

Android

The Android operating system is released under the open source Apache License and is built on Linux kernel version 2.6. Android is a project of the Open Handset Alliance (OHA). Founded by Google, OHA is an association that includes 65 hardware/software companies and operators, such as KDDI, NTT DoCoMo, Sprint Nextel, Telefónica, Dell, HTC, Intel, Motorola, Qualcomm, Texas Instruments, Samsung, LG, T-Mobile, and Nvidia.

The first Android phone, T-Mobile G1 (also marketed as HTC Dream), was released in October 2008, followed by the release of 12 additional android phones in 2009. There are now dozens of Android mobile devices, including both phones and tablets. In addition to the natural fragmentation of screen size, capabilities, and OS version, developers saw incompatibilities between devices that require specific workarounds for both native applications and browser-based applications.

The Android mobile operating system has a rich set of features. 2D and 3D graphics are supported, based on OpenGL ES 2.0 specifications, and there is good media support for common audio, video, and image formats. Animated transitions and high-resolution, colorful graphics are integrated in the operating system and commonly seen in applications. The Android operating system supports multi-touch input (although it is not supported in every Android device). The web browser is based on the powerful WebKit engine and includes Chrome's V8 JavaScript runtime.

Multitasking of applications is supported. In Android, multitasking is managed by structuring applications as “activities.” Activities have a distinct visual presentation and should be single-purpose, such as taking a photo, searching and presenting results, or editing a contact. Activities may be accessed by other applications as well. A simple application may implement a single activity, but more complex applications may be implemented as a number of activities cohesively presented as a single application.

Android lacks authoritative human interface guidelines, except for fairly narrow icon, widget, and menu design guidelines and broad advice about structuring activities.¹ This lack of standards can make it more challenging to design and develop for Android;

¹ http://developer.android.com/guide/practices/ui_guidelines/index.html

however, Android does include a set of common user interface components that are comparable to those available on the iPhone.

Android Development

To develop for the Android, you can use Windows, Linux, or Mac. Android applications are typically written in Java, but there is no Java Virtual Machine on the platform; instead, Java classes are recompiled in to Dalvik bytecode and run on a Dalvik virtual machine. Dalvik was specially designed for Android to reduce battery consumption and work well with the limited memory and CPU-power of a mobile phone. (Note that Android does not support J2ME.) Since the release of the Android NDK (Native Development Kit) in June 2009, developers may also create native libraries in C and C++ to reuse existing code or gain performance.

The most commonly used and recommended editor is Eclipse with the Android Development Tools plug-in. The plug-in provides a full-featured development environment that is integrated with the emulator. It provides debugging capabilities and lets you easily install multiple versions of the Android platform. As you will see in this chapter, the plug-in makes it easy to get a simple app up and running. If you don't want to use Eclipse, there are command-line tools to create a skeleton app, emulator, debugger, and bridge to an actual device.

In this chapter, you will learn how to set up your Eclipse development environment, create a simple “Hello World” application, launch the application in the emulator, and then build and install the application on an Android device. We also review Android distribution options are also reviewed at the end of this chapter.

Setting Up The Development Environment With Eclipse

You will need to install/set up the following components for your development environment to follow the tutorials in this chapter. Note that Android does not require that you use Eclipse, but it is an easy way to get started with native Android development.

- The Eclipse IDE. Any of the package downloads for the IDE should work fine. <http://www.eclipse.org/downloads/>
- Android Development Tools (ADT) Eclipse plug-in. <http://developer.android.com/sdk/eclipse-adt.html#installing>
- The Android SDK.

Install the Android SDK by following the instructions in the Android developer site:

<http://developer.android.com/sdk/installing.html>.

The tutorial in this chapter assumes the tools are available on your system PATH:

- On Mac or Linux (in ~/.profile or ~/.bashrc): export PATH=\${PATH};<your_sdk_dir> /tools
 - On Windows, add the tools path to your environment variables.
 - One or more versions of the Android platform (to simulate different devices). Unless you know that you'll be using new APIs introduced in the latest SDK, you should select a target with the lowest platform version possible. For compatibility with all devices, we recommend SDK 1.5, API 3.
1. On Mac and Linux, if you have set up your \$PATH as described previously, you can just type on the command line: android (note: if you use the command-line tool, you will need to restart Eclipse to see the installed targets).

On Windows, double-click *SDK Setup.exe* at the root of the SDK directory.

Or in Eclipse, select **Window** ► **Android SDK and AVD Manager**.

