SpiraTest / SpiraTeam | Automated Unit Testing

Integration & User Guide

Inflectra Corporation

Date: October 3rd, 2014

Contents

1. Introduction 1

2. Integrating with NUnit 2

3. Integrating with JUnit 7

4. Integrating with TestNG 14

5. Integrating with Selenium-RC 21

6. Integrating with PyUnit 25

7. Integrating with MS-Test 29

8. Integrating with Ruby Test::Unit . 35

9. Integrating with Perl TAP 39

10. Integrating with PHPUnit 43

1. Introduction

SpiraTest® provides an integrated, holistic Quality Assurance (QA) management solution that manages requirements, tests and incidents in one environment, with complete traceability from inception to completion.

SpiraTeam® is an integrated Application Lifecycle Management (ALM) system that manages your project's requirements, releases, test cases, issues and tasks in one unified environment. SpiraTeam® contains all of the features provided by SpiraTest® - our highly acclaimed quality assurance system and SpiraPlan® - our agile-enabled project management solution.

This guide outlines how to use either SpiraTest® or SpiraTeam® in conjunction with a variety of automated unit testing tools. This guide assumes that the reader is familiar with both SpiraTest/SpiraTeam and the appropriate automated unit testing tool. For information regarding how to use SpiraTest, please refer to the *SpiraTest User Manual*.

Note: This guide does *not* cover use of the Spira RemoteLaunch™ automated testing system that can be used to remotely schedule and launch automated functional and load tests. For information on using Spira RemoteLaunch™ please refer to the *Spira RemoteLaunch User Guide*.

2. Integrating with NUnit

2.1. Installing the NUnit Add-In

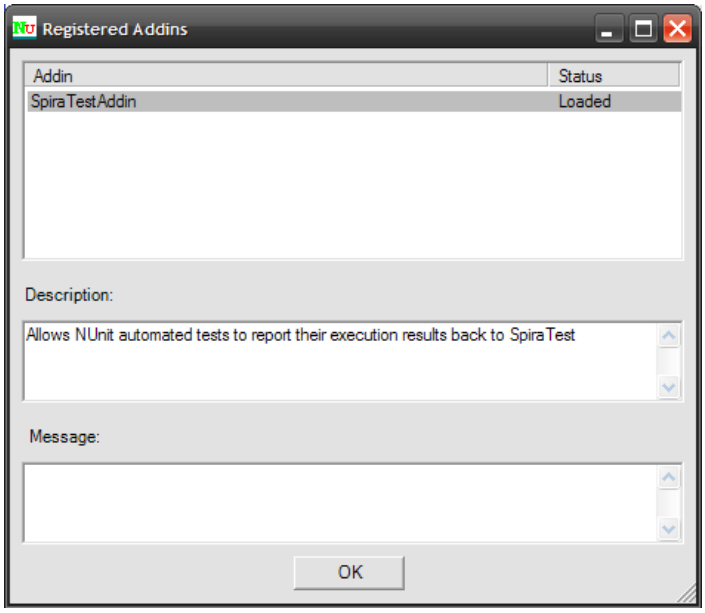
This section outlines how to install the SpiraTest Add-In for NUnit onto a workstation so that you can then run automated NUnit tests against a .NET application and have the results be recorded as test runs inside SpiraTest. It assumes that you already have a working installation of SpiraTest v2.2 or later. If you have an earlier version of SpiraTest you will need to upgrade to at least v2.2 before trying to use this add-in. You will also need to have either version **v2.5.5** or **v2.6.3 of NUnit**, since there are two versions of the add-in that have been compiled with the v2.5.5 and v2.6.3 NUnit APIs. If you are using a different version, please visit www.nunit.org to obtain the appropriate version (2.5.5 or 2.6.3).

To obtain the version of the add-in that is compatible with your version of SpiraTest, you simply need to go to http://www.inflectra.com/SpiraTest/Downloads.aspx or http://www.inflectra.com/SpiraTeam/Downloads.aspx and download the NUnit Add-In zipfile.

Once you have obtained the NUnit Zipfile from our website, you should extract all the files from zip archive into a temporary folder on your computer (e.g. C:\Temp).

Next, you should copy the add-in libraries to the folder NUnit expects to find them in. First, if you are running any instances of the NUnit GUI, close them. Then, copy the `SpiraTestNUnitAddIn.dll` assembly from its location in the temporary folder to the NUnit Add-In folder (typically `C:\Program Files\NUnit 2.5.5\bin\net-2.0\addins`).

Now you can restart the NUnit GUI application. To check that the add-in was loaded successfully, click on Tools > Addins… to bring up the list of loaded add-ins:



You should see an entry marked “SpiraTest Addin” listed with its detailed description and status “Loaded”. If this does not happen, try closing and reopening NUnit.

2.2. Using NUnit with SpiraTest

The typical code structure for an NUnit test fixture coded in C# is as follows:

```
using System;
using NUnit.Framework;
```

```
namespace Inflectra.SpiraTest.AddOns.SpiraTestNUnitAddIn.SampleTestSuite
{
	/// <summary>
	/// Sample test fixture that tests the NUnit SpiraTest integration
	/// </summary>
	[TestFixture]
	public class SampleTestFixture
	{
		[SetUp]
		public void Init()
		{
			//Do Nothing
		}

		/// <summary>
		/// Sample test that asserts a failure
		/// </summary>
		[Test]
		public void _01_SampleFailure()
		{
			//Failure Assertion
			Assert.AreEqual (1, 0);
		}

		/// <summary>
		/// Sample test that succeeds
		/// </summary>
		[Test]
		public void _02_SamplePass()
		{
			//Successful assertion
			Assert.AreEqual (1, 1);
		}

		/// <summary>
		/// Sample test that fails
		/// </summary>
		[Test]
		public void _03_SampleIgnore()
		{
			//Failure Assertion
			Assert.AreEqual (1, 0);
		}
	}
}
```

The .NET class is marked as an NUnit test fixture by applying the [TestFixture] attribute to the class as a whole, and the [Test] attribute to each of the test assertion methods individually – highlighted in yellow above. When you open up the class in NUnit and click the <Run> button it loads all the test classes marked with [TestFixture] and executes all the methods marked with [Test] in turn.

Each of the Assert statements is used to test the state of the application after executing some sample code that calls the functionality being tested. If the condition in the assertion is true, then execution of the test continues, if it is false, then a failure is logged and NUnit moves on to the next test method.

So, to use SpiraTest with NUnit, each of the test cases written for execution by NUnit needs to have a corresponding test case in SpiraTest. These can be either existing test cases that have manual test steps or they can be new test cases designed specifically for automated testing and therefore have no defined test steps. In either case, the changes that need to be made to the NUnit test fixture for SpiraTest to record the NUnit test run are illustrated below:

```
using System;
using NUnit.Framework;
using Inflectra.SpiraTest.AddOns.SpiraTestNUnitAddIn.SpiraTestFramework;

namespace Inflectra.SpiraTest.AddOns.SpiraTestNUnitAddIn.SampleTestSuite
{
	/// <summary>
	/// Sample test fixture that tests the NUnit SpiraTest integration
	/// </summary>
	[
	TestFixture,
	SpiraTestConfiguration (
		"http://<server name>/SpiraTest",
		"<username>",
		"<password>",
		<project id>,
		<release id>,
		<test set id>,
		<runner name>
	)
	]
	public class SampleTestFixture
	{
		[SetUp]
		public void Init()
		{
			//Do Nothing
		}

		/// <summary>
		/// Sample test that asserts a failure
		/// </summary>
		[
		Test,
		SpiraTestCase (<test case id>)
		]
		public void _01_SampleFailure()
		{
			//Failure Assertion
			Assert.AreEqual (1, 0);
		}

		/// <summary>
		/// Sample test that succeeds
		/// </summary>
		[
		Test,
		SpiraTestCase (<test case id>))
		]
		public void _02_SamplePass()
		{
			//Successful assertion
			Assert.AreEqual (1, 1);
		}

		/// <summary>
		/// Sample test that does not log to SpiraTest
		/// </summary>
		[
		Test
		]
		public void _03_SampleIgnore()
		{
			//Failure Assertion
			Assert.AreEqual (1, 0);
```

```
            }
        }
}
```

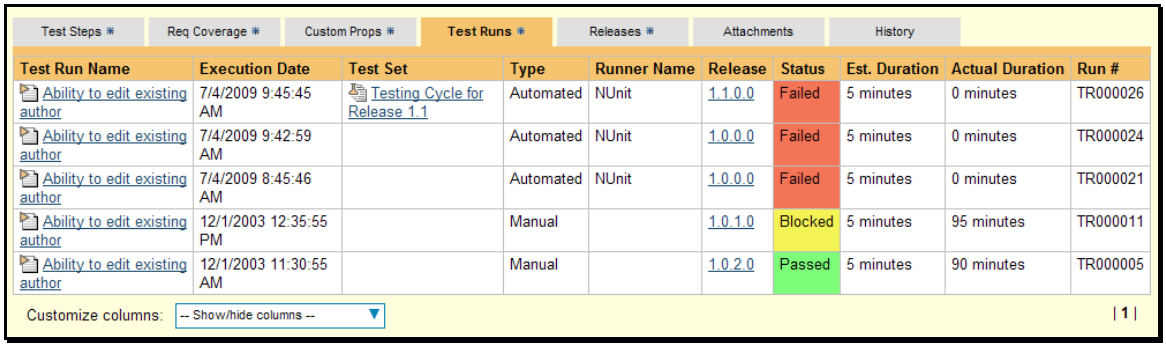
The overall class is marked with a new [SpiraTestConfiguration] attribute that contains the following pieces of information needed to access the SpiraTest test repository:

- **URL** - The URL to the instance of SpiraTest being accessed. This needs to start with `http://` or `https://`.
- **User Name** - A valid username for the instance of SpiraTest.
- **Password** - A valid password for the instance of SpiraTest.
- **Project Id** - The ID of the project (this can be found on the project homepage in the "Project Overview" section)
- **Release Id** (Optional) - The ID of the release to associate the test run with. This can be found on the releases list page (click on the Planning > Releases tab). If you don't want to specify a release, just use the value -1.
- **Test Set Id** (Optional) – The ID of the test set to associate the test run with. This can be found on the test set list page (click on the Testing > Test Sets tab). If you don't want to specify a test set, just use the value -1. If you choose a test set that is associated with a release, then you don't need to explicitly set a release id (i.e. just use -1). However if you do set a release value, it will override the value associated with the test set.
- **Runner Name** – This should be set to NUnit so that the test results recorded in SpiraTest have the name 'NUnit' associated with them.

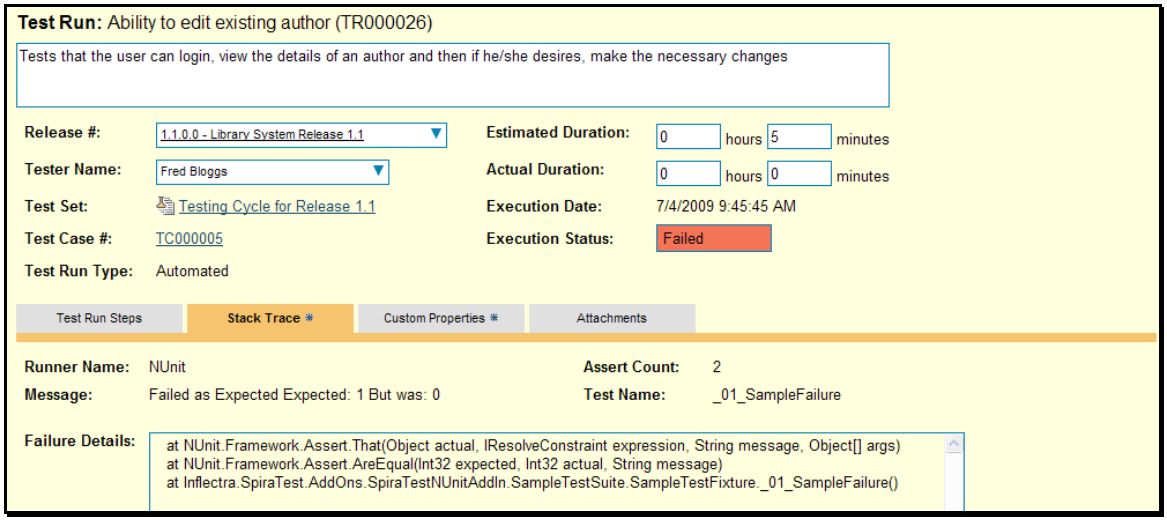
In addition, each of the individual test methods needs to be mapped to a specific test case within SpiraTest. This is done by adding a [SpiraTestCase] attribute to the test method together with the ID of the corresponding test case in SpiraTest. The Test Case ID can be found on the test cases list page (click the "Test Cases" tab).

For these attributes to be available in your test fixture, you also need to add a reference to the SpiraTestFramework.dll assembly. This assembly can be found in the temporary folder that you extracting the add-in to. It is recommended that you move this file from the temporary folder into a permanent folder located within your .NET project.

Now all you need to do is compile your code, launch NUnit, run the test fixtures as you would normally do, and when you view the test cases in SpiraTest, you should see an NUnit automated test run displayed in the list of executed test runs:



Clicking on one of the NUnit test runs will bring up a screen that provides information regarding what NUnit test method failed, what the error was, together with the associated code stack-trace:



Congratulations… You are now able to run NUnit automated tests and have the results be recorded within SpiraTest. The sample test fixture SampleTestSuite.cs is provided with the installation.

3. Integrating with JUnit

3.1. Installing the JUnit Extension

This section outlines how to install the SpiraTest Extension for JUnit onto a workstation so that you can then run automated JUnit tests against a Java application and have the results be recorded as test runs inside SpiraTest. It assumes that you already have a working installation of SpiraTest v3.0 or later. If you have an earlier version of SpiraTest you will need to upgrade to at least v3.0 before trying to use this extension. You will also need to have at least version 4.0 of JUnit, since earlier versions do not support the use of Java metadata and reflection. If you are using an earlier version, please visit www.junit.org to obtain the latest version.

