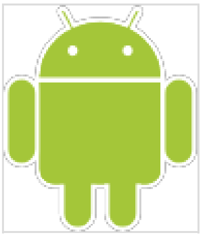
TABLE OF CONTENTS (HIDE)

Android SDK

How to Install and Get Started

1. Introduction

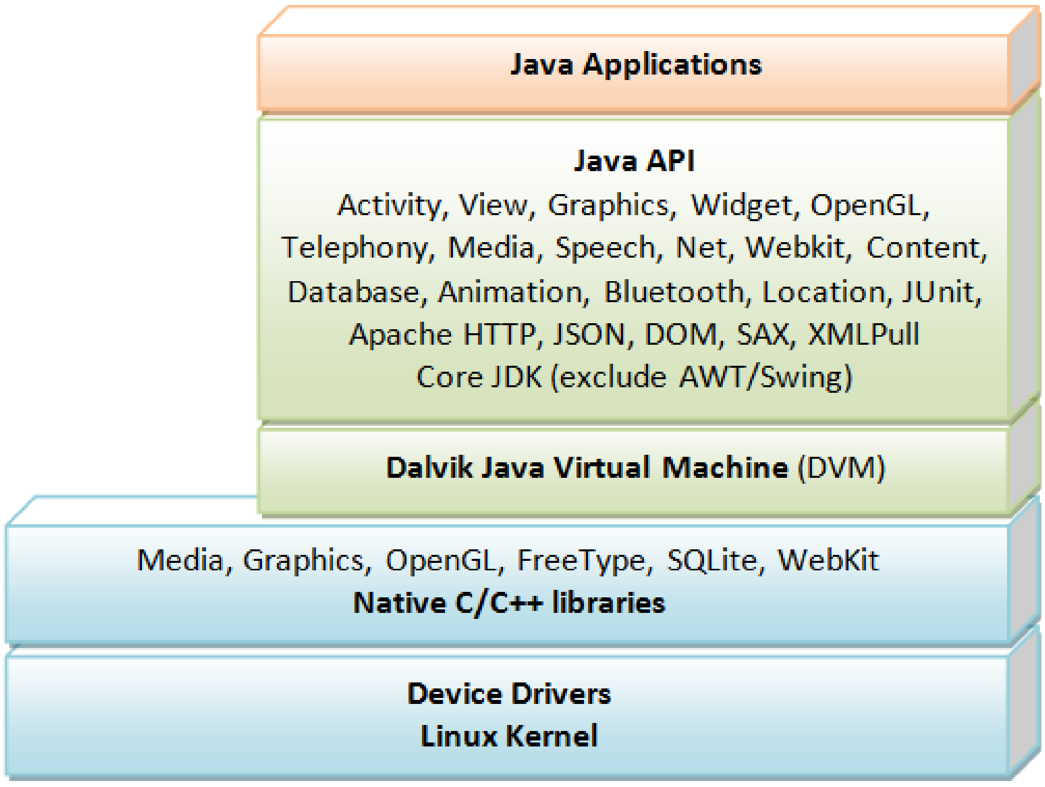


Android is an *Operating System for mobile devices* developed by Google, which is built upon Linux kernel. Android competes with Apple's iOS (for iPhone/iPad), RIM's Blackberry, Microsoft's Windows Phone (previously called Windows Mobile), Sambian OS, and many other proprietary mobile OSes.

Android Platform

Android is based on Linux with a set of native core C/C++ libraries. Android applications are written in Java. However, they run on Android's own Java Virtual Machine, called Dalvik Virtual Machine (DVM), which is optimized to operate on the mobile devices.

The mother site for Android is www.android.com. For developers, visit http://developer.android.com to download the SDK, Android Training, API Gudies and API documentation.


Java Applications
Java API
Activity, View, Graphics, Widget, OpenGL, Telephony, Media, Speech, Net, Webkit, Content, Database, Animation, Bluetooth, Location, JUnit, Apache HTTP, JSON, DOM, SAX, XMLPull
Core JDK (exclude AWT/Swing)
Dalvik Java Virtual Machine (DVM)
Media, Graphics, OpenGL, FreeType, SQLite, WebKit
Native C/C++ libraries
Device Drivers
Linux Kernel

2. How to Install Android SDK

NOTE: Installing all the necessary software needed for Android programming takes times - It could take hours?!

