

---

# Develop Android applications with Eclipse

## Get started with Google's Android Development Tools Eclipse plug-in

Skill Level: Intermediate

Frank Ableson ([fableson@msiservices.com](mailto:fableson@msiservices.com))  
Software designer

26 Feb 2008

Android is Google's oft-discussed mobile, wireless, computer, and communications platform. You can take advantage of the powerful Eclipse environment to build Android applications using the Android Eclipse plug-in. This tutorial introduces Android application development with the Eclipse plug-in, otherwise known as Android Development Tools. The tutorial provides an introduction to Android development with a quick introduction to the platform, a tour of Android Development Tools, and includes the construction of two example applications.

## Section 1. Before you start

This tutorial introduces Android application development within the Eclipse environment, including the construction of two example applications. The first is a basic starter application, complete with all phases of building and debugging. The second application examines more complex features of Android, including contacts searching and Google Maps address lookup. To get the most from this tutorial, mobile-development experience is helpful, but not required. Java™ programming skills are required for Android applications, but are not an explicit requirement for this tutorial.

### About this tutorial

Why do we care about Android? Android is an important platform for two reasons. First, the fact that Google is introducing it and the mind-share Android has garnered in such a small amount of time. Google is flexing its muscles and attempting to make a play for the crowded mobile market. Its first salvo into this market, Android and the

Open Handset Alliance, is an impressive starting point. The second reason Android is important is because it isn't just another mobile platform with a phone, menus, and a touchscreen. As you will learn in this tutorial, Android takes a different approach to applications. The architecture of Android permits a highly customizable software environment thanks to its runtime binding of requested actions and the code to satisfy those requests. Whether it's market-driven considerations or the technical aspects of Android, it is a platform worth examination.

This tutorial is organized in the following sections:

- Android basics and required tools
- The Android software developer kit
- Building and debugging the SaySomething Android application
- Creating the content provider and Google Maps application

## System requirements

This tutorial requires several technologies that work together. You need all of them for this tutorial.

### Eclipse Platform

Eclipse is the platform upon which the plug-in runs. Get the latest version of Eclipse Classic (V3.3.1 was used in this tutorial).

### Android Developer Tools

The Android Developer Tools (the Eclipse plug-in) may be installed by following the instructions found at [Installing the Android SDK](#).

### Source code

Source code snippets in this tutorial include:

- `AndroidManifest.xml` snippet — This file is the application deployment descriptor for Android applications.
- `IntentReceiver` — This demonstrates the implementation of an `IntentReceiver`, which is the class that processes intents as advertised by the `IntentFilter` tag in the `AndroidManifest.xml` file.
- `SaySomething.java` — This implements an Android activity, the primary entry point to the sample application of this tutorial.
- `Main.xml` — This contains the visual elements, or resources, for use by Android activities.
- `R.java` — This file is automatically generated by Android Developer Tools and "connects" the visual resources to the Java source code.
- `AndroidManifest.xml` complete — This lists a full `AndroidManifest.xml` file, along with a description of each of the important elements.

- `MobileServiceCallContacts.java` — This contains the code necessary to display contacts as well as react to user input to subsequently perform a Google Maps address lookup.
- 

## Section 2. Introduction to Android

Before diving right into the ins and outs of the Eclipse plug-in and developing Android applications, let's have a look at the architecture of Android and some of the key terms that will be helpful in the tutorial and beyond, as you begin to build Android applications for yourself.

### Android terminology

Android application development under the Eclipse environment requires knowledge of the Eclipse environment and the Android platform. An understanding of the terms below is helpful in Android application development with the Eclipse plug-in.

#### **Open Handset Alliance**

This is the organization led by Google Inc., consisting of numerous public and private organizations.

#### **Android**

The flagship product of the Open Handset Alliance. This is an open source operating environment targeted for mobile devices.

#### **Emulator**

A software tool representative of another system — This is often an environment that runs on a personal computer (IBM®, Mac, Linux®) that emulates another environment, such as a mobile-computing device.

#### **Linux**

An open source operating system kernel at the heart of many computing platforms, including servers, desktop computers, networking appliances, and mobile-computing devices. Android runs on top of a Linux kernel.

#### **Dalvik Virtual Machine**

The Dalvik VM is an operating environment found in the Android stack, which interprets application code at runtime. The Dalvik VM is similar to a compliant Java VM, but the two are not compatible.

## Android basics and required tools

Android is an open source operating system targeted for mobile platforms. At the time of this writing, it is a software-only platform with no publicly available hardware devices.

The Android platform is best described as a *stack* because it is a collection of components, including:

- Linux kernel-based operating system
- Java programming environment
- Tool chain, including compiler, resource compiler, debugger, and emulator
- Dalvik VM for running applications

Now that we've briefly introduced the Android platform architecture, let's look at some important characteristics of the platform from a market perspective.

## Why is Android important?

The computer technology press has lavished attention on Android since its announcement and initial SDK release. Android is important as a platform for two disparate, yet compelling, reasons, among many others.

Android is a market-mover. The mobile-application space is crowded and difficult to gain footing for a newcomer. Google has the resources and the mind-share to make a splash in any market it puts in its sights. Google's entry into the mobile space has been in the works for a few years. Android was a separate and distinct company purchased by Google to give it a jump-start on a mobile presence. Anything Google is doing gets attention, and publicity is good for introducing new platforms. Score one for Android.

The second reason Android is important is because of its application model. Android applications are not monolithic, menu-laden applications that require a great deal of clicking and tapping to operate. Sure, there are menus and buttons to be tapped, but Android has an innovative design element to its architecture known as an *intent*.

## The intent

An intent is a construct that permits an application to issue a request, which is somewhat like a help-wanted sign. It might look like this:

"Wanted: An application to help me look up a contact" or "Wanted: An application to help me display this image" or "Wanted: An application to perform this geographic-based search."

In a similar and complementary fashion, applications can register themselves as capable and interested in performing satisfying various requests or intents. To follow

the classified advertising paradigm, these might look like this:

"Available: Application ready and willing to present contact records in clear, concise manner," or "Available: Application ready and willing to perform a geographic search."

These are examples of `IntentFilters`, which are discussed next.

## The IntentFilter

Applications announce their availability to perform these types of operations via a construct known as an `IntentFilter`. The `IntentFilter` is either registered at runtime or is enumerated in the `AndroidManifest.xml` file. The following snippet comes from an Android application that responds to incoming SMS (text) messages:

### Listing 1. Android application responding to incoming SMS

```
<receiver class=".MySMSMailBox" >
  <intent-filter>
    <action android:value="android.provider.Telephony.SMS_RECEIVED" />
  </intent-filter>
</receiver>
```

After this brief introduction to the intent and `IntentFilter`, the next section introduces the four main types of Android applications.

---

## Section 3. Android applications A quick survey

Let's take a moment to examine the four main types of Android applications: activity, services, receivers, and `ContentProvider`. We will also take a look at the views to display the user-interface (UI) elements.

### Activity

The activity is the most visible and prominent form of an Android application. An activity presents the UI to an application, along with the assistance of a class known as a *view*. The view class is implemented as various UI elements, such as text boxes, labels, buttons, and other UIs typical in computing platforms, mobile or otherwise.

An application may contain one or more activities. They are typically on a one-to-one relationship with the screens found in an application.

An application moves from one activity to another by calling a method known as

`startActivity()` or `startSubActivity()`. The former method is used when the application desires to simply "switch" to the new activity. The latter is used when a synchronous call/response paradigm is desired. In both cases, an intent is passed as an argument to the method.

It is the operating system's responsibility to determine the best-qualified activity to satisfy the specified intent.

## Services and receivers

Like other multitasked computing environments, there are applications running "in the background" that perform various duties. Android calls these types of applications "services." The service is an Android application that has no UI.

The receiver is an application component that receives requests to process intents. Like the service, a receiver does not, in normal practice, have a UI element. Receivers are typically registered in the `AndroidManifest.xml` file. The snippet shown in Listing 1 is an example of a receiver application. Note that the class attribute of the receiver is the Java class responsible for implementing the receiver. Listing 2 is an example of receiver code.

### Listing 2. Receiver code

```
package com.msi.samplereceiver;

import android.content.Context;
import android.content.Intent;
import android.content.IntentReceiver;

public class myreceiver extends IntentReceiver
{
    public void onReceiveIntent(Context arg0, Intent arg1)
    {
        // do something when this method is invoked.
    }
}
```

## Data management with ContentProvider

The `ContentProvider` is the Android mechanism for data-store abstraction. Let's look at a specific type of data found on a mobile device: the address book or contacts database. The address book contains all the contacts and phone numbers a person might require when using a mobile phone. The `ContentProvider` is a mechanism to abstract access to a particular data store. In many ways, the `ContentProvider` acts in the role of a database server. Operations to read and write content to a particular data store should be passed through an appropriate `ContentProvider`, rather than accessing a file or database directly. There may be both "clients" and "implementations" of the `ContentProvider`.