To obtain the version of the JUnit extension that is compatible with your version of SpiraTest, you simply need to log-in as a project-level administrator to SpiraTest, go to the Administration home page and download the JUnit Extension compressed archive (.zip). This process is described in the *SpiraTest Administration Guide* in more detail.

The JUnit extension is provided as a compressed zipfile that includes both the binaries (packaged as a JAR-file) and the source code (stored in a folder structure that mirrors the Java classpath). The JAR-file binary was compiled for use on a Windows x86 platform, other platforms (e.g. Linux) will require you to compile the Java source files into the appropriate Java classfiles before using the extension. The rest of this section will assume that you are using the pre-compiled JAR-file.

Once you have downloaded the Zip archive, you need to uncompress it into a folder of your choice on your local system. Assuming that you uncompressed it to the `C:\Program Files\JUnit Extension` folder, you should have the following folder structure created:

```
C:\Program Files\JUnit Extension
C:\Program Files\JUnit Extension\com
C:\Program Files\JUnit Extension\com\inflectra
C:\Program Files\JUnit Extension\com\inflectra\spiratest
C:\Program Files\JUnit Extension\com\inflectra\spiratest\addons
C:\Program Files\JUnit Extension\com\inflectra\spiratest\addons\junitextension
C:\Program Files\JUnit Extension\com\inflectra\spiratest\addons\junitextension\samples
```

The JAR-file is located in the root folder, the source-code for the extension can be found in the “junitextension” subfolder, and the sample test fixture can be found in the “samples” subfolder.

Now to use the extension within your test cases, you need to first make sure that the JAR-file is added to the Java classpath. The method for doing this is dependent on the platform you’re using, so please refer to FAQ on www.junit.org for details on the appropriate method for your platform. As an example, on a Windows platform, the JAR-file would be added to the classpath by typing the following:

```
set CLASSPATH=%CLASSPATH%; C:\Program Files\JUnit Extension\JUnitExtension.jar
```

Once you have completed this step, you are now ready to begin using your JUnit test fixtures with SpiraTest.

2.2. Using JUnit with SpiraTest

The typical code structure for a JUnit test fixture coded in Java is as follows:

```
package com.inflectra.spiratest.addons.junitextension.samples;

import static org.junit.Assert.*;
import junit.framework.JUnit4TestAdapter;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.*;
import org.junit.runner.notification.*;

import java.util.*;

/**
 * Some simple tests using JUnit 4
 *
 * @author            Inflectra Corporation
 * @version           2.3.0
 *
 */
public class SimpleTest
{
      protected int fValue1;
      protected int fValue2;

      /**
       * Sets up the unit test
       */
      @Before
      public void setUp()
      {
            fValue1= 2;
            fValue2= 3;
      }

      /**
       * Tests the addition of the two values
       */
      @Test
      public void testAdd()
      {
            double result = fValue1 + fValue2;

            // forced failure result == 5
            assertTrue (result == 6);
      }

      /**
       * Tests division by zero
       */
      @Test
      public void testDivideByZero()
      {
            int zero = 0;
            int result = 8 / zero;
            result++; // avoid warning for not using result
      }

      /**
       * Tests two equal values
       */
      @Test
```

```
        public void testEquals()
        {
                assertEquals(12, 12);
                assertEquals(12L, 12L);
                assertEquals(new Long(12), new Long(12));

                assertEquals("Size", 12, 13);
                assertEquals("Capacity", 12.0, 11.99, 0.0);
        }

        /**
         * Tests success
         */
        @Test
        public void testSuccess()
        {
                //Successful test
                assertEquals(12, 12);
        }

        /**
         * Entry point for command line execution
         *
         * @param args The command line arguments
         */
        public static void main (String[] args)
        {
                //Instantiate the JUnit core
                JUnitCore core = new JUnitCore();

                //Finally run the test fixture
                core.run (SimpleTest.class);
        }

        /**
         * Entry point for JUnit 4.x runners
         *
         * @return              Handle to the test framework
         */
        public static junit.framework.Test suite()
        {
                return new JUnit4TestAdapter(SimpleTest.class);
        }
}
```

The Java class is marked as a JUnit test fixture by applying the @Before attribute to the setup method, and the @Test attribute to each of the test assertion methods individually – highlighted in yellow above. When you open up the class in a JUnit runner or execute from the command line it loads all the test classes and executes all the methods marked with @Test in turn.

Each of the Assert statements is used to test the state of the application after executing some sample code that calls the functionality being tested. If the condition in the assertion is true, then execution of the test continues, if it is false, then a failure is logged and JUnit moves on to the next test method.

So, to use SpiraTest with JUnit, each of the test cases written for execution by JUnit needs to have a corresponding test case in SpiraTest. These can be either existing test cases that have manual test steps or they can be new test cases designed specifically for automated testing and therefore have no defined test steps. In either case, the changes that need to be made to the JUnit test fixture for SpiraTest to record the JUnit test run are illustrated below:

```
package com.inflectra.spiratest.addons.junitextension.samples;
```

```
import static org.junit.Assert.*;
import junit.framework.JUnit4TestAdapter;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.*;
import org.junit.runner.notification.*;

import com.inflectra.spiratest.addons.junitextension.*;

import java.util.*;

/**
 * Some simple tests using the ability to return results back to SpiraTest
 *
 * @author           Inflectra Corporation
 * @version          2.3.0
 *
 */
@SpiraTestConfiguration(
      url="http://sandman/SpiraTest",
      login="fredbloggs",
      password="fredbloggs",
      projectId=1,
      releaseId=1,
      testSetId=1
)
public class SimpleTest
{
      protected int fValue1;
      protected int fValue2;

      /**
       * Sets up the unit test
       */
      @Before
      public void setUp()
      {
            fValue1= 2;
            fValue2= 3;
      }

      /**
       * Tests the addition of the two values
       */
      @Test
      @SpiraTestCase(testCaseId=5)
      public void testAdd()
      {
            double result = fValue1 + fValue2;

            // forced failure result == 5
            assertTrue (result == 6);
      }

      /**
       * Tests division by zero
       */
      @Test
      @SpiraTestCase(testCaseId=5)
      public void testDivideByZero()
      {
            int zero = 0;
            int result = 8 / zero;
            result++; // avoid warning for not using result
```

```
        }

        /**
         * Tests two equal values
         */
        @Test
        @SpiraTestCase(testCaseId=6)
        public void testEquals()
        {
                assertEquals(12, 12);
                assertEquals(12L, 12L);
                assertEquals(new Long(12), new Long(12));

                assertEquals("Size", 12, 13);
                assertEquals("Capacity", 12.0, 11.99, 0.0);
        }

        /**
         * Tests success
         */
        @Test
        @SpiraTestCase(testCaseId=6)
        public void testSuccess()
        {
                //Successful test
                assertEquals(12, 12);
        }

        /**
         * Entry point for command line execution
         *
         * @param args The command line arguments
         */
        public static void main (String[] args)
        {
                //Instantiate the JUnit core
                JUnitCore core = new JUnitCore();

                //Add the custom SpiraTest listener
                core.addListener(new SpiraTestListener());

                //Finally run the test fixture
                core.run (SimpleTest.class);
        }

        /**
         * Entry point for JUnit 4.x runners
         *
         * @return            Handle to the test framework
         */
        public static junit.framework.Test suite()
        {
                return new JUnit4TestAdapter(SimpleTest.class);
        }
}
```

The overall class is marked with a new @SpiraTestConfiguration attribute that contains the following pieces of information needed to access the SpiraTest test repository:

- **URL** - The URL to the instance of SpiraTest being accessed. This needs to start with `http://` or `https://`.
- **Login** - A valid username for the instance of SpiraTest.

- **Password** - A valid password for the instance of SpiraTest.
- **Project Id** - The ID of the project (this can be found on the project homepage in the “Project Overview” section)
- **Release Id** (Optional) - The ID of the release to associate the test run with. This can be found on the releases list page (click on the Planning > Releases tab). If you don’t want to specify a release, just use the value -1.
- **Test Set Id** (Optional) – The ID of the test set to associate the test run with. This can be found on the test set list page (click on the Testing > Test Sets tab). If you don’t want to specify a test set, just use the value -1. If you choose a test set that is associated with a release, then you don’t need to explicitly set a release id (i.e. just use -1). However if you do set a release value, it will override the value associated with the test set.

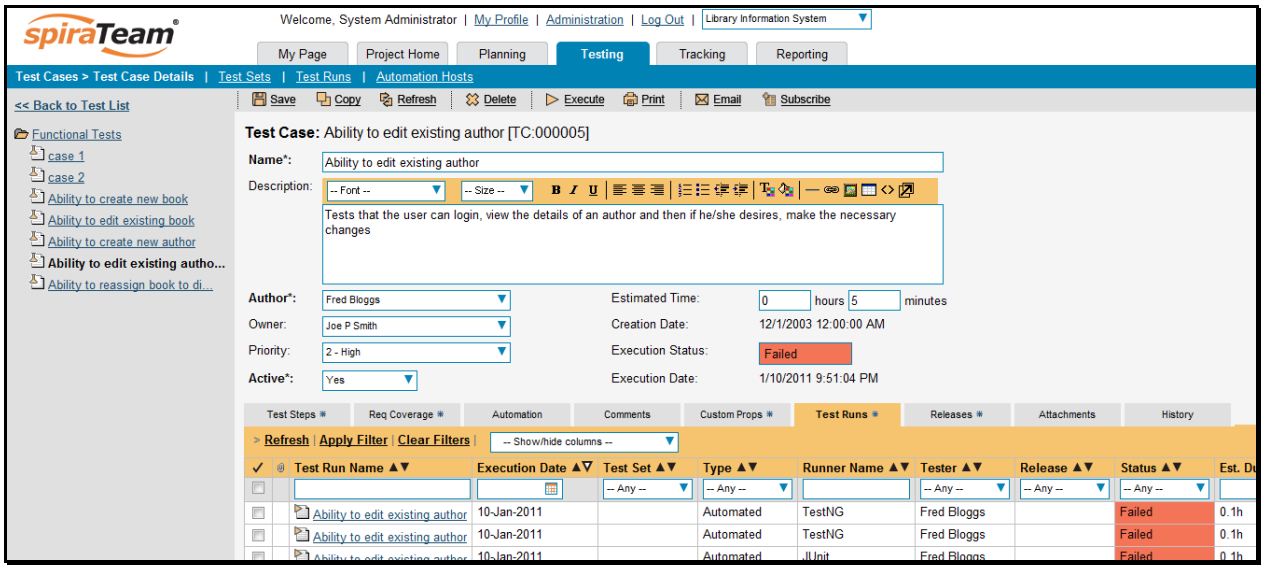
In addition, each of the individual test methods needs to be mapped to a specific test case within SpiraTest. This is done by adding a @SpiraTestCase attribute to the test method together with the ID of the corresponding test case in SpiraTest. The Test Case ID can be found on the test cases list page (click the “Test Cases” tab).

For these attributes to be available in your test fixture, you also need to add a reference to the *com.inflectra.spiratest.addons.junitextension* package. This package is bundled within the supplied JAR-file library for Windows, and can be compiled from the provided source .java files on other platforms.

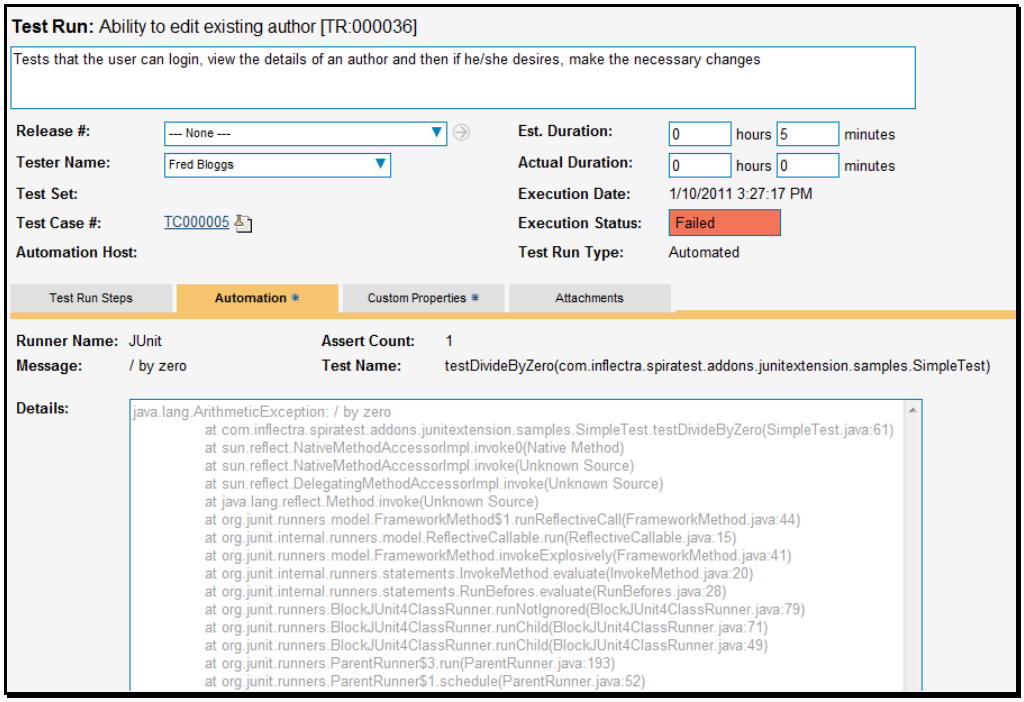
Now all you need to do is compile your code and then launch JUnit by executing the test fixture through the command line (or through your choice of IDE, e.g. Eclipse). E.g. for our sample test, you would use the following command:

```
java com.inflectra.spiratest.addons.junitextension.samples.SimpleTest
```

Once the test has run, you can view the test cases in SpiraTest, you should see a JUnit automated test run displayed in the list of executed test runs:



Clicking on one of the JUnit test runs will bring up a screen that provides information regarding what JUnit test method failed, what the error was, together with the associated code stack-trace:



Congratulations… You are now able to run JUnit automated tests and have the results be recorded within SpiraTest. The sample test fixture SimpleText.java is provided with the installation.

