Android Application Programming

Tutorial 1: Introduction to Android and Eclipse

Introduction

In this tutorial, you will setup your Android development environment and go through the Hello World tutorial on the Android Developer website at http://developer.android.com/resources/tutorials/hello-world.html. You will finally learn about using the Eclipse debugger and logging errors.

Setup Your Environment

The best way to develop applications for Android is using the Eclipse IDE and the Android plug-in for Eclipse called Android Development Tools (ADT). You can test your applications by running them on an Android Virtual Device (AVD), aka Android emulator.

To setup your environment, follow each of the steps below in order. Detailed installation instructions are available here: http://developer.android.com/sdk/installing.html

1. Download and install JDK 6
   http://java.sun.com/javase/downloads/index.jsp

2. Download and install Eclipse IDE for Java EE Developers (version 3.5 - Galileo)
   http://www.eclipse.org/downloads/

3. Download the Android SDK and unzip it onto your hard drive in a location of your choosing.
   http://developer.android.com/sdk/index.html

4. Download and install the Android Development Tools (ADT) for Eclipse
   http://developer.android.com/sdk/eclipse-adt.html

Now your environment is ready for development.

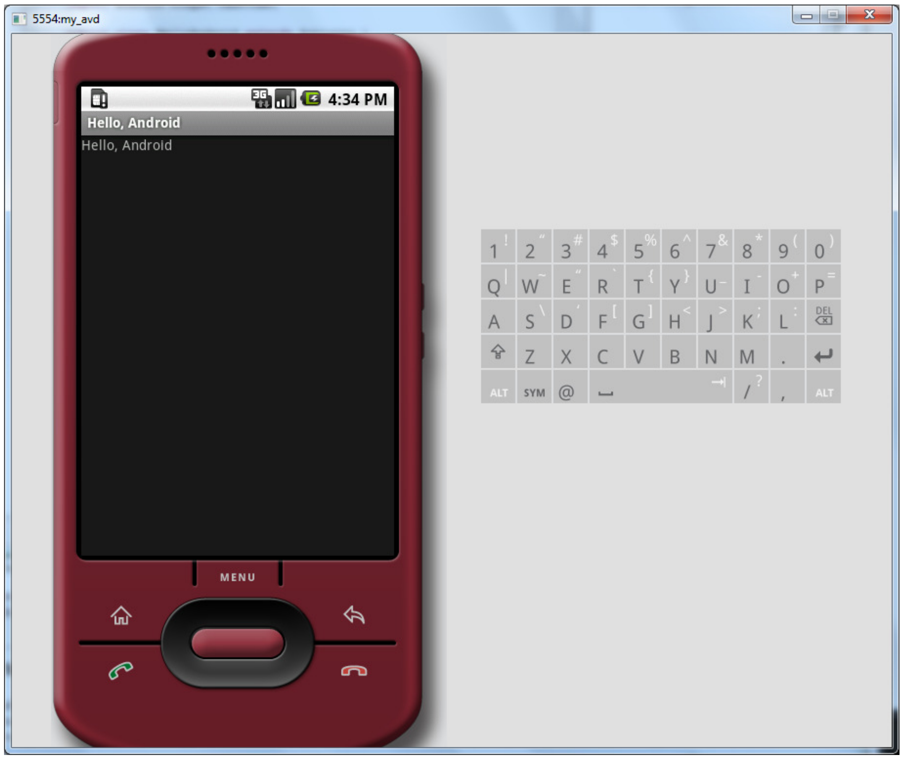
Hello World Tutorial

Run through the “Hello World” tutorial at http://developer.android.com/resources/tutorials/hello-world.html which takes you through creating an Android Virtual Device (AVD), creating an Android project, writing some code to display “Hello, Android”, and running the application in the emulator.

When you run the Android Application, Eclipse will first start an Android emulator if one isn’t already running. It may take a few minutes for this window to appear because of a lengthy initialization period… you’ll have to be patient. You can watch the status of the emulator from Eclipse’s Console window. It will look something like this (not all lines are displayed):

```
[2009-12-29 16:26:29 - HelloAndroid]------------------------------
[2009-12-29 16:26:29 - HelloAndroid]Android Launch!
[2009-12-29 16:26:29 - HelloAndroid]adb is running normally.
[2009-12-29 16:26:29 - HelloAndroid]Performing com.example.helloandroid.HelloAndroid activity
launch
```

```
[2009-12-29 16:26:29 - HelloAndroid]Automatic Target Mode: launching new emulator with
compatible AVD 'my_avd'
[2009-12-29 16:26:29 - HelloAndroid]Launching a new emulator with Virtual Device 'my_avd'
[2009-12-29 16:26:43 - HelloAndroid]New emulator found: emulator-5554
[2009-12-29 16:26:43 - HelloAndroid]Waiting for HOME ('android.process.acore') to be
launched...
```



Once the emulator has been initialized, the compiled package is installed on the device. Finally the application is executed as shown in the image to the right.