2. Under Settings, select “Force https://...” (Figure 3–1).

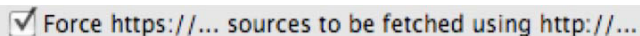


Figure 3–1. Force https

3. Then, under Available packages, select the SDK 1.5, API 3 and Google APIs for Android API 3 (Figure 3–2).

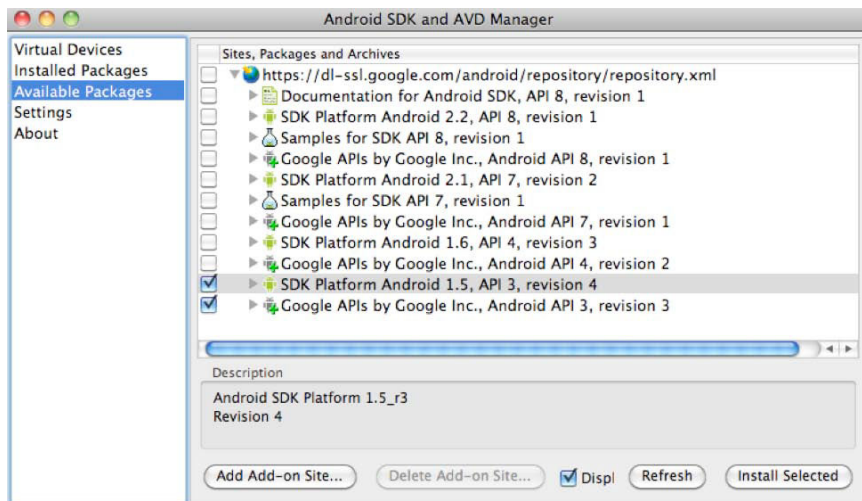


Figure 3–2. Android SDK and AVD Manager: selecting packages to install

4. Create an Android Virtual Device (AVD), as shown in Figure 3–3.

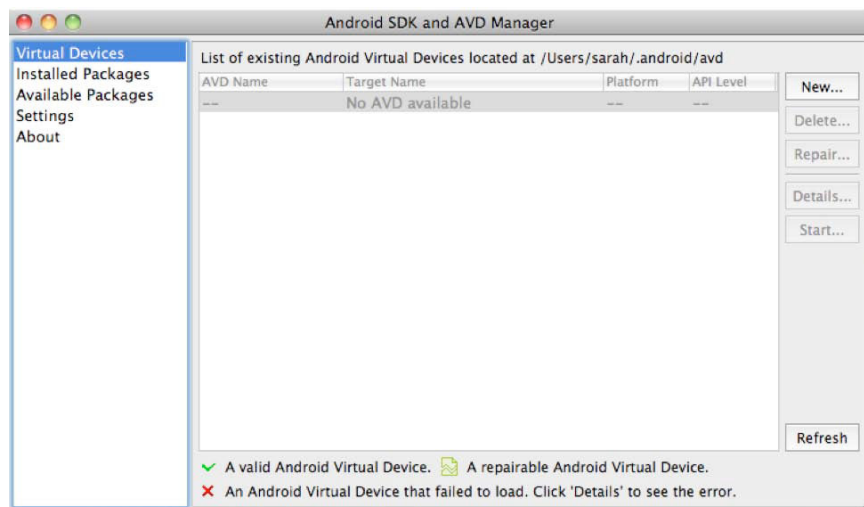


Figure 3–3. *Android SDK and AVD Manager: creating a virtual device*

5. Click **New** and fill in your desired values for virtual device properties (Figure 3–4).

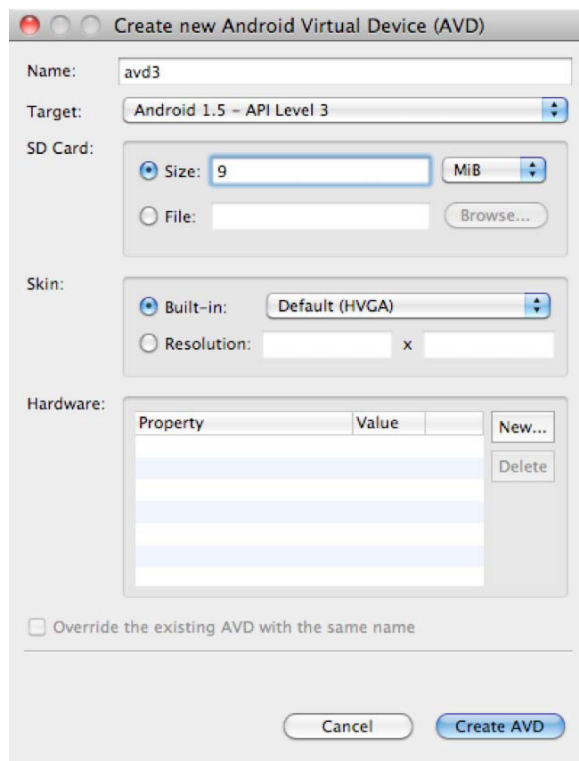


Figure 3–4. *Virtual device details*

Building a Simple Android Application

We will build a simple Hello World application and test it in the Android emulator. While there is a native development kit (NDK) that allows you to build code in C or C++, it is only for creating high-performance libraries. Android applications are always written in Java. This short tutorial will introduce you to building an Android application in Java using the Eclipse IDE.

The goal of this application is to have the user enter his or her name into a text box, press a button, and have the application greet them by name.

1. Select **File** ► **New** ► **Project**.
2. Select **Android** ► **Android Project**, and click **Next** (Figure 3–5).

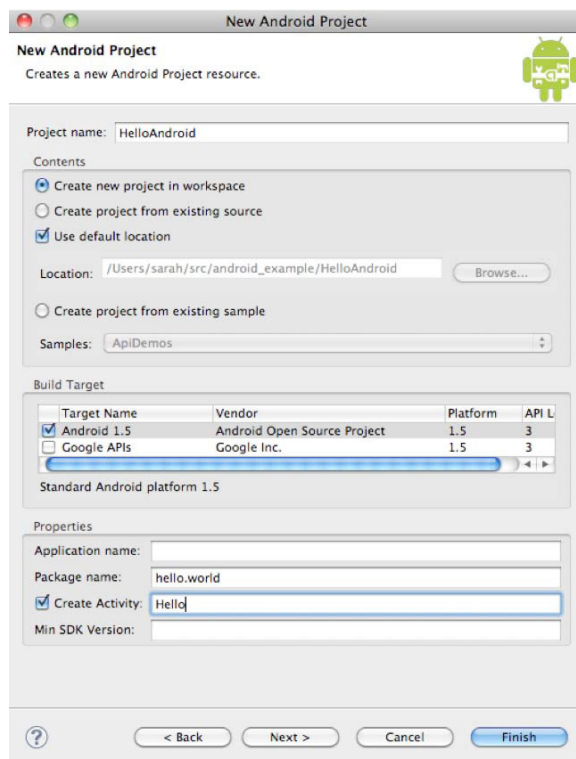