4. Integrating with TestNG

4.1. Installing the TestNG Listener

This section outlines how to install the SpiraTest Listener for TestNG onto a workstation so that you can then run automated TestNG unit tests against a Java application and have the results be recorded as test runs inside SpiraTest. It assumes that you already have a working installation of SpiraTest v3.0 or later. If you have an earlier version of SpiraTest you will need to upgrade to at least v3.0 before trying to use this listener. You will also need to have at least version 1.0 of TestNG running under JDK 1.5 or later, since earlier versions do not have support for annotations and custom listeners. If you are using an earlier version, please visit www.testng.org to obtain the latest version.

To obtain the latest version of the TestNG listener, you simply need to log-in as a project-level administrator to SpiraTest, go to the Administration home page and download the SpiraTest TestNG Listener compressed archive (.zip) from the section that lists downloads and add-ons. This process is described in the *SpiraTest Administration Guide* in more detail.

The TestNG listener is provided as a compressed zipfile that includes both the binaries (packaged as a JAR-file) and the source code (stored in a folder structure that mirrors the Java classpath). The JAR-file binary was compiled for use on a Windows x86 platform, other platforms (e.g. Linux) will require you to compile the Java source files into the appropriate Java classfiles before using the extension. The rest of this section will assume that you are using the pre-compiled JAR-file.

Once you have downloaded the Zip archive, you need to uncompress it into a folder of your choice on your local system. Assuming that you uncompressed it to the `C:\Program Files\SpiraTestListener` folder, you should have the following folder structure created:

```
C:\Program Files\SpiraTestListener
C:\Program Files\SpiraTestListener\com
C:\Program Files\SpiraTestListener\com\inflectra
C:\Program Files\SpiraTestListener\com\inflectra\spiratest
C:\Program Files\SpiraTestListener\com\inflectra\spiratest\addons
C:\Program Files\SpiraTestListener\com\inflectra\spiratest\addons\testnglistener
C:\Program Files\SpiraTestListener\Extension\com\inflectra\spiratest\addons\testnglistener\samples
```

The JAR-file is located in the root folder, the source-code for the extension can be found in the “testnglistener” subfolder, and the sample test fixture can be found in the “samples” subfolder.

Now to use the listener within your test cases, you need to first make sure that the JAR-file is added to the Java classpath. The method for doing this is dependent on the platform you’re using, so please refer to FAQ on www.testngorg for details on the appropriate method for your platform. As an example, on a Windows platform, the JAR-file would be added to the classpath by typing the following:

```
set CLASSPATH=%CLASSPATH%; C:\Program Files\SpiraTestListener\TestNGListener.jar
```

Once you have completed this step, you are now ready to begin using your TestNG test fixtures with SpiraTest.

2.2. Using TestNG with SpiraTest

The typical code structure for a TestNG test fixture coded in Java is as follows:

```
package com.inflectra.spiratest.addons.testnglistener.samples;

import org.testng.annotations.*;
import static org.testng.AssertJUnit.*;

import java.util.*;

/**
 * Some simple tests using the ability to return results back to SpiraTest
 *
 * @author              Inflectra Corporation
 * @version             2.3.0
 *
 */
@Test(groups={"unittest"})
public class SimpleTest
{
        protected int fValue1;
        protected int fValue2;

        /**
         * Sets up the unit test
         */
        @BeforeClass
        public void setUp()
        {
                fValue1= 2;
                fValue2= 3;
        }

        /**
         * Tests the addition of the two values
         */
        @Test(groups={"unittest"})
        public void testAdd()
        {
                double result = fValue1 + fValue2;

                // forced failure result == 5
                assertTrue (result == 6);
        }

        /**
         * Tests division by zero
         */
        @Test(groups={"unittest"})
        public void testDivideByZero()
        {
                int zero = 0;
                int result = 8 / zero;
                result++; // avoid warning for not using result
        }

        /**
         * Tests two equal values
         */
        @Test(groups={"unittest"})
        public void testEquals()
        {
                assertEquals(12, 12);
                assertEquals(12L, 12L);
                assertEquals(new Long(12), new Long(12));

                assertEquals("Size", 12, 13);
                assertEquals("Capacity", 12.0, 11.99, 0.0);
        }
```

```
        /**
         * Tests success
         */
        @Test(groups={"unittest"})
        public void testSuccess()
        {
                //Successful test
                assertEquals(12, 12);
        }
}
```

The Java class is marked as a TestNG test fixture by applying the @Test attribute to the class definition, and the @Test attribute to each of the test assertion methods individually – highlighted in yellow above. In addition, special setup methods are marked with annotations such as @BeforeClass. When you open up the class in a TestNG runner or execute from the command line it loads all the test classes and executes all the methods marked with @Test in turn.

Each of the Assert statements is used to test the state of the application after executing some sample code that calls the functionality being tested. If the condition in the assertion is true, then execution of the test continues, if it is false, then a failure is logged and TestNG moves on to the next test method.

So, to use SpiraTest with TestNG, each of the test cases written for execution by TestNG needs to have a corresponding test case in SpiraTest. These can be either existing test cases that have manual test steps or they can be new test cases designed specifically for automated testing and therefore have no defined test steps. In either case, the changes that need to be made to the TestNG test fixture for SpiraTest to record the TestNG test run are illustrated below:

```
package com.inflectra.spiratest.addons.testnglistener.samples;

import org.testng.annotations.*;
import static org.testng.AssertJUnit.*;

import com.inflectra.spiratest.addons.testnglistener.*;

import java.util.*;

/**
 * Some simple tests using the ability to return results back to SpiraTest
 *
 * @author              Inflectra Corporation
 * @version             2.3.0
 *
 */
@SpiraTestConfiguration(
        url="http://localhost/SpiraTest",
        login="fredbloggs",
        password="fredbloggs",
        projectId=1,
        releaseId=1,
        testSetId
)
@Test(groups={"unittest"})
public class SimpleTest
{
        protected int fValue1;
        protected int fValue2;

        /**
         * Sets up the unit test
         */
        @BeforeClass
        public void setUp()
        {
                fValue1= 2;
                fValue2= 3;
        }
```

```
        /**
         * Tests the addition of the two values
         */
        @Test(groups={"unittest"})
        @SpiraTestCase(testCaseId=5)
        public void testAdd()
        {
                double result = fValue1 + fValue2;

                // forced failure result == 5
                assertTrue (result == 6);
        }

        /**
         * Tests division by zero
         */
        @Test(groups={"unittest"})
        @SpiraTestCase(testCaseId=5)
        public void testDivideByZero()
        {
                int zero = 0;
                int result = 8 / zero;
                result++; // avoid warning for not using result
        }

        /**
         * Tests two equal values
         */
        @Test(groups={"unittest"})
        @SpiraTestCase(testCaseId=6)
        public void testEquals()
        {
                assertEquals(12, 12);
                assertEquals(12L, 12L);
                assertEquals(new Long(12), new Long(12));

                assertEquals("Size", 12, 13);
                assertEquals("Capacity", 12.0, 11.99, 0.0);
        }

        /**
         * Tests success
         */
        @Test(groups={"unittest"})
        @SpiraTestCase(testCaseId=6)
        public void testSuccess()
        {
                //Successful test
                assertEquals(12, 12);
        }
}
```

The overall class is marked with a new @SpiraTestConfiguration attribute that contains the following pieces of information needed to access the SpiraTest test repository:

- **URL** - The URL to the instance of SpiraTest being accessed. This needs to start with `http://` or `https://`
- **Login** - A valid username for the instance of SpiraTest.
- **Password** - A valid password for the instance of SpiraTest.
- **Project Id** - The ID of the project (this can be found on the project homepage in the “Project Overview” section)
- **Release Id** (Optional) - The ID of the release to associate the test run with. This can be found on the releases list page (click on the Planning > Releases tab). If you don’t want to specify a release, just use the value -1.

- **Test Set Id** (Optional) – The ID of the test set to associate the test run with. This can be found on the test set list page (click on the Testing > Test Sets tab). If you don't want to specify a test set, just use the value -1. If you choose a test set that is associated with a release, then you don't need to explicitly set a release id (i.e. just use -1). However if you do set a release value, it will override the value associated with the test set.

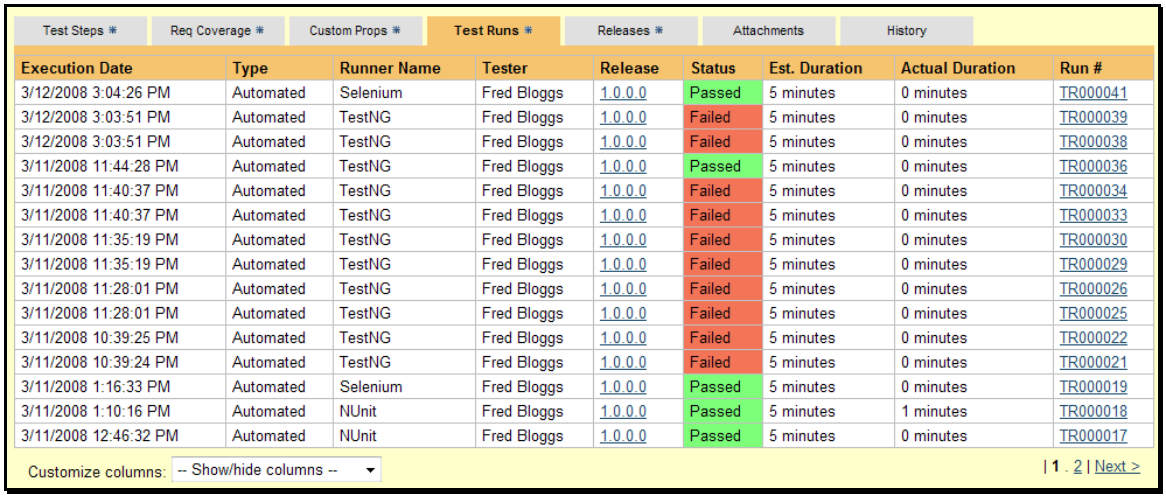
In addition, each of the individual test methods needs to be mapped to a specific test case within SpiraTest. This is done by adding a @SpiraTestCase attribute to the test method together with the ID of the corresponding test case in SpiraTest. The Test Case ID can be found on the test cases list page (click the "Test Cases" tab).

For these attributes to be available in your test fixture, you also need to add a reference to the *com.inflectra.spiratest.addons.testnglistener* package. This package is bundled within the supplied JAR-file library for Windows machines, and can be compiled from the provided source .java files on other platforms.

Now all you need to do is compile your code and then launch TestNG by executing the test fixture through the command line (or through your choice of IDE, e.g. Eclipse) with the SpiraTest listener option specified as a command argument. E.g. for our sample test, you would use the following command:

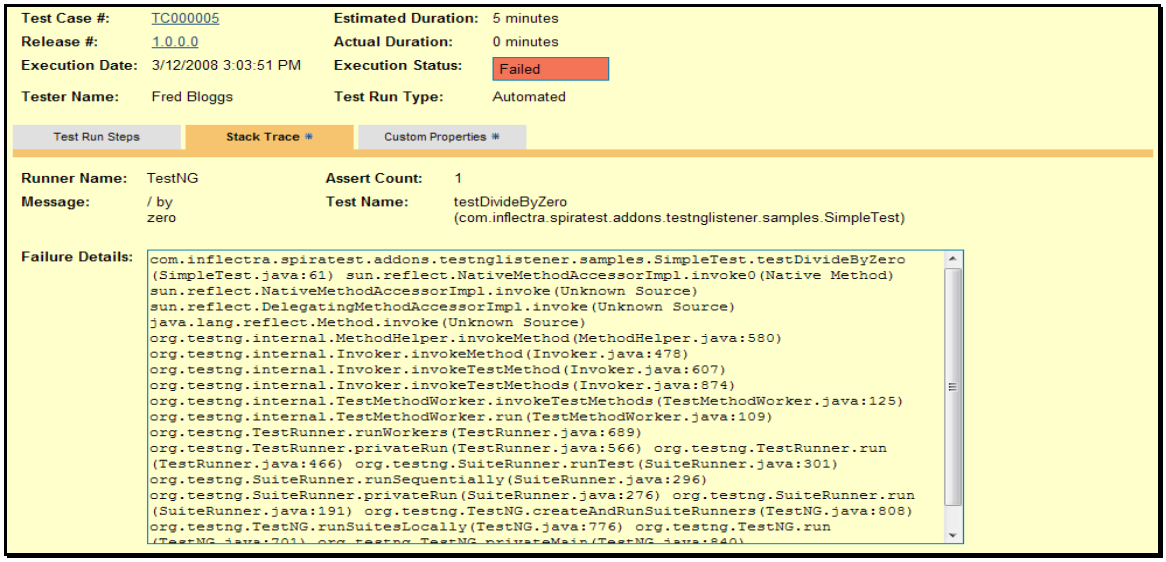
```
java
-classpath " C:\Program Files\Selenium-RC-1.0.0\selenium-remote-control-1.0.0\selenium-java-
client-driver-1.0.0\selenium-java-client-driver.jar;C:\Program
Files\SpiraTestListener\TestNGListener.jar;C:\Program Files\TestNG-5.7\testng-5.7\testng-5.7-
jdk15.jar" org.testng.TestNG
-listener com.inflectra.spiratest.addons.testnglistener.SpiraTestListener
com\inflectra\spiratest\addons\testnglistener\samples\unittests.xml
```

Once the test has run, you can view the test cases in SpiraTest, you should see a TestNG automated test run displayed in the list of executed test runs:



Execution Date	Type	Runner Name	Tester	Release	Status	Est. Duration	Actual Duration	Run #
3/12/2008 3:04:26 PM	Automated	Selenium	Fred Bloggs	1.0.0.0	Passed	5 minutes	0 minutes	TR000041
3/12/2008 3:03:51 PM	Automated	TestNG	Fred Bloggs	1.0.0.0	Failed	5 minutes	0 minutes	TR000039
3/12/2008 3:03:51 PM	Automated	TestNG	Fred Bloggs	1.0.0.0	Failed	5 minutes	0 minutes	TR000038
3/11/2008 11:44:28 PM	Automated	TestNG	Fred Bloggs	1.0.0.0	Passed	5 minutes	0 minutes	TR000036
3/11/2008 11:40:37 PM	Automated	TestNG	Fred Bloggs	1.0.0.0	Failed	5 minutes	0 minutes	TR000034
3/11/2008 11:40:37 PM	Automated	TestNG	Fred Bloggs	1.0.0.0	Failed	5 minutes	0 minutes	TR000033
3/11/2008 11:35:19 PM	Automated	TestNG	Fred Bloggs	1.0.0.0	Failed	5 minutes	0 minutes	TR000030
3/11/2008 11:35:19 PM	Automated	TestNG	Fred Bloggs	1.0.0.0	Failed	5 minutes	0 minutes	TR000029
3/11/2008 11:28:01 PM	Automated	TestNG	Fred Bloggs	1.0.0.0	Failed	5 minutes	0 minutes	TR000026
3/11/2008 11:28:01 PM	Automated	TestNG	Fred Bloggs	1.0.0.0	Failed	5 minutes	0 minutes	TR000025
3/11/2008 10:39:25 PM	Automated	TestNG	Fred Bloggs	1.0.0.0	Failed	5 minutes	0 minutes	TR000022
3/11/2008 10:39:24 PM	Automated	TestNG	Fred Bloggs	1.0.0.0	Failed	5 minutes	0 minutes	TR000021
3/11/2008 1:16:33 PM	Automated	Selenium	Fred Bloggs	1.0.0.0	Passed	5 minutes	0 minutes	TR000019
3/11/2008 1:10:16 PM	Automated	NUnit	Fred Bloggs	1.0.0.0	Passed	5 minutes	1 minutes	TR000018
3/11/2008 12:46:32 PM	Automated	NUnit	Fred Bloggs	1.0.0.0	Passed	5 minutes	0 minutes	TR000017

Clicking on one of the TestNG test runs will bring up a screen that provides information regarding what TestNG test method failed, what the error was, together with the associated code stack-trace:



Congratulations… You are now able to run TestNG automated tests and have the results be recorded within SpiraTest. The sample test fixture SimpleText.java is provided with the installation.

