

Android SDK 2.2

How to Install and Get Started

Introduction

Android is a *mobile operating system* developed by Google, which is based upon Linux. Android competes with Apple's iPhone, RIM's Blackberry, and Microsoft's Windows Phone 7 (previously Windows Mobile 6).

Android applications are written in the Java programming language. However, it uses a JIT compiler to compile into special bytecodes and run on its own Java Virtual Machine (called Dalvik Virtual Machine (DVM)). The DVM bytecodes, unfortunately, are not compatible with the Java Virtual Machine (JVM).

Android supports 2D graphics via its custom library, 3D graphics via the OpenGL ES, databases via the lightweight SQLite, web browser based on the open-source WebKit engine. It supports common audio, video, and image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF), GSM Telephony, Bluetooth, EDGE, 3G, and WiFi, Camera, GPS, compass, and accelerometer.

The mother site for android is <http://www.android.com>. For developers, visit <http://developer.android.com> to download the SDK, tutorials and sample codes.

How to Install Android SDK

Step 0: Read "Installing the SDK" @ <http://developer.android.com/sdk/installing.html>.

Step 1: Download - Download the Android SDK from <http://developer.android.com> ⇒ SDK ⇒ "android-sdk_r07-windows.zip" (about 23M).

Step 2: Unzip - Unzip the downloaded file into a directory of your choice. DO NOT unzip onto the Desktop (its path is hard to locate). I suggest using "d:\myproject". Android SDK will be unzipped into directory "d:\myproject\android-sdk-windows". For ease of use, we shall shorten and rename this directory to "d:\myproject\android". Hereafter, I shall denote the android installed directory as \$ANDROID_HOME.

Step 3: Setup PATH - Include the android's tools directory (\$ANDROID_HOME\tools) to your PATH environment variable.

For Windows: Start "Control Panel" ⇒ "System" ⇒ (Vista/7) "Advanced system settings" ⇒ Switch to "Advanced" tab ⇒ "Environment variables" ⇒ Choose "System Variables" for all users (or "User Variables" for this login user only) ⇒ Select variable "PATH" ⇒ Choose "Edit" for modifying an existing variable ⇒ In variable "Value", APPEND your \$ANDROID_HOME\tools directory (e.g., "d:\myproject\android\tools"), followed by a semi-colon ';', IN FRONT of all the existing path entries. DO NOT remove any entry; otherwise, some programs may not run.

Step 4: Install Eclipse ADT Plugin - Eclipse is an open-source IDE (Integrated Development Environment) that supports Android program development via a plugin called ADT (Android Development

Tools).

If you have yet to install Eclipse, read "[How to Install Eclipse and Get Started](#)".

To install Eclipse Android Development Tools (ADT): Launch Eclipse ⇒ "Help" ⇒ "Install New Software" ⇒ In "Work with" box, enter <https://dl-ssl.google.com/android/eclipse/> ⇒ Check "Development Tools" ⇒ "Next" ⇒ "Finish" ⇒ Restart Eclipse to use ADT plugin.

Step 5: Add Android Platform and Other Components - Launch Android's "SDK manager" (at \$ANDROID_HOME) ⇒ "Select All" ⇒ "Install".

Write your First Android Program Using Eclipse

Step 0: Read "Hello, world" tutorial at <http://developer.android.com/resources/tutorials/hello-world.html>.

Step 1: Create a AVD (Android Virtual Device)

- Launch Eclipse. From "Window" menu ⇒ "Preferences" ⇒ "Android" ⇒ In "SDK Location", enter your Android SDK installed directory (e.g., "d:\myproject\android").
- From "Window" menu ⇒ "Android SDK and AVD Manager" ⇒ "Virtual Devices" ⇒ "New" ⇒ "Create New Android Virtual Device (AVO) dialog appears.
- In "Name" field, enter a name such as "my_avd". Choose a target platform, e.g., Android 2.2 ⇒ "Create AVD".

Step 2: Create a new Android Project

- From Eclipse's "File" menu ⇒ "New" ⇒ "Project..." ⇒ "Android Project" ⇒ "Next" ⇒ The "New Android Project" dialog appears.
- In "Project Name", enter "Hello" (this is the Eclipse's project name). In "Build Target", select "Android 2.2" (or your target device) ⇒ In "Application Name", enter a name that will show up with the program icon, e.g., "Hello". In "Package Name", enter "com.test" (this is the Java package name). In "Create Activity", enter "HelloAndroid" (this is the main Java class name for the android application) ⇒ "Finish". Ignore the warning/error messages, if any.

Step 3: First Android Program to say "Hello, world"

In Eclipse's "Package Explorer" (left-most panel), expand the "Hello" project node ⇒ "src" ⇒ package "com.test" ⇒ Open "HelloAndroid.java" ⇒ Modify the codes as follows:

```
package com.test;

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

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView textView = new TextView(this); // Construct a TextView UI
        textView.setText("Hello, world!");     // Set text for TextView
        setContentView(textView);           // This Activity sets content to the TextView
    }
}
```

You may select "Source" menu ⇒ "Organize Imports" (or ctrl-shift-O) to ask Eclipse to resolve the import.

Step 4: Run the Android Program - Run the application by selecting the "Run" menu ⇒ "Run" ⇒ "Android Application". Be patient, as it takes quite sometimes for the emulator to fire up. Watch the Eclipse's console for messages. Unlock the Android device, by holding and sliding the unlock bar to the right. You shall see "Hello, world!" displayed on the emulator.

