Enhanced Junit 4 Test Runner

Synopsis

The Enhanced Android Test Runner is part of the RySource suite of solutions designed to simplify and enhance the testing and quality assurance of software development. It gives the test developer the tools to better align with AGILE practices amongst adding more metadata and verbosity to their existing framework without sacrificing the standardised approach in which the ITR is deployed.

How it works?

The EATR extends the existing InstrumentationTestRunner class to provide the same setup and configuration process of a traditional setup. When extended, the end developer can configure and retrieve callbacks on events by using the provided annotations or interfaces.

Getting Started

This guide is designed to aim at beginners with little to no experience with Junit 4; it will however provide an easy implementation guide for the EATR.

Step 1 – Download an IDE of your choice

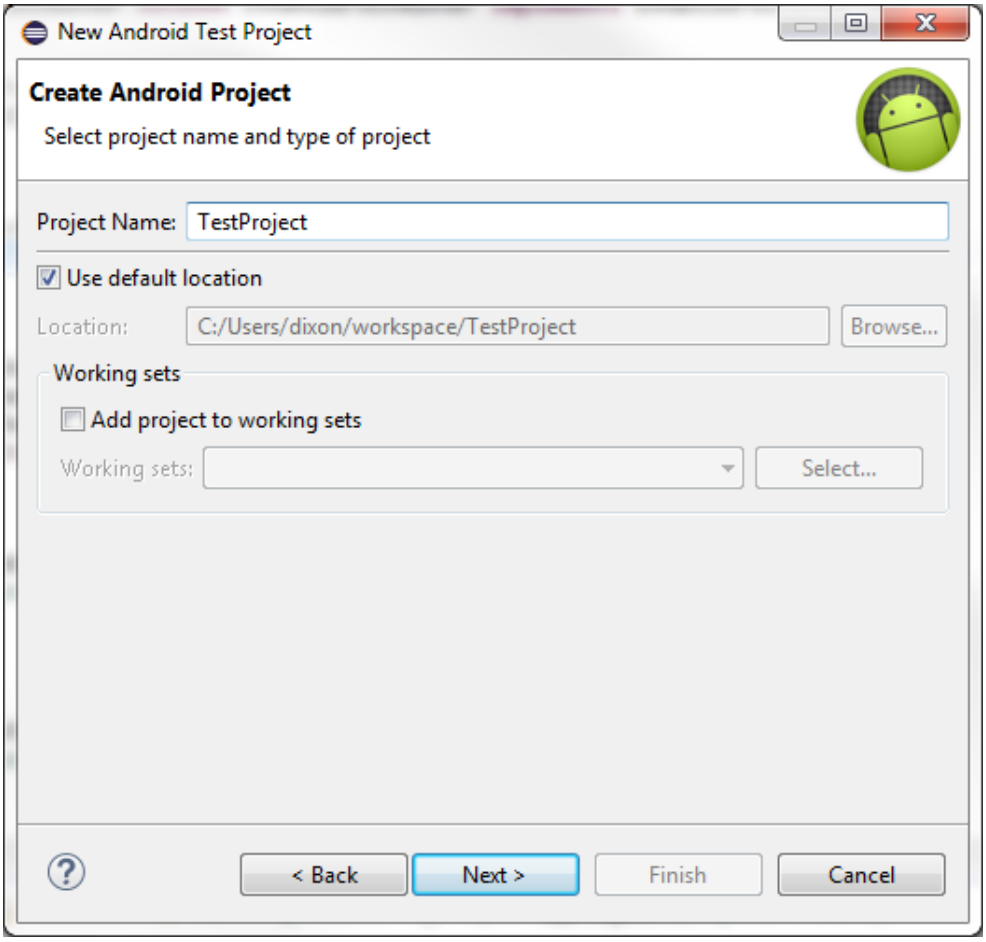
For this example I will be using the Eclipse IDE environment.

You can download the latest release from their website https://eclipse.org/downloads/

Note: *Select the IDE for the task you wish to accomplish; for the purpose of this demo I will be using the standard "Eclipse IDE for Java Developers"*

Step 2 – Create a new Android test project

Once eclipse is open, use the navigation menu to select and create a new project.