The next section introduces Android views, the UI mechanism for putting things on the screen of an Android device.

## Views

The Android activity employs views to display UI elements. Views follow one of the following layout designs:

### **LinearVertical**

Each subsequent element follows its predecessor by flowing beneath it in a single column.

### **LinearHorizontal**

Each subsequent element follows its predecessor by flowing to the right in a single row.

### **Relative**

Each subsequent element is described in terms of offsets from the prior element.

### **Table**

A series of rows and columns similar to HTML tables. Each cell can hold one view element.

Once a particular layout (or combination of layouts) has been selected, individual views are used to present the UI.

View elements consist of familiar UI elements, including:

- Button
- ImageButton
- EditText
- TextView (similar to a label)
- CheckBox
- Radio Button
- Gallery and ImageSwitcher for displaying multiple images
- List
- Grid
- DatePicker
- TimePicker
- Spinner (similar to a combo box)
- AutoComplete (EditText with auto text-complete feature)

Views are defined in an XML file. Listing 3 shows an example of a simple `LinearVertical` layout.

### Listing 3. Simple LinearVertical layout

```
<?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"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Activity 1!"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Activity 1, second text view!"
    />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Switch To Activity 2"
    id="@+id/switchto2"
    />
</LinearLayout>
```

Note that each element has one or more attributes in the Android name space.

The next section walks through obtaining the Android SDK and configuring it for use with Eclipse.

---

## Section 4. Android Software Developer Kit

Now that we have a feel for the Android platform, let's get the Eclipse environment set up for Android development so we can create our example applications. This section walks through obtaining the Android SDK and configuring it for use with Eclipse.

### Obtaining and installing Eclipse

If Eclipse is not installed, download it and install the latest stable release of the Eclipse IDE from the Eclipse Foundation (see [Resources](#)). The installation is a compressed folder. Extract the contents of the folder to a convenient place on your computer. The installer does not create any icons or shortcuts on Windows®. For purposes of this tutorial, the Eclipse folder will be located in the c:\software\eclipse directory.

To start Eclipse, double-click on eclipse.exe found in the eclipse installation directory. This will start the IDE. The software prompts for a "workspace" and suggests a default location, such as c:\documents and settings\username\workspace. Choose this location or specify an alternative



workspace location.

Once Eclipse is loaded, click the **Workbench - Go to the workbench** icon on the main screen.

Now it's time to obtain the Android SDK.

## Obtaining and installing the Android SDK

Find the Android downloads in [Resources](#).

There are SDK installation versions available for Windows, Mac OS X (Intel® only), and Linux (i386). Select the latest version of the SDK for the desired platform. Note that at the time of this writing, the latest Android SDK version is marked m3-rc37a.

The Android SDK is a compressed folder. Download and extract the contents of this file to a convenient place on your computer. For purposes of this tutorial, the SDK is installed to c:\software\google\android\_m3-rc37a. Obviously, if you are installing this on Mac OS X and Linux, you should install the SDK where you usually put your development tools.

Both Eclipse and the Android SDK are installed. It's time to install the Eclipse plug-in to take advantage of the Eclipse environment.

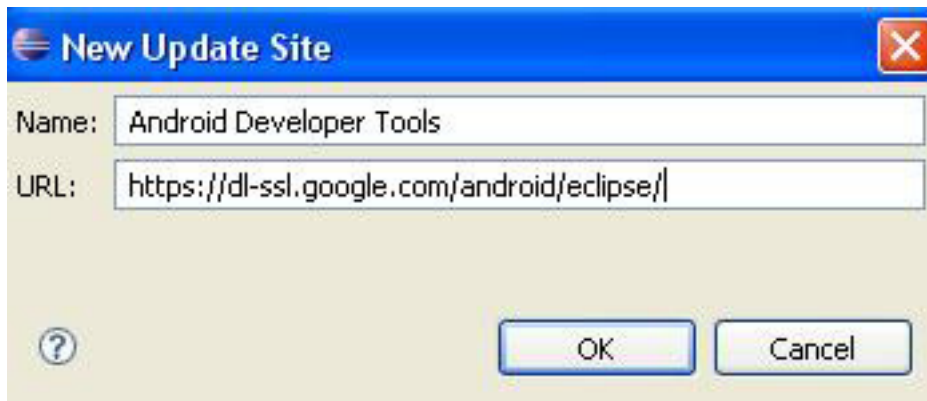
## Obtaining and installing the Eclipse plug-in

The following steps demonstrate the installation of the Eclipse plug-in, officially known as Android Developer Tools. Note that alternative installation directions are available from the Android Web site. See [Resources](#) for more information.

Install the Android Developer Tools:

1. Run the "Find and Install" feature in Eclipse, found under the **Help > Software Updates** menu.
2. Select the **Search for new features to install** option.
3. Select **New Remote Site**. Give this site a name, such as "Android Developer Tools," for example. Use the following URL in the dialog: <https://dl-ssl.google.com/android/eclipse>. Please note the HTTPS in the URL. This is a secure download.

### Figure 1. New Update Site



4. A new entry is added to the list and is checked by default. Click **Finish**. The search results display the Android Developer Tools. Select the **Developer Tools** and click **Next**.
5. After reviewing and accepting the license agreement, click **Next**. Note that the license agreement includes a special requirement for use of the Google Maps API.
6. Review and accept the installation location, then click **Finish**.

The plug-in is now downloaded and installed. The plug-in is not signed (at the time of writing), so proceed at your own comfort level by clicking on **Install All**, then restart Eclipse.

## Configuring the Eclipse plug-in

Once Eclipse is restarted, it is time to connect the plug-in to the SDK installation. Select **Preferences** under the Window menu. Click on the **Android** item in the tree view to the left. In the right-hand pane, specify the SDK installation location. The value used for this tutorial is `c:\software\google\android\m3-rc37a` (again, use appropriate locations in Mac OS X and Linux installations).

Once the SDK location is specified, there are three other sections that may be configured. They are mentioned here briefly:

- The Build section has options for rebuilding resources automatically. Leave this checked. The Build option can change the level of verbosity. Normal is the default.
- DDMS — Dalvik Debug Monitor Service is used for peering into a running VM. These settings specify TCP/IP port numbers used for connecting to a running VM with the debugger and various logging levels and options. The default settings should be just fine.
- LogCat — This is a log file created on the underlying Linux kernel. The font is selectable in this dialog. Adjust this as desired.

Congratulations! The Eclipse environment is ready to create Android applications.

---

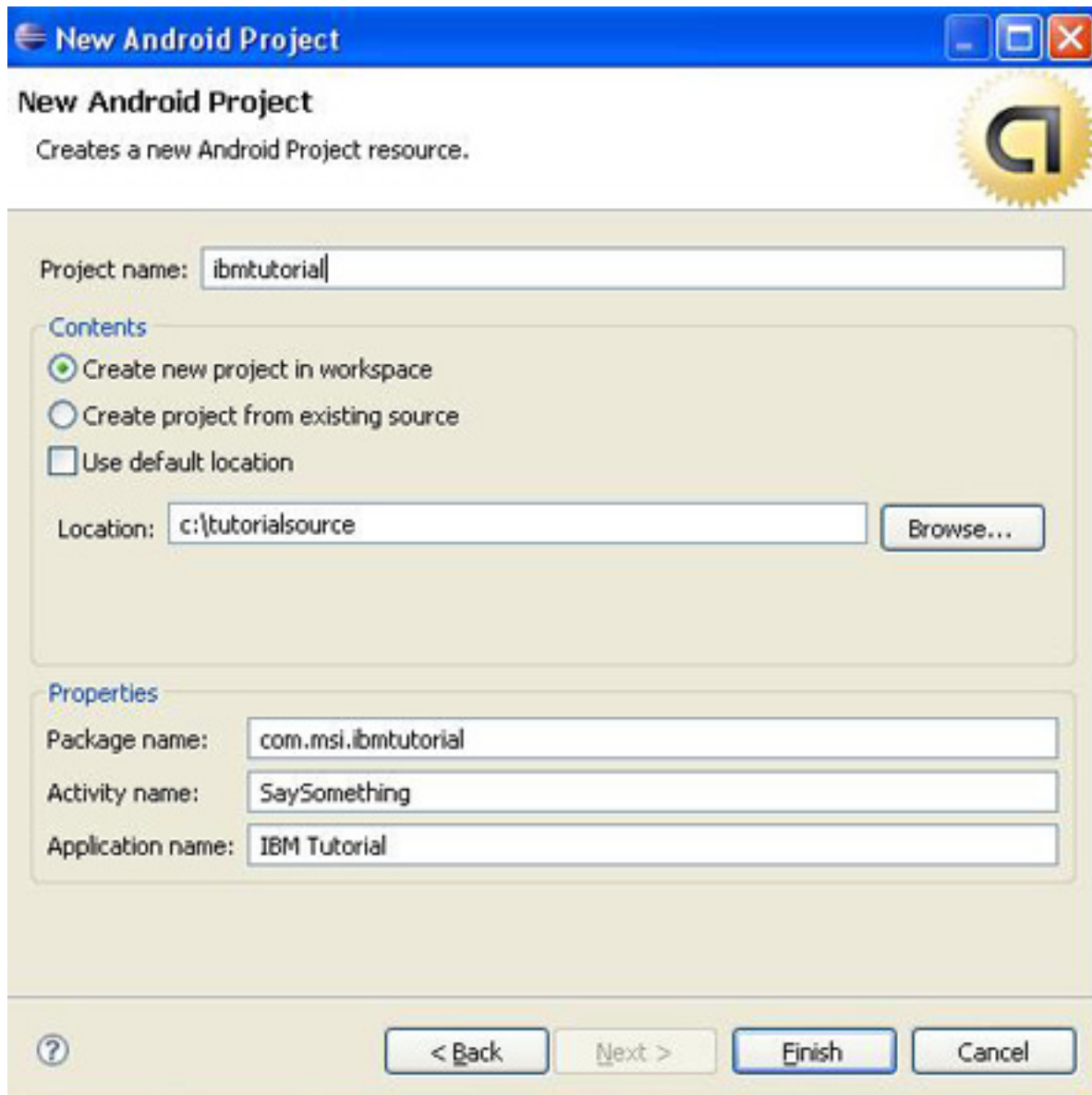
## Section 5. Building the SaySomething Android application