5. Integrating with Selenium-RC

Selenium Remote Control (RC) is a test tool that allows you to write automated web application user interface tests in any programming language against any HTTP website using any mainstream JavaScript-enabled browser[1]. Selenium RC comes in two parts.

- A server which can automatically launch and kill supported browsers, and acts as a HTTP proxy for web requests from those browsers.
- Client libraries for various computer languages - these are referred to as 'client drivers'.

Therefore to use SpiraTest with Selenium Remote Control (hereafter referred to as just Selenium), you need to decide which client driver you want to use, and then use the appropriate integration between SpiraTest and that driver's underlying platform/language. The drivers that are currently supported by SpiraTest are: .NET, Java and Python.

5.1. Using the .NET Driver

To use the .NET driver for Selenium with SpiraTest, you will need to run your Selenium tests within the context of a NUnit test fixture. The details of how to configure NUnit to operate with SpiraTest are described in section 2 above. Once you have configured NUnit for use with SpriaTest, there is one change that needs to be made to the SpiraTest NUnit configuration so that the Selenium tests report back to SpiraTest as 'Selenium' rather than "NUnit' so you can distinguish between them.

Supplied with the SpiraTest NUnit add-in is a sample test for using Selenium with SpiraTest:

```
using System;
using NUnit.Framework;
using Inflectra.SpiraTest.AddOns.SpiraTestNUnitAddIn.SpiraTestFramework;
using Selenium;

namespace SeleniumSampleTest
{
	/// <summary>
	/// Sample test fixture that tests the NUnit SpiraTest integration with the
	/// Selenium-RC .NET Driver
	/// </summary>
	[
	TestFixture,
	SpiraTestConfiguration("http://localhost/SpiraTest", "fredbloggs", "fredbloggs", 1, 1, 2,
SpiraTestConfigurationAttribute.RunnerName.Selenium)
	]
	public class GoogleTest
	{
		private static ISelenium selenium;

		[SetUp]
		public void SetupTest()
		{
			//Instantiate the selenium .NET proxy
			selenium = new DefaultSelenium("localhost", 4444, "*iexplore",
"http://www.google.com");
			selenium.Start();
		}

		[TearDown]
		public void TeardownTest()
		{
			selenium.Stop();
		}

		/// <summary>
```

[1] *Selenium RC Home Page*. OpenQA. 2008 <http://selenium-rc.openqa.org>

```
                /// Sample test that searches on Google, passes correctly
                /// </summary>
                [
                Test,
                SpiraTestCase (5)
                ]
                public void GoogleSearch()
                {
                        //Opens up Google
                        selenium.Open("http://www.google.com/webhp");

                        //Verifies that the title matches
                        Assert.AreEqual("Google", selenium.GetTitle());
                        selenium.Type("q", "Selenium OpenQA");

                        //Verifies that it can find the Selenium website
                        Assert.AreEqual("Selenium OpenQA", selenium.GetValue("q"));
                        selenium.Click("btnG");
                        selenium.WaitForPageToLoad("5000");
                        Assert.IsTrue(selenium.IsTextPresent("www.openqa.org"));
                        Assert.AreEqual("Selenium OpenQA - Google Search", selenium.GetTitle());
                }
        }
}
```

The details of the sample itself are described in more detail on the Selenium website, and you can see that we have added the SpiraTest specific attributes onto the test case and test methods to indicate that they need to report back to SpiraTest.

However there is one change that has been made to the *SpiraTestConfiguration* attribute applied to the test fixture – an extra **SpiraTestConfigurationAttribute.RunnerName.Selenium** parameter has been specified. This tells the SpiraTest NUnit add-in to report the results back as being generated by Selenium rather than NUnit.

5.2. Using the Java Driver

To use the Java driver for Selenium with SpiraTest, you will need to run your Selenium tests within the context of a TestNG test fixture. The details of how to configure TestNG to operate with SpiraTest are described in section 4 above. Once you have configured TestNG for use with SpriaTest, there is one change that needs to be made to the SpiraTest TestNG configuration so that the Selenium tests report back to SpiraTest as 'Selenium' rather than "TestNG' so you can distinguish between them.

Supplied with the SpiraTest TestNG listener is a sample test for using Selenium with SpiraTest:

The details of the sample itself are described in more detail on the Selenium website, and you can see that we have added the SpiraTest specific attributes onto the test case and test methods to indicate that they need to report back to SpiraTest.

```
package com.inflectra.spiratest.addons.testnglistener.samples;

import org.testng.annotations.*;
import static org.testng.AssertJUnit.*;
import com.thoughtworks.selenium.*;

import com.inflectra.spiratest.addons.testnglistener.*;

/**
 * A sample Selenium test using the ability to return results back to SpiraTest
 *
 * @author              Inflectra Corporation
 * @version             2.3.0
 *
 */
@SpiraTestConfiguration(
url="http://localhost/SpiraTest",
```

```
login="fredbloggs",
password="fredbloggs",
projectId=1,
releaseId=1,
testSetId=-1
runner=RunnerName.Selenium
)
@Test(groups={"seleniumtest"})
public class SeleniumTest
{
    private Selenium selenium;

        @BeforeClass
    public void setUp()
        {
                //Instantiate the selenium Java proxy
         String url = "http://www.google.com";
         selenium = new DefaultSelenium("localhost", 4444, "*firefox", url);
         selenium.start();
    }

        @AfterClass
    protected void tearDown()
        {
         selenium.stop();
    }

        // Sample test that searches on Google, passes correctly
        @Test(groups={"seleniumtest"})
        @SpiraTestCase(testCaseId=5)
    public void testGoogle()
        {
                //Opens up Google
         selenium.open("http://www.google.com/webhp?hl=en");

                //Verifies that the title matches
         assertEquals("Google", selenium.getTitle());
         selenium.type("q", "Selenium OpenQA");

                //Verifies that it can find the Selenium website
         assertEquals("Selenium OpenQA", selenium.getValue("q"));
         selenium.click("btnG");
         selenium.waitForPageToLoad("5000");
         assertEquals("Selenium OpenQA - Google Search", selenium.getTitle());
    }
}
```

However there is one change that has been made to the *SpiraTestConfiguration* attribute applied to the test fixture – an extra **runner=RunnerName.Selenium** parameter has been specified. This tells the SpiraTest TestNG listener to report the results back as being generated by Selenium rather than TestNG.

5.3. Using the Python Driver

To use the Python driver for Selenium with SpiraTest, you will need to run your Selenium tests within the context of a PyUnit unit-test fixture. The details of how to configure PyUnit to operate with SpiraTest are described in section 6 below. Once you have configured PyUnit for use with SpriaTest, there is one change that needs to be made to the SpiraTest PyUnit configuration so that the Selenium tests report back to SpiraTest as 'Selenium' rather than "PyUnit' so you can distinguish between them.

Supplied with the SpiraTest PyUnit extension is a sample test for using Selenium with SpiraTest:

```
from selenium import selenium
import unittest
import sys, time
import spiratestextension

#     A sample Selenium test using the ability to return results back to SpiraTest
```

```
#
#     Author         Inflectra Corporation
#     Version        2.3.0
#
class TestSeleniumSample(unittest.TestCase):

    seleniumHost = 'localhost'
    seleniumPort = str(4444)
    browserStartCommand = "*firefox"
    browserURL = "http://www.google.com"

    def setUp(self):
        print "Using selenium server at " + self.seleniumHost + ":" + self.seleniumPort
        self.selenium = selenium(self.seleniumHost, self.seleniumPort,
self.browserStartCommand, self.browserURL)
        self.selenium.start()

    def testGoogle__4(self):
        selenium = self.selenium
        selenium.open("http://www.google.com/webhp?hl=en")

        #Verifies that the title matches
        self.assertEqual ("Google", selenium.get_title())
        selenium.type("q", "Selenium OpenQA")

        #Verifies that it can find the Selenium website
        self.assertEqual("Selenium OpenQA", selenium.get_value("q"))
        selenium.click("btnG")
        selenium.wait_for_page_to_load("5000")
        self.assertEqual("Selenium OpenQA - Google Search", selenium.get_title())

    def tearDown(self):
        self.selenium.stop()

suite = unittest.TestLoader().loadTestsFromTestCase(TestSeleniumSample)
testResult = unittest.TextTestRunner(verbosity=2).run(suite)
releaseId = 1
testSetId = -1
spiraTestExtension = spiratestextension.SpiraTestExtension()
spiraTestExtension.projectId = 1
spiraTestExtension.server = "localhost"
spiraTestExtension.port = 80
spiraTestExtension.path = "SpiraTest"
spiraTestExtension.userName = "fredbloggs"
spiraTestExtension.password = "fredbloggs"
spiraTestExtension.recordResults(TestSeleniumSample, testResult, releaseId, testSetId,
"Selenium")
```

The details of the sample itself are described in more detail on the Selenium website, and you can see that we have added the SpiraTest specific test case suffixes and reporting code into the test methods to indicate that they need to report back to SpiraTest.

However there is one change that has been made to the *spiraTestExtension.recordResults* method called at the end of the test case. An extra string parameter has been specified that contains “Selenium”. This tells the SpiraTest PyUnit extension to report the results back as being generated by Selenium rather than PyUnit.

6. Integrating with PyUnit

6.1. Installing the PyUnit Extension

This section outlines how to install the SpiraTest Extension for PyUnit onto a workstation so that you can then run automated PyUnit tests against a Python application and have the results be recorded as test runs inside SpiraTest. It assumes that you already have a working installation of SpiraTest v2.3 or later, and a working Python development environment. If you have an earlier version of SpiraTest you will need to upgrade to at least v2.3 before trying to use this extension.

To obtain the version of the PyUnit extension that is compatible with your version of SpiraTest, you simply need to log-in as a project-level administrator to SpiraTest, go to the Administration home page and download the PyUnit Extension compressed archive (.zip). This process is described in the *SpiraTest Administration Guide* in more detail.

The PyUnit extension is provided as a set of Python source files that can be imported into your existing unit tests to add the SpiraTest reporting functionality. Once you have downloaded the Zip archive, you simply need to uncompress it into a folder of your choice on your local system (e.g. C:\Program Files\SpiraTest\PyUnit Extension)

Now to use the extension within your test cases, you need to first make sure that the folder is added to the Python *PYTHONPATH*. The method for doing this is dependent on the platform you're using, so please refer to the documentation on python.org for details on the appropriate method for your platform. As an example, on a Windows platform, the folder would be added to the PYTHONPATH by typing the following:

```
set PYTHONPATH=%PYTHONPATH%; C:\Program Files\SpiraTest\PyUnit Extension
```

Once you have completed this step, you are now ready to begin using your PyUnit test fixtures with SpiraTest.

6.2. Using PyUnit with SpiraTest

The typical code structure for a PyUnit test fixture coded in Python is as follows:

```
import random
import unittest

# sample PyUnit test case
class TestSequenceFunctions(unittest.TestCase):

      def setUp(self):
            self.seq = range(10)

      def testshuffle(self):
      # make sure the shuffled sequence does not lose any elements
            random.shuffle(self.seq)
            self.seq.sort()
            self.assertEqual(self.seq, range(10))

      def testchoice(self):
            element = random.choice(self.seq)
            self.assert_(element in self.seq)

      def testfail(self):
            self.assertEqual(1, 2, "1==2 Should fail")

      def testsample(self):
            self.assertRaises(ValueError, random.sample, self.seq, 20)
            for element in random.sample(self.seq, 5):
```

```
                self.assert_(element in self.seq)

suite = unittest.TestLoader().loadTestsFromTestCase(TestSequenceFunctions)
testResult = unittest.TextTestRunner(verbosity=2).run(suite)
```

The Python class is marked as a PyUnit test fixture by inheriting from the unittest.TestCase base class, and the individual test methods are identified by using the 'test' prefix, with special setUp() and tearDown() methods reserved for the respective purposes. When you open up the class in a PyUnit runner or execute from the command line it loads all the test classes and executes all the methods marked with 'test…' in turn.

Each of the Assert statements is used to test the state of the application after executing some sample code that calls the functionality being tested. If the condition in the assertion is true, then execution of the test continues, if it is false, then a failure is logged and PyUnit moves on to the next test method.

