. If you were to run this application now, you would see what's shown in the graphical layout in Figure 12-20. The buttons here are just objects that do nothing when clicked. You could put stickers on your Android device's touchscreen and get the same effect as what you have just programmed. We need to give them methods so they can do their thing.

So how do we make these buttons work? Fortunately, help is easy to find. When I was in this situation, I simply ran a Google search on "Creating a Button on Android." The first result was the Android Developers web site, at `http://developer.android.com/reference/android/widget/Button.html`. You can check it out in Figure 12-21.



Class Overview

Represents a push-button widget. Push-buttons can be pressed, or clicked, by the user to perform an action.

A typical use of a push-button in an activity would be the following:

```
public class MyActivity extends Activity {
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        setContentView(R.layout.content_layout_id);

        final Button button = (Button) findViewById(R.id.button_id);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Perform action on click
            }
        });
    }
}
```

***Figure 12-21.** The Android Developers site entry on how to create a push-button widget; one of many entries that can help your application*

In all honesty, I wasn't really certain what this code did when I read the entry in Figure 12-21. But I found that with a little tweaking, I was able to come up with some code that allows a button to perform an action on a click. You will find that the Android Developers site and other similar programmer sites can only give us the materials we need to make our programs work, but the assembly of them is up to you. Here is how I altered the instructions:

```
public class NewProjectActivity extends Activity implements OnClickListener{
```

`OnClickListener` is a particular method that will not work unless we use an import statement. So we need to use it in the above section like so:

```
import android.view.View;
import android.view.View.OnClickListener;
```

Now it is time to really bring these buttons to life. We need to tweak the code yet again to adjust for two buttons:

```
Button btnWelcome = (Button)findViewById(R.id.btnWelcome);
        btnWelcome.setOnClickListener(this);
Button btnWebsite = (Button)findViewById(R.id.btnWebsite);
        btnWebsite.setOnClickListener(this);
    }
public void onClick(View v) {
        Button button = (Button)v;
        //Intent intent;
```

What we are going to need to do now is invoke a Java construct known as a `switch`, which is like an `if...then` statement in Android. The case is what we want to happen when our buttons are clicked. For now, we will deal only with the case for the Welcome button, as we need to handle the Website button differently. The default command that you see is what happens if the Website button is clicked.

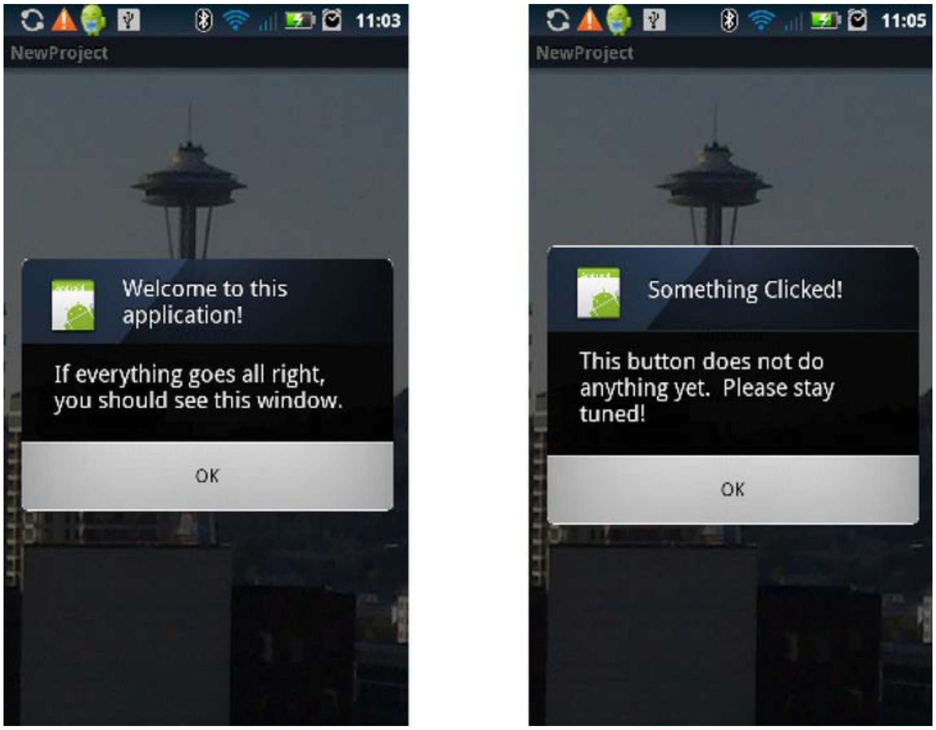
```
switch (button.getId()) {

case R.id.btnWelcome:
PopupMessage("Welcome to this Application!",  "If everything goes right, you should see this
window.");
                        break;
default:
                        PopupMessage("Something Clicked!", "This button does not do anything
yet.  Please stay tuned!");
                        break;
        }
    }
```