Pre-Installation Check List

Before installing Android SDK, you need to install:

1. Java Development Kit (JDK): Read "How to install JDK".
2. Eclipse: Read "How to install Eclipse".

Now, you are ready to install the Android SDK.

Step 0: Read the "Android Training"

Start from Android's mother site @ http://www.android.com.

Goto "Developers" @ http://developer.android.com. There are three main menus: Design, Develop, and Distribute. Start with "Develop", you can find the "Android Training", "API Guides", "Reference" and "Tools". For beginners, browse through the "Android Training".

Step 1: Download the Android SDK

Download the Android SDK from http://developer.android.com/sdk/index.html. For novices, choose the installer version by clicking the button "Download the SDK for Windows" (`installer_r20-windows.exe`). For Lunix and Mac, select "Other Platforms".
For advanced programmers, I recommend that you choose the ZIP version (under the "Other Platforms") (`android-sdk_r20-windows.zip`). I prefer the ZIP version, as you could install many versions, rename the directory, and remove the installation easily.

Read "Installing the SDK".

Step 2: Install Android SDK

Run the downloaded installer into a directory of your choice, e.g., "`d:\bin\android-sdk`"; or unizp the downloaded ZIP file. Take note of the installed directory. Hereafter, I shall denote the android installed directory as `<ANDROID_SDK_HOME>`.

Step 3: Install Android Platforms and Add-ons via "SDK Manager"

The Android SDK comprises 2 parts: the "tools" and the "Platforms & Add-ons". After running the installer (in the pervious step), the basic "tools" are installed, which are executables that support app development. The "Platforms & Add-ons" consist of ALL Android platforms (from Android 1.x to 4.x) and various Google Add-ons (such as Google Map API), which could be selectively installed.

Now, we have to choose our Android "Platforms & Add-ons".

1. Launch Android's "SDK Manager", which is responsible for managing the software components. If you have run the installer, it should have started the SDK Manager after the installation. Otherwise, launch the SDK manager by running (double-clicking) "`SDK Manager.exe`" under the Android installed directory.
2. In "Add Platforms and Packages", select your target Android platforms and add-ons packages. For novices, select "Android SDK Platform-Tools", and at least one Android platform (e.g., Android 4.1 (API 16)) ⇒ "Install".

Step 4: Install Eclipse Android Development Tool (ADT) Plugin

I suppose that you have installed Eclipse.

1. Launch Eclipse.
2. Install Eclipse ADT: From Eclipse's "Help" menu ⇒ "Install New Software..." ⇒ In "Work with", enter `https://dl-ssl.google.com/android/eclipse/` ⇒ Check ALL boxes ⇒ Next ⇒ Finish ⇒ Restart Eclipse to use ADT plugin.
3. Configure Eclipse ADT: From Eclipse's "Window" menu ⇒ Preferences ⇒ Android ⇒ In "SDK Location", select the Android SDK installed directory (e.g., "`d:\bin\android`-sdk").

Step 5: Create a Android Virtual Device (AVD) (or Emulator) via "AVD Manager"

AVDs are *emulators* that allow you to test your application without the real devices. You can create AVDs for different android platforms (from Android 1.x to Android 4.x) and configurations (e.g., screen size, orientation, SD card and its capacity).

1. From Eclipse's "Window" menu ⇒ Preferences ⇒ Android ⇒ In "SDK Location", enter your Android SDK installed directory (e.g., "`d:\bin\android`-sdk").
2. From "Window" menu ⇒ AVD Manager. (You could also start the AVD manager by running "`AVD Manager.exe`" under the Android SDK installed directory.)
3. In "Android Virtual Device Manager" dialog ⇒ "New".
4. The "Create New Android Virtual Device (AVD)" dialog appears. In "Name", enter "`Android41_Phone`". Select the "Target" Android platform, "SD Card Size" (e.g., 10MB, do not set a huge SD Card size, which would take hours to create.) Skin (screen resolution, e.g., `WVGA800x480` for smart phone - Wiki "Graphics display resolution" for the various resolution) ⇒ "Create AVD".

