**Android SDK with Eclipse Tutorial**

**CS5260 – Parallel Processing**

**Lawrence Kalisz**

**September 19, 2011**

This tutorial covers the tools required to create an Android application to install on mobile devices running the Android operating system.

All material in this tutorial has been referenced from <http://developer.android.com/index.html>.

The tutorial sections:

1. System Requirements
2. Installing Android SDK
3. Installing Eclipse IDE and Android ADT plugin
4. Installing an Android Platform
5. Creating an Android Virtual Device (AVD)
6. Creating a new Android Project
7. Deploying application to mobile device

# System Requirements

The sections below describe the system and software requirements for developing Android applications using the Android SDK.

**Supported Operating Systems**

* Windows XP (32-bit), Vista (32- or 64-bit), or Windows 7 (32- or 64-bit)
* Mac OS X 10.5.8 or later (x86 only)
* Linux (tested on Ubuntu Linux, Lucid Lynx)
	+ GNU C Library (glibc) 2.7 or later is required.
	+ On Ubuntu Linux, version 8.04 or later is required.
	+ 64-bit distributions must be capable of running 32-bit applications. For information about how to add support for 32-bit applications, see the [Ubuntu Linux installation notes](http://developer.android.com/sdk/installing.html#troubleshooting).

**Supported Development Environments**

***Eclipse IDE***

* Eclipse 3.5 (Galileo) or greater

**Note:** Eclipse 3.4 (Ganymede) is no longer supported with the latest version of ADT.

* Eclipse [JDT](http://www.eclipse.org/jdt) plugin (included in most Eclipse IDE packages)
* If you need to install or update Eclipse, you can download it from <http://www.eclipse.org/downloads/>.

Several types of Eclipse packages are available for each platform. For developing Android applications, we recommend that you install one of these packages:

* + Eclipse IDE for Java Developers
	+ Eclipse Classic (versions 3.5.1 and higher)
	+ Eclipse IDE for Java EE Developers
* [JDK 5 or JDK 6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) (JRE alone is not sufficient)
* [Android Development Tools plugin](http://developer.android.com/sdk/eclipse-adt.html) (recommended)
* **Not** compatible with Gnu Compiler for Java (gcj)

***Other development environments or IDEs***

* [JDK 5 or JDK 6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) (JRE alone is not sufficient)
* [Apache Ant](http://ant.apache.org/) 1.8 or later

# Download and Install the Android SDK

Here's an overview of the steps you must follow to set up the Android SDK:

1. Prepare your development computer and ensure it meets the system requirements.

Before installing the SDK, the latest Java JDK needs to be installed.

1. Install the SDK starter package.

The SDK starter package is not a full development environment—it includes only the core SDK Tools, which you can use to download the rest of the SDK components (such as the latest Android platform).

Get the latest version of the SDK starter package from the [SDK download page](http://developer.android.com/sdk/index.html).

1. Install the ADT Plugin for Eclipse (if you'll be developing in Eclipse).

Android offers a custom plugin for the Eclipse IDE, called Android Development Tools (ADT), which is designed to give you a powerful, integrated environment in which to build Android applications. It extends the capabilities of Eclipse to let you quickly set up new Android projects, create an application UI, debug your applications using the Android SDK tools, and even export signed (or unsigned) APKs in order to distribute your application. In general, developing in Eclipse with ADT is a highly recommended approach and is the fastest way to get started with Android.

1. Add Android platforms and other components to your SDK.

The last step in setting up your SDK is using the Android SDK and AVD Manager (a tool included in the SDK starter package) to download essential SDK components into your development environment.

You can launch the Android SDK and AVD Manager in one of the following ways:

* From within Eclipse, select **Window > Android SDK and AVD Manager**.
* On Windows, double-click the SDK Manager.exe file at the root of the Android SDK directory.
* On Mac or Linux, open a terminal and navigate to the tools/ directory in the Android SDK, then execute: android

To download components, use the graphical UI of the Android SDK and AVD Manager to browse the SDK repository and select new or updated components (see figure 1). The Android SDK and AVD Manager installs the selected components in your SDK environment.



