Core Android Test Automation Tools and Techniques

Part of the Android Test Automation Series

Julian Harty

18th January 2011

# Introduction

As the Android platform has matured, so have the automated testing tools and frameworks provided by the SDK. The documentation is also improving on [http://developer.android.com](http://developer.android.com/) and there are various references and tutorials available online. Here’s my attempt to describe the tools and my experience in using them.

I do not claim to be competent in any of these tools rather I hope my experiences and notes will help other learners to conquer the basics and understand the context of each tool so they can then reach a basic level of competence before surpassing my work and writing the expert documentation we need.

These notes are still incomplete drafts at this stage. More material will be added in future revisions.

# Topics

This document will cover the following topics. All are provided in the current version of the Android SDK, version 8 which includes version 9 of the Android platform <http://developer.android.com/sdk/index.html>

* Android JUnit Tests & Test Runner
* Instrumentation
  + Activity
  + Service
* Monkey
* Monkeyrunner

For an overview of the tools and concepts, see <http://developer.android.com/guide/topics/testing/index.html>

# Android JUnit and Test Runner

Android uses JUnit 3 for both the JUnit and Instrumentation tests.

Here are several articles online which provide examples of using Android JUnit testing.

* <http://developer.android.com/resources/tutorials/testing/helloandroid_test.html> is part of the official documentation, and it will help you understand the fundamentals of how to structure, create and run tests using eclipse.
* <http://www.bogotobogo.com/Android/android11JUnitTest.html>
* <http://marakana.com/forums/android/examples/121.html>

## Instrumentation Test Runner

The Instrumentation Test Runner underpins all the unit testing classes on Android, and independent projects such as Robotium. Once you start to progress beyond running simple automated tests, you may decide to invest time learning more about the capabilities of the Instrumentation Test Runner e.g. how to run only the ‘small’ tests (typically these run quickly and therefore are often run as part of a continuous integration process); how to generate code coverage, etc. Some features are only documented when the instrumentation is run from the command-line (rather than in an IDE such as eclipse). Someone who knows more about eclipse than I do may be able to configure eclipse to take advantage of these features. Please let me know if you do so and I would be happy to include the information in this document.

More documentation can be found in the official Android project documentation at:

<http://developer.android.com/reference/android/test/InstrumentationTestRunner.html>

* Test sizes: purpose and how to run them.
* Code Coverage (EMMA)

### Test sizes

Test sizes were a deceptively simple concept envisioned at Google to try to clarify the purpose and resources a given test would require. The existing nomenclature in the software industry seemed over complex, so the first three sizes were small, medium, and large. You can read more about the concepts on the Google Testing Blog, at <http://googletesting.blogspot.com/2010/12/test-sizes.html>

The Instrumentation Test Runner for Android provides support for Java annotations e.g. @SmallTest <http://developer.android.com/reference/android/test/suitebuilder/annotation/SmallTest.html> and a mechanism to run only those tests <http://developer.android.com/reference/android/test/InstrumentationTestRunner.html> Currently I only know how to run these tests from the command line, not from an IDE such as eclipse.

### Code coverage

EMMA is one of several well known opensource code-coverage tools, and it has been adopted for the Android project to provide code coverage data. The results can be processed with a wide range of tools i.e. any that can ‘consume’ EMMA output files.