You can test your AVD by launching the emulator. Start the AVD manager ⇒ Select a AVD ⇒ Click the "Start" button ⇒ Check "Scale display to real size" to get a smaller screen that could fit in your display ⇒ Launch. Wait patiently! The emulator is very slow and take a few MINUTES to launch. You can change the orientation (between portrait and landscape) of the the emulator via "Ctrl-F11".

We typically create different AVDs to emulate different real devices, e.g., `Android41_tablet` of resolution (1024x768 XGA).

Step 6: Setup PATH

You can skip this step now if you are not familiar with PATH, but it is needed later.

Include the android's `tools` directory (`<ANDROID_SDK_HOME>\tools`) and `platform-tools` directory (`<ANDROID_SDK_HOME>\platform-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_SDK_HOME>\tools` directory (e.g., "`d:\bin\android-sdk\tools`"), followed by a semi-colon `';'`, IN FRONT of all the existing path entries. DO NOT remove any existing entry; otherwise, some programs may not run.

Add the `platform-tools` directory to the PATH too.

3. Write your First Android Program Using Eclipse ADT

Android apps are written in Java, and use XML extensively. I shall assume that you have basic knowledge of Java programming and XML.

3.1 Hello-world

Step 0: Read

Go to "Android Training" @ http://developer.android.com/training/index.html, Read "Get Started", "Build your first app".

Step 1: Create a new Android Project

1. Launch Eclipse.
2. From "File" menu ⇒ New ⇒ Project.. ⇒ Android Application Project ⇒ Next.
3. The "New Android Project" dialog appears:
    a. In "Application Name", enter "`Hello Android`" (this is the Android appliation name that shows up on the real device).
    b. In "Project Name", enter "`HelloAndroid`" (this is the Eclipse's project name).
    c. In "Package Name", enter "`com.example.helloandroid`".
    d. In "Build SDK", select the latest version (e.g., Android 4.1 (API 16)).
    e. In "Minimum Required SDK", select "API 8 Android 2.2 (Froyo)" - almost all of the Android devices meet this minimum requirement ⇒ Next.
4. The "Configure Launcher Icon" dialog appears, which allows you to set the application's icon to be displayed on the devices ⇒ Next.
5. The "Create Activitiy" dialog appears. Check "Create Activity" Box ⇒ Select "BlankActivity" ⇒ Next.
6. The "New Blank Activity" dialog appears.
    a. In "Activity Name", enter "`HelloActivity`".
    b. In "Layout Name", enter "`activity_hello`" (default).
    c. In "Title", enter "Hello" (this title will appear as the screen title) ⇒ Finish.

Eclipse ADT creates a default Hello-world Android app.