So, to use SpiraTest with PyUnit, each of the test cases written for execution by PyUnit needs to have a corresponding test case in SpiraTest. These can be either existing test cases that have manual test steps or they can be new test cases designed specifically for automated testing and therefore have no defined test steps. In either case, the changes that need to be made to the PyUnit test fixture for SpiraTest to record the PyUnit test run are illustrated below:

```
import random
import unittest
import spiratestextension

# sample PyUnit test case
class TestSequenceFunctions(unittest.TestCase):

      def setUp(self):
            self.seq = range(10)

      def testshuffle__2(self):
      # make sure the shuffled sequence does not lose any elements
            random.shuffle(self.seq)
            self.seq.sort()
            self.assertEqual(self.seq, range(10))

      def testchoice__3(self):
            element = random.choice(self.seq)
            self.assert_(element in self.seq)

      def testfail__4(self):
            self.assertEqual(1, 2, "1==2 Should fail")

      def testsample__5(self):
            self.assertRaises(ValueError, random.sample, self.seq, 20)
            for element in random.sample(self.seq, 5):
                self.assert_(element in self.seq)

suite = unittest.TestLoader().loadTestsFromTestCase(TestSequenceFunctions)
testResult = unittest.TextTestRunner(verbosity=2).run(suite)
releaseId = 1
testSetId = 1
spiraTestExtension = spiratestextension.SpiraTestExtension()
spiraTestExtension.projectId = 1
spiraTestExtension.server = "localhost"
spiraTestExtension.port = 80
spiraTestExtension.path = "SpiraTest"
spiraTestExtension.userName = "fredbloggs"
spiraTestExtension.password = "fredbloggs"
spiraTestExtension.recordResults(TestSequenceFunctions, testResult, releaseId, testSetId)
```

Firstly, each of the individual test methods is appended with two underscores followed by the ID of the corresponding test case in SpiraTest. So for example testshuffle() is now testshuffle__2() as it maps to test case TC00002 inside SpiraTest.

Second, at the end of the test run, the testResults object generated by the test run is passed to a special SpiraTestExtension() class via the recordResults() method. This class takes the results from the test run and uses it to generate the web-service messages that are sent to SpiraTest to communicate the test results.

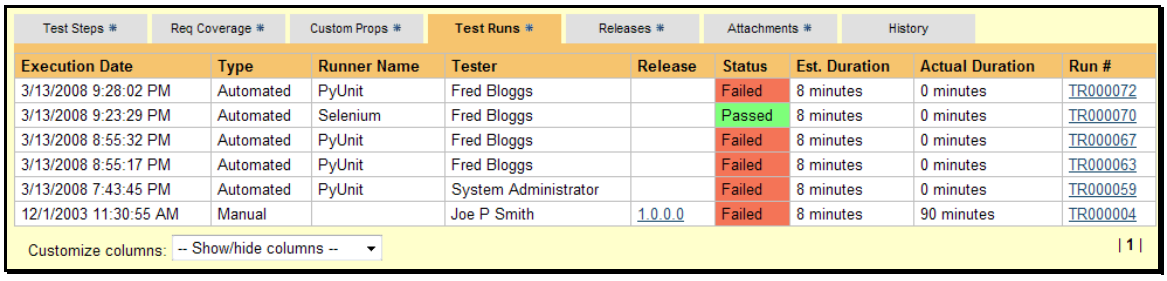
The following attributes need to be set on the instance of the SpiraTestExtension() object so that the extension can access the SpiraTest repository:

- **spiraTestExtension.projectId** – The ID of the project inside SpiraTest (this can be found on the project homepage in the "Project Overview" section)
- **spiraTestExtension.server** - The name of the web server that SpiraTest is installed on
- **spiraTestExtension.port** – The port used to access SpiraTest over the network (typically 80 unless you have a custom port setup)
- **spiraTestExtension.path** – The path to SpiraTest on your webserver (typically just 'SpiraTest')
- **spiraTestExtension.userName** - A valid username for the instance of SpiraTest that has access to the project specified above
- **spiraTestExtension.password** - A valid password for the user specified above

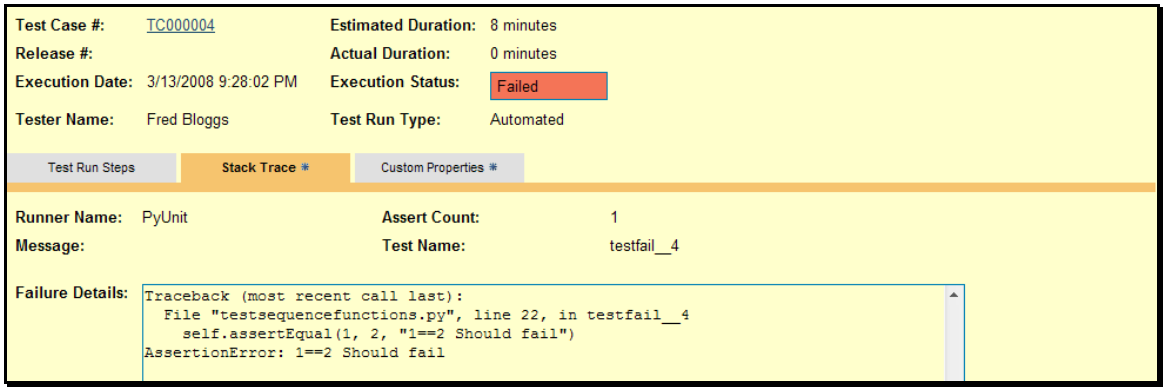
In addition, when calling the `recordResults()` method, you should also pass the Release ID and the Test Set ID which is used to tell SpiraTest which release and/or test set to associate the test execution with.

The Release ID can be found on the releases list page (click on the Planning > Releases tab) – just remove the RL prefix from the number as well as any leading zeros. Similarly, the Test Set ID can be found on the test set list page (click on the Testing > Test Sets tab) – just remove the TX prefix from the number as well as any leading zeros. If you don't want to associate the test run with a specific release or test set, just use the special value -1 to indicate N/A.

Now all you need to do is save your code, launch PyUnit, run the test fixtures as you would normally do, and when you view the test cases in SpiraTest, you should see a PyUnit automated test run displayed in the list of executed test runs:



Clicking on one of the PyUnit test runs will bring up a screen that provides information regarding what PyUnit test method failed, what the error was, together with the associated code stack-trace:



Congratulations… You are now able to run PyUnit automated tests and have the results be recorded within SpiraTest. The sample test fixture testsequencefunctions.py is provided with the installation.

7. Integrating with MS-Test

This section describes how to use SpiraTest in conjunction with the unit testing features provided by Microsoft Visual Studio Team System (MS VSTS) Test – commonly known as MS-Test.

7.1. Installing the MS-Test Extension

This section outlines how to install the SpiraTest extension for Microsoft Visual Studio Team System Test (MS-Test) onto a workstation so that you can then run automated MS-Test tests against a .NET application and have the results be recorded as test runs inside SpiraTest. It assumes that you already have a working installation of SpiraTest v2.3 or later. If you have an earlier version of SpiraTest you will need to upgrade to at least v2.3 before trying to use this add-in. You will also need to have either Visual Studio Team System 2005 or later or Visual Studio 2008 Professional or later, since earlier versions do not come with the test automation features.

To obtain the latest version of the SpiraTest extension, you can either access the administration section of SpiraTest, and click on the Add-Ins & Downloads link or simply visit the Inflectra corporate downloads webpage (http://www.inflectra.com/Products/Downloads.aspx) and then download the compressed archive (.zip) that contains the extension and associated sample files.

The MS-Test extension is provided as a compressed zipfile that includes both the binaries (packaged as a .NET dll assembly) and the source code (stored in a Visual Studio project folder structure). The rest of this section will assume that you are using the pre-compiled assembly.

Once you have downloaded the Zip archive, you need to uncompress it into a folder of your choice on your local system. Assuming that you uncompressed it to the `C:\Program Files\SpiraTest\MSTest Extension` folder, you should have the following folder structure created:

```
C:\Program Files\SpiraTest\MSTest Extension
C:\Program Files\SpiraTest\MSTest Extension\SampleMSTest
C:\Program Files\SpiraTest\MSTest Extension\SampleMSTest\Properties
C:\Program Files\SpiraTest\MSTest Extension\SpiraTestMSExtension
C:\Program Files\SpiraTest\MSTest Extension\SpiraTestMSExtension\Properties
C:\Program Files\SpiraTest\MSTest Extension\SpiraTestMSExtension\Web References
```

The pre-built assembly SpiraTestMSTestExtension.dll is located in the root folder, the source-code for the extension can be found in the “SpiraTestMSExtension” subfolder, and the sample test fixture can be found in the “SampleMSTest” subfolder.

Now to use the extension within your test cases, you need to first make sure that the SpiraTestMSTestExtension.dll assembly is added to the project references. Once you have completed this step, you are now ready to begin using your MS-Test test fixtures with SpiraTest.

7.2. Using MS-Test with SpiraTest

The typical code structure for a Visual Studio Team System Test (MS-Test) test fixture coded in C# is as follows:

```
using System;
using System.Threading;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Inflectra.SpiraTest.AddOns.SpiraTestMSTestExtension.SampleMSTest
{
    /// <summary>
    /// Sample test fixture that tests the SpiraTest integration
    /// Written by Paul Tissue. Packed by Inflectra Corporation
    /// </summary>
    [
```

```
    TestClass
    ]
    public class SpiraTestCaseAttributeTest
    {
        /// <summary>
        /// Test fixture state
        /// </summary>
        protected static int testFixtureState = 1;

        /// <summary>
        /// Constructor method
        /// </summary>
        public SpiraTestCaseAttributeTest()
        {
            //Delegates to base

            //Set the state to 2
            testFixtureState = 2;
        }

        /// <summary>
        /// Sample test that asserts a failure and overrides the default configuration
        /// </summary>
        [
        TestMethod
        ]
        public void SampleFail()
        {
            //Verify the state
            Assert.AreEqual(2, testFixtureState, "*Real Error*: State not persisted");

            //Failure Assertion
            Assert.AreEqual(1, 0, "Failed as Expected");
        }

        /// <summary>
        /// Sample test that succeeds - uses the default configuration
        /// </summary>
        [
        TestMethod
        ]
        public void SamplePass()
        {
            //Verify the state
            Assert.AreEqual(2, testFixtureState, "*Real Error*: State not persisted");

            //Successful assertion
            Assert.AreEqual(1, 1, "Passed as Expected");
        }

        /// <summary>
        /// Sample test that does not log to SpiraTest
        /// </summary>
        [
        TestMethod,
        ExpectedException(typeof(AssertFailedException))
        ]
        public void SampleIgnore()
        {
            //Verify the state
            Assert.AreEqual(2, testFixtureState, "*Real Error*: State not persisted");

            //Failure Assertion
            Assert.AreEqual(1, 0, "Failed as Expected");
        }
    }
}
```

The .NET class is marked as a MS-Test unit test fixture by applying the [TestClass] attribute to the class as a whole, and the [TestMethod] attribute to each of the test assertion methods individually – shown above. When you open up the class in Visual Studio and click Tests > Run > Run All Tests in Solution it

loads all the test classes marked with [TestClass] and executes all the methods marked with [TestMethod] in turn.

Each of the Assert statements is used to test the state of the application after executing some sample code that calls the functionality being tested. If the condition in the assertion is true, then execution of the test continues, if it is false, then a failure is logged and MS-Test moves on to the next test method.

So, to use SpiraTest with MS-Test, each of the test cases written for execution by MS-Test needs to have a corresponding test case in SpiraTest. These can be either existing test cases that have manual test steps or they can be new test cases designed specifically for automated testing and therefore have no defined test steps. In either case, the changes that need to be made to the MS-Test test fixture for SpiraTest to record the MS-Test test run are illustrated below:

```
using System;
using System.Threading;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Inflectra.SpiraTest.AddOns.SpiraTestMSTestExtension.SampleMSTest
{
    /// <summary>
    /// Sample test fixture that tests the SpiraTest integration
    /// Written by Paul Tissue. Packed by Inflectra Corporation
    /// </summary>
    [
    TestClass
    ]
    public class SpiraTestCaseAttributeTest : MSTestExtensionsTestFixture
    {
        /// <summary>
        /// Test fixture state
        /// </summary>
        protected static int testFixtureState = 1;

        /// <summary>
        /// Constructor method
        /// </summary>
        public SpiraTestCaseAttributeTest()
        {
            //Delegates to base

            //Set the state to 2
            testFixtureState = 2;
        }

        /// <summary>
        /// Sample test that asserts a failure and overrides the default configuration
        /// </summary>
        [
        TestMethod,
        SpiraTestCase(5),
        SpiraTestConfiguration("http://localhost/SpiraTest", "fredbloggs", "fredbloggs", 1, 1,
2)
        ]
        public void SampleFail()
        {
            //Verify the state
            Assert.AreEqual(2, testFixtureState, "*Real Error*: State not persisted");

            //Failure Assertion
            Assert.AreEqual(1, 0, "Failed as Expected");
        }

        /// <summary>
        /// Sample test that succeeds - uses the default configuration
        /// </summary>
        [
        TestMethod,
```

```
        SpiraTestCase(6)
        ]
        public void SamplePass()
        {
            //Verify the state
            Assert.AreEqual(2, testFixtureState, "*Real Error*: State not persisted");

            //Successful assertion
            Assert.AreEqual(1, 1, "Passed as Expected");
        }

        /// <summary>
        /// Sample test that does not log to SpiraTest
        /// </summary>
        [
        TestMethod,
        ExpectedException(typeof(AssertFailedException))
        ]
        public void SampleIgnore()
        {
            //Verify the state
            Assert.AreEqual(2, testFixtureState, "*Real Error*: State not persisted");

            //Failure Assertion
            Assert.AreEqual(1, 0, "Failed as Expected");
        }

    }
}
```

And the following settings need to be added to the .config file associated with the test fixture assembly:

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <sectionGroup name="applicationSettings" type="System.Configuration.ApplicationSettingsGroup,
System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
      <section name="Inflectra.SpiraTest.AddOns.SpiraTestMSTestExtension.Properties.Settings"
type="System.Configuration.ClientSettingsSection, System, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
    </sectionGroup>
  </configSections>
  <applicationSettings>
    <Inflectra.SpiraTest.AddOns.SpiraTestMSTestExtension.Properties.Settings>
      <setting name="Url" serializeAs="String">
        <value>http://localhost/SpiraTest</value>
      </setting>
      <setting name="Login" serializeAs="String">
        <value>fredbloggs</value>
      </setting>
      <setting name="Password" serializeAs="String">
        <value>fredbloggs</value>
      </setting>
      <setting name="ProjectId" serializeAs="String">
        <value>1</value>
      </setting>
      <setting name="ReleaseId" serializeAs="String">
        <value>1</value>
      </setting>
      <setting name="TestSetId" serializeAs="String">
        <value>1</value>
      </setting>
      <setting name="RecordTestRun" serializeAs="String">
        <value>True</value>
      </setting>
    </Inflectra.SpiraTest.AddOns.SpiraTestMSTestExtension.Properties.Settings>
  </applicationSettings>
```

Firstly the settings in the .config file indicate the following information to the test fixture:

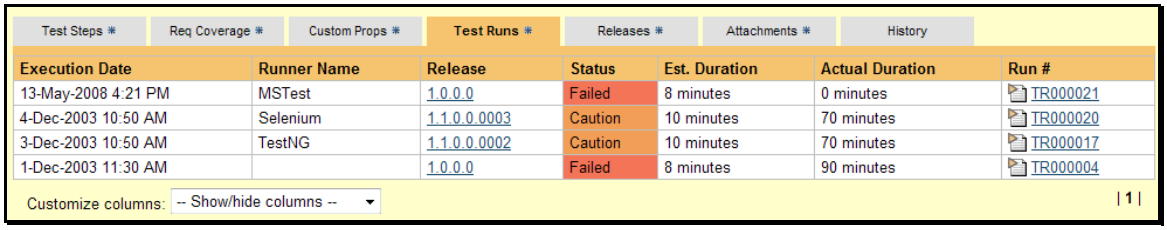
- The URL to the instance of SpiraTest being accessed. This needs to start with `http://` or `https://`.
- A valid username and password for the instance of SpiraTest.
- The ID of the project (this can be found on the project homepage in the “Project Overview” section)
- (Optional) The ID of the release to associate the test-run with. This can be found on the releases list page (click on the Planning > Releases tab)
- (Optional) The ID of the test set to associate the test-run with. This can be found on the test set list page (click on the Testing > Test Sets tab)
- A flag that tells MS-Test whether or not to record the results in SpiraTest.

Next, the test fixture class needs to be derived from the *MSTestExtensionsTestFixture* base class so that the test runner knows that the test class will be reporting back to SpiraTest.

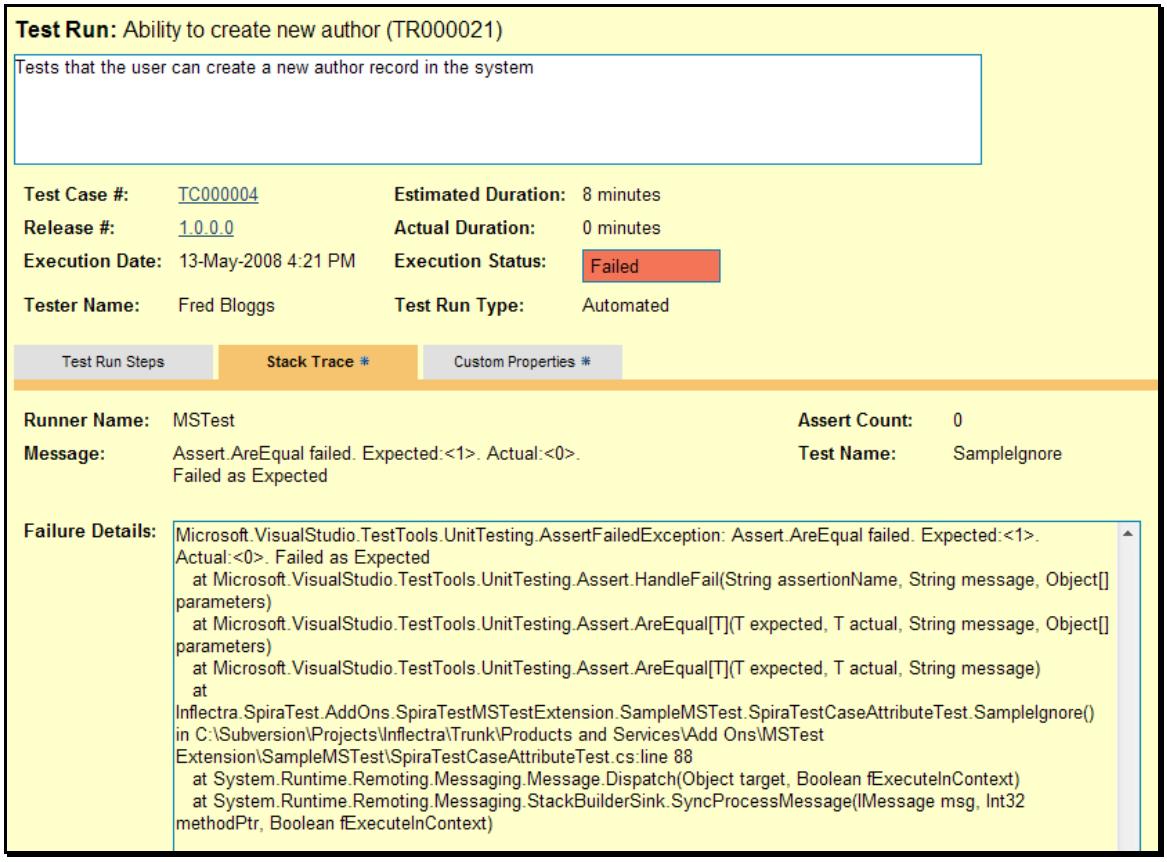
In addition, each of the individual test methods needs to be mapped to a specific test case within SpiraTest. This is done by adding a [SpiraTestCase] attribute to the test method together with the ID of the corresponding test case in SpiraTest. The Test Case ID can be found on the test cases list page (click the “Test Cases” tab).

In addition you can also override the default SpiraTest configuration settings from the .config file by adding the [SpiraTestConfiguration] attribute directly to the test method and specifying the SpiraTest URL, authentication information, project id, release id (optional) and test set id (optional). An example of this is shown above for the SampleFail() method.

Now all you need to do is compile your code, launch Visual Studio, run the test fixtures as you would normally do, and when you view the test cases in SpiraTest, you should see a MSTest automated test run displayed in the list of executed test runs:



Clicking on one of the MSTest test runs will bring up a screen that provides information regarding what MSTest test method failed, what the error was, together with the associated code stack-trace:



Congratulations. You are now able to run MSTest automated tests and have the results be recorded within SpiraTest. The sample test project SampleMSTest is provided with the installation.

8. Integrating with Ruby Test::Unit

8.1. Installing the Ruby Test::Unit Test Runner

This section outlines how to install the SpiraTest custom Test Runner for Test::Unit onto a workstation so that you can then run automated Test::Unit tests against a Ruby application and have the results be recorded as test runs inside SpiraTest. It assumes that you already have a working installation of SpiraTest v2.3 or later, and a working Ruby development environment. If you have an earlier version of SpiraTest you will need to upgrade to at least v2.3 before trying to use this extension.

To obtain the version of the Test::Unit test runner that is compatible with your version of SpiraTest, you simply need to log-in as a project-level administrator to SpiraTest, go to the Administration home page and download the Test::Unit test runner compressed archive (.zip). This process is described in the *SpiraTest Administration Guide* in more detail.

The Test::Unit test runner is provided as a set of Ruby source files that can be imported into your existing unit tests to add the SpiraTest reporting functionality. Once you have downloaded the Zip archive, you simply need to uncompress it into a folder of your choice on your local system (e.g. C:\Program Files\SpiraTest\RubyTestUnitRunner)

Now to use the custom test runner within your test cases, you need to first make sure that the folder is added to the Ruby *RUBYPATH (or just the system PATH)*. The method for doing this is dependent on the platform you're using, so please refer to the documentation on http://ruby-lang.org for details on the appropriate method for your platform. As an example, on a Windows platform, the folder would be added to the RUBYPATH by typing the following:

```
set RUBYPATH=%RUBYPATH%; C:\Program Files\SpiraTest\RubyTestUnitRunner
```

Once you have completed this step, you are now ready to begin using your Test::Unit test fixtures with SpiraTest.

8.2. Using Ruby Test::Unit with SpiraTest

The typical code structure for a Test::Unit test suite and test case coded in Ruby is as follows:

```
#this is a test case that tests addition operations
class TC_Adder < Test::Unit::TestCase
  def setup
    @adder = Adder.new(5)
  end
  def test_add
    assert_equal(7, @adder.add(2), "Should have added correctly")
  end
  def test_addfail
    assert_equal(7, @adder.add(3), "Test failure")
  end
  def teardown
    @adder = nil
  end
end

#this is a test suite that calls the test case
class TS_Examples
       def self.suite
              suite = Test::Unit::TestSuite.new
              suite << TC_Adder.suite
              return suite
       end
end
Test::Unit::UI::Console::TestRunner.run(TS_Examples)
```

The Test::Unit test case is marked as a Test::Unit test case by inheriting from the `Test::Unit::TestCase` base class, and the individual test methods are identified by using the 'test' prefix, with special setup and teardown methods reserved for the respective purposes. When you open up the class in a Ruby Test::Unit runner or execute from the command line it loads all the test classes and executes all the methods marked with 'test…' in turn.

Each of the Assert statements is used to test the state of the application after executing some sample code that calls the functionality being tested. If the condition in the assertion is true, then execution of the test continues, if it is false, then a failure is logged and Test::Unit moves on to the next test method.

So, to use SpiraTest with Test::Unit, each of the test cases written for execution by Test::Unit needs to have a corresponding test case in SpiraTest. These can be either existing test cases that have manual test steps or they can be new test cases designed specifically for automated testing and therefore have no defined test steps. In either case, the changes that need to be made to the Test::Unit test case and test suite for SpiraTest to record the Test::Unit test run are illustrated below:

```
#this is a test case that tests addition operations
class TC_Adder < Test::Unit::TestCase
  def setup
    @adder = Adder.new(5)
  end
  def test_add__2
    assert_equal(7, @adder.add(2), "Should have added correctly")
  end
  def test_addfail__3
    assert_equal(7, @adder.add(3), "Test failure")
  end
  def teardown
    @adder = nil
  end
end

#this is a test suite that calls the test case
class TS_Examples
       def self.suite
              suite = Test::Unit::TestSuite.new
              suite << TC_Adder.suite
              return suite
       end
end
projectId = 1
releaseId = 2
testSetId = -1
testRunner = Test::Unit::SpiraTest::TestRunner.new(TS_Examples,
"http://servername/SpiraTest", "fredbloggs", "fredbloggs", projectId, releaseId,
testSetId)