This section creates a basic Android application, called SaySomething, using the Android Developer Tools. Once the application is created, we will debug and run it.

### New project wizard

The first step is to create a new project. Select the wizard for Android project, as shown below.

#### **Figure 2. New project wizard**

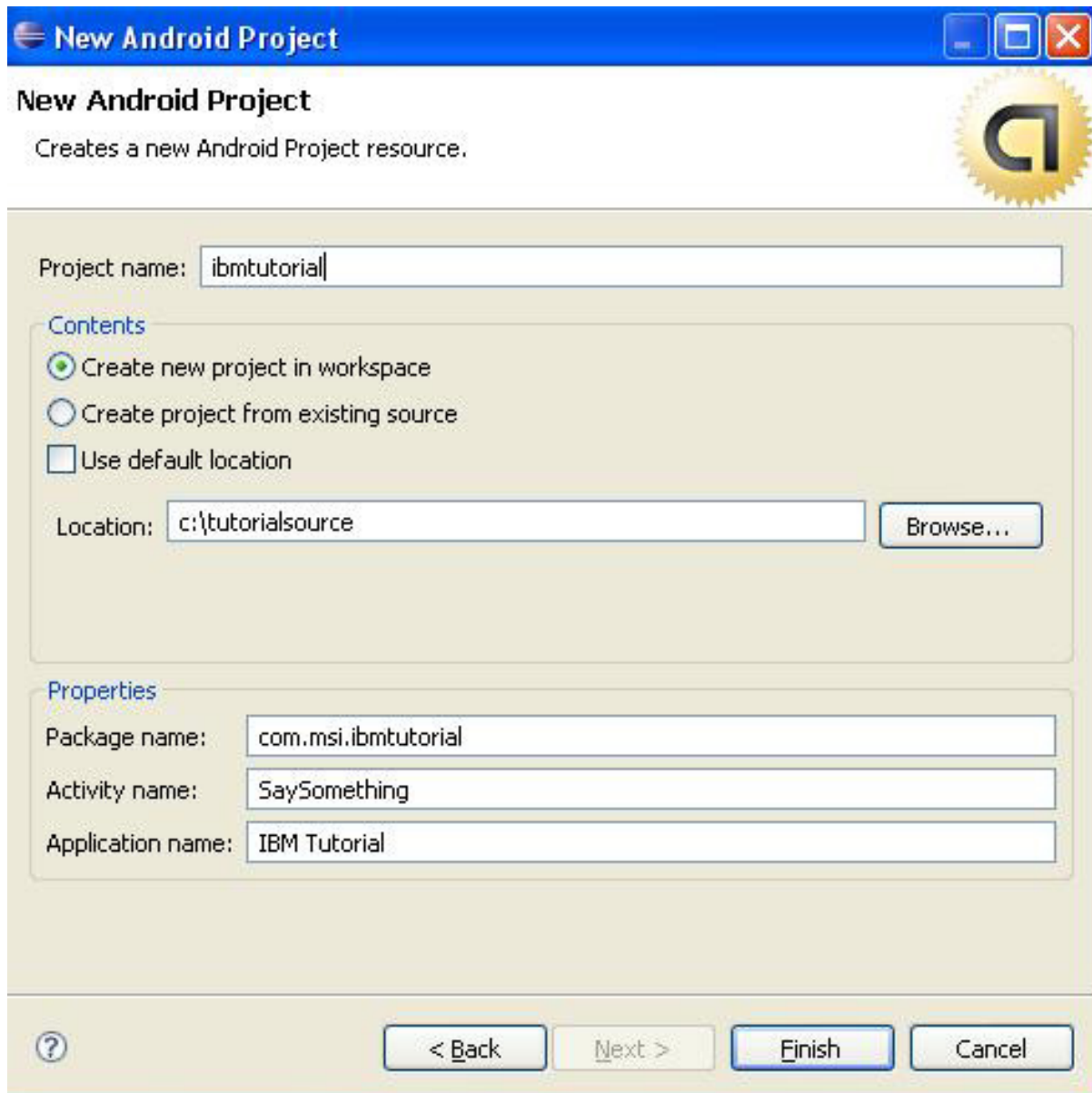


The requirements for the application are:

- Name
- Location
- Package name
- Activity name — Think of this as the main "form" or screen of the application
- Application name

Take a look at the new project.