# Setting up the Eclipse IDE

If you need to install or update Eclipse, you can download it from this location:

<http://www.eclipse.org/downloads/>

The "Eclipse Classic" version is recommended. Otherwise, a Java or RCP version of Eclipse is recommended.

**Downloading the ADT Plugin**

Use the Update Manager feature of your Eclipse installation to install the latest revision of ADT on your development computer.

Follow these steps to download the ADT plugin and install it in your Eclipse environment.

1. Start Eclipse, then select **Help** > **Install New Software...**.
2. Click **Add**, in the top-right corner.
3. In the Add Repository dialog that appears, enter "ADT Plugin" for the *Name* and the following URL for the *Location*: <https://dl-ssl.google.com/android/eclipse/>
4. Click **OK**

Note: If you have trouble acquiring the plugin, try using "http" in the Location URL, instead of "https" (https is preferred for security reasons).

1. In the Available Software dialog, select the checkbox next to Developer Tools and click **Next**.
2. In the next window, you'll see a list of the tools to be downloaded. Click **Next**.
3. Read and accept the license agreements, then click **Finish**.

Note: If you get a security warning saying that the authenticity or validity of the software can't be established, click **OK**.

1. When the installation completes, restart Eclipse.

After you've successfully downloaded the ADT as described above, the next step is to modify your ADT preferences in Eclipse to point to the Android SDK directory:

1. Select **Window** > **Preferences...** to open the Preferences panel (Mac OS X: **Eclipse** > **Preferences**).
2. Select **Android** from the left panel.

You may see a dialog asking whether you want to send usage statistics to Google. If so, make your choice and click **Proceed**. You cannot continue with this procedure until you click **Proceed**.

1. For the *SDK Location* in the main panel, click **Browse...** and locate your downloaded SDK directory.
2. Click **Apply**, then **OK**.

Done! If you haven't encountered any problems, then the installation is complete.

# Install a Platform

To install a platform in Eclipse:

1. In the Android SDK and AVD Manager, choose **Available Packages** in the left panel.
2. Click the repository site checkbox to display the components available for installation.
3. Select at least one platform to install, and click **Install Selected**. If you aren't sure which platform to install, use the latest version.

Within the install manager, you can select which platforms you want to install:



# Creating an Android Virtual Device (AVD)

In this tutorial, you will run your application in the Android Emulator. Before you can launch the emulator, you must create an Android Virtual Device (AVD). An AVD defines the system image and device settings used by the emulator.

To create an AVD:

1. In Eclipse, choose **Window > Android SDK and AVD Manager**.
2. Select **Virtual Devices** in the left panel.
3. Click **New**.

The **Create New AVD** dialog appears.

1. Type the name of the AVD, such as "my\_avd".
2. Choose a target. The target is the platform (that is, the version of the Android SDK, such as 2.1) you want to run on the emulator.

You can ignore the rest of the fields for now.

1. Click **Create AVD**.

# Creating a new Android Project

After you've created an AVD, the next step is to start a new Android project in Eclipse.

1. From Eclipse, select **File > New > Project**.

If the ADT Plugin for Eclipse has been successfully installed, the resulting dialog should have a folder labeled "Android" which should contain "Android Project". (After you create one or more Android projects, an entry for "Android XML File" will also be available.)

1. Select "Android Project" and click **Next**.



1. Fill in the project details with the following values:
* *Project name:* HelloAndroid
* *Application name:* Hello, Android
* *Package name:* com.example.helloandroid (or your own private namespace)
* *Create Activity:* HelloAndroid

Click **Finish**.



Here is a description of each field:

***Project Name***

This is the Eclipse Project name — the name of the directory that will contain the project files.

***Application Name***

This is the human-readable title for your application — the name that will appear on the Android device.

***Package Name***

This is the package namespace (following the same rules as for packages in the Java programming language) that you want all your source code to reside under. This also sets the package name under which the stub Activity will be generated.

Your package name must be unique across all packages installed on the Android system; for this reason, it's important to use a standard domain-style package for your applications. The example above uses the "com.example" namespace, which is a namespace reserved for example documentation — when you develop your own applications, you should use a namespace that's appropriate to your organization or entity.