Figure 3–5. *New Android project*

You will need to provide a package name for your app. This can be something like *hello.world* or whatever you want it to be.

Make sure the box labeled **Create Activity** is checked and give your activity a name such as *Hello*. An activity is a UI class that allows you to display things on the screen and get user input. We will modify this class to create a simple UI.

If the box labeled **Min SDK Version** is empty, just click on the lowest SDK version you want to support in the list labeled **Build Target**. This will automatically fill in the correct number for you. This number will be important when you publish your app because it will enable devices to determine if they are able to run your application.

3. Click **Finish**.

Once you have completed the steps to create your application, take a look at the resulting structure in the Eclipse Package Explorer. It should look like Figure 3–6. Navigate into the `src` directory and find your activity class `Hello.java`. Double-click on it to open the file in the editor.

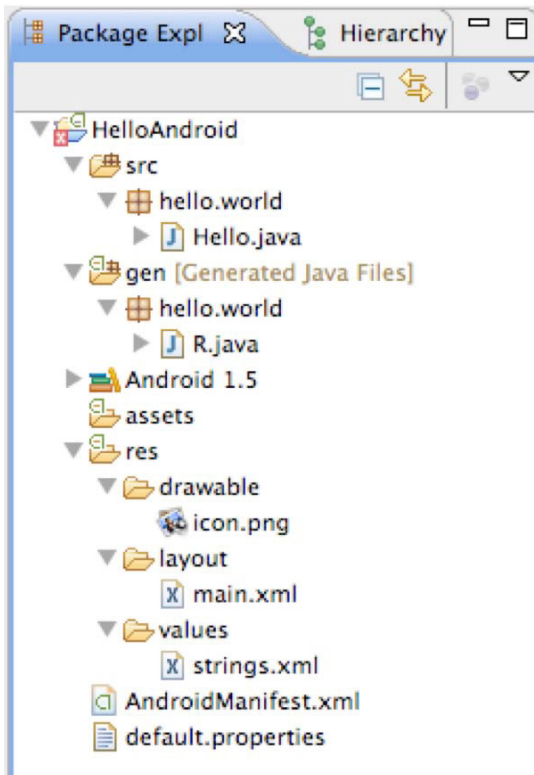


Figure 3–6. Eclipse Package Explorer

4. This class contains a method called “onCreate,” which calls the method “setContentView” passing in “R.layout.main.” This loads the layout that is defined in `res/layout/main.xml` (Figure 3–7).

```

package hello.world;

import android.app.Activity;

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