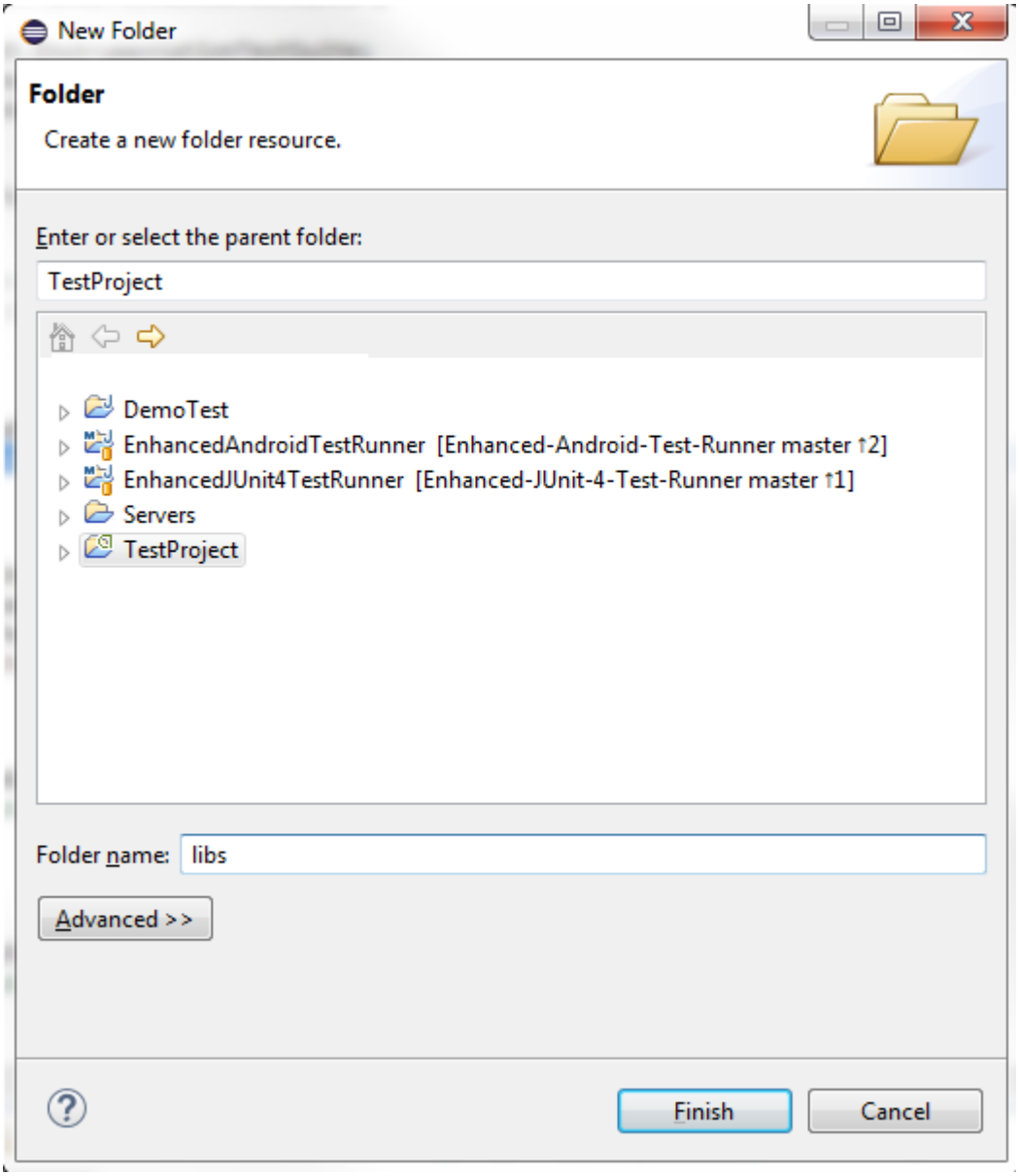


Once you have done this, give the project a name and a location for your new project and select the Android project you wish to test.

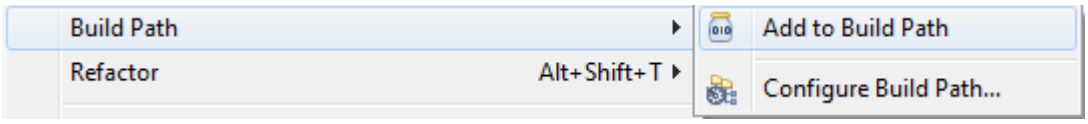
Step 3 – Add the correct libraries

Create a new folder by right clicking on your project and selecting new ->folder.

You can name this anything but for standardisation I will call it “libs”, which is an abbreviation for libraries.



Once this is created, drag the version of the EATR library you have into this folder. Finally right click this and select “add to build path”.



Once this is done you can add the JavaDoc or Sources packages if you so require, but for the purpose of this demonstration, this step has been ignored as it is not required.

You can also find the JavaDocs on our website.

Step 4 – Create your test runner

As like with the Android Instrumentation Test Runner, create a new class to extend the custom "EnhancedTestRunner". You should be given three methods to override as part of this.

getSetup

The getSetup method asks for a class which implements the @Setup annotation. For the purpose of this demo, I will add the @Setup annotation and call getClass().

For information regarding the Setup annotation, please see the JavaDoc.

getTestInterface

This provides callbacks on test events; it will be covered further in the building out stage. For now leave this null.

getTestSuites

This is the core method in which you add your test classes. In here please add an InstrumentationTestSuite and add the classes you wish to test.

```
import com.rysource.annotations.Setup;

@Setup(
        application = "My Test Application",
        version = "1.0.0",
        attempt = 1,
        features = {
                "Playback of a clear HLS v1 Stream"
        },
        knownDefects = {
                "Some devices might lag with playback"
        },
        reportType = ReportType.EXCEL_REPORT,
        retainSuppressedTests = true
        )
public class DemoTestRunner extends EnhancedTestRunner {

    @Override
    public Class<?> getSetup() {
        return getClass();
    }

    @Override
    public EnhancedTestInterface getTestInterface() {
        return null;
    }

    @Override
    public TestSuite getTestSuites() {
        InstrumentationTestSuite suite = new InstrumentationTestSuite(this);
        suite.addTestSuite(ClearPlaybackTests.class);
        return suite;
    }

}
```

Step 5 – Create your first test

Next go ahead and create a new class to hold your first test suite, once you have created your tests you can then annotate them with the SuiteInformation and TestInformation annotations.