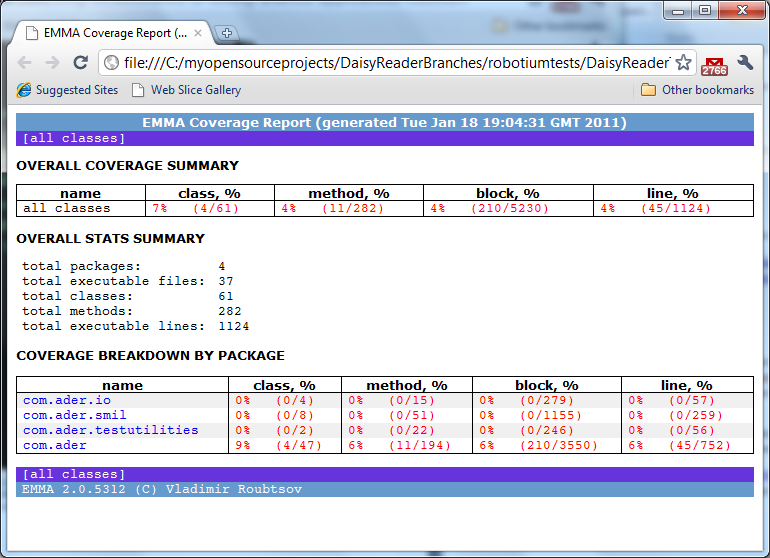
As far as I can tell we need to use the ant scripts to generate coverage data and reports. We cover ant later in this document.

Here are some of the issues I discovered (I was using r8 of the SDK tools, the SkeletonApp from the API Demos samples that come with the SDK, and Ubuntu 10.10):

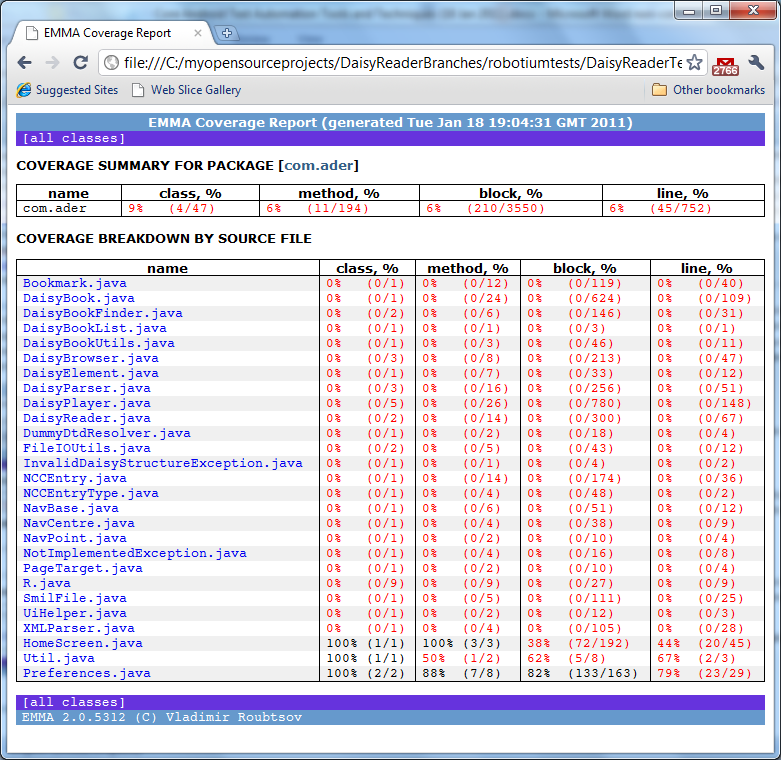
* I spent several hours fruitlessly trying to convince ant there’s a target called coverage. Somehow when I was in the tests folder, and after running another android update test-project -m ../ -p . I was then able to run ant coverage. Note: I had created a file called build.properties in both the main project folder and the tests folders with the value of emma.enabled=true
* Coverage data doesn’t seem to be available on real devices as permission is denied when trying to access /data/data/ (where the coverage file is generated in a project-specific sub-folder). The coverage report does seem to be available when the tests are run on an emulator. The contents of the coverage.ec file are processed and a coverage.html file is created in a folder of tests/coverage The coverage file is coverage.html which you can open in your web browser. You can expand each source java file for the application under test to see which lines have been covered (they have a green background) and those not covered (these have a red background).
* When you try to build the tests using ant (which happens when you run ant coverage) the compilation may fail to find classes and libraries belonging to the target application. See <http://stackoverflow.com/questions/2472059/cant-build-and-run-an-android-test-project-created-using-ant-create-test-projec> for the fundamental issues. I’ve also described how to address, including some revisions to the <compile> ant task described in the article on stackoverflow, for Robotium tests at <http://code.google.com/p/robotium/wiki/QuestionsAndAnswers> search for the question related to code coverage for the details.