```

Figure 3–7. *Hello.java generated source code*

5. Double-click on *main.xml* to open it up in the Layout Editor. (You may need to click on the **Layout** tab in the lower left corner of the *main.xml* panel to see the Layout Editor, as illustrated in Figure 3–8.) The Layout Editor is a tool provided by the ADT plug-in for laying out UI widgets in your application. Notice that the main layout contains only a text widget that displays the text “Hello World, Hello!”.

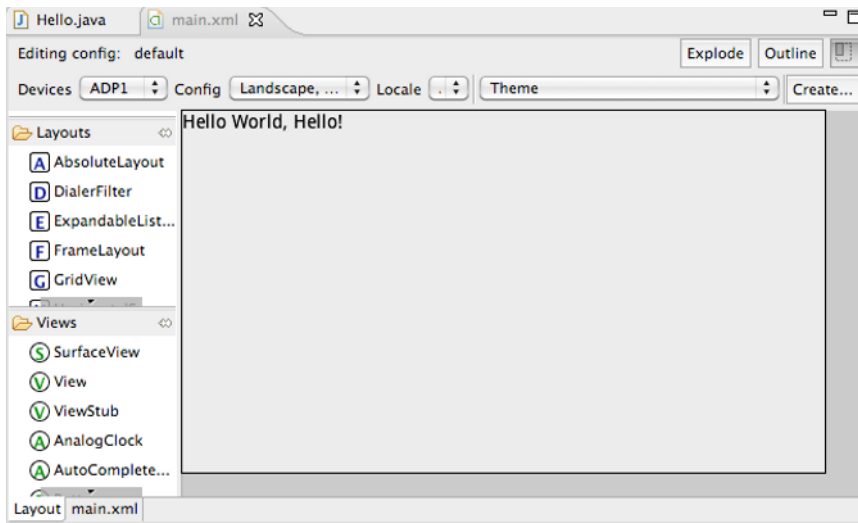


Figure 3–8. *ADT Layout Editor*

6. At this point, you can run your application. Go to the **Run** menu and click **Run**. Select **Android Application** from the list and click **OK**. This will launch the emulator and install your application.

NOTE: If your project contains errors such as: “The project cannot be built until build path errors are resolved.”

Clean the project by choosing: **Project** ► **Clean**

Then click **Run**.

7. The emulator takes a while to start up, and it may start up in a locked state and say “Screen locked, Press Menu to unlock.” Just click the **Menu** button and your application will be launched. Your running application should look like the emulator shown in Figure 3–9.

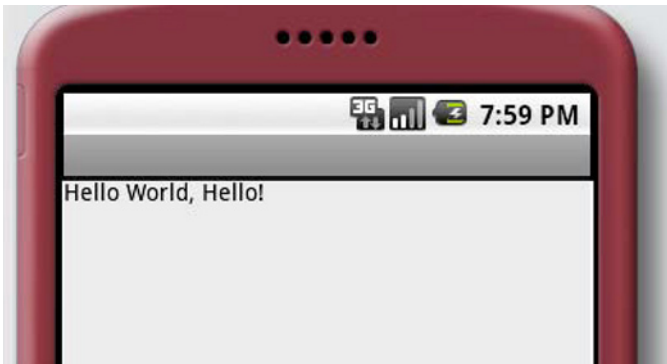


Figure 3–9. *Application running in Emulator*

8. Now that we have a simple application up and running, let’s make it do something a little more interesting. We will add a text box in which the user enters his or her name, and a button that will prompt the Android device to say hello to the user. In the Eclipse editor, open up `res/layout/main.xml` in the Layout Editor. Remove the “Hello World” text from the screen by right-clicking on it and selecting **Remove** from the menu (and confirm when prompted by a pop-up message box).
9. Then add an input text field. Scroll through the **Views** menu (shown in Figure 3–10) to get to the **EditText** item. Click and drag **EditText** into the black layout window. You will now see an editable text item.