testRunner.start
```

Firstly, each of the individual test methods is appended with two underscores followed by the ID of the corresponding test case in SpiraTest. So for example `test_add` is now `test_add__2` as it maps to test case TC00002 inside SpiraTest.

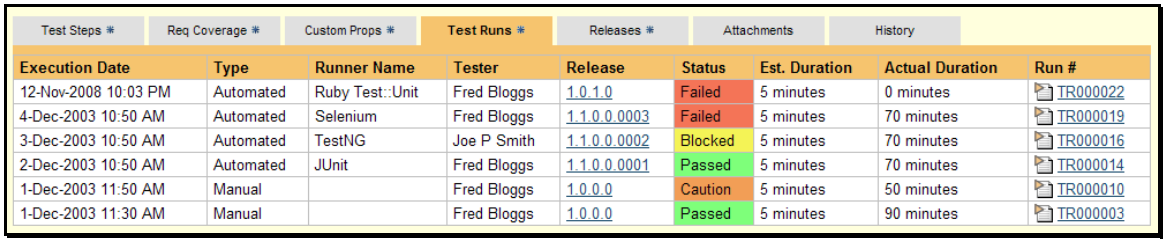
Second, at the end of the test suite, instead of just creating the standard Console Test Runner class and passing it a reference to the test suite (e.g. TS_Examples), we now create an instance of the special `Test::Unit::SpiraTest::TestRunner` class, passing it a reference to the test suite as well as specifying the SpiraTest connection information.

This class takes the results from the test suite being executed and uses it to generate the web-service messages that are sent to SpiraTest to communicate the test results.

The following parameters need to be passed during the instantiation of the `Test::Unit::SpiraTest::TestRunner` object so that the custom test runner can access the SpiraTest repository:

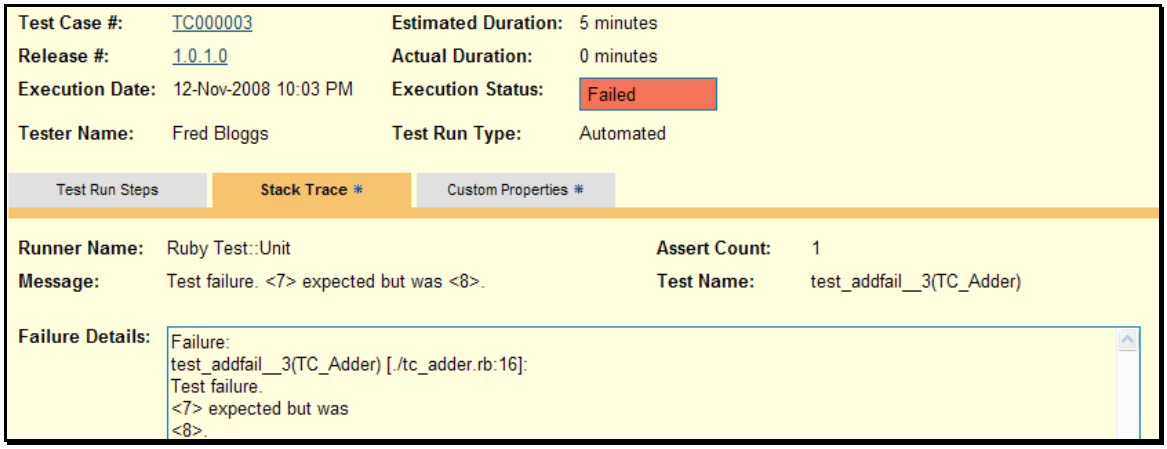
- **suite** – the reference to the Test::Unit test suite that contains the test cases being executed. In our example above, this is the TS_Examples class.
- **baseUrl**– The base URL used to access your instance of SpiraTest (e.g. http://myserver/SpiraTest). It should include the protocol (e.g. http/https), the server-name, the port number (if not 80/443) and the virtual directory (if there is one).
- **userName** - A valid username for the instance of SpiraTest that has access to the project specified above
- **password** - A valid password for the user specified above
- **projectId** - The ID of the project inside SpiraTest (this can be found on the project homepage in the "Project Overview" section)
- **releaseId** - The ID of the SpiraTest release to associate the test run with. This can be found on the releases list page (click on the Planning > Releases tab). If you don't want to associate the test run with a specific release, just use the value -1 to indicate N/A.
- **testSetId** - The ID of the SpiraTest test set to associate the test run with. This can be found on the test set list page (click on the Testing > Test Sets tab). If you don't want to associate the test run with a specific test set, just use the value -1 to indicate N/A.

Now all you need to do is save your code, launch Test::Unit, run the test fixtures as you would normally do (e.g. by executing the TS_Examples ruby file from the command line), and when you view the test cases in SpiraTest, you should see a Ruby Test::Unit automated test run displayed in the list of executed test runs:



Execution Date	Type	Runner Name	Tester	Release	Status	Est. Duration	Actual Duration	Run #
12-Nov-2008 10:03 PM	Automated	Ruby Test::Unit	Fred Bloggs	1.0.1.0	Failed	5 minutes	0 minutes	TR000022
4-Dec-2003 10:50 AM	Automated	Selenium	Fred Bloggs	1.1.0.0.0003	Failed	5 minutes	70 minutes	TR000019
3-Dec-2003 10:50 AM	Automated	TestNG	Joe P Smith	1.1.0.0.0002	Blocked	5 minutes	70 minutes	TR000016
2-Dec-2003 10:50 AM	Automated	JUnit	Fred Bloggs	1.1.0.0.0001	Passed	5 minutes	70 minutes	TR000014
1-Dec-2003 11:50 AM	Manual		Fred Bloggs	1.0.0.0	Caution	5 minutes	50 minutes	TR000010
1-Dec-2003 11:30 AM	Manual		Fred Bloggs	1.0.0.0	Passed	5 minutes	90 minutes	TR000003

Clicking on one of the Ruby Test::Unit test runs will bring up a screen that provides information regarding what Ruby Test::Unit test method failed, what the error was, together with the associated code stack-trace:



Congratulations… You are now able to run Test::Unit automated tests and have the results be recorded within SpiraTest. The sample test suite ts_examples.rb together with two test cases (tc_adder and tc_subtracter) is provided with the installation.

9. Integrating with Perl TAP

9.1. Installing the Perl TAP Extension

This section outlines how to install the SpiraTest extensions for Perl's Test Anything Protocol (TAP) so that you can then run automated Perl TAP unit tests against a Perl application and have the results be recorded as test runs inside SpiraTest. It assumes that you already have a working installation of SpiraTest v2.3 or later, and a working Perl development environment. If you have an earlier version of SpiraTest you will need to upgrade to at least v2.3 before trying to use this extension.

To obtain the latest version of the TAP extension you simply need to go to http://www.inflectra.com/SpiraTest/Downloads.aspx page and download the Perl TAP Extension compressed archive (.zip). This process is described in the *SpiraTest Administration Guide* in more detail.

The TAP extension is provided as a set of Perl library files (.pm) that can be imported into your existing TAP test harnesses to add the SpiraTest reporting functionality. Once you have downloaded the Zip archive, you simply need to uncompress it and copy the **Inflectra folder** (and subfolders) into the standard Perl library location (e.g. `C:\Perl\lib` on Windows). The sample files (the ones ending in .pl) that are not located in a folder can be put into a folder of your choice.

Once you have completed this step, you are now ready to begin running one of the provided samples or use your existing TAP unit tests with SpiraTest.

9.2. Using Perl TAP Extension with SpiraTest

The typical code structure for a Perl TAP test harness is as follows:

a) The sample test harness - SampleHarness.pl

```
#this is a test case that tests addition operations
#!/usr/bin/perl -w
use TAP::Harness;

#instantiate the harness
my $harness = TAP::Harness ->new;

#define the list of tests to be executed
my @tests = ("SampleTest1.pl", "SampleTest2.pl");

$harness->runtests(@tests);
```

b) One of the sample test fixtures – Sample1Test.pl

```
#!/usr/bin/perl -w

# Specify our plan, how many tests we're writing
 use Test::More tests => 9;

 # or alternately, if we don't know how many:
 # use Test::More qw(no_plan);

 # Check that our module compiles and can be "use"d.
 BEGIN { use_ok( 'Inflectra::SpiraTest::Addons::Samples::TestMe' ); }

 # Check our module can be required. Very similar test to that above.
 require_ok( 'Inflectra::SpiraTest::Addons::Samples::TestMe' );

 # There are a number of ways to generate the "ok" tests. These are:
 # ok: first argument is true, second argument is name of test.
 # is: first argument equals (eq) second argument, third argument is name of test.
```

```
 # isnt: first argument does not equal (ne) the second, third is name of test
 # like: first argument matches regexp in second, third is name of test
 # unlike: first argument does not match regexp, third is name of test
 # cmp_ok: compares first and third argument with comparison in second. Forth is test
name.

 # Here are some examples that should PASS
 ok( add(1,1) == 2, "Basic addition is working");

 is ( subtract(2,1), 1, "Basic subtraction is working");
 isnt( multiply(2,2), 5, "Basic multiplication doesn't fail");

# Here are some examples that should FAIL
 ok( add(1,1) == 3, "Basic addition is working");

 is ( subtract(2,1), 0, "Basic subtraction is working");
 isnt( multiply(2,2), 4, "Basic multiplication doesn't fail");

# Here is an example of a test that throws an ERROR
is($notdeclared, 1, "Undeclared variable test");
```

The TAP test cases in the sample code use the Test::More library which provides the necessary assertion methods for testing results from the code under test. The tests are themselves executed by adding their filenames to an array passed to the default TAP::Harness class. To run the test cases, you just need to execute the SampleHarness.pl file from the command line, and the test output will be echoed onto the screen.

Now, to use SpiraTest with TAR, each of the TAP test case files (e.g. SampleTest1.pl, SampleTest2.pl in our example) needs to have a corresponding test case in SpiraTest. These can be either existing test cases that have manual test steps or they can be new test cases designed specifically for automated testing and therefore have no defined test steps. In either case, *no changes need to be made to the individual test cases*, but the following changes need to be made to the test harness (illustrated in yellow below):

```
#this is a test case that tests addition operations
#!/usr/bin/perl -w
use Inflectra::SpiraTest::Addons::SpiraHarness::Harness;

#instantiate the harness
my $harness = Inflectra::SpiraTest::Addons::SpiraHarness::Harness->new;
#specify the spiratest custom harness properties
$spira_args = {};
$spira_args->{"base_url"} = "http://localhost/SpiraTest";
$spira_args->{"user_name"} = "fredbloggs";
$spira_args->{"password"} = "fredbloggs";
$spira_args->{"project_id"} = 1;
$spira_args->{"release_id"} = 1;
$spira_args->{"test_set_id"} = 1;
$harness->{"spira_args"} = $spira_args;

#define the list of tests and their SpiraTest Mapping
#Hash is of the format: TestFile => Test Case ID
my $tests = {};
$tests->{"SampleTest1.pl"} = 2;
$tests->{"SampleTest2.pl"} = 3;

$harness->runtests($tests);
```

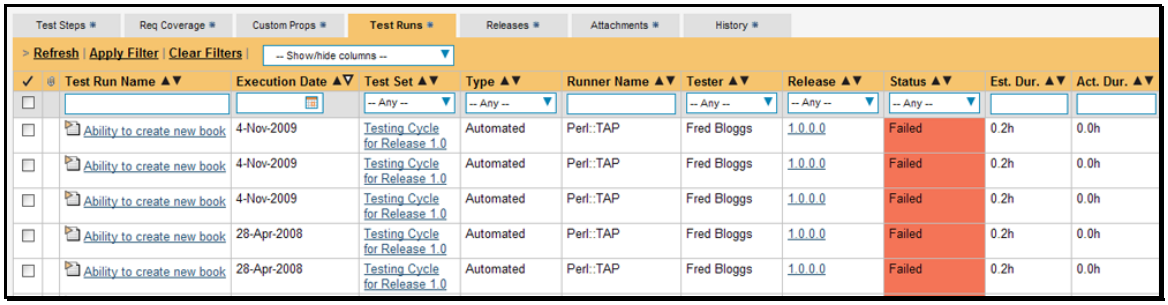
Firstly you need to use the SpiraTest specific harness rather than the general TAP::Harness library. This new class is actually a subclass of the standard one, so it supports all the same methods, with the exception of the runtests command, which now accepts a Perl hashref rather than a simple array.

Also you need to create and pass a hashref of arguments to the test harness (the spira_args property on the instantiated harness class) so that it knows how to access the SpiraTest server during test execution:

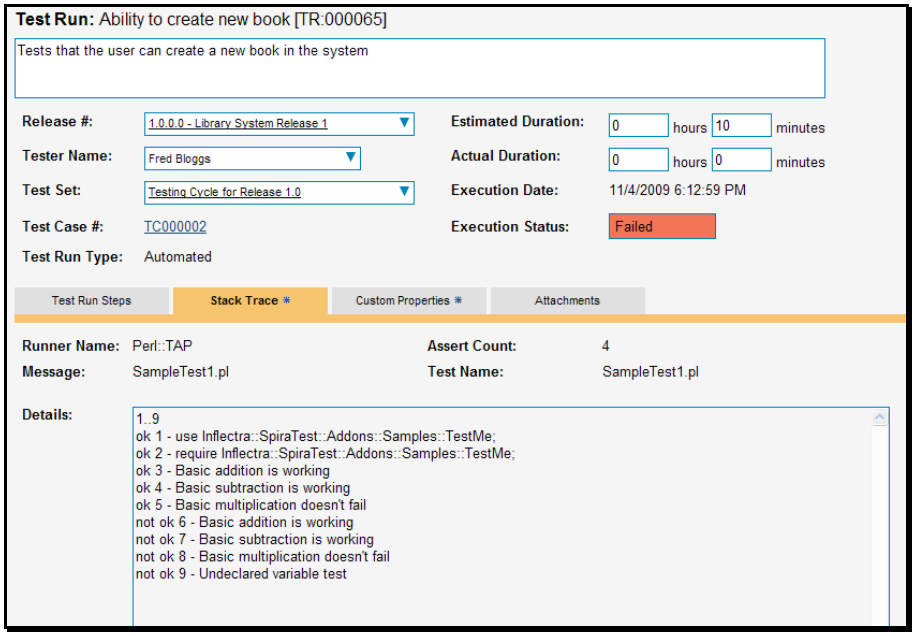
- **base_url**– The base URL used to access your instance of SpiraTest (e.g. http://myserver/SpiraTest). It should include the protocol (e.g. http/https), the server-name, the port number (if not 80/443) and the virtual directory (if there is one).
- **user_name** - A valid username for the instance of SpiraTest that has access to the project specified above
- **password** - A valid password for the user specified above
- **project_id** - The ID of the project inside SpiraTest (this can be found on the project homepage in the "Project Overview" section)
- **release_id** - The ID of the SpiraTest release to associate the test run with. This can be found on the releases list page (click on the Planning > Releases tab). If you don't want to associate the test run with a specific release, just comment out the line.
- **test_set_id** - The ID of the SpiraTest test set to associate the test run with. This can be found on the test set list page (click on the Testing > Test Sets tab). If you don't want to associate the test run with a specific test set, just comment out the line.

Finally instead of passing a simple array of the test case files to be executed, you instead need to create a Perl hashref and pass that to the runtests(…) method. The hashref needs to contain a list of the various test case files and their associated SpiraTest Test Case ID with the TC prefix removed (e.g. test case TC00005 would be just 5).

Now all you need to do is save your code, run the test fixtures as you would normally do (e.g. by executing from the command line), and when you view the test cases in SpiraTest, you should see a Perl::TAP automated test run displayed in the list of executed test runs:



Clicking on one of the Perl::TAP test runs will bring up a screen that provides information regarding what Perl::TAP test method failed, what the error was, together with the associated code stack-trace:



Congratulations… You are now able to run Perl TAP unit tests and have the results be recorded within SpiraTest. The sample test suite `SampleHarness.pl` together with its two test cases (`SampleTest1.pl` and `SampleTest2.pl`) is provided with the installation.

10. Integrating with PHPUnit

10.1. Installing the PHPUnit Extension

This section outlines how to install the SpiraTest Extension for PHPUnit onto a workstation so that you can then run automated PHPUnit tests against a PHP application and have the results be recorded as test runs inside SpiraTest. It assumes that you already have a working installation of SpiraTest v2.3 or later, and a working PHP development environment. If you have an earlier version of SpiraTest you will need to upgrade to at least v2.3 before trying to use this extension.

To obtain the latest version of the SpiraTest PHPUnit extension you simply need to go to http://www.inflectra.com/SpiraTest/Downloads.aspx page and download the PHPUnit Extension compressed archive (.zip). This process is described in the *SpiraTest Administration Guide* in more detail.

The PHPUnit extension is provided as a set of PHP source files that can be imported into your existing unit tests to add the SpiraTest reporting functionality. Once you have downloaded the Zip archive, you simply need to uncompress it into a folder of your choice on your local system (e.g. C:\Program Files\SpiraTest\PHPUnit Extension)

Now to use the extension within your test cases, you need to first make sure that the folder is added to the PHP *include_path*. The method for doing this is dependent on the platform you're using, so please refer to the documentation on php.org for details on the appropriate method for your platform. Alternatively you can copy the PHPUnit extension files to the root of the PHP installation and then just include the extension files using the root folder syntax.

Once you have completed these steps, you are now ready to begin using your PHPUnit test fixtures with SpiraTest.

10.2. Using PHPUnit with SpiraTest

The typical code structure for a PHPUnit test suite and test case coded in PHP is as follows:

a) Sample Test Suite