Note: It is still important to use the “SmallTest”, “MediumTest”, “LargeTest” or “Suppress” as this identifies them as tests, whereas the information annotations simply identify metadata.

See the below example for a finalised test suite, ignoring the AutomationWrapper.

```
@SuiteInformation(
        suiteName = "Clear Playback",
        suiteDescription = "Custom Suite Description",
        priority = SuitePriority.MEDIUM,
        suiteAcceptanceCriteria =
            {
                "All tests pass",
                "No tests fail"
            }
        )
public class DemoPlaybackTests extends AutomationWrapper {

    @Override
    protected void setUp() throws Exception {
        super.setUp();
    }

    @Override
    protected void tearDown() throws Exception {
        super.tearDown();
    }

    @TestInformation(
            testName = "PlayHLSV1Master",
            testDescription = "Playback of a HLS v1 Master Playlist",
            expectedBehaviour = "Playback occurs successfully",
            priority = TestPriority.HIGH,
            type = TestType.AUTOMATIC
            )
    @SmallTest
    public void testHlsV1MasterPlayback() throws Exception {
        prepareTest("Tests/ClearPlayback/HLSV1/HLS1MasterPlaylist.html");
        CommonUtils.waitForTest();
    }

    @SmallTest
    public void testHlsV1MediaPlayback() throws Exception {
        prepareTest("Tests/ClearPlayback/HLSV1/HLS1MediaPlaylist.html");
        CommonUtils.waitForTest();
    }
```

Building Out

This section is designed to cover the use of custom annotations and interfaces. It will give the test developer an idea as to how to implement and output an AGILE based test report using acceptance criteria to base your tests at a feature level.

Adding the Enhanced Test Interface

The first feature provided by this SDK is the ETI or “Enhanced Test Interface”.

The class which implements this interface should be passed back in the getTestInterface abstract method as invoked earlier when you extend the EnhancedTestRunner.

These call-backs can be used to parse live results to any business intelligence system of your choosing. For example, send an email on a test failure during a continuous integration build of your software.

For information regarding the API’s, please refer to the JavaDoc.

```
public class DemoTestRunner extends EnhancedTestRunner implements EnhancedTestInterface {

    @Override
    public void onExecutionFailure(String arg0, String arg1, String arg2) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onTestFailure(String arg0, String arg1, String arg2) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onTestFinished(boolean arg0, String arg1, String arg2) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onTestPassed(String arg0, String arg1) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onTestStarted(String arg0, String arg1) {
        // TODO Auto-generated method stub

    }
```

Reporting

One of the core features of the EATR is the ability to output custom reports based on the metadata for each test you provided in the Suite collection.

Before you can output tests, please fill in as much as possible to the @Setup annotation which you defined earlier. Inside of this annotation, add basic information regarding the application you are writing. This information is used as a front page of the test report and will also be used in the future for retention and comparative purposes.

```
@Setup(
        application = "My Test Application",
        version = "1.0.0",
        attempt = 1,
        features = {
                "Playback of a clear HLS v1 Stre
        },
        knownDefects = {
                "Some devices might lag with pla
        },
        reportType = ReportType.EXCEL_REPORT,
        retainSuppressedTests = true
        )
```

This annotation is the only requirement for a report to be outputted. An example of the Excel report front page can be seen below.

1						
2	Name	Test Application				
3	Version	Beta 1.0				
4	Attempt	1				
5						
6	Features	Tests basic addition of two hard coded integers				
7		Tests basic subtraction of two hard coded integers				
8						
9	Defects	No known defects in this feature				
10						
11	Passed Tests	2	66%			
12	Failed Tests	1	33%			
13	Tests Not Run	0	0%			
14	Total	3	100%			
15						
16	**Suite ID**	**Description**	**Passed**	**Failed**	**Not Tested**	**Total**
17	Subtraction Tests	Test various subtraction calculations	1	0	0	1
18	Addition Tests	Test various addition calculations	1	1	0	2

Versioning

This section covers some changes over release version, for further information please refer to the GitHub releases page: https://github.com/SKLn-Rad/Enhanced-Android-Test-Runner/releases

Version 0.2.0

All call-backs working as expected

Limitations and Enhancements

This section covers the features allowed in each version of the EATR runner and limitations of the runner itself.

Version 0.2.0

Excel Reporting still in development (Expected 1.0.0 Release)

Feedback and Contact

Have ideas for improvements or feedback regarding the solution? Please contact us on the details below to discuss.

Any feedback, comments, questions or recommendations; please email ryandixon1993@gmail.com or leave an issue on the GitHub page: https://github.com/SKLn-Rad/Enhanced-Android-Test-Runner