If you enter in the code above, then you will notice an error at `PopupMessage`. This is because we haven't created the code for `PopupMessage` yet. We can do that right after our code above, like this:

```
    public void PopupMessage(CharSequence title, CharSequence message) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle(title)
                .setMessage(message)
                .setIcon(R.drawable.icon)
                .setNeutralButton("OK", new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int which) {
                                return;
                        }
                });
        builder.show();

     }
}
```

In case you are wondering what `PopupMessage` will do, it should create a dialog window when you click the button. This window will have a title and a message, which you can input in the parentheses after `PopupMessage` in two sets of quotes, separated by a comma. The `PopupMessage` window will also bring up the icon, as well as an OK button so we can head back to the main menu.

Were you to run this program, you would discover that the Welcome button puts out whatever message you want. On the right in Figure 12-22, you can see what happens when you click the Website, at least for now.



Welcome to this application!

If everything goes all right, you should see this window.

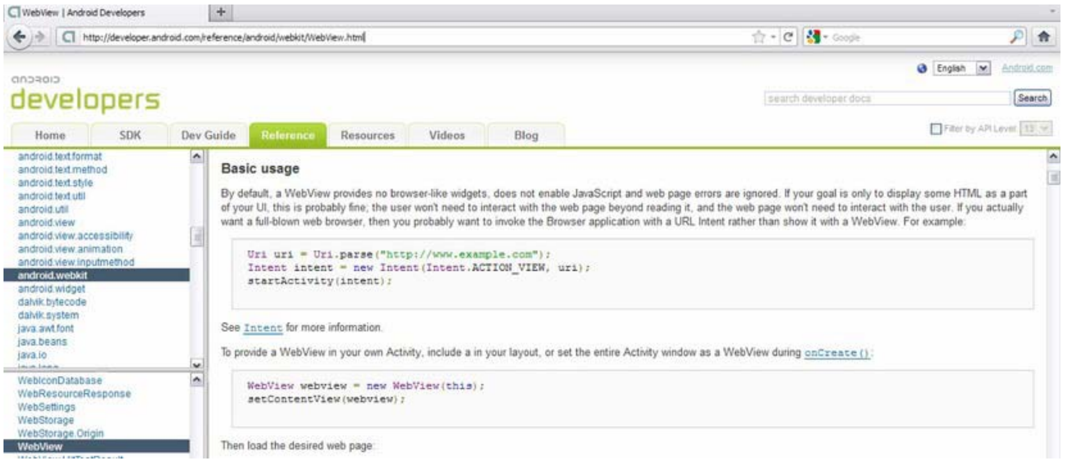
Something Clicked!

This button does not do anything yet. Please stay tuned!

Figure 12-22. *The dialog window on the left is displayed when the Welcome button is clicked, and the dialog window on the right is displayed when the Website button is clicked.*

The reason why I set up the program to display "Something Clicked" after the Website button is clicked was because I did not know how to make my Android application display a web site view. For situations like this, it is helpful to have some placeholder code until you can get the entire program working properly. This is another trick to programming on Android.