```
<?php
/**
 *
 * @author           Inflectra Corporation
 * @version          2.3.0
 *
 */

 require_once 'PHPUnit/Framework.php';
 require_once 'PHPUnit/TextUI/ResultPrinter.php';
 require_once './SimpleTest.php';

 // Create a test suite that contains the tests
 // from the ArrayTest class
  $suite = new PHPUnit_Framework_TestSuite('SimpleTest');

 // Create a test result and attach the default console text listener
 $result = new PHPUnit_Framework_TestResult;
 $textPrinter = new PHPUnit_TextUI_ResultPrinter;
 $result->addListener($textPrinter);

 // Run the tests and print the results
 $result = $suite->run($result);
 $textPrinter->printResult($result);

 ?>
```

b) Sample Test Case

```
<?php
require_once 'PHPUnit/Framework/TestCase.php';

/**
 * Some simple tests
 *
 * @author           Inflectra Corporation
 * @version          2.2.0
 *
 */

class SimpleTest extends PHPUnit_Framework_TestCase
{
      protected $fValue1;
      protected $fValue2;

      /**
       * Sets up the unit test
       */
      protected function setUp()
      {
            $this->fValue1= 2;
            $this->fValue2= 3;
      }

      /**
       * Tests the addition of the two values
       */
      public function testAdd()
      {
            $result = $this->fValue1 + $this->fValue2;

            // forced failure result == 5
            $this->assertTrue ($result == 6);
      }

      /**
       * Tests division by zero
       */
      /*
      public function testDivideByZero()
      {
            $zero = 0;
            $result = 8 / $zero;
            $result++; // avoid warning for not using result
      }

      /**
       * Tests two equal values
       */
      /*
      public function testEquals()
      {
            $this->assertEquals(12, 12);
            $this->assertEquals(12.0, 12.0);
            $num1 = 12;
            $num2 = 12;
            $this->assertEquals($num1, $num2);

            $this->assertEquals("Size", 12, 13);
            $this->assertEquals("Capacity", 12.0, 11.99, 0.0);
      }
```

```
	/**
	 * Tests success
	 */
	/*
	public function testSuccess()
	{
		//Successful test
		$this->assertEquals(12, 12);
	}
}
?>
```

The PHP class is marked as a PHPUnit test case by inheriting from the *PHPUnit_Framework_TestCase* base class, and the individual test methods are identified by using the 'test' prefix, with special setUp() and tearDown() methods reserved for the respective purposes. When you open up the class in a PHPUnit runner or execute from the command line it loads all the test classes and executes all the methods marked with 'test…' in turn.

Each of the Assert statements is used to test the state of the application after executing some sample code that calls the functionality being tested. If the condition in the assertion is true, then execution of the test continues, if it is false, then a failure is logged and PHPUnit moves on to the next test method.

So, to use SpiraTest with PHPUnit, each of the test cases written for execution by PHPUnit needs to have a corresponding test case in SpiraTest. These can be either existing test cases that have manual test steps or they can be new test cases designed specifically for automated testing and therefore have no defined test steps. In either case, the changes that need to be made to the PHPUnit test case and test suite for SpiraTest to record the PHPUnit test run are illustrated below:

a) Sample Test Suite

```
<?php
/**
 * Passes a list of tests to be executed to PHPUnit and adds the custom SpiraTest Listener
 *
 * @author           Inflectra Corporation
 * @version          2.3.0
 *
 */

 require_once 'PHPUnit/Framework.php';
 require_once 'PHPUnit/TextUI/ResultPrinter.php';
 require_once './SimpleTest.php';
 require_once '../SpiraListener/Listener.php';

 // Create a test suite that contains the tests
 // from the ArrayTest class
  $suite = new PHPUnit_Framework_TestSuite('SimpleTest');

 //Set the timezone identifier to match that used by the SpiraTest server
 date_default_timezone_set ("US/Eastern");

 //Create a new SpiraTest listener instance and specify the connection info
 $spiraListener = new SpiraListener_Listener;
 $spiraListener->setBaseUrl ('http://localhost/SpiraTeam');
 $spiraListener->setUserName ('fredbloggs');
 $spiraListener->setPassword ('fredbloggs');
 $spiraListener->setProjectId (1);
 $spiraListener->setReleaseId (1);
 $spiraListener->setTestSetId (1);

 // Create a test result and attach the SpiraTest listener
```

```
// object as an observer to it (as well as the default console text listener)
$result = new PHPUnit_Framework_TestResult;
$textPrinter = new PHPUnit_TextUI_ResultPrinter;
$result->addListener($textPrinter);
$result->addListener($spiraListener);

// Run the tests and print the results
$result = $suite->run($result);
$textPrinter->printResult($result);

?>
```

b) Sample Test Case

```
<?php
require_once 'PHPUnit/Framework/TestCase.php';

/**
 * Some simple tests using the ability to return results back to SpiraTest
 *
 * @author           Inflectra Corporation
 * @version          2.2.0
 *
 */

class SimpleTest extends PHPUnit_Framework_TestCase
{
      protected $fValue1;
      protected $fValue2;

      /**
       * Sets up the unit test
       */
      protected function setUp()
      {
             $this->fValue1= 2;
             $this->fValue2= 3;
      }

      /**
       * Tests the addition of the two values
       */
      public function testAdd__2()
      {
             $result = $this->fValue1 + $this->fValue2;

             // forced failure result == 5
             $this->assertTrue ($result == 6);
      }

      /**
       * Tests division by zero
       */
      /*
      public function testDivideByZero__3()
      {
             $zero = 0;
             $result = 8 / $zero;
             $result++; // avoid warning for not using result
      }

      /**
       * Tests two equal values
       */
      /*
```

```
        public function testEquals__4()
        {
                $this->assertEquals(12, 12);
                $this->assertEquals(12.0, 12.0);
                $num1 = 12;
                $num2 = 12;
                $this->assertEquals($num1, $num2);

                $this->assertEquals("Size", 12, 13);
                $this->assertEquals("Capacity", 12.0, 11.99, 0.0);
        }

        /**
         * Tests success
         */
        /*
        public function testSuccess__5()
        {
                //Successful test
                $this->assertEquals(12, 12);
        }
}
?>
```

Firstly, each of the individual test methods is appended with two underscores followed by the ID of the corresponding test case in SpiraTest. So for example testSuccess() is now testSuccess__5() as it maps to test case TC00005 inside SpiraTest.

Second, in the Test Suite class, the PHPUnit TestResult object is passed an additional PHPUnit listener (in addition to the default one). This special listener class intercepts the results from the test run during execution and uses it to generate the web-service messages that are sent to SpiraTest to communicate the test results.

The following attributes need to be set on the instance of the *SpiraListener_Listener*() object so that the extension can access the SpiraTest repository:

- **baseUrl** – The base URL used to access your instance of SpiraTest (e.g. http://myserver/SpiraTest). It should include the protocol (e.g. http/https), the server-name, the port number (if not 80/443) and the virtual directory (if there is one).
- **userName** - A valid username for the instance of SpiraTest that has access to the project specified above
- **password** - A valid password for the user specified above
- **projectId** – The ID of the project inside SpiraTest (this can be found on the project homepage in the “Project Overview” section)
- **releaseId** - The ID of the SpiraTest release to associate the test run with. This can be found on the releases list page (click on the Planning > Releases tab). If you don’t want to associate the test run with a specific release, just use the value -1 to indicate N/A.
- **testSetId** - The ID of the SpiraTest test set to associate the test run with. This can be found on the test set list page (click on the Testing > Test Sets tab). If you don’t want to associate the test run with a specific test set, just use the value -1 to indicate N/A.

The SpiraListener_Listener class can also be called with the parameters as the constructor arguments:

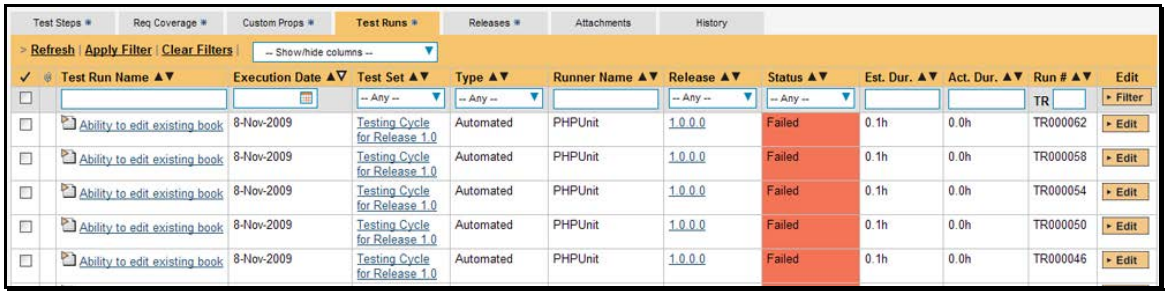
```
//Create a new SpiraTest listener instance and specify the connection info
 $spiraListener = new SpiraListener_Listener (
      'http://localhost/SpiraTeam',
      'fredbloggs',
```

```
        'fredbloggs',
        1,
        1,
        1);
```

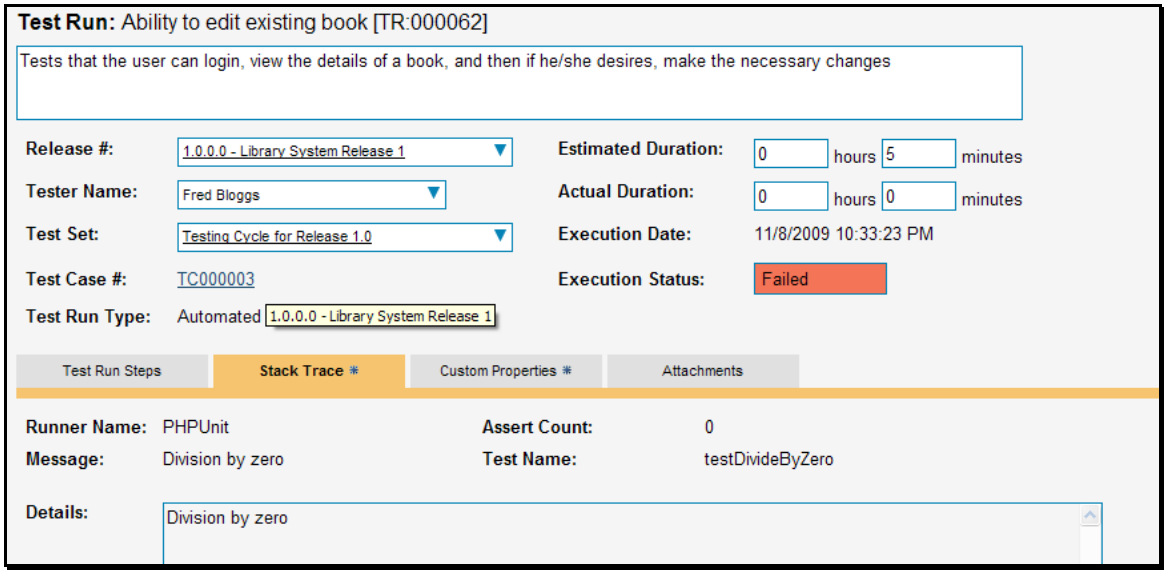
You can also attach the listener to the class declaratively by adding it to the `phpunit.xml` configuration file instead of adding through PHP code:

```
<phpunit>
  <listeners>
    <listener class="SpiraListener_Listener" file="../SpiraListener/Listener.php">
          <arguments>
              <!-- URL -->
              <string>http://localhost/SpiraTeam</string>
              <!-- User Name -->
              <string>fredbloggs</string>
              <!-- User Password -->
              <string>fredbloggs</string>
              <!-- Project ID -->
              <integer>1</integer>
              <!-- Release ID -->
              <integer>1</integer>
              <!-- Test Set ID -->
              <integer>1</integer>
            </arguments>
    </listener>
  </listeners>
  <testsuites>
    <testsuite name="Sample Suite">
      <directory>.</directory>
      <file>./SampleSuite.php</file>
    </testsuite>
  </testsuites>
</phpunit>
```

Now all you need to do is save your code, launch PHPUnit, run the test suite as you would normally do, and when you view the test cases in SpiraTest, you should see a PHPUnit automated test run displayed in the list of executed test runs:



Clicking on one of the PHPUnit test runs will bring up a screen that provides information regarding what PHPUnit test method failed, what the error was, together with the associated code stack-trace:



Congratulations… You are now able to run PHPUnit automated tests and have the results be recorded within SpiraTest. The sample test suite SampleSuite.php and sample test case SampleTest.php are provided with the installation in the Samples subfolder.

Legal Notices

This publication is provided as is without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

This publication could include technical inaccuracies or typographical errors. Changes are periodically added to the information contained herein; these changes will be incorporated in new editions of the publication. Inflectra Corporation may make improvements and/or changes in the product(s) and/or program(s) and/or service(s) described in this publication at any time.

The sections in this guide that discuss internet web security are provided as suggestions and guidelines. Internet security is constantly evolving field, and our suggestions are no substitute for an up-to-date understanding of the vulnerabilities inherent in deploying internet or web applications, and Inflectra cannot be held liable for any losses due to breaches of security, compromise of data or other cyber-attacks that may result from following our recommendations.

SpiraTest®, SpiraPlan®, SpiraTeam® and Inflectra® are registered trademarks of Inflectra Corporation in the United States of America and other countries. Microsoft®, Windows®, Explorer® and Microsoft Project® are registered trademarks of Microsoft Corporation. QuickTest Pro® is a registered trademark of Hewlett-Packard Development Company, L.P. All other trademarks and product names are property of their respective holders.

Please send comments and questions to:

Technical Publications

Inflectra Corporation

8121 Georgia Ave, Suite 504

Silver Spring, MD 20910-4957

U.S.A.

support@inflectra.com