**Figure 3. New Android project**

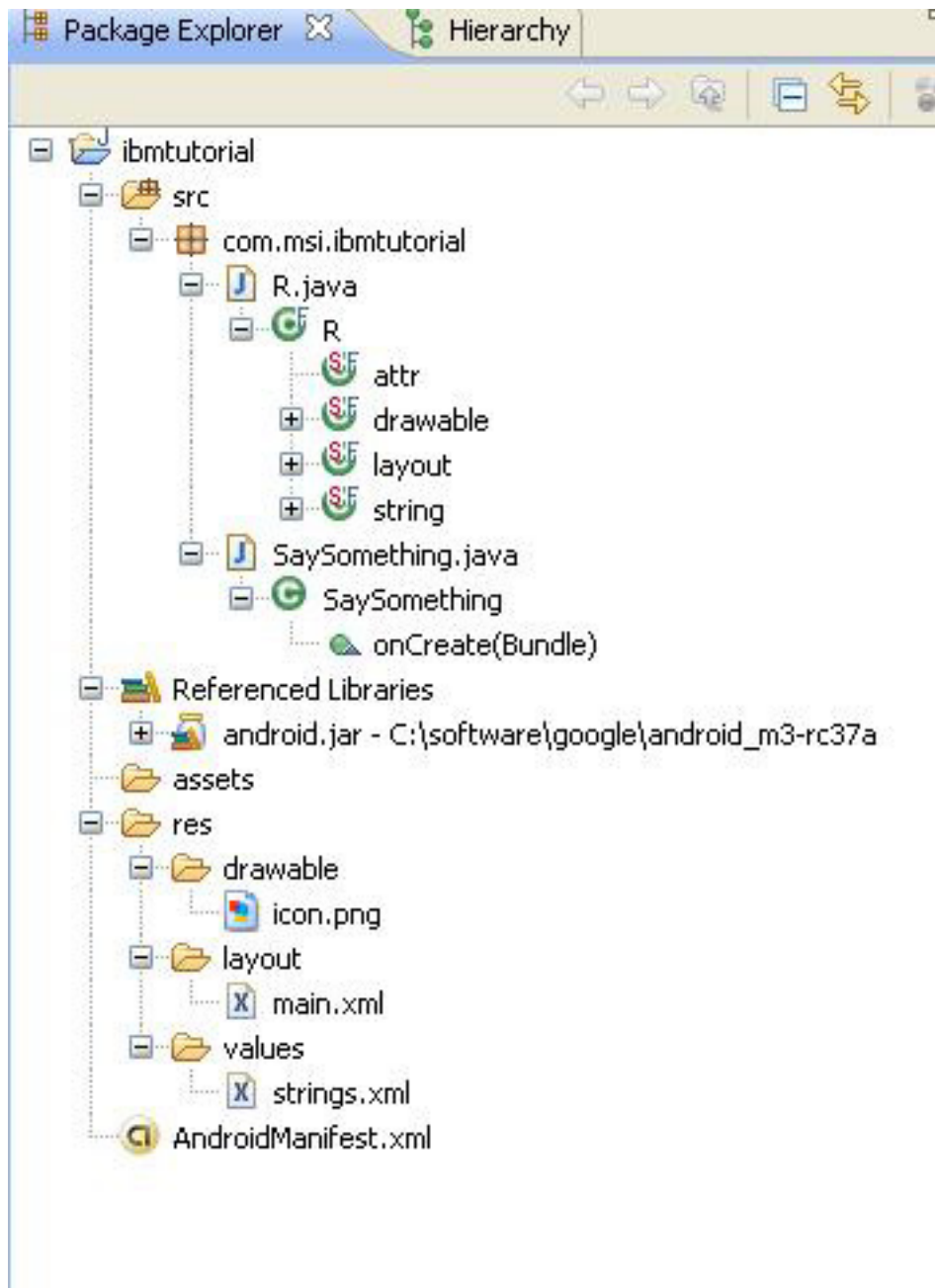


This will create a default application ready to be built and run. The components may be seen in the Package Explorer, which we discuss next.

## The Package Explorer

The Package Explorer (found in the Java perspective in Eclipse) displays all the components of the sample Android application (see Figure 4).

### Figure 4. Package Explorer



Items of note include:

### src folder

Includes the package for the sample application, namely `com.msi.ibmtutorial`

### R.java

The Android Developer Tools create this file automatically and represents the constants needed to access various resources of the Android application. More on the relationship between the R class and resources is found below.

### SaySomething.java

Implementation of the application's primary activity class.

### Referenced libraries

Contains `android.jar`, which is the Android runtime class jar file, found in the Android SDK.

### res folder

Contains the resources for the application, including:

- Icons
- Layout files
- Strings

### AndroidManifest.xml

Deployment descriptor of the sample application.

Next, we'll examine the source code in further detail.

## The primary activity of the application

The sample application consists of a single activity, namely `SaySomething`. As described above, the `SaySomething` class is implemented in the file `SaySomething.java`.

### Listing 4. SaySomething.java

```
package com.msi.ibmtutorial;

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

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

Things to note about this source snippet:

- `SaySomething` is a normal Java class, with a package and imports, as expected.
- `SaySomething` extends a base Android class named `Activity`, which is located in the `android.app` package.
- The `onCreate()` method is the entry point to this activity, receiving an argument of type `Bundle`. The `Bundle` is a class which is essentially a wrapper around a map or hashmap. Elements required for construction are passed in this parameter. This tutorial does not examine this

parameter.

- The `setContentView(...)` is responsible for creating the primary UI using the `R.layout.main` argument. This is an identifier representing the main layout found in the resources of the application.

The next section reviews the resources for the sample application.

## Resources for the application

Resources in Android are organized into a subdirectory of the project named `res`, as described previously. Resources fall into three primary categories:

### Drawables

This folder contains graphics files, such as icons and bitmaps

### Layouts

This folder contains XML files that represent the layouts and views of the application. These will be examined in detail below.

### Values

This folder contains a file named `strings.xml`. This is the primary means for string localization for the application.

The next section dissects the `main.xml` file to review the sample application's primary UI resources.

## main.xml

The sample application contains a single activity and a single view. The application contains a file named `main.xml` that represents the visual aspects of the primary UI of the activity. Note that there is no reference in the `main.xml` where the layout is used. This means it may be used in more than one activity, if desired. Listing 5 contains the content of the layout file.

### Listing 5. Layout file

```
<?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"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World, SaySomething"
    />
</LinearLayout>
```

This is the most simplistic of layouts. There is a single linear layout, which is oriented as a vertical layout, meaning all contained elements are in a single column. There is



a single `TextView` element, which can be likened to a label in other development environments. A `TextView` represents static text that is not editable.

Note that each view element (`layout` and `TextView` in this example) have attributes in the Android name space. Some attributes are common to all views — the `android:layout_width` and `android:layout_height` attributes, for example. The values available for these attributes are:

### Fill Parent

This extends the view element to take the maximum space available. This can also be thought of as meaning "stretch."

### Wrap Content

This value tells Android to paint the elements one after the next without stretching.

During the build process, all resources are compiled. One of the products of that process is the `R.java` file, which represents the resources to the remainder of the application. The `R.java` file is discussed next.

## R.java

The `R.java` file is created upon build automatically, so be sure to not modify it by hand as all changes will be lost. Listing 6 contains the `R.java` file for the sample application.

### Listing 6. R.java file

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */

package com.msi.ibmtutorial;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
    }
}
```

The `R` class contains anonymous subclasses, which each contain identifiers for the various resources previously described. Note that all of these classes are static.

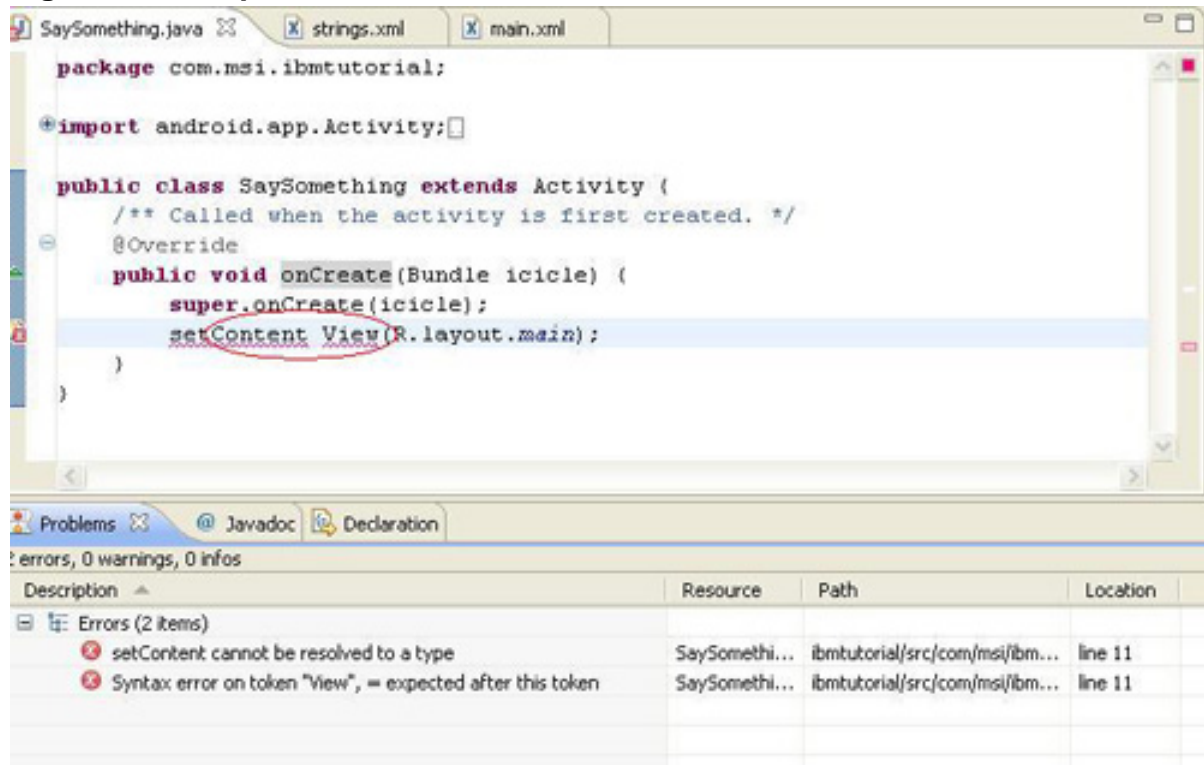
Note the element represented by: `R.layout.main`. This identifier represents the layout

defined by main.xml. Recall that this value is used in the `onCreate` method of the activity as follows: `setContentView(R.layout.main);`. This is the point at which a specific activity (in this case, `SayAnything`) and a specific layout (`main`) are bound together at runtime.

## Building applications

Files are compiled every time they are saved by default.

**Figure 5. Error pane**



We introduced an error into the source code where we added an extra space between `setContent` and `View`. When the file is saved, it is compiled and any errors appear in the **Problems** pane at the bottom of the screen. Upon fixing the error in the source code, the application builds properly and the errors are removed from the problems list.