Again, we can find out how to display a web site view on the Android Developers site. If you run a search on `WebView` on the web site's search engine, you'll find this URL helpful (see Figure 12-23): `http://developer.android.com/reference/android/webkit/WebView.html`.



Basic usage

By default, a WebView provides no browser-like widgets, does not enable JavaScript and web page errors are ignored. If your goal is only to display some HTML as a part of your UI, this is probably fine; the user won't need to interact with the web page beyond reading it, and the web page won't need to interact with the user. If you actually want a full-blown web browser, then you probably want to invoke the Browser application with a URL Intent rather than show it with a WebView. For example:

```
Uri uri = Uri.parse("http://www.example.com");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
```

See `Intent` for more information.

To provide a WebView in your own Activity, include a in your layout, or set the entire Activity window as a WebView during `onCreate()`:

```
WebView webview = new WebView(this);
setContentView(webview);
```

Then load the desired web page:

Figure 12-23. *The Android Developers site's instructions for WebView*

We're going to apply the basic usage in Figure 12-23 and make a case for when the Website button is clicked. Go into the NewActivityJava file and add this case to the switch area:

```
case R.id.btnWebsite:
                intent = new Intent(v.getContext(), WebsiteActivity.class);
                startActivity(intent);
                break;
```

The code here states that you are creating a new activity, but you are going to need to do a few things first. Notice that WebsiteActivity is underlined red, because you still need to create this activity. Before that, though, double-check to make certain that you have all your code for NewActivityJava correct.

```
package New.Project;


import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import android.view.View;
import android.view.View.OnClickListener;

public class NewProjectActivity extends Activity implements OnClickListener{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button btnWelcome = (Button)findViewById(R.id.btnWelcome);
        btnWelcome.setOnClickListener(this);
        Button btnWebsite = (Button)findViewById(R.id.btnWebsite);
        btnWebsite.setOnClickListener(this);
    }
    public void onClick(View v) {
        Button button = (Button)v;
        Intent intent;

        switch (button.getId()) {
                case R.id.btnWelcome:
                        PopupMessage("Welcome to this application!", "If everything goes all
right, you should see this window.");
                        break;
                case R.id.btnWebsite:
                intent = new Intent(v.getContext(), WebsiteActivity.class);
                startActivity(intent);
                break;

                        default:
                                PopupMessage("Something Clicked!", "This button does not do
anything yet.  Please stay tuned!");
                                break;
        }
    }

    public void PopupMessage(CharSequence title, CharSequence message) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle(title)
                .setMessage(message)
                .setIcon(R.drawable.icon)
                .setNeutralButton("OK", new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int which) {
                                return;
                        }
                });
        builder.show();

    }
}
```

Now that you have the `NewProjectActivity.java` file taken care of, you should make certain that this program will give you a good view of a web site. You will have to open up a new screen for it in your layout file. What you need to do is open `res/layout`, right-click, select New, and then select Other. You will see a window like in Figure 12-24.

