# Enhanced Junit 4 Test Runner

## Synopsis

The Enhanced Junit 4 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 Junit 4 is deployed.

## How it works?

The EJ4TR extends the existing Suite class to provide the same setup and configuration process of a traditional Junit 4 implementation. When enabled with the @RunWith annotation, the end developer can configure and retrieve call-backs on events by using the provided annotations or interfaces as described in the JavaDoc or Getting Started section.

## Contents

# 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 EJ4TR.

## Step 1 – Download an IDE of your choice

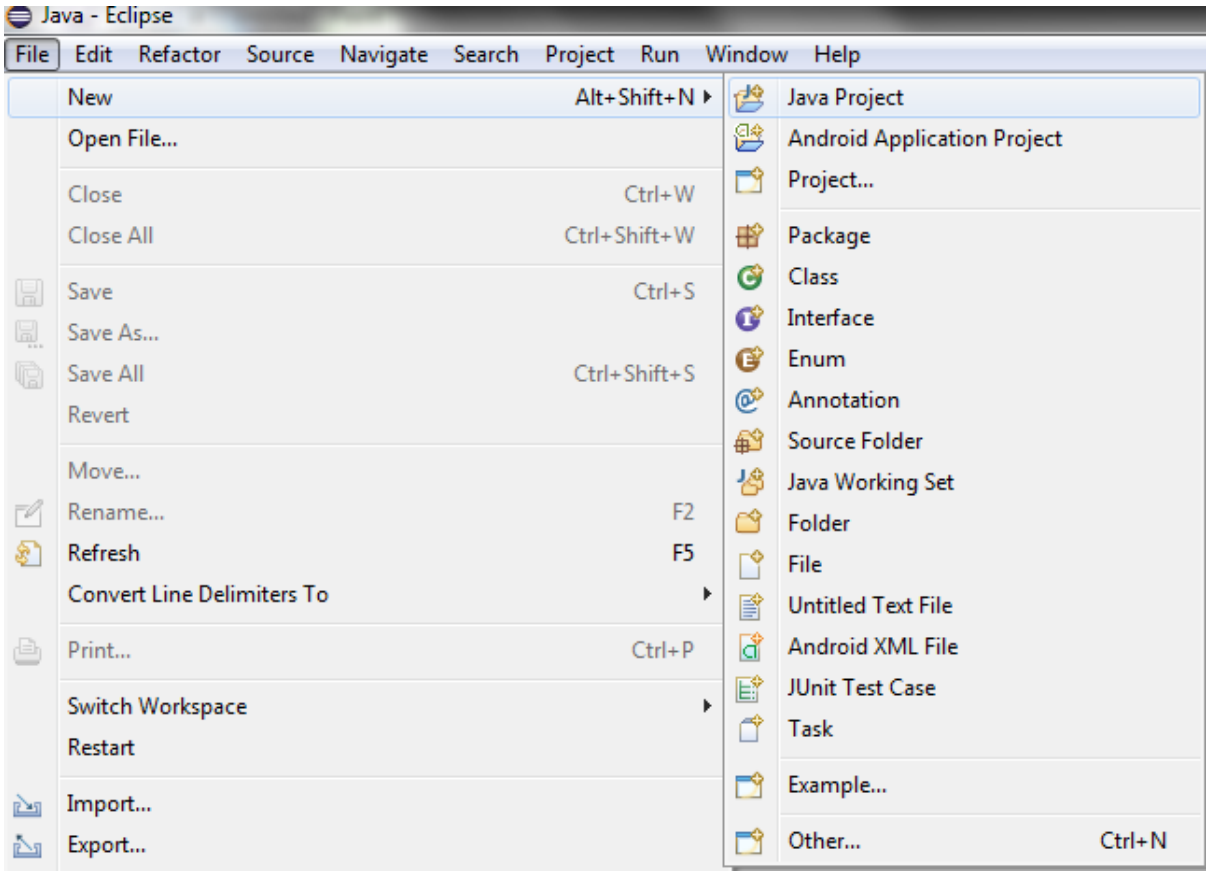
For this example I will be using the Eclipse IDE environment.