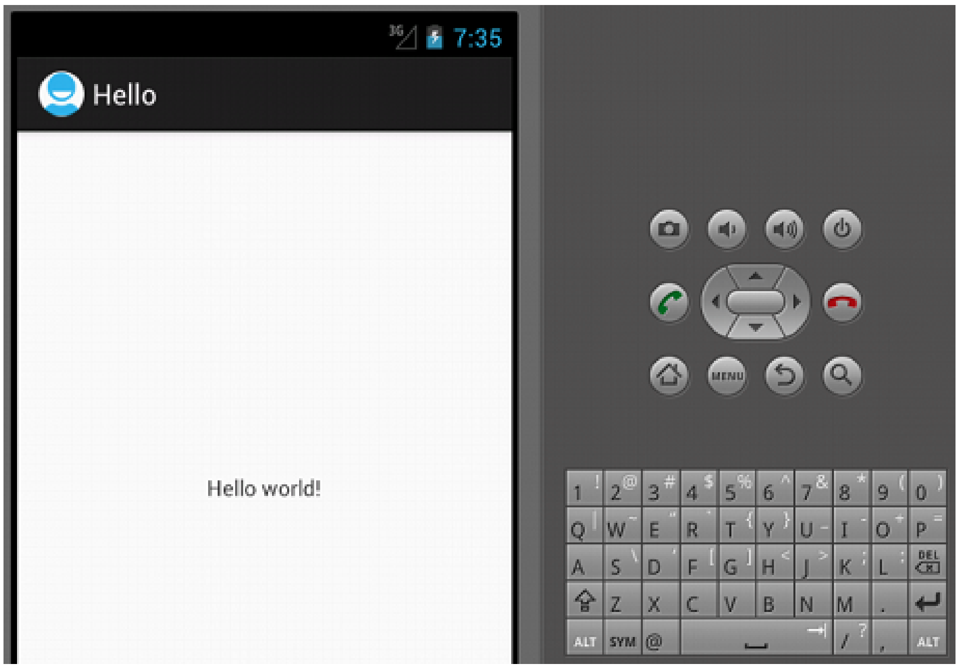
Step 2: Run the Android App

Run the application by right-click on the project node ⇒ "Run As" ⇒ "Android Application".

Be patient! It takes a few MINUTES to fire up the emulator! Watch the Eclipse's status bar for the launching progress; and the console view (or LogCat view) for messages.

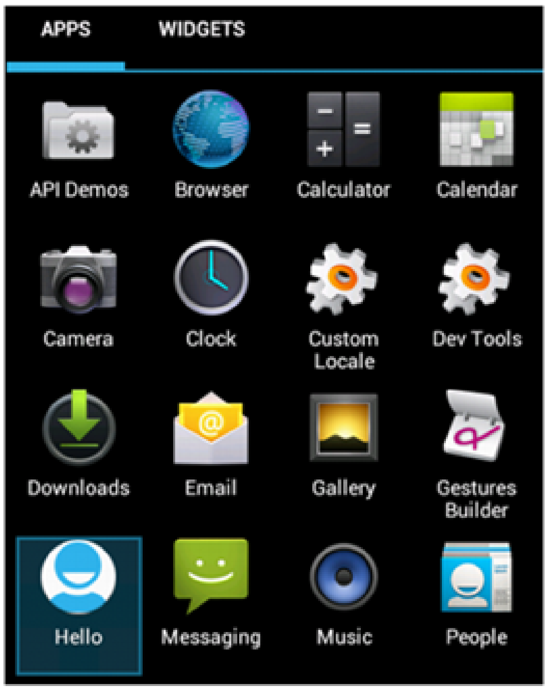
Once the emulator started, unlock the device by holding and sweeping the "lock" to the right (or left). It shall launch your Hello-world app, and displays "Hello, world!" on the screen with a title "`Hello`".

If your program is not launched automatically, try launching it from the "app menu" manually, after the emulator is started. Look for the icon "`Hello`".

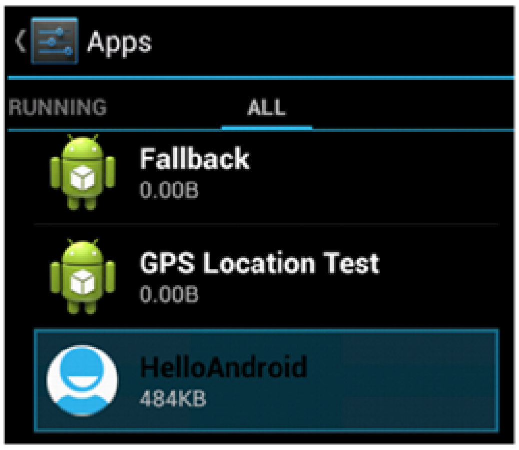


Trying launching the app from "HOME" ⇒ "..." ⇒ Look for the icon "Hello".

Also try "HOME" ⇒ "..." ⇒ "MENU" ⇒ "Manage Apps" ⇒ Select "HelloAndroid" ⇒ Uninstall.



In “Apps”, shown as “Hello” (Title of the Main Activity)



In “manage app”, shown as “HelloAndroid” (Application Name)

NOTE: DO NOT CLOSE the emulator, as it really takes a long time to start. You could always re-run or run new applications on the same emulator.

Run the Android App on Real Devices

To run the Andriod app on the real devices:

1. Connect the real device to your computer. Make sure that you have the "USB Driver" for your device installed

on your computer. You can find the "Google USB Driver" @ http://developer.android.com/sdk/win-usb.html, and Google's certified "OEM USB Drivers" @ http://developer.android.com/tools/extras/oem-usb.html. If you device is not certified there, good luck! It took me many hours to find a compatible driver for my cheap Pad.