#### Examples of a code coverage report

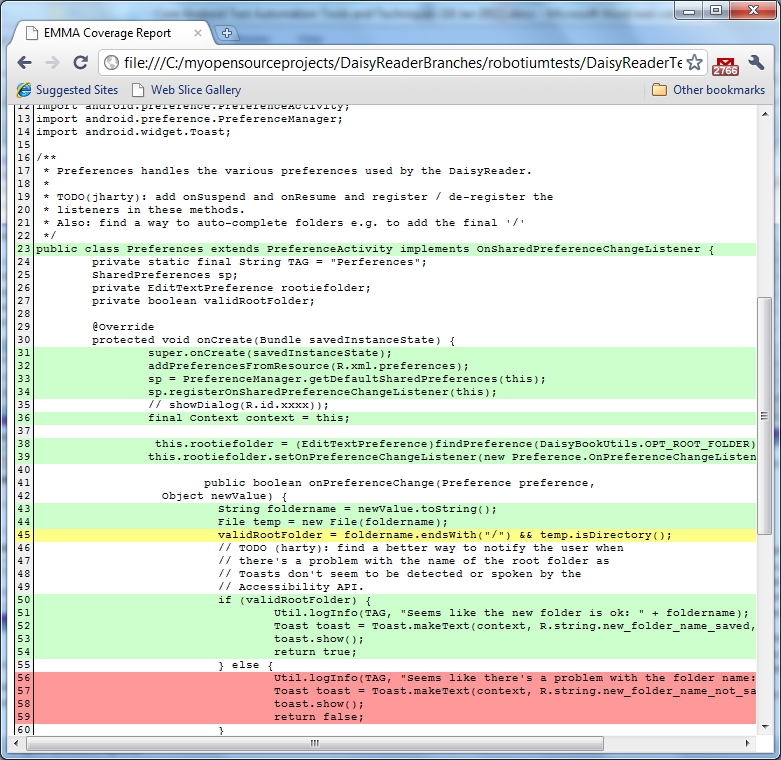
Here’s a screenshot of running the Robotium tests (which use Instrumentation) for one of my current opensource projects DaisyReader. We can see the coverage for these large tests (which test through the UI) is low at 4% line coverage. There are separate small tests for many of the classes so the overall coverage would be higher when we run all the tests.



By expanding the coverage for the com.ader package we can see that HomeScreen.java and Preferences.java are the only classes exercised by the current Robotium tests.



Let’s expand the results for Preferences.java



For Preferences.java we can see some of the lines have a white background - these are not executable lines of code. Those with a green background are covered by the tests i.e. they have been executed at least once. Those with a red background have not been executed, either because no tests exist, or in this case because one of the tests failed. Finally the line with a yellow background was partly executed according to EMMA, hovering over the line in the web browser, using the mouse pointer, displays more information.

### Using Ant

Ant is a well established opensource tool used to build software and perform various tasks. Although many software developers now prefer to use an IDE such as eclipse, learning about how to use Ant to build, deploy and test software can be helpful, especially as some features of the Android tools don’t appear to be available in the IDE. In practice, for Android development and testing you will also need to learn about the ‘android’ and ‘adb’ commands.

Let’s investigate what it takes to run the single, simple Instrumentation test case provided for the SkeletonApp in the Android API demos. I recommend you make a copy of the relevant folder into a fresh folder so you can experiment without corrupting the original source code.

In the fresh folder, open a command-line (in Windows) or a terminal window (in Linux or OSX) .

run:

android update project --path . --target 4 --subprojects

This uses the android command to update the project in the current folder (specified by --path), including sub-projects (the tests are in a sub-project) specifying a target version of the Android platform of 4 (version 1.6 also known as Donut). You should see several lines of output stating the project has been updated and various build files have been created.

ant debug

will build the SkeletonApp application.