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 Java project

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

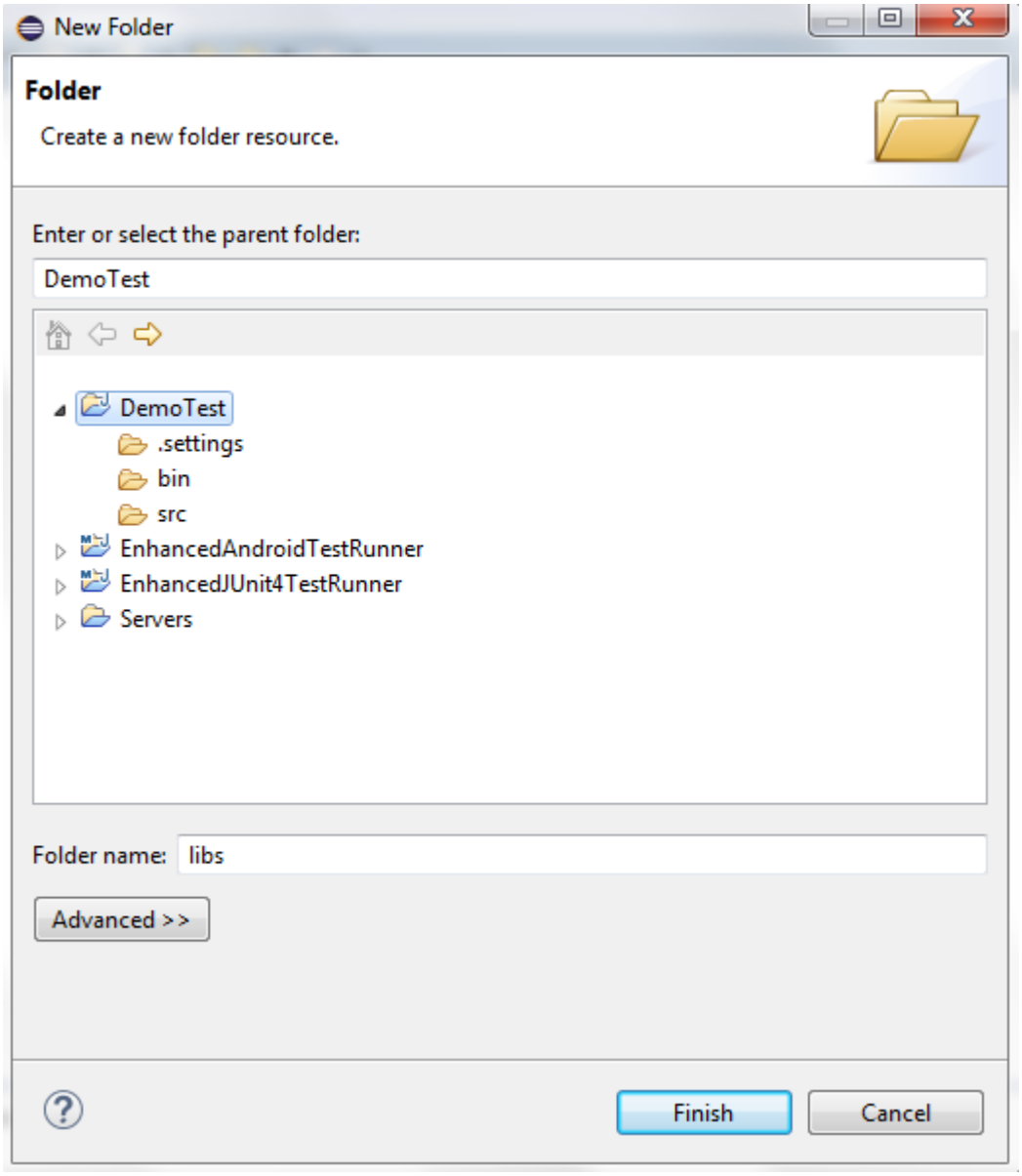



Once you have done this, give the project a name and select a JDK version. In this example I will be using JavaSE 1.8.

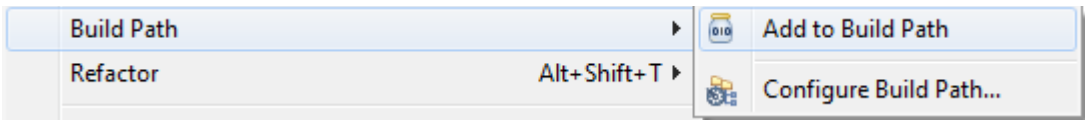
## Step 3 – Add the correct libraries

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 EJ4TR 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 handler

As like in Junit 4, you must have a method with utilises the @RunWith and @SuiteClasses annotation.

In Eclipse, create a new class named whatever you want and add the following annotations.

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite.SuiteClasses;

import com.rad.runner.EnhancedSuite;

@RunWith(EnhancedSuite.class)
@SuiteClasses({

})
public class TestController {

}
```

### *@RunWith*

This must point towards our EnhancedSuite class as this provides the hooks for the interface and test case management provided as part of this solution.

### *@SuiteClasses*

This String[] contains all of the Classes in which tests can be found.

## Step 5 – Create your first test

Create a new class for your test; this will be used in this demonstration throughout all of the sections including the building out phase.

Once created, add your test and add this class to your @SuiteClasses String[] as seen below.