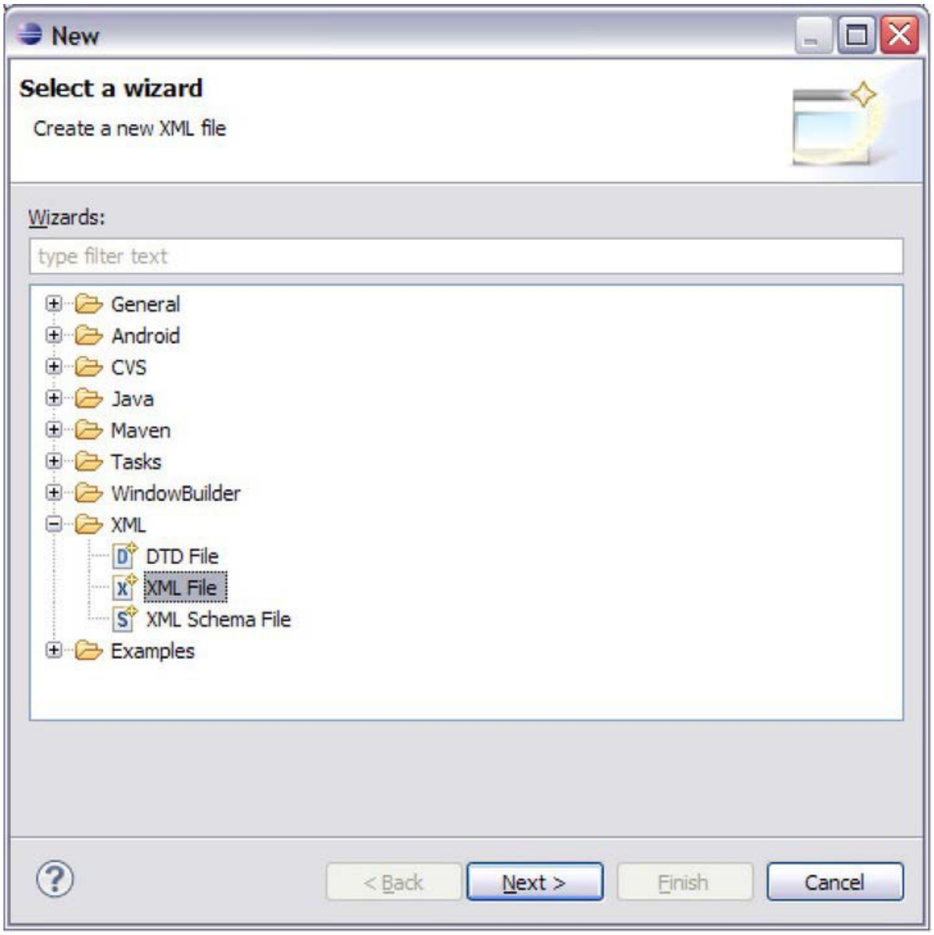


Figure 12-24. Creating a new file in an Android program

Open the XML folder and select XML File. Click Next. You should see a window like in Figure 12-25. Name the file `website.xml` and click Finish.