***Create Activity***

This is the name for the class stub that will be generated by the plugin. This will be a subclass of Android's [Activity](http://developer.android.com/reference/android/app/Activity.html) class. An Activity is simply a class that can run and do work. It can create a UI if it chooses, but it doesn't need to. As the checkbox suggests, this is optional, but an Activity is almost always used as the basis for an application.

***Min SDK Version***

This value specifies the minimum API Level required by your application.

Your Android project is now ready. It should be visible in the Package Explorer on the left. Open the HelloAndroid.java file, located inside *HelloAndroid > src > com.example.helloandroid*). It should look like this:

package com.example.helloandroid;

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

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

Notice that the class is based on the [Activity](http://developer.android.com/reference/android/app/Activity.html) class. An Activity is a single application entity that is used to perform actions. An application may have many separate activities, but the user interacts with them one at a time. The [onCreate()](http://developer.android.com/reference/android/app/Activity.html#onCreate%28android.os.Bundle%29) method will be called by the Android system when your Activity starts — it is where you should perform all initialization and UI setup. An activity is not required to have a user interface, but usually will.

**Construct the UI**

Take a look at the revised code below and then make the same changes to your HelloAndroid class. The bold items are lines that have been added.

package com.example.helloandroid;

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 tv = new TextView(this);
       tv.setText("Hello, Android");
       setContentView(tv);**
   }
}

An Android user interface is composed of hierarchies of objects called Views. A [View](http://developer.android.com/reference/android/view/View.html) is a drawable object used as an element in your UI layout, such as a button, image, or (in this case) a text label. Each of these objects is a subclass of the View class and the subclass that handles text is [TextView](http://developer.android.com/reference/android/widget/TextView.html).

In this change, you create a TextView with the class constructor, which accepts an Android [Context](http://developer.android.com/reference/android/content/Context.html) instance as its parameter. A Context is a handle to the system; it provides services like resolving resources, obtaining access to databases and preferences, and so on. The Activity class inherits from Context, and because your HelloAndroid class is a subclass of Activity, it is also a Context. So, you can pass this as your Context reference to the TextView.

Next, you define the text content with [setText()](http://developer.android.com/reference/android/widget/TextView.html#setText%28java.lang.CharSequence%29).

Finally, you pass the TextView to [setContentView()](http://developer.android.com/reference/android/app/Activity.html#setContentView%28android.view.View%29) in order to display it as the content for the Activity UI. If your Activity doesn't call this method, then no UI is present and the system will display a blank screen.

There it is — "Hello, World" in Android! The next step, of course, is to see it running.

**Run the Application**

The Eclipse plugin makes it easy to run your applications:

1. Select **Run > Run**.
2. Select "Android Application".

To learn more about creating and editing run configurations in Eclipse, refer to [Developing In Eclipse, with ADT](http://developer.android.com/guide/developing/eclipse-adt.html#RunConfig).

The Eclipse plugin automatically creates a new run configuration for your project and then launches the Android Emulator. When the emulator is booted, the Eclipse plugin installs your application and launches the default Activity. You should now see something like this:



The "Hello, Android" you see in the grey bar is actually the application title. The Eclipse plugin creates this automatically (the string is defined in the res/values/strings.xml file and referenced by your AndroidManifest.xml file). The text below the title is the actual text that you have created in the TextView object.

That concludes the basic "Hello World" tutorial.

**Upgrade the UI to an XML Layout**

The "Hello, World" example you just completed uses what is called a "programmatic" UI layout. This means that you constructed and built your application's UI directly in source code. If you've done much UI programming, you're probably familiar with how brittle that approach can sometimes be: small changes in layout can result in big source-code headaches. It's also easy to forget to properly connect Views together, which can result in errors in your layout and wasted time debugging your code.