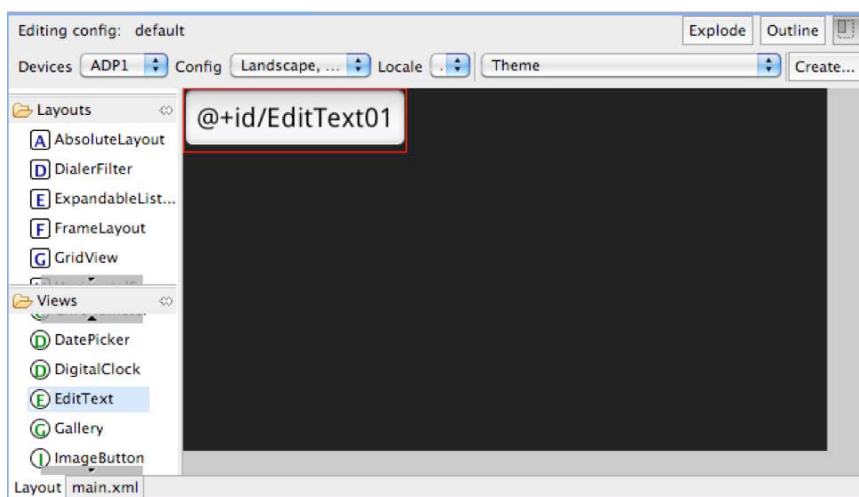


Figure 3–10. *Edit Text added in the Layout Editor*

10. The text that appears in the box is the default text. You can use this to provide guidance to the user about what they should enter in the box. In this application, we will ask the user to input his or her name into the box so we will make the default text say “Name.” To do this, click on the **Properties** tab (shown in Figure 3–11). To make it appear, you may need to double-click on the **EditText** item on the **Outline** tab.

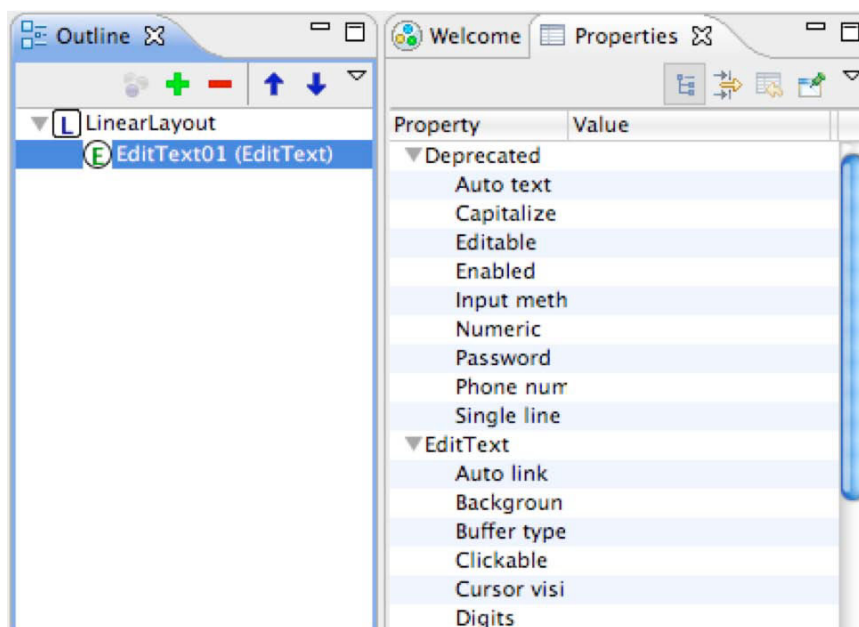


Figure 3–11. *Properties panel*

Scroll down until you see the **Text** property. Click on its value to edit it and change the value to “Name.” We will also change the size of the text box to be more appropriate for a name. Scroll until you see the **width** property. Click on the row to set its value. Give it a value of “300px.”

11. Next, find the **Button** control and add one to the layout. Edit the button’s Text property to say “Hello Android!”. When this button is clicked, we want to grab the contents of the name input box and display text that says hello to the user. We will need to add an empty text control to the layout to hold this text. Find the **TextView** control in the list and drag it under the button. Then delete the default text in the **TextView**.
12. Launch your app and see how the UI Widgets look in the layout. Your emulator should look like Figure 3–12. You should be able to type in the text field, but if you click the button at this point nothing will happen.