You can use the emulator as you would a real Android device by using your mouse to click on the screen and using your keyboard to type.

When you make changes to your program, **do not close this window**. Just edit your program, save it (which makes Eclipse re-build it), and re-run your new program (Ctrl-F11); the emulator will replace the previous program with the new one and execute it (you'll have to be patient… it can take up to a minute to reload). If you do close the emulator, you can start it again by running your app from Eclipse (Eclipse will start the emulator) or by running the emulator tool from the SDK's tools directory: `emulator -avd my_avd`

The Android Package

The .apk file, the Android Package, is the application file that was installed and executed on the emulator. The .apk file is produced by Eclipse in a number of steps as illustrated in the figure below.

1. The Java source code is compiled into Java bytecode (.class files) as is traditionally done in Java programming.
2. The dx tool converts the Java bytecode into Android bytecode (.dex files), a compact format that can be executed by a Dalvik virtual machine.
3. Finally, the Android Asset Packaging Tool (aapt) is used to combine .dex files with the AndroidManifest.xml and other resources making up your application. It produces the .apk which is really just a zip file.

You can find .apk files on the Web and install them directly to your emulator or Android device although you must be careful about installing un-trusted .apk files that could contain malicious code.

XML Layout

Continue following the tutorial where it describes how to change from a "programmatic" layout to an XML-based layout. In this section, you'll learn about main.xml, strings.xml, and the auto-generated R class. Specifying the layout of your Android apps with XML is considered a "best practice" and is ideally how you will design all your applications in the future.

Using the Debugger

Continue following the tutorial where it describes how to debug your project in Eclipse. In this part of the tutorial, you'll introduce a `NullPointerException` that will cause your program to terminate abruptly. You'll also add a breakpoint so that you can step through your code.

When Eclipse pauses at a break point, you'll notice that Eclipse has shifted around some sub-windows and introduced new ones. This orientation is called the "Debug" perspective, and this is the ideal perspective to use when debugging. The perspective you were seeing before debugging is the "Java" perspective.

When stepping through your code, use **F6** to step to the next line, **F5** to step into a function call, and **F8** to resume execution. This might take some getting used to if you are transitioning from Visual Studio.

When debugging, sometimes you'll notice that Eclipse tries to display source code that it cannot find and will display a "Source not found" message. Press F8 to continue when this happens. Sometimes you'll need to press F8 several times because Eclipse will break at several locations. In this example, eventually you'll notice that

Eclipse will not continue running your app when pressing F8, and that's because the modal dialog box complaining about the error in the emulator must be dismissed before Eclipse can continue to execute any code.

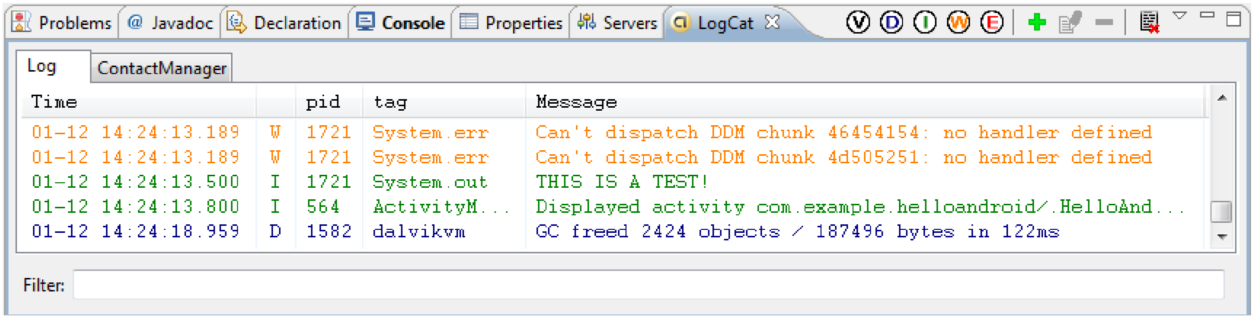
Poor Man's Debugger

You can also use output statements to debug your application. The tutorial does not mention this technique, but you may find it helpful at times.