## AndroidManifest.xml

The `AndroidManifest.xml` file represents the deployment descriptor for an Android application. The file lists any activity, service, content provider, or receiver contained in the application, along with the appropriate `IntentFilters` supported by the application. Here is the complete `AndroidManifest.xml` file for the sample application:

### Listing 5. AndroidManifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.msi.ibmtutorial">
    <application android:icon="@drawable/icon">
        <activity class=".SaySomething" android:label="@string/app_name">
            <intent-filter>
                <action android:value="android.intent.action.MAIN" />
                <category android:value="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

### Things to note:

- The package name from the source file is represented here. This follows a similar pattern to a Java source file and imports. The `<manifest>` tag is in essence "importing" classes from this package. All non-fully qualified classes in this file are found in the package identified in the package attribute.
- The `<application>` tag has an attribute that references a resource from the application's resources. Note the `@` symbol preceding the drawable identifier. This is a hint for the file to look in the drawable folder of the application's resources for a resource called "icon."
- The `<activity>` tag contains the following attributes and values of note:
  - `class` represents the Java class implementing this activity
  - `android:label` is the name of the application. Note that it is coming from one of the string resources. The `string.xml` file contains localized strings for the application.
  - `<intent-filter>` represents the `IntentFilter` available in the sample application. This is the most common `IntentFilter` seen in Android applications. This filter essentially says that it implements the "main" action (or entry point) and is located in the launcher of the OS. In English, this means it can be started as an application from the primary list of applications on an Android device.

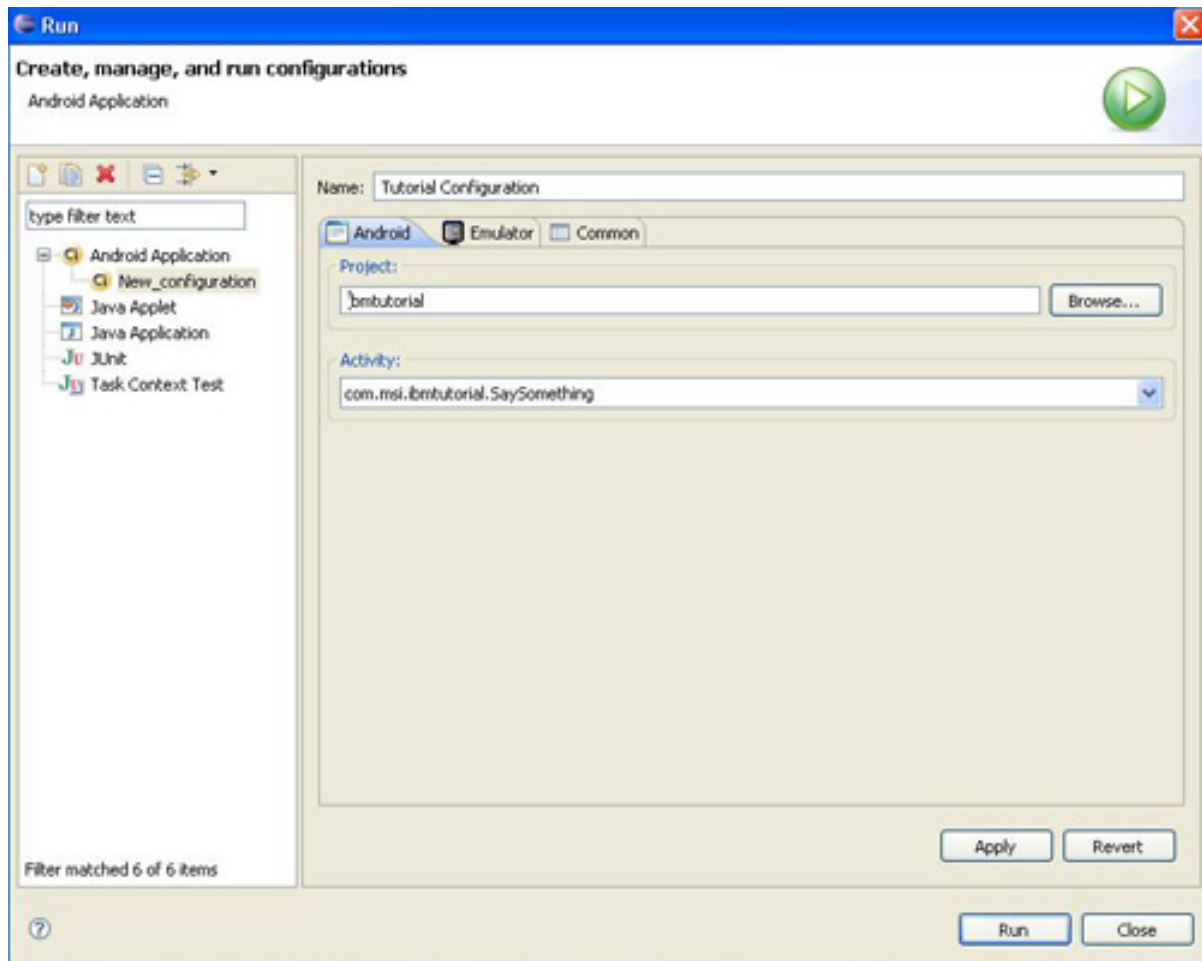
The next section describes starting the application on the Android Emulator from within Eclipse.

## Running the application

Now that the application has compiled successfully, it's time to run the sample application. Select **Open Run Dialog** or shortcut on the toolbar within Eclipse. This opens a dialog where startup configurations are created. Highlight the Android Application option and click the icon for **New**.

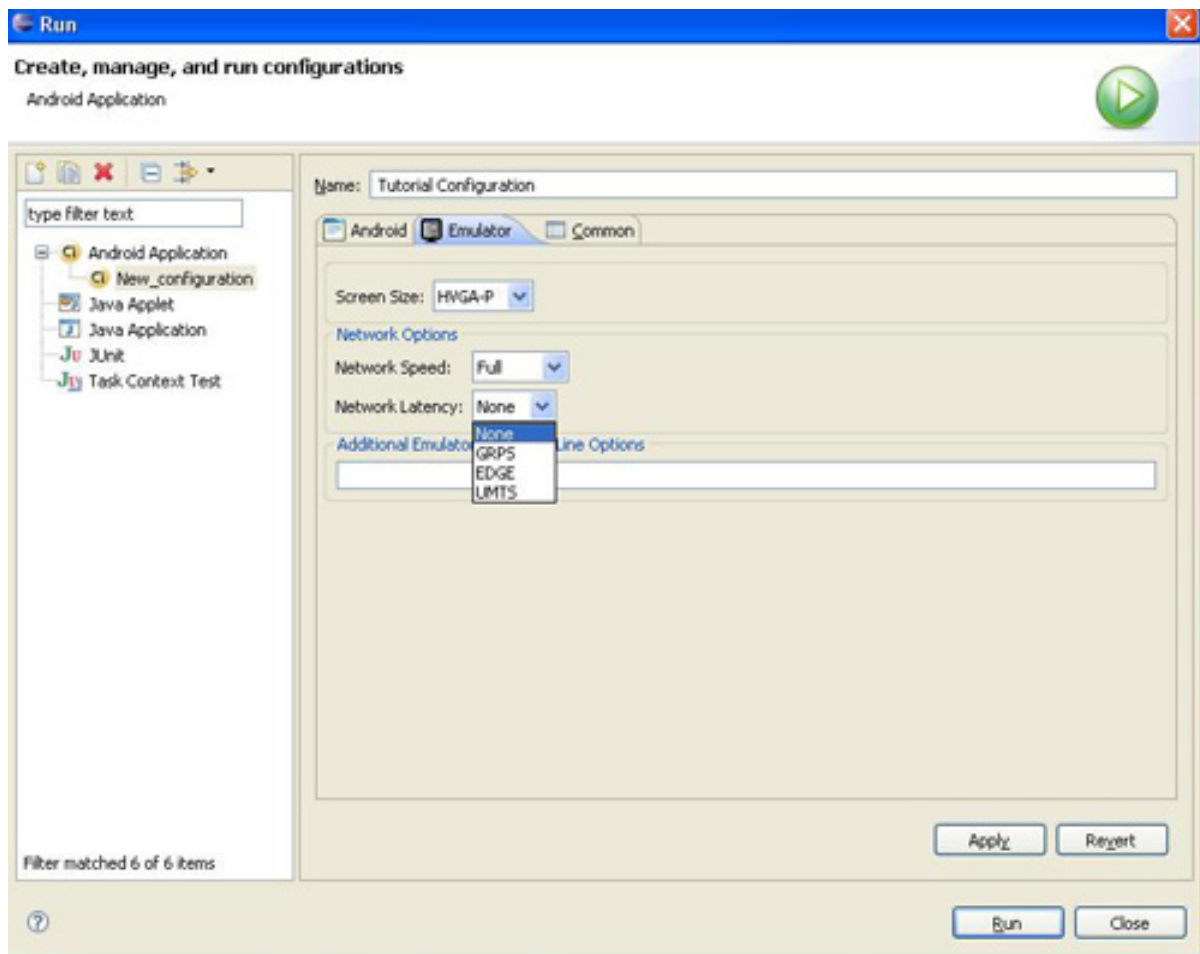
Figure 6 show the values used for the tutorial sample.