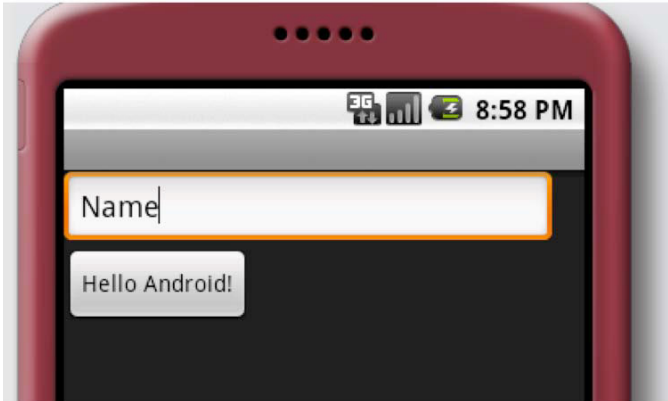


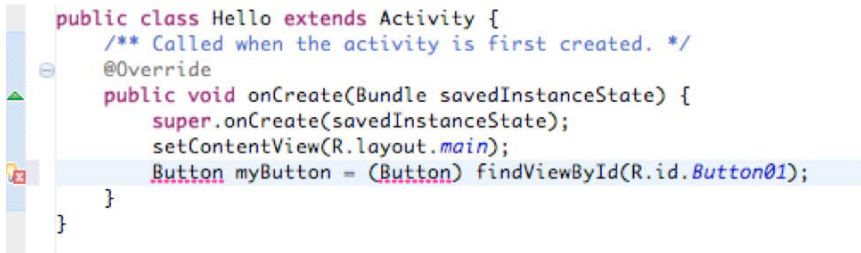
Figure 3–12. Application running in emulator with UI Widgets.

13. To make the button perform an action, you need to attach an event listener. Open up your activity class file *Hello.java* in the *src* directory. You can attach an event listener in the *onCreate* method. First, get a reference to the button using its ID, as shown in Listing 3–1.

Listing 3–1. Button reference

```
Button myButton = (Button) findViewById(R.id.Button01);
```

You can find the ID of your button by looking at its ID property in the properties list. When you first add this code, Eclipse will complain that it doesn’t recognize the type `Button`. Eclipse will automatically add an import statement for you to import the `Button` class. Just click on the red **x** that appears to the left of that line of code and select **Import 'Button' (android.widget)** (Figure 3–13).



```
public class Hello extends Activity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        Button myButton = (Button) findViewById(R.id.Button01);  
    }  
}
```

Figure 3–13. Find button reference

Now that you have a reference to the button, you can add an event listener for the onclick event (see Listing 3–2).

Listing 3–2. *onClickListener*

```
myButton.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
    }  
});
```

Listing 3–2 shows the code to create an empty event listener. Any code you add inside the onClick method will be executed when the button is clicked. Get references to the EditText and TextView controls using the same method used to get the button object (shown in Listing 3–3).

Listing 3–3. *References to EditText and TextView from the Layout*

```
EditText et = (EditText) findViewById(R.id.EditText01); TextView tv = (TextView)  
findViewById(R.id.TextView01);
```

Then set the text in the TextView using the name that was entered into the EditText (with code shown in Listing 3–4).

Listing 3–4. *References to EditText and TextView from the Layout*

```
tv.setText("Hello " + et.getText());
```

The onCreate method should now look like Figure 3–14.

```
package hello.world;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class Hello extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button myButton = (Button) findViewById(R.id.Button01);
        myButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                EditText et = (EditText) findViewById(R.id.EditText01);
                TextView tv = (TextView) findViewById(R.id.TextView01);
                tv.setText("Hello " + et.getText());
            }
        });
    }
}
```

Figure 3–14. *Hello.java*

Now run your application and type your name into the box and click “Hello Android!”. The device will display a customized hello message, including the name you typed.

Simple Application Using Android WebView

This section shows how to embed a WebView, which could allow you to add HTML UI to your native Android application. Create a project, as you did in the previous tutorial (Figure 3–15).