Add the following line of code to your onCreate method:

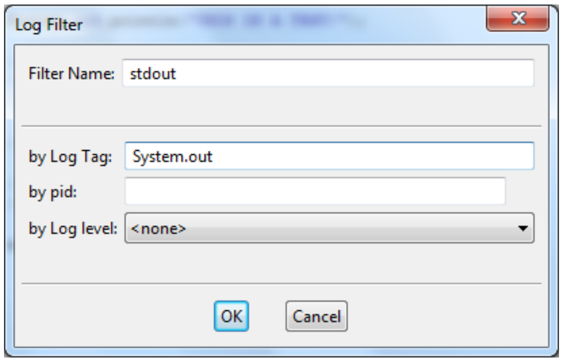
```
System.out.println("THIS IS A TEST!");
```

Now re-run your application. You won't see the output in the Console tab in Eclipse; it will instead appear in the LogCat. To view LogCat, select **Window > Show view > Other..., Android > LogCat**. Now you'll see the LogCat filled with many different messages, including the one you just output to stdout:



```
Time                   pid   tag           Message
01-12 14:24:13.189  W  1721  System.err    Can't dispatch DDM chunk 46454154: no handler defined
01-12 14:24:13.189  W  1721  System.err    Can't dispatch DDM chunk 4d505251: no handler defined
01-12 14:24:13.500  I  1721  System.out    THIS IS A TEST!
01-12 14:24:13.800  I  564   ActivityM...  Displayed activity com.example.helloandroid/.HelloAnd...
01-12 14:24:18.959  D  1582  dalvikvm      GC freed 2424 objects / 187496 bytes in 122ms
```

Because there are so many messages, you may want to filter out only those to stdout, so you'll need to create a filter that shows only messages tagged with "System.out". Press the green + which will display the Log Filter dialog box displayed on the right. Enter "stdout" for the filter name and "System.out" for the Log Tag.



Filter Name: stdout
by Log Tag: System.out
by pid:
by Log level: <none>

Now you will have a tab in LogCat called "stdout" which will only contain messages sent to stdout.

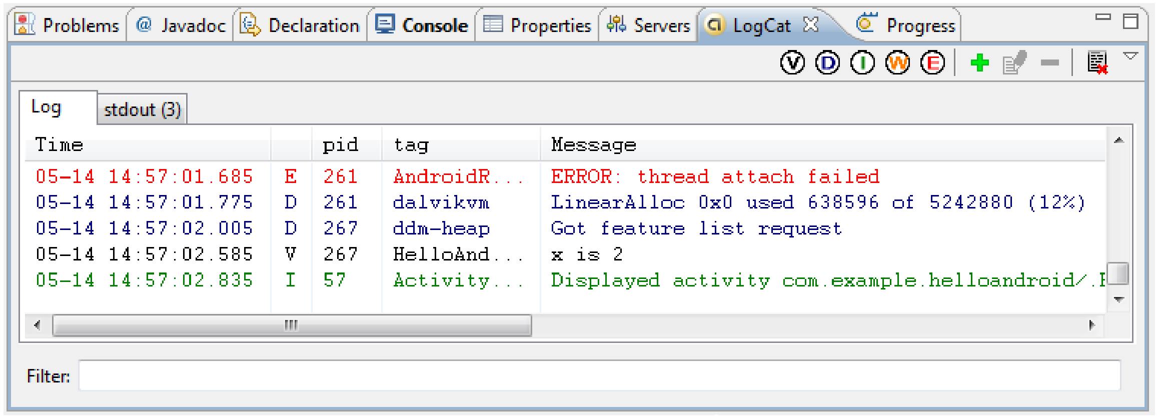
Logging Messages

Rather than send output to stdout, you may also use the Android logging class (android.util.Log) which allows for more fine-grained logging and better filtering.

Import `android.util.Log`, declare a log tag, and add a call to `Log.v()` in your program like so:

```java
package com.example.helloandroid;

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

public class HelloAndroid extends Activity {

    private static final String LOG_TAG = "HelloAndroid";

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

        int x = 2;
        Log.v(LOG_TAG, "x is " + x);
    }
}
```

When you run your program, the `Log.v()` call will output “x is 2” to LogCat as shown below.



```
Time                 pid  tag          Message
05-14 14:57:01.685 E 261  AndroidR...  ERROR: thread attach failed
05-14 14:57:01.775 D 261  dalvikvm     LinearAlloc 0x0 used 638596 of 5242880 (12%)
05-14 14:57:02.005 D 267  ddm-heap     Got feature list request
05-14 14:57:02.585 V 267  HelloAnd...  x is 2
05-14 14:57:02.835 I 57   Activity...  Displayed activity com.example.helloandroid/.I
```

Note that there are different levels of debugging. The `Log.v()` method outputs **verbose** or low-priority log messages which appear in black. **Debug** messages (using `Log.d()`) appear in dark blue, **information** messages (`Log.i()`) appear green, warning messages (`Log.w()`) appear orange, and **error** messages (`Log.e()`) appear red. Typically you’ll want to use the appropriate method to log the severity of the error message.

In order to see only messages logged with your `LOG_TAG`, create a filter just like you did for stdout previously, but this time use the Log Tag of “HelloAndroid”.

Conclusion

You should now be familiar with setting up an emulator and running an Android app on the emulator. You also know how to use the Eclipse debugger to step through code and can sending debug information to the LogCat. In the next tutorial, you will create a more sophisticated Android application that plays tic-tac-toe.