### Figure 6. Run dialog



Give the configuration a name. The tutorial sample uses the name *Tutorial Configuration*. Select the **ibmtutorial** project from the list of available projects (click **Browse** to see available projects). Select the startup activity in the drop-down. Now select the **Emulator** tab to specify Emulator settings, as desired. The default can be left alone. There are a couple of items to note, as described in Figure 7.

**Figure 7. Run dialog, Emulator tab**



There are a few screen sizes and orientations to choose from, as well as network choices. The network choices are important when building applications that employ Internet connectivity as mobile devices have varying network speed capabilities. Choose full network speed and no latency when prototyping an application. Once the main functionality is present, it's a good idea to test with less-than-ideal network conditions to see how the application responds in situations with suboptimal network connectivity.

Select **Run** to see the sample application in action.

### Figure 8. Emulator



Now that the application is running on the Emulator, it's time to see what's happening behind the scenes. The Dalvik Debug Monitor Service (DDMS) will assist with this.

## Debugging the application

To see what is happening with a running application, it is helpful to tap into the running Dalvik VM. To enable this from Eclipse, select **Window > Open Perspective > Other**. This displays a dialog box where the DDMS may be selected. This opens a new perspective in Eclipse with a number of interesting windows. Here is a quick introduction to the available resources in the DDMS perspective:

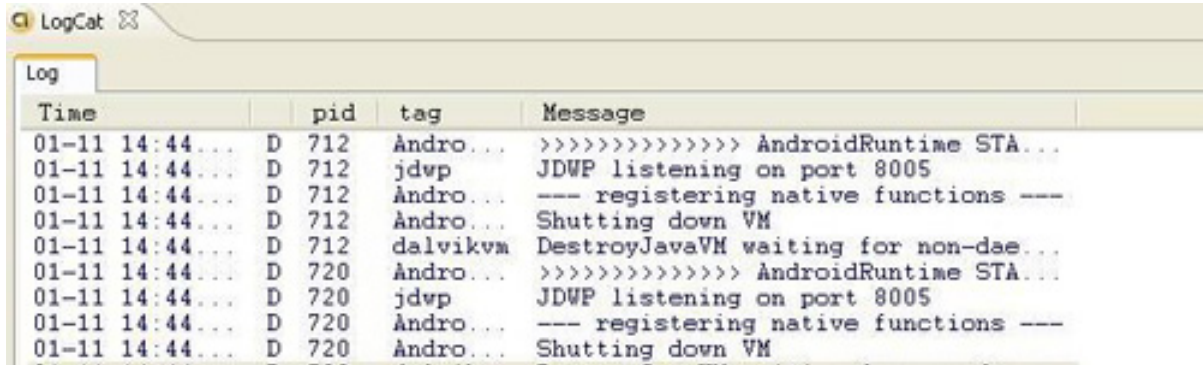
The LogCat is a running log file of activity taking place in the VM. Applications can make their own entries to this list with a simple line of code as follows:



Log.i(tag,message);, where tag and message are both Java strings. The Log class is part of the android.util.Log package.

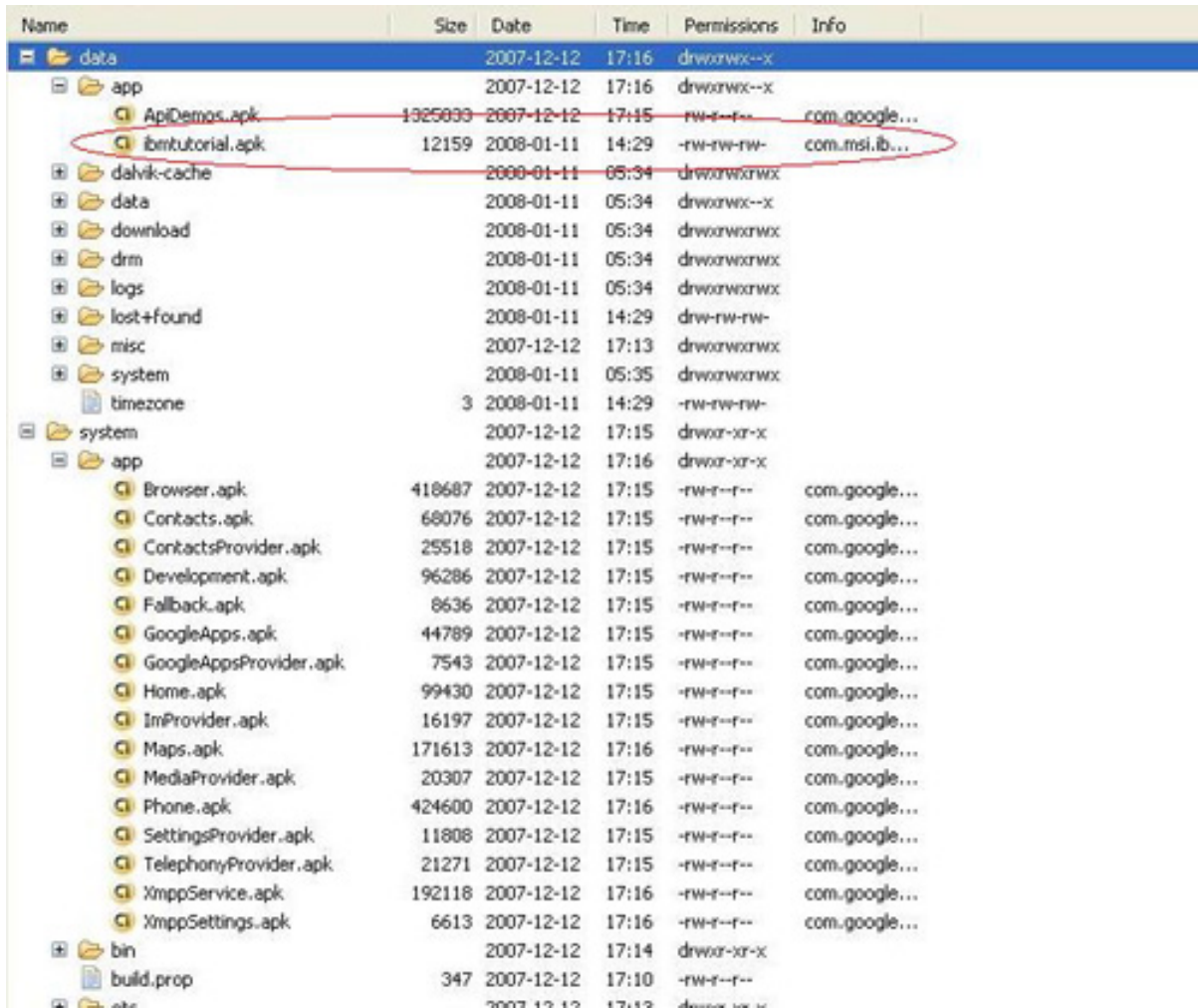
Figure 9 shows the LogCat in action.

Figure 9. LogCat in action



Another handy tool in the DDMS is the file explorer, which permits file system access of the Emulator. Figure 10 shows where the tutorial's sample application is deployed on the Emulator.