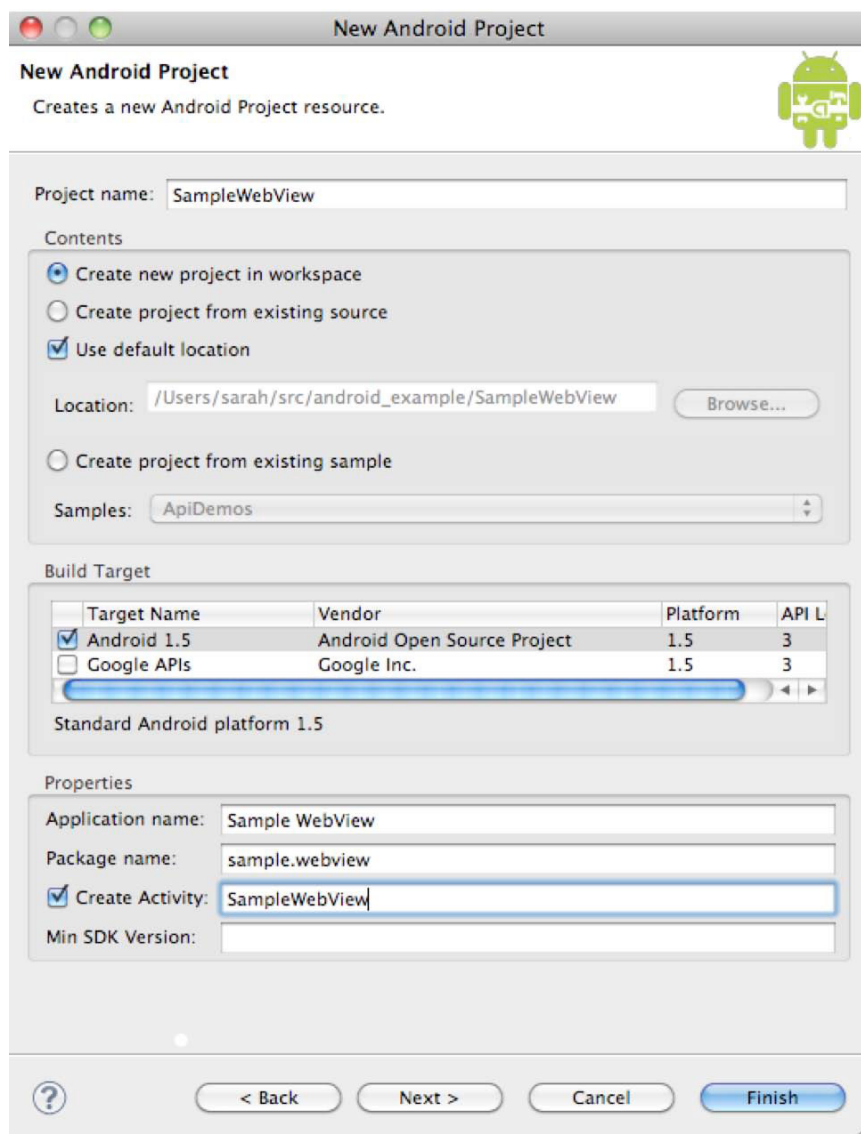


Figure 3–15. *Create Project.*

In this example, we don't use a layout (although you could). Instead, we simply create a new `WebView` and then set the `ContentView` to that instance of the `WebView`. Then we dynamically create some html and load it into the `WebView` (Figure 3–16). This is a very simply example of a powerful concept (Figure 3–17).

```
package sample.webview;

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

public class SampleWebView extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        WebView webView = new WebView(this);
        setContentView(webView);

        String hello = "<html><body><p>This could be HTML UI</p></body></html>";
        webView.loadData(hello, "text/html", "utf-8");
    }
}
```

Figure 3–16. Code for adding a *WebView* to *SampleWebView.java*



Figure 3–17. Application with *WebView* running in the Android simulator

For more details on different ways to use *WebView* in Android, see <http://developer.android.com/reference/android/webkit/WebView.html>.

Building for an Android Device

It is important to test your application on a range of target devices to understand its usability and responsiveness. For example, a G1 is significantly slower than a Nexus One. For some applications, that may make no difference, but for most it will be noticeable. Also, some device features (such as the accelerometer) cannot be tested in

the emulator. Building for an Android Device is easier than other mobile platforms. You do not need to sign up for a developer program or sign your executable just to run it on a device. This section walks you through installing your application on an Android device with USB.

1. Set your application as “debuggable.” In the *manifest.xml*, under the **Application** tab, set **Debuggable** to “true,” as shown in Figure 3–18.