That's why Android provides an alternate UI construction model: XML-based layout files. The easiest way to explain this concept is to show an example. Here's an XML layout file that is identical in behavior to the programmatically-constructed example:

<?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"/>

The general structure of an Android XML layout file is simple: it's a tree of XML elements, wherein each node is the name of a View class (this example, however, is just one View element). You can use the name of any class that extends [View](http://developer.android.com/reference/android/view/View.html) as an element in your XML layouts, including custom View classes you define in your own code. This structure makes it easy to quickly build up UIs, using a more simple structure and syntax than you would use in a programmatic layout. This model is inspired by the web development model, wherein you can separate the presentation of your application (its UI) from the application logic used to fetch and fill in data.

These XML layout files belong in the res/layout/ directory of your project. The "res" is short for "resources" and the directory contains all the non-code assets that your application requires. In addition to layout files, resources also include assets such as images, sounds, and localized strings.

The Eclipse plugin automatically creates one of these layout files for you: main.xml. In the "Hello World" application you just completed, this file was ignored and you created a layout programmatically. This was meant to teach you more about the Android framework, but you should almost always define your layout in an XML file instead of in your code. The following procedures will instruct you how to change your existing application to use an XML layout.

1. In the Eclipse Package Explorer, expand the /res/layout/ folder and open main.xml (once opened, you might need to click the "main.xml" tab at the bottom of the window to see the XML source). Replace the contents with the following XML:

<?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"/>

Save the file.

1. Inside the res/values/ folder, open strings.xml. This is where you should save all default text strings for your user interface. If you're using Eclipse, then ADT will have started you with two strings, *hello* and *app\_name*. Revise *hello* to something else. Perhaps "Hello, Android! I am a string resource!" The entire file should now look like this:

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello, Android! I am a string resource!</string>
    <string name="app\_name">Hello, Android</string>
</resources>

1. Now open and modify your HelloAndroid class and use the XML layout. Edit the file to look like this:

package com.example.helloandroid;

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

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

When you make this change, type it by hand to try the code-completion feature. As you begin typing "R.layout.main" the plugin will offer you suggestions. You'll find that it helps in a lot of situations.

Instead of passing setContentView() a View object, you give it a reference to the layout resource. The resource is identified as R.layout.main, which is actually a compiled object representation of the layout defined in /res/layout/main.xml. The Eclipse plugin automatically creates this reference for you inside the project's R.java class. If you're not using Eclipse, then the R.java class will be generated for you when you run Ant to build the application. (More about the R class in a moment.)

Now re-run your application — because you've created a launch configuration, all you need to do is click the green arrow icon to run, or select **Run > Run History > Android Activity**. Other than the change to the TextView string, the application looks the same. After all, the point was to show that the two different layout approaches produce identical results.

**R class**

In Eclipse, open the file named R.java (in the gen/ [Generated Java Files] folder).

A project's R.java file is an index into all the resources defined in the file. You use this class in your source code as a sort of short-hand way to refer to resources you've included in your project. This is particularly powerful with the code-completion features of IDEs like Eclipse because it lets you quickly and interactively locate the specific reference you're looking for.

# Deploying application to mobile device

If you are developing on Windows and would like to connect an Android-powered device to test your applications, then you need to install the appropriate USB driver.

Setting up each device may be different in terms of allowing your computer to install applications onto the device.

As an example I am using a Samsung Vibrant smartphone running the Android 2.1 OS.

To allow installs:

* Install OEM driver on computer.
* In the phone, goto **Settings > Applications > Development >** enable USB debugging.
* Eclipse automatically finds the mobile device and installs the application when Run is used in testing.

If you wish to deploy your application for the world to use, you have to register as a developer.

**Distribute your applications to users of Android mobile phones.**

|  |  |
| --- | --- |
|

|  |
| --- |
| **Android Market** enables developers to easily publish and distribute their applications directly to users of Android-compatible phones. https://www.google.com/accounts/android-developer-minidroid.png Android Market is open to all Android application developers. Once registered, developers have complete control over when and how they make their applications available to users. Easy and simple to use. Start using Android Market in 3 easy steps: register, upload, and publish.  |

 |

Before you can publish software on the Android Market, you must do three things:

* Create a developer profile
* Pay a registration fee ($25.00) with your credit card (using Google Checkout)
* Agree to the [Android Market Developer Distribution Agreement](http://www.android.com/us/developer-distribution-agreement.html%22%20%5Ct%20%22policy)