I then ended up running the following command to make sure the link between the tests and the application was correctly established:

android update test-project -m . -p tests

This uses the android command to update the relationship between the tests, in the tests sub-folder and the application in the current folder. We then need to change directory to the tests sub-folder

cd tests

ant run-tests

And run the tests using the ‘run-tasks’ Ant task (which is created by the Ant scripts provided as part of the Android SDK and tools). If all is well then you will eventually see confirmation the tests have run successfully*. However...*

However, I discovered I first needed to install the application to be tested on the device. The following commands will achieve this:

cd ..

ant install

Followed by

cd tests

ant run-tests

The essential output from the hundreds of lines of output from the ant run-tests command is something like

[exec] OK (1 test)

Of course the number of tests will depend on how many exist for your project ☺

# Instrumentation

## Testing Activities

TBC. Here are some initial references to get you started in the interim.

* <http://developer.android.com/resources/tutorials/testing/activity_test.html>
* <http://www.bogotobogo.com/Android/android12ActivityTestingB.html#Creating Test Case Class>

## Testing Services

Android provides the ServiceTestCase to enable services to be tested. The ServiceTestCase allows a service to be started and stopped, and supports bindService. bindService returns an IBinder interface which we can use to call public methods declared in the respective public interface used by the service.

There seems to be very little information on ServiceTestCase and few examples online. I ended up taking several example services and creating tests for them e.g. for:

* <http://www.androidcompetencycenter.com/2009/01/basics-of-android-part-iii-android-services/> where I modified the service slightly so it incremented a counter each time getStatusCode() was called.
* <http://developer.android.com/resources/samples/ApiDemos/src/com/example/android/apis/app/LocalService.html> However once I realised I didn’t know how to intercept Toasts or Notifications I deferred my work on creating tests for this service.
* <http://marakana.com/forums/android/examples/60.html> Here the service was so simple that the test didn’t seem worthwhile.

Trap: If the <service> tag is not within the <application> tag then the service will not be incorporated into the application. Clues include:

* Messages in the eclipse error log saying “AndroidManifest: ignoring unknown ‘service’ XML element”
* LogCat reporting for the tag ActivityManager “Unable to start service intent { ... } not found

Simply move the <service> tag in AndroidManifest.xml to within the <application> tag, rebuild and install the updated application, and the service should now be included in the application.

### Next Steps to explore when testing services

* If the service creates Toasts or Notifications there doesn’t seem to be any way to detect or intercept these. Potentially a technique such as the strategy for Mocking UI components such as the TextView, described in <http://www.connorgarvey.com/blog/?p=141> might work; something we can try as the need arises.
* Create Tests for services that run in a remote process.
* Experimenting with multi-threading, long-running tasks, etc.
* Testing services that interact with external sites, e.g. RSS feeds. We should be able to point the service to use a RSS feed we control programmatically on the local computer to test polling, handling multiple queued messages, etc.

# Android Tools

Here are some Android tools relevant for testing and profiling your applications.

## Monkey

TBC

## Monkeyrunner

TBC Meanwhile here are some links to relevant articles online

* <http://mobilebytes.wordpress.com/2010/12/21/android-agile-patterns/>

## Traceview

Another TBC. See the following in the interim:

* <http://stackoverflow.com/questions/1957135/how-to-test-the-performance-of-an-android-application>

# Useful adb commands

adb shell pm list packages | *grep example*

This adb command lists the packages (applications) installed on the running emulator or connected device. The output is piped to the grep command (standard on Macs and Linux, you need to install a grep program on Windows computers e.g. using cygwin). The grep command searches for lines that contain the word example (many of our code examples use a java package name that includes the word example).

# Summary & Conclusion

Android offers a range of Specific test classes to help us test our core code, our Activities, our Services, etc. Expect to spend some time debugging problems getting the tools to run and generate the output you expect. Sites such as [www.stackoverflow.com](http://www.stackoverflow.com) are handy when researching issues and asking for help. Thankfully, the test automation classes are useful and can help us automate many aspects of our testing. There are tools that augment and simplify the Android test classes e.g. Robotium and replace them e.g. the TEMA tools. These will both be described separately.