```
@SuiteClasses({
    TestAddition.class
})
```

```
import org.junit.Assert;
import org.junit.Test;

public class TestAddition {

    @Test
    public void testBasicAddition() {
        boolean passed = true;
        Assert.assertTrue(passed);
    }

}
```

Once created, you should be able to right click and run the test as a Junit Test with a successful outcome. This can be seen in the Junit panel located where your Project Panel was previously.

# 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".

This interface must be implemented in the same Class as the @RunWith annotation and can be used to provide live callbacks during test events directly from the RunNotifier itself.

To implement this, simply add the statement to the existing RunController Class as seen below.

Once complete, you should be prompted to add the missing methods.

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite.SuiteClasses;

import com.rad.runner.EnhancedSuite;
import com.rad.runner.interfaces.EnhancedTestInterface;

@RunWith(EnhancedSuite.class)
@SuiteClasses({
    TestAddition.class
})
public class TestController implements EnhancedTestInterface {

    @Override
    public void onTestFailure(String arg0, String arg1, String arg2, String arg3) {
        // TODO Auto-generated method stub

    }

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

    }

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

    }
```

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.

## Adding Suite and Test Metadata

One of the other features is the ability to add Test and Feature level metadata into your tests directly. This can be used by the user to pass directly into BI systems or it can be used in the EJ4TR to provide more verbose and AGILE tailored information into the test report generated locally on the device after the test run.

To start, add the @SuiteInformation annotation into the class your wish to annotate, and the @TestInformation annotation into the test case. You can manually adjust the information using the available API's, or just use the defaults as can be seen in the JavaDoc.

```
import org.junit.Assert;
import org.junit.Test;

import com.rad.runner.annotations.SuiteInformation;
import com.rad.runner.annotations.SuiteInformation.SuitePriority;
import com.rad.runner.annotations.TestInformation;
import com.rad.runner.annotations.TestInformation.TestPriority;
import com.rad.runner.annotations.TestInformation.TestType;

@SuiteInformation(
        suiteName = "Addition Tests",
        suiteDescription = "Test various addition calculations",
        priority = SuitePriority.HIGH,
        suiteAcceptanceCriteria = {
                "All calculations must be correct",
                "At least one addition based calculation should be included in the suite"
        })
public class TestAddition {

    @TestInformation(
            testName = "Test Basic Addition",
            testDescription = "Test what happens when you add one and two together",
            expectedResult = "The result should equal 3",
            priority = TestPriority.HIGH,
            type = TestType.AUTOMATIC)
    @Test
    public void testBasicAddition() {
        int one = 1;
        int two = 2;
        Assert.assertTrue((one + two) == 3);
    }

}
```

See the above image for an example of how a test should be added to the suite; notice how both the @Test and @TestInformation are required. All annotations ending in "Information" do not replace the original as a declaration of the intention of the methods execution, rather as an indication that the information is available post-test and should be used in the report.

If you run the test, no information should be provided into the console. Please use the EnhancedTestInterface to get the Class name and Method name of the test and use this to reflectively get the annotation for that test if you wish to use it in your own logic.

## Reporting

One of the core features of the EJ4TR 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 add the @Setup annotation to the TestController Class we created earlier. Inside of this method, 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.

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite.SuiteClasses;

import com.rad.runner.EnhancedSuite;
import com.rad.runner.annotations.Setup;
import com.rad.runner.annotations.Setup.ReportType;
import com.rad.runner.interfaces.EnhancedTestInterface;

@RunWith(EnhancedSuite.class)
@SuiteClasses({
    TestAddition.class
})
@Setup(
        application = "",
        attempt = 1,
        version = "",
        reportType = ReportType.EXCEL,
        retainIgnoredTests = true,
        features = {
                "Tests basic addition of two hard coded integers"
        },
        knownDefects = {
                "No known defects in this feature"
        })
public class TestController implements EnhancedTestInterface {
```

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

*Version 1.1.0*

First release, includes Test annotation, Suite annotation, basic Excel reporting and the ETI

# Limitations and Enhancements

This section covers the features allowed in each version of the EJ4TR runner and limitations of the runner itself.

*Version 1.2.0*

Excel Reporter Finalised (Beta)

Junit XML Style Reporting (Stub added, to be completed in 1.3.0)

*Version 1.1.0*

Excel report only includes summary page (To be fixed in 1.2.0)

No tester passed into report (To be fixed in 1.2.0)

Hard coded reporting location (To be fixed in 1.2.0)

# 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