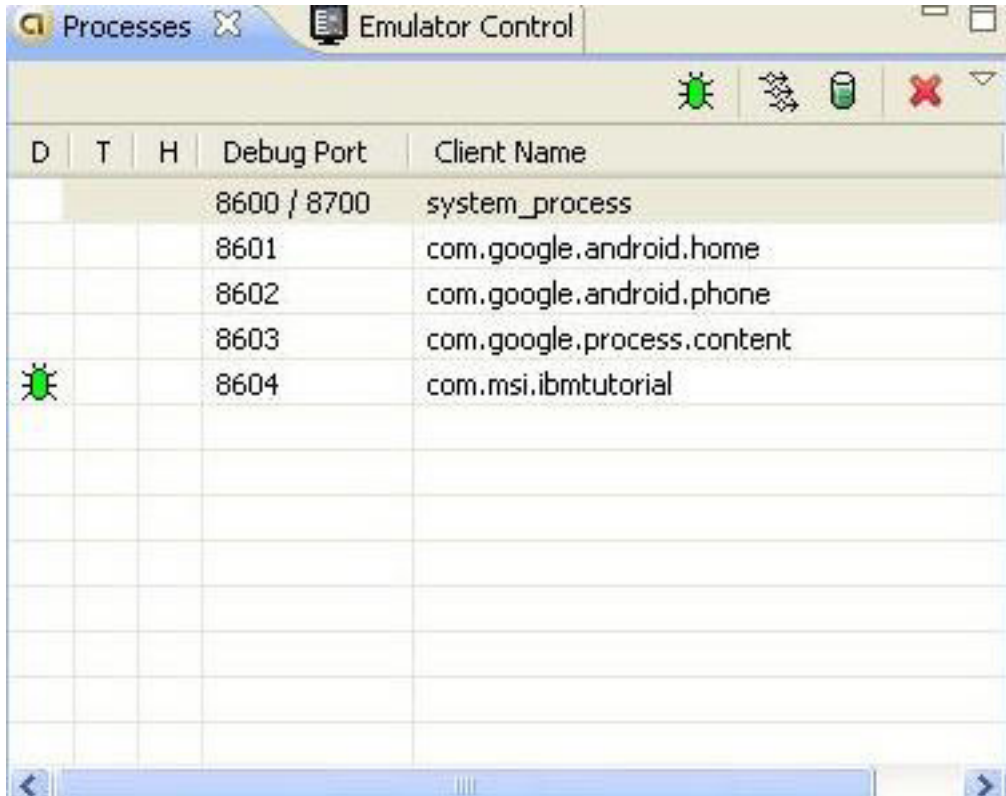
Figure 10. Sample application deployed on the Emulator



User applications are deployed in `/data/app` while Android built-in applications are found in the `/system/app` directory.

A running process list is also available in the DDMS.

**Figure 11. Running process list**



The screenshot shows the 'Processes' window in the DDMS tool. The window title is 'Processes' and it has a sub-tab 'Emulator Control'. The table below lists the running processes:

D	T	H	Debug Port	Client Name
			8600 / 8700	system_process
			8601	com.google.android.home
			8602	com.google.android.phone
			8603	com.google.process.content
			8604	com.msi.ibmtutorial

Full-scale debugging of an Android application is beyond the scope of this tutorial. For more information, see [Resources](#).

---

## Section 6. Building the content provider and Google Maps example

Now that you have seen a complete application example, let's take a quick look at a more complex application.

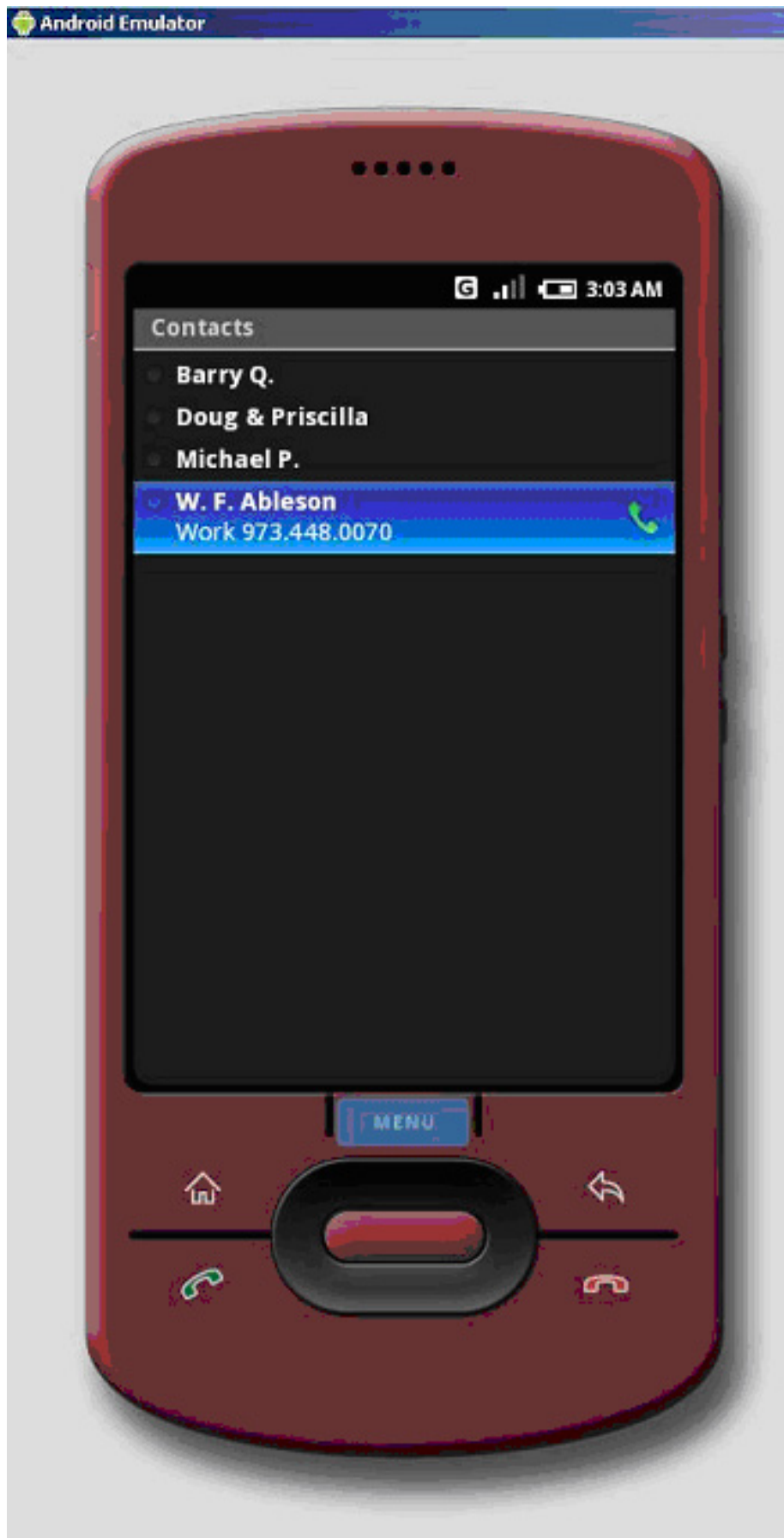
### Content provider and Google Maps

This second application examined in this tutorial is built with a theme of a mobile service professional (perhaps an appliance repair technician) who must map his way to the next service call. The application leverages Android's built-in contacts database as a record store. This tutorial will give you a feel for accessing data from



a content provider, as well as a glance at an intent in action as we use the address data found in the contacts database to perform a Google Maps search. For this tutorial to work properly on your Android Emulator, be sure to have one or more contacts recorded and be sure to populate the home address field. Figure 12 shows the Emulator with a few entries in the contact application.

**Figure 12. Emulator with entries in contact application**



Here is the first of two code snippets for the second application. Note that this application's main `Activity` class extends the `ListActivity`. This is because we're going to display information in a list.