Dissecting HelloAndroid.java: An application could have one or more Activity. An Activity is a single, focused thing that the user can do. Our Android program HelloAndroid extends from Activity, so as to override the onCreate() for our own implementation. onCreate() is a call-back method, which will be called by the Android System when the activity is first created. A View is a drawable object (UI component). We construct a TextView (which is a subclass of View), and set it to contain the String "Hello, world". We then set the Activity screen to this TextView.

Android Application Structure: The Android project (under Eclipse ADT) consists of several folders:

- src: Java Source codes. The Java classes must be kept in a proper package with at least two levels of identifiers (e.g., com.test).
- res: Resources, such as drawable (e.g., images, icons), layout (UI components and layout), values (e.g., locale strings).
- gen: Generated Java codes (e.g., to reference the resources) by Eclipse's ADT.
- AndroidManifest.xml: Describe the structure of the application.
- bin: Compiled bytecodes and packaging information.

The binaries together with its resources are bundled into an *Android Package Archive* file (with ".apk" file extension) via the "aapt" tool, for distribution and installation into the mobile devices.

Android Application Manifest File - AndroidManifest.xml: Each Android application has a *manifest file* named AndroidManifest.xml in the project's root directory. It declares the application components.

For example, our "Hello" application, with activity HelloAndroid, has the following manifest (generated automatically by the Eclipse ADT):

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.test"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".HelloAndroid"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- The <manifest> element specifies the package name and android version. It contains one <application> element.
- The <application> element specifies the icon and label used in mobile device's "apps" menu. It contains one or more <activity> elements.
- This application has one activity named HelloAndroid. The <activity> element declares its program

name (.HelloAndroid where . is relative to the package com.test) and label (diplayed as window title). It may contain `<intent-filter>`.

- The `<intent-filter>` declares that this activity is the entry point (`intent.action.MAIN`) of the application when launched (`intent.category.LAUNCHER`).

Beside declaring the application's components, it also provides references to external libraries, and specifies the permissions.

Using XML Layout

Instead of using program codes to create UI. It is better, more flexible, and recommended to layout your UI components via a descriptive XML files. Let's rewrite our hello-world to use XML layout.

- Create a new Android project called "HelloXmlLayout". Use "Hello in XML" for the application name, "com.test" for package name, "HelloAndroidXML" for the activity.
- Select the "HelloXmlLayout" project. Expand the `res/layout` node and open the `main.xml`. Eclipse provides two different views: layout and xml. Select the xml view and enter the followings:

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

Ignore the warning on "no DTD/XML Schema".

- The `main.xml` file declares a `TextView` that holds a text string. Instead of hardcoding the string content, we create a string reference called `hello`, with the actual string coded in `res/values/strings.xml`. This approach is particular useful to support internationalization (i18n) and localization (l10n), as you can customize different strings for different locales.
- Expand `res/values` node and open `strings.xml`, and enter the followings:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello, world! (via XML layout)</string>
    <string name="app_name">Hello in XML</string>
</resources>
```

We write the actual string for the reference `hello`. The reference "app_name" holds the string for the application name (defined while we create the project).

- Next, modify the `HelloAndroidXML.java` to use the XML layout (instead using programming codes to produce the layout), as follows:

```
package com.test;

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

public class HelloAndroidXML extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); // Use "res\layout\main.xml" to layout UI components
    }
}
```

- Run the application, again (use "Run" ⇒ "Run" or "Run" ⇒ "Run History" or push the "run" button). You shall see the new string displayed on the emulator.
- The Eclipse ADT automatically generates a R.java, which keeps track of all the application resources, in hello\gen as follows:

```
package com.test;

public final class R {
    public static final class attr {
    }
    public static final class drawable {           // Inner class R.drawable
        public static final int icon=0x7f020000; // Resource ID R.drawable.icon
    }
    public static final class id {                // Inner class R.id
        public static final int textview=0x7f050000; // Resource ID R.id.textview
    }
    public static final class layout {           // Inner class R.layout
        public static final int main=0x7f030000; // Resource ID R.layout.main
    }
    public static final class string {          // Inner class R.string
        public static final int app_name=0x7f040001; // Resource ID R.string.app_name
        public static final int hello=0x7f040000; // Resource ID R.string.hello
    }
}
```

The R.java indexes all the resources used in this application (R stands for resources). For example, the inner class layout's main (R.layout.main used in the program) reference res\layout\main.xml; the inner class string references res\values\strings.xml; the inner class drawable's icon references res\drawable\icon which keeps the images used as icons in "apps" menu.

- Check out the manifest AndroidManifest.xml. Note the change in activity name.

Next?

Read the "Hello-xxx" tutorials in Android Developers, under the "resource" tag (<http://developer.android.com/resources/index.html>).

Study the sample codes (in Android's "samples" directory, with some explanations in "Sample Code" as above), especially "API Demos".

To run the sample program in Eclipse:

1. Select "File" menu ⇒ "New" ⇒ "Project..." ⇒ "Android" ⇒ "Android Project" ⇒ the "New Android Project" dialog appears.
2. First, choose your "Build Target" ⇒ Then, check "Create project from existing sample" ⇒ Browse and select the desired sample. Once the sample is selected, the rest of the fields will be filled automatically ⇒ Finished.
3. Select "Run" menu ⇒ "Run" ⇒ "Android Application".

REFERENCES & RESOURCES

- Android mother site @ www.android.com
- Android Developers @ developer.android.com.

Latest version tested: Android SDK 2.2, Eclipse 3.6

Last modified: October, 2010

Feedback, comments, corrections, and errata can be sent to Chua Hock-Chuan (ehchua@ntu.edu.sg) | [HOME](#)