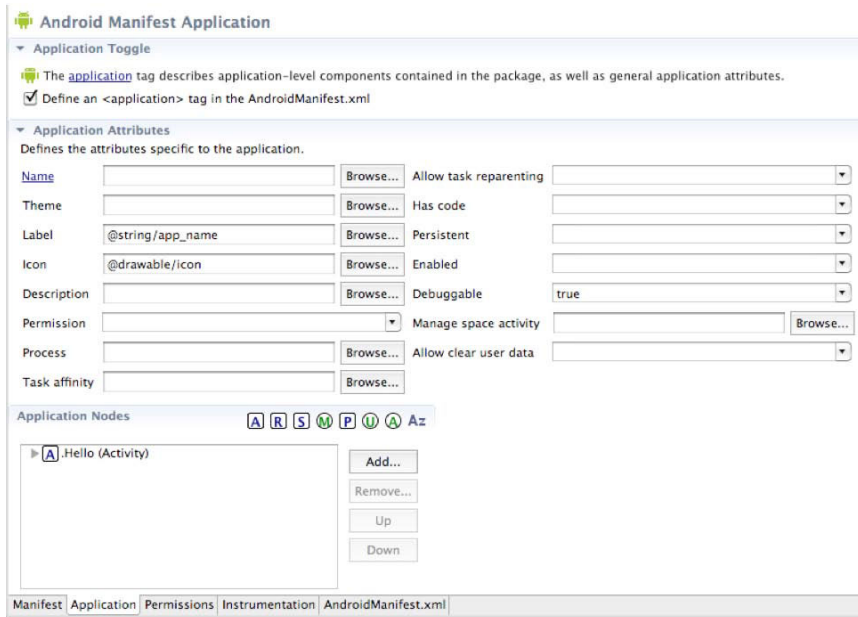


Figure 3–18. Set application to debuggable in manifest.

2. Set your device so it allows **USB Debugging**. In settings, select **Application ► Development**, and make sure **USB Debugging** is checked.
3. Set your system to detect your device. On Mac, this just works. On Windows, you need to install a driver.² On Linux, you need to set up USB rules.³ You can verify that your device is connected by executing *adb devices* from your *SDK tools/* directory. If connected, you’ll see the device name listed as a “device.”
4. Then using Eclipse, run or debug as usual. You will be presented with a **Device Chooser** dialog that lists the available emulator(s) and connected device(s). Select the device upon which you want to install and run the application.

² Download driver: <http://developer.android.com/sdk/win-usb.html>.

³ See detailed Linux instructions: <http://developer.android.com/guide/developing/device.html>.

Distribution on the Web

In order to publish your application, you will need to digitally sign it with a private key. This is a key that you can generate using standard tools, and that you, the developer, hold on to. Self-signed certificates are valid. You can easily generate your private key using Keytool and Jarsigner, both of which are standard Java tools. You can also use an existing key if you already have one.

The Eclipse ADT plug-in makes signing your application very easy, as it provides a wizard that will walk you through creating a private key if you don't already have one, and using it to sign your application. There is a wizard to sign and compile your application for release. For more information on signing your application, see the documentation in the Android developer site.⁴

Once you have a signed .apk file, you can place it on a web site and if you browse to it from the web browser on an Android device, you will be prompted to install the application.

Android Market

The Android Market is the official Google directory for applications (Figure 3–19). With web distribution described previously, this marketplace is just one option for distributing your application. Some Android devices come preinstalled with an application called “Market,” which allows people to access the Android Market. You may also access applications from the Android Market web site.

For developers who would like to submit their applications to the Market, there is a simple sign-up process with a \$25 fee that must be paid with Google checkout.

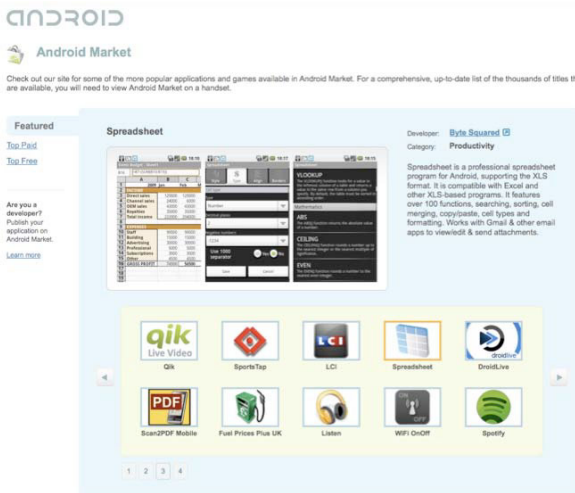


Figure 3–19. *The Android Market*

⁴ Signing your app: <http://developer.android.com/guide/publishing/app-signing.html>.