### Listing 6. First snippet of second application

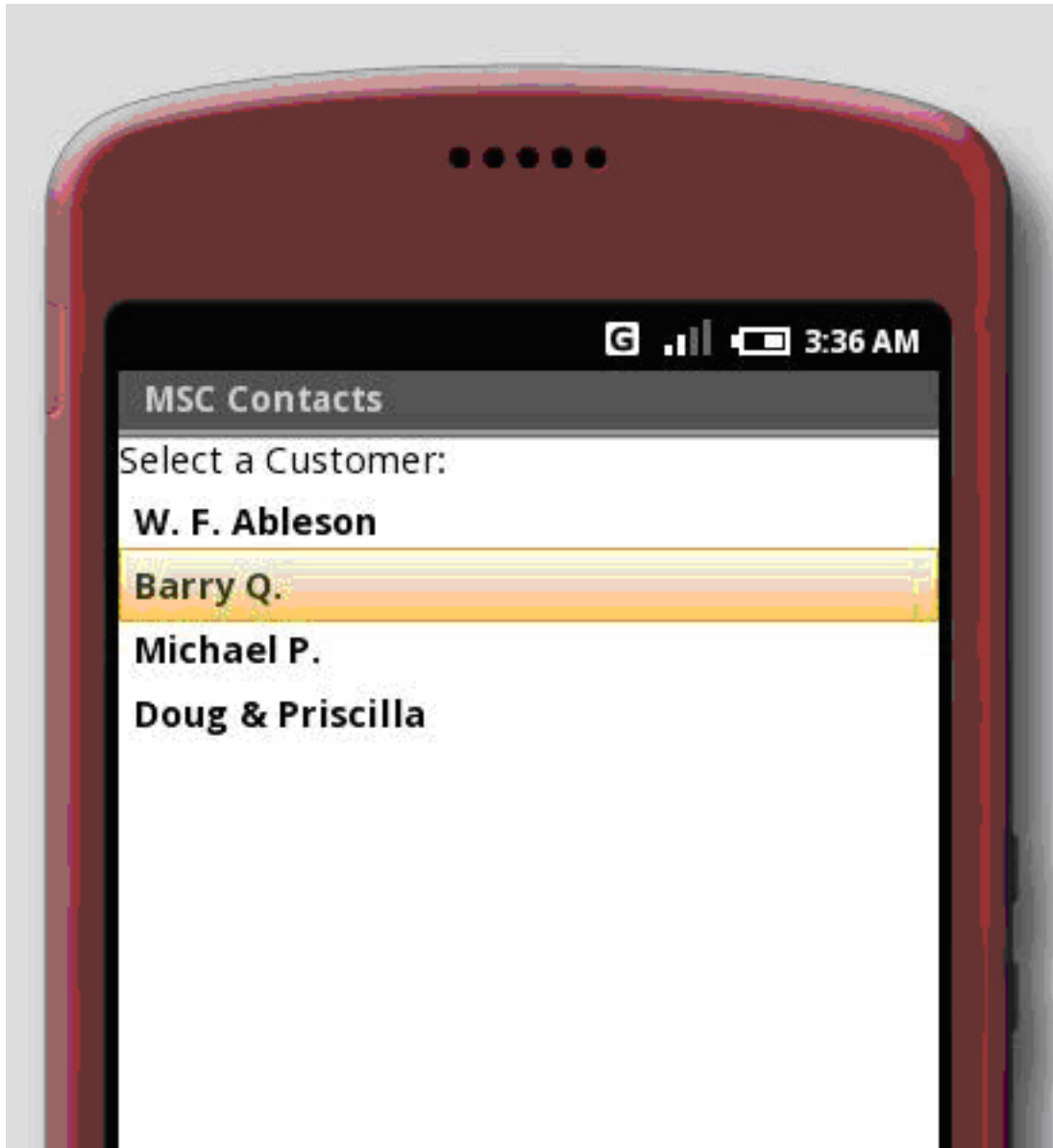
```
public class MobileServiceCallContacts extends ListActivity
{
    final String tag = "MSCC";
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icle)
    {
        super.onCreate(icle);
        setContentView(R.layout.main);

        // Get a cursor with all people
        Cursor c = getContentResolver().query(People.CONTENT_URI, null, null,
                                             null, null);
        startManagingCursor(c);

        ListAdapter adapter = new SimpleCursorAdapter(this, android.R.
            layout.simple_list_item_1, c, new String[] {People.NAME}, new int[]
            {android.R.id.text1});
        setListAdapter(adapter);
    }
    ...
}
```

Note the use of the cursor class to query the contacts database. This "result set" cursor is linked to the UI via a class known as a `ListAdapter`. Figure 13 shows the application in action as it presents the contacts available on the device. Note that there is no sort order in use in this display.

### Figure 13. Application in action



Any one of the contacts may be selected by a tap (click of the mouse), the center button on the emulator, or by pressing the **Enter** key on your keyboard. Once this entry is selected, the code must perform a lookup to obtain the address of the selected contact. This is where the `onListItemClick()` overridden method comes into play. This method's implementation has four important arguments. The one of most interest here is the `dbidentifier` method. Because the cursor was bound to the UI, when this method is invoked, it actually receives an identifier to the underlying data source. The `dbidentifier` field may be used to query the contacts database for desired information. It can also be used to simply launch the contacts application using an intent as shown in the commented-out code in Listing 7.

#### Listing 7. The `onListItemClick()` overridden method

```
@Override
protected void onListItemClick(ListView list,View view,int position,long
```

```

                                                                    dbidentifier)
    {
        super.onListItemClick(list,view,position,dbidentifier);

        try
        {
            // this commented out code below will launch the Contacts application
            // and "view" the contact Intent myIntent = \
new Intent(android.content.
            // Intent.VIEW_ACTION,new ContentURI("content://contacts/people/"
            // + dbidentifier)); startSubActivity(myIntent,position);

            // let's lookup specifics on this record
            ContentURI theContact = \
new ContentURI(android.provider.Contacts.ContactMethods.CONTENT_URI.toURI());

            // IMPORTANT
            // in order to use this sample application, you need to have at least
            // one Contact record on your Android emulator\
            // and be sure to have populated the 'Home Address field'
            //
            // this "where clause" is for HOME address and for the person record
            // selected in the GUI (id, dbidentifier)
            Cursor c = managedQuery(theContact,null," type = 1 and person = " +
                dbidentifier,null);

            if (!c.first())
            {
                showAlert("MSCC","No Contact Methods Available!","",true);
                return;
            }

            String address = c.getString(c.getColumnIndex("data"));

            address = address.replace("\n","");
            address = address.replace(",","");
            address = address.replace(" ","+");

            Intent geoIntent = new Intent("android.intent.action.VIEW",
                new ContentURI\
("geo:0,0?q=" + address));
            startActivity(geoIntent);
        }
        catch (Exception ee)
        {
            Log.i(tag,ee.getMessage());
        }
    }
}

```

Once the address has been obtained, a few simple string operations are required to clean up the data to prepare it for a query to Google Maps. The `geoIntent` is a new Intent created to perform a geo search, which in the default Android Emulator image is satisfied by a call to Google Maps.

All of the major elements of the first application still hold true for this application. There is a single activity launched from the main application screen. There is, of course, the `AndroidManifest.xml` file identifying our new application. Remember, complete source code is available in the [Download](#) section.

There is one final piece of information that is important to this second example application. In the `AndroidManifest.xml` file, there is an additional entry that gives the application permission to read the contacts database: `<uses-permission id="android.permission.READ_CONTACTS" />`. Without this explicit

permission, the Linux kernel will prevent the application from accessing the contacts database.

---

## Section 7. Summary

This tutorial introduced the Android platform, the Android Developer Tools, and the key elements of Android development in Eclipse. The Android Developer Tools allows you to leverage the rich Eclipse development environment for building and testing Android applications. You should now be ready to create your own Android applications.

# Downloads

Description	Name	Size	Download method
Example source code	os-eclipse-android-examples.zip	57KB	<a href="#">HTTP</a>

[Information about download methods](#)

# Resources

## Learn

- The author's Android book is available in part online at [Manning Publications](#).
- The authoritative Android information source is [Google's Android Web site](#), where you can find project documentation and links to download the Android SDK.
- Check out the "[Recommended Eclipse reading list](#)."
- Read the tutorial "[Build a mobile RSS reader](#)" to learn how to read, parse, and display RSS or other XML data in mobile apps, including your own mash-ups, using the Android Developer Tools.
- Browse all the [Eclipse content](#) on developerWorks.
- New to Eclipse? Read the developerWorks article "[Get started with Eclipse Platform](#)" to learn its origin and architecture, and how to extend Eclipse with plug-ins.
- Expand your Eclipse skills by checking out IBM developerWorks' [Eclipse project resources](#).
- To listen to interesting interviews and discussions for software developers, check out [developerWorks podcasts](#).
- Stay current with developerWorks' [Technical events and webcasts](#).
- Watch and learn about IBM and open source technologies and product functions with the no-cost [developerWorks On demand demos](#).
- Check out upcoming conferences, trade shows, webcasts, and other [Events](#) around the world that are of interest to IBM open source developers.
- Visit the developerWorks [Open source zone](#) for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.

## Get products and technologies

- Check out the latest [Eclipse technology downloads](#) at IBM [alphaWorks](#).
- [Install the Eclipse Android Development Tools \(ADT\) plug-in](#) using Eclipse software update. You can also download the [Android SDK](#) and learn how to [install, configure, and use the Android SDK](#).
- Download [Eclipse Platform and other projects](#) from the Eclipse Foundation.
- Download [IBM product evaluation versions](#), and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Innovate your next open source development project with [IBM trial software](#), available for download or on DVD.



## Discuss

- The [Eclipse Platform newsgroups](#) should be your first stop to discuss questions regarding Eclipse. (Selecting this will launch your default Usenet news reader application and open eclipse.platform.)
- The [Eclipse newsgroups](#) has many resources for people interested in using and extending Eclipse.
- Participate in [developerWorks blogs](#) and get involved in the developerWorks community.

## About the author

Frank Ableson

Frank Ableson is an entrepreneur and software developer in northern New Jersey, specializing in mobile and embedded application software. He is currently authoring a book about Android application development for Manning Publications. His professional interests are embedded systems, wireless communications, and automotive electronics. His biggest fans are his wife, Nikki, and their children.