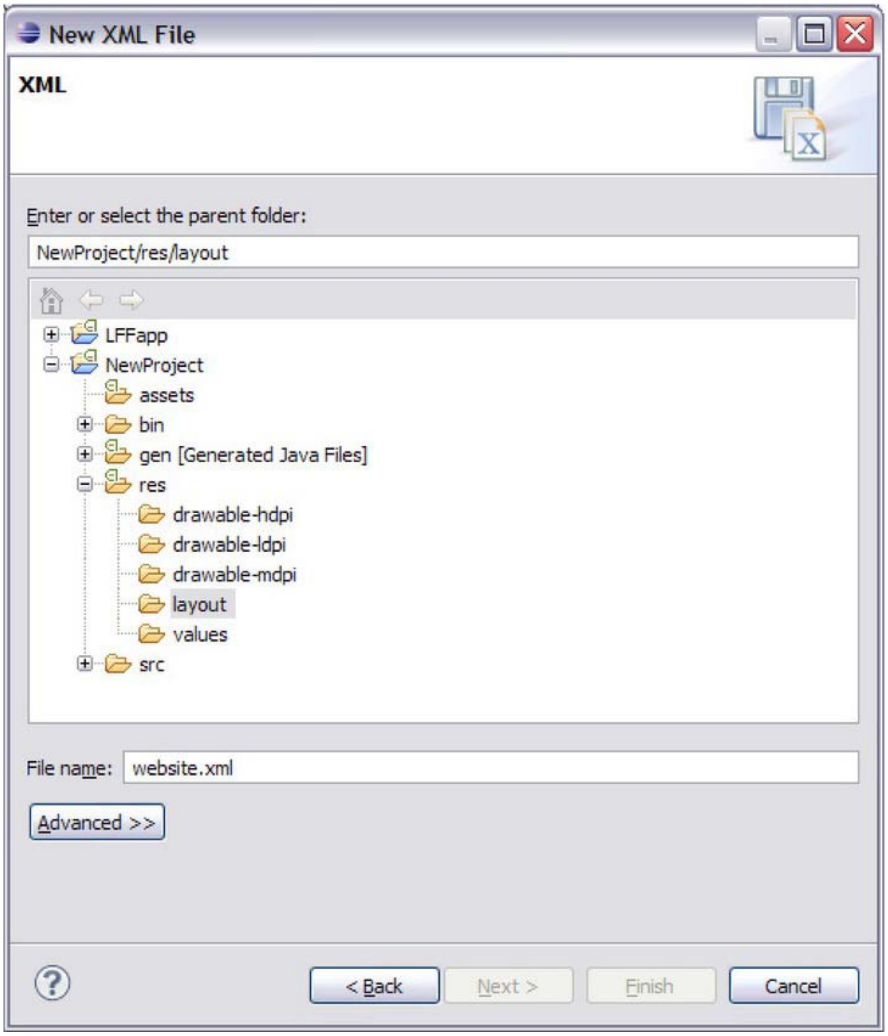


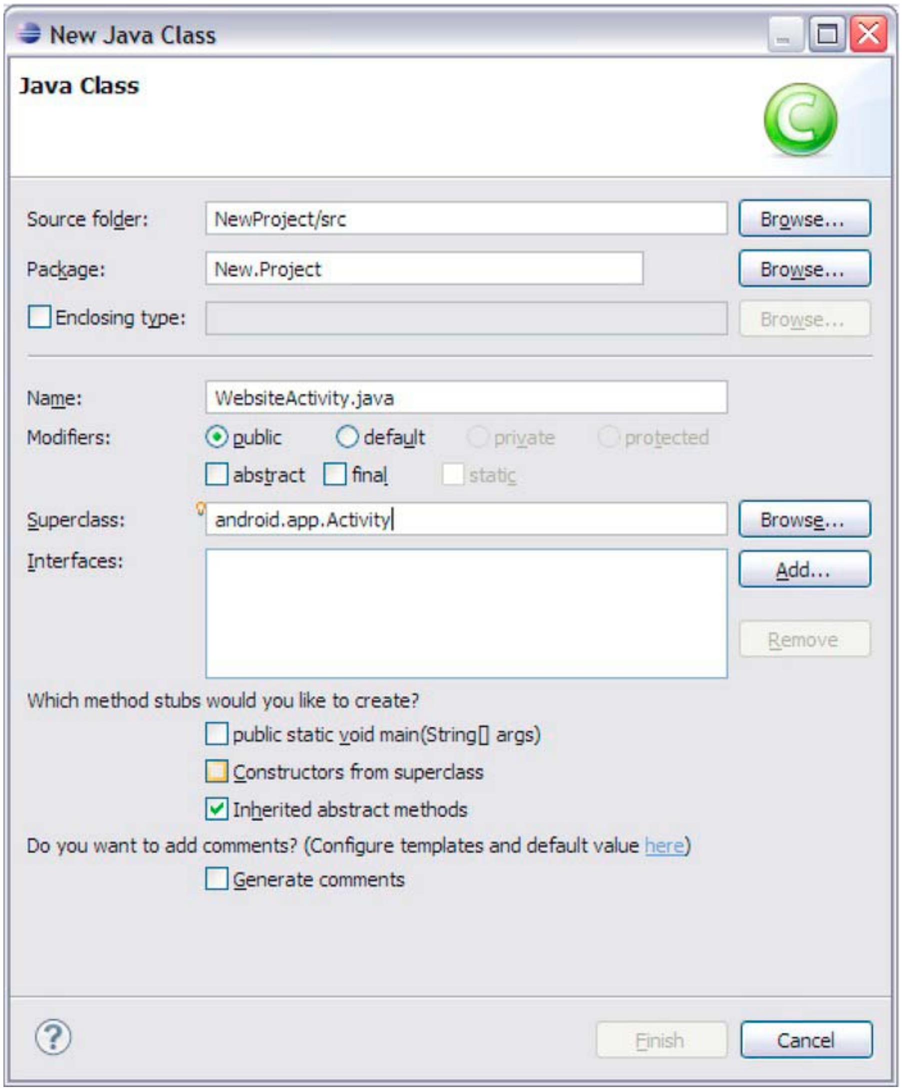
Figure 12-25. *What to name your new XML file*