2. Enable "USB Debugging" mode on your real device: from "Settings" ⇒ "Applications" ⇒ "Development" ⇒ Check "USB Debugging". This allows Android SDK to transfer data between your computer and your device. Also enable "Unknown source" from "Applications". This allows applications from unknown sources to be installed on the device.
3. You shall see the message "USB Debugging Connected" when you plugs the USB cable into your computer.
4. From Eclipse, right-click on the project node ⇒ Run As ⇒ Android Application.
5. The "Android Device Chooser" dialog appears. Select your real device (instead of the AVD emulator) ⇒ OK.
6. Eclipse ADT installs the app on the connected device and starts it.

You can also use the "`adb`" (Android Debug Bridge) tool (under "`<ANDROID_SDK_HOME>\platform-tools`") to install the "`.apk`" file ("`HelloAndroid.apk`") onto the real devices:

```
// Change directory to <project-root>\bin, where the ".apk" is located
// -d option for real device
> adb -d install filename.apk
2402 KB/s (157468 bytes in 0.064s)
        pkg: /data/local/tmp/filename.apk
Success

> adb --help
```

3.2 Hello-world by Coding

Let's continue from the previous example.

Expand the "`src`" node. Expand the "`com.example.helloandroid`" package node. Open the "`HelloActivity.java`", and replace it with the following codes:

```
package com.example.helloandroid;

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

public class HelloActivity 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 component
        textView.setText("Hello, from my code!"); // Set the text message for TextView
        setContentView(textView);  // this Activity sets its content to the TextView
    }
}
```

Run the application by right-clicking the project node ⇒ "Run As" ⇒ "Android Application". You shall see the message "Hello, from my code!".

Dissecting the "`HelloActivity.java`" - Application, Activity & View

An application could have one or more `Activity`.

An `Activity`, which usually has a screen, is a single, focused thing that the user can *interact* with the applicatoin. The `HelloActivity` extends the `android.app.Activity` class, and overrides the `onCreate()` method. The `onCreate()` is a call-back method, which will be called by the Android system when the activity is launched.

A `View` is a UI component (or widget, or control). We construct a `TextView` (which is a subclass of `android.view.View`), and set its text message. We then set the content-view of the `HelloActivity` 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.example`).
- `res`: Resources, including drawable (e.g., images and icons), layout (UI components and layout), values (e.g., locale strings for internationalization).
- `asset`: where you store raw files (e.g., configuration, audio and image files).
- `gen`: Generated Java codes.
- `bin`: Compiled bytecodes (in sub-directory `classes`), and the "`.apk`" (*Android Package Archive* file).
- `AndroidManifest.xml`: The manifest to describe this app, such as its activities and services.
- `default.properties`: holds various settings for the build system, updated by the ADT.
- `Android 4.1`: the build target platform, with link to Android API ("`android.jar`").

Android Application Descriptor File - "`AndroidManifest.xml`"

Each Android application has a *manifest file* named `AndroidManifest.xml` in the project's root directory. It descibes the application.

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

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.helloandroid"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".HelloActivity"
            android:label="@string/title_activity_hello" >
            <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, version-code and version-name. The version-code is an integer uses by the Android Market to keep track of new version, usually starts at 1. The version-name is a string for your own identification. It is interesting and confusing that two elements are used to identify a version, e.g., Android platform 4.1 has API Level 16.
- The `<manifest>` contains one `<application>` element.

- The `<application>` element specifies the icon, label (the application's title) and theme of this application. It contains one ore more `<activity>` elements.
- This application has one activity named `HelloAndroid`. The `<activity>` element declares its program name ("`.HelloActivity`" where `'.'` is relative to the package `com.example.helloandroid`, you can also use fully-qualified name); and label (the activity's screen title). It may contain `<intent-filter>`.
- The `<intent-filter>` declares that this activity is the entry point (`android.intent.action.MAIN`) of the application. This activity is to be added to the application launcher (`android.intent.category.LAUNCHER`).