Go on and open this file in your editor, and you will see some default code, but not much. Go ahead and add the lines below.:

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

This code tells Android to prepare for a web site view. What web site you want there is up to you. However, you will need to prepare for a new activity in the Java code.

In order to do this, you need to create a new directory by right-clicking the `NewActivity.java` file. Select New, and then select Class. You'll then see a window like in Figure 12-26.



Source folder: NewProject/src
Package: New.Project
Enclosing type: [ ]
Name: WebsiteActivity.java
Modifiers: [✓ public] default private protected
abstract final static
Superclass: android.app.Activity
Interfaces:

Which method stubs would you like to create?
[ ] public static void main(String[] args)
[ ] Constructors from superclass
[✓] Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
[ ] Generate comments

Figure 12-26. Setting up a new Java class in Eclipse

The source folder and package can remain the same. You should choose something like `WebsiteActivity.java` for the name, and create a superclass of `android.app.Activity`. Go ahead and click Finish.

Open up WebsiteActivity.java in your editor and copy this code:

```
package New.Project;

import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class WebsiteActivity extends Activity implements OnClickListener  {
        WebView webView;

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

        webView = (WebView) findViewById(R.id.webview);
        webView.getSettings().setJavaScriptEnabled(true);
        webView.loadUrl("http://google.com");

        webView.setWebViewClient(new HelloWebViewClient());
    }

    public void onClick(View v) {
    }

    private class HelloWebViewClient extends WebViewClient {
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String url) {
            view.loadUrl(url);
            return true;
        }
    }

    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if ((keyCode == KeyEvent.KEYCODE_BACK) && webView.canGoBack()) {
            webView.goBack();
            return true;
        }
        return super.onKeyDown(keyCode, event);
    }
}
```

Now that we have the XML and Java written, there is one more thing that we need to do. Open up the AndroidManifest.xml program. Right before the last section of

```
</application>
</manifest> :
```

put this piece of code to set up the web site:

```
<activity android:name=".WebsiteActivity" android:label="Welcome to the Website"
android:theme="@android:style/Theme.NoTitleBar">
                <intent-filter android:label="Welcome to the Website">
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
```

This lets the manifest know that there will be other activities going on that will cause the changing of screens.

There is one more thing that you need to do before you are ready to publish. You have to ask permission to get on the Internet.

Go ahead and enter in this last line of code at the beginning:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="New.Project"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.INTERNET"/>
```

Now run the program again. When you click the Website button, it will now direct you to the web site of your choosing (I have chosen Google for this example). You can even click the Back button on your Android device, and you will be directed to the beginning user interface.

If you've made it this far, then you are now an Android developer. You could go to Chapter 9 and publish this on the Android Market if you wished. This is, of course, a very limited program, but in its defense, I have seen Android applications that do less than this. You can see that with a lot of tweaking and more lines of code, the program would be able to do much, much more.

Summary

Programming an Android application with the Eclipse IDE can be difficult, and the user needs to know how Java SE works. Like any OOP language, it uses self-contained constructs known as objects, which are subclassed by a process known as inheritance. The attributes of the objects are determined by the user, and they are put into action by methods, often with the help of package declarations that acquire code from elsewhere.

It really is all about the proper code when writing an application in Eclipse. In fact, it will not work if there are any errors. If you are having trouble figuring out how to program, check out the Android Developers web site, as well as other various online sources; they might have the exact code you are looking for to get your application to do what you want it to do.

You are going to have to use these basic principles to develop an application that is going to make a killing on the Android market. I wish the best of luck to you as you create, innovate, and market your application using the principles in the previous chapters.