3.3 Hello-world using XML Layout

Instead of writing program codes to create the UI. It is more flexible and recommended to layout your UI components via a descriptive XML layout file. In this way, you don't need to hardcode the views, and you can easily modify the look and feel of the application by editing the XML markups. The programming codes can therefore focus on the business logic. (This is similar to the Model-View-Control (MVC) framework used in web applications where the views are written in JSPs using markups and the controls in Servlets, which are clearly separated. This is also similar to client-side programming where HTML is used for contents, CSS for presentation and JavaScript for programming logic. Again, views, contents and programs are clearly separated).

To improve the performance, the XML files are compiled into binary using the Android Asset Packaging Tool (aapt). The devices store them as binary, and the file is read in binary, instead of converting back to XML.

Let's rewrite our hello-world to use XML layout.

Step 1: Create a New Android Application

Create a new Android project called "`HelloAndroidXML`". Use "`HelloAndroidXML`" for the application name and project name, "`com.example.helloandroidxml`" for package name. Create a "`BlankActivity`" called "`HelloActivity`", with layout name "`activity_hello`" and title "`HelloXML`".

Step 2: Define the Layout in XML - "`res\layout\activity_hello.xml`"

Expand the "`HelloAndroidXML`" project node. Expand the "`res`" node and "`layout`" node. Open the "`activity_hello.xml`". Eclipse provides different views for a XML file: Graphics Layout and XML. Select the "`activity_hello.xml`" tab (at the bottom of the panel) and study the code:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:padding="@dimen/padding_medium"
        android:text="@string/hello_world"
        tools:context=".HelloActivity" />
</RelativeLayout>
```

The `activity_hello.xml` file declares a `TextView` that holds a text string. Instead of hardcoding the string content, a string reference called `@string/hello_world` is used, with the actual string coded in `res/values/strings.xml`. This approach is particular useful to support internationalization, as you can customize different strings for different locales.

This activity's uses a "relative layout", where its components are arranged relative to each other. It has width and height matching its parent ("`match_parent`").

The screen contains a `TextView` component, with its text obtained from the string reference "`@string/hello_world`". The `TextView` component has width and height big enough to hold its content ("`wrap_content`"), and is centralized horizontally and vertically.

Step 3: Defining String References and Values - "res\values\string.xml"

Expand `res/values` node. Open `strings.xml`, and study the code:

```
<resources>
    <string name="app_name">HelloAndroidXml</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">HelloXML</string>
</resources>
```

This "`string.xml`" defines these string references and values:

- The string reference "`hello_world`" contains string value of "Hello world!". Change it to "Hello, from XML!".
- The string reference "`app_name`" contains the application's name, that you entered when you created the project. This reference is used in "`AndroidManifest.xml`".
- The string reference "`title_activity_main`" contains the activity's title. This reference is also used in "`AndroidManifest.xml`". Change it to "Hello in XML Layout"

Step 4: The Activity - "HelloActivity.java"

Next, study the "`HelloActitivy.java`" (right-click on the project ⇒ Expand "`src`" node ⇒ Expand package node "`com.example.helloandriodxml`"), as follows:

```
package com.example.helloandroidxml;

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

public class HelloActivity extends Activity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_hello, menu);
        return true;
    }
}
```

The "`HelloActivity`" sets its content-view to "`R.layout.activity_hello`", which is mapped to the XML layout file "`res\layout\activity_hello.xml`" that we have studied earlier.

Step 5: Run the App

Run the application (select "Run As" menu ⇒ "Android Application"). You shall see the new string "Hello, from XML!" and new title "Hello in XML Layout" displayed on the emulator.

Step 6: The Generated Resource Reference Class - "gen\R.java"

The Eclipse ADT automatically generates a "`R.java`", which keeps track of all the application resources, in "`gen`" directory as follows:

```
package com.example.helloandroidxml;

public final class R {
    public static final class attr {
    }
    public static final class dimen {
        public static final int padding_large=0x7f040002;
        public static final int padding_medium=0x7f040001;
        public static final int padding_small=0x7f040000;
    }
    public static final class drawable {
        public static final int ic_action_search=0x7f020000;
        public static final int ic_launcher=0x7f020001;
    }
    public static final class id {
        public static final int menu_settings=0x7f080000;
    }
    public static final class layout {
        public static final int activity_hello=0x7f030000;
    }
    public static final class menu {
        public static final int activity_hello=0x7f070000;
    }
    public static final class string {
        public static final int app_name=0x7f050000;
        public static final int hello_world=0x7f050001;
        public static final int menu_settings=0x7f050002;
        public static final int title_activity_hello=0x7f050003;
    }
    public static final class style {
        public static final int AppTheme=0x7f060000;
    }
}
```

The `R.java` ("`R`" stands for resources) indexes all the resources used in this application in the static *nested classes* such as `layout` and `string`. For example, the inner class `layout`'s property `activity_hello` (`R.layout.activity_hello`) references `res\layout\activity_hello.xml`; the inner class `string` references `res\values\strings.xml`.

4. Publishing Your App

To publish your apps on Android Maket/Google Play, you need to sign your app with your digital certificate. (During development, you app is signed using a debug certificate by the Eclipse ADT.)

Signing Android Apps

1. From Eclipse, right-click on the project ⇒ Export... ⇒ Android ⇒ Export Android Application.
2. In "Project Checks" dialog ⇒ Next.
3. In "Keystore selection" ⇒ Create new keystore ⇒ In "Location", enter the filename for the keystore (e.g., `keystore.db`) ⇒ Set your password for the keystore.
4. In "Key creation", enter the data ⇒ Finish.

References:

1. Signing Your Applications @ http://developer.android.com/tools/publishing/app-signing.html.
2. For keystore management, read JDK documentation on "keytool - Key and Certificate Management Tool".

5. Using Android Debug Bridge (ADB)

ADB lets you manage the connected real devices and emulator AVDs. ADB actually consists of three parts:

- A client program called "`adb.exe`" (kept under the "`<ANDROID_SDK_HOME>\platform-tools`").
- A server that that runs on your development machine, which is responsible for communicating between an "`adb`" client program and the connected real devices or emulator AVDs.
- The ADB daemon, which runs as a background process on every real device and emulator AVD. The ADB server connects to this daemon for communication.

You can use the "`adb`" client to install Android apps (in "`.apk`" format), or copy files between your development machine and real device's internal storage and external SD card.

From a CMD shell, launch the "`adb`" client program (you need to include the "`<ANDROID_SDK_HOME>\platform-tools`" in PATH environment variable):

```
// List the command options
> adb --help
......

// List all the devices connected, take note of the device serial number
> adb devices
......

// Install app in the device of the given serial number
> adb -s device_id install app_filename.apk
// Install app on real device
> adb -d install app_filename.apk
// Install app on emulator
> adb -e install app_filename.apk

// Copy the file from the development machine to the device
> adb push computer_filename sdcard_filename
// Copy the file from the device to development machine
> adb pull sdcard_filename computer_filename
```

6. Next?

Continue with the "Android Training" @ http://developer.android.com/training/index.html.

Read the Android "API Guides" @ http://developer.android.com/guide/components/index.html.

Study the Android sample codes (in Android SDK "samples" directory), especially the "API Demos".

To run the sample programs in Eclipse: Select "File" menu ⇒ "New" ⇒ "Project..." ⇒ "Android" ⇒ "Android Sample Project" ⇒ E.g., "API Demos".

To import an existing project:

1. Select "File" menu ⇒ "New" ⇒ "Project..." ⇒ "Android" ⇒ "Android Project from Existing Code" ⇒ Next.
2. In "Import Project" dialog, browse and select the desired project from the Android SDK's "samples" (e.g., "`ApiDemos`"). Check "Copy projects into workspace".
3. Run the application. Right-click on the project ⇒ "Run As" ⇒ "Android Application".

REFERENCES & RESOURCES

1. Android SDK @ http://developer.android.com/sdk/index.html.
2. "Android Training" @ http://developer.android.com/training/index.html.
3. Android "API Guides" @ http://developer.android.com/guide/components/index.html.

Link to Android's References and Resources

Latest version tested: Android 4.1, Android SDK r20, Eclipse 4.2 (Juno)
Last modified: July, 2012