Torry Harris

TORRY HARRIS BUSINESS SOLUTIONS

Unit Testing webMethods Integrations using JUnit

Practicing TDD for EAI projects

Ganapathi Nanjappa

4/28/2010

Table of Contents

1. Abstract 3
2. Introduction 3
3. JUnit wrapper for webMethods 3
4. JUnit test suite for webMethods Services 5
5. Limitations of the JUnit wrapper 6
6. Conclusion 6

1. Abstract

In a very short time, Test Driven Development (TDD) has branched off from Extreme Programming (XP) and gained a significant prominence among non-XP projects. It is now seen as a practice by itself and applied to various programming methodologies including RUP and AUP.

In addition to the quality benefits offered by practicing TDD, the unit tests provide a working specification of the functional code, and a fully functional test suite that can be reused many times over.

This white paper presents a solution to unit test webMethods Flow/Java services using the popular JUnit test framework, and helps put TDD into practice for Enterprise Application Integration projects.

2. Introduction

JUnit is a unit testing framework for the Java programming language and has played a significant role in the practice of TDD in Java projects. It provides reusable components to organize and execute test cases and verify the results.

webMethods Developer is the stand-alone graphical development tool used to build, edit and test integration logic implemented in the webMethods flow language.

While the webMethods Developer provides various tools for testing and debugging the integration solutions, it does not provide support for using JUnit from within the Developer. This white paper details how JUnit can be used for testing services implemented in the webMethods flow/java language.

3. JUnit wrapper for webMethods

As part of the ongoing efforts at the EAI Centre of Excellence at Torry Harris, a JUnit wrapper for webMethods was developed to test and verify the flow services implemented in the webMethods flow language. The JUnit Wrapper consists of a single java class named “WMTestClient” and provides methods to invoke any flow/java service deployed on the webMethods Integration Server (IS). The wrapper method accepts the input filename for the service under test and converts the file into a format that webMethods IS understands. The screen shot of a test method is shown below:

```
@Test
public void testConnectionStatistics() throws Exception {

    // Create the test client to invoke the service being tested
    WMTestClient client = new WMTestClient(
            properties.getProperty("server"),
            properties.getProperty("username"),
            properties.getProperty("password"));

    // Invoke the service under test
    Map<String, String> results = client.invoke(
            "pub.art.connection",
            "getConnectionStatistics",
            properties.getProperty("testConnectionStatistics.input"),
            properties.getProperty("testConnectionStatistics.output"),
            true);

    // Get the pipeline variable for comparison
    String valueString = results.get("TotalConnections");

    // Assert the results
    assertNotNull(valueString);
}
```

A sample JUnit test method for testing the "getConnectionStatistics" service.

The wrapper provides all the result variables of the service in a java "Map" and any variable can be used for asserting the test results. In the above test case, the output variable "TotalConnections" is used for asserting the test case. The wrapper is generic enough to allow assertion of multiple output cases, as may be required for services that return multiple values.

```
public Map<String, String> invoke(String packageName, String serviceName,
        String inputFileName, String outputFileName, boolean saveOutput)
        throws Exception {
```

The generic wrapper method to test any webMethods flow/java service.

The wrapper accepts input XML file, package and service names as input apart from the optional output file and location to save the output file. The input files and the test cases are generated by the developer for the services developed/modified. Input files can be generated from the webMethods Developer console by selecting the "Save Pipeline" option for the service under test. The output file can be used for closer inspection of the results and/or for comparing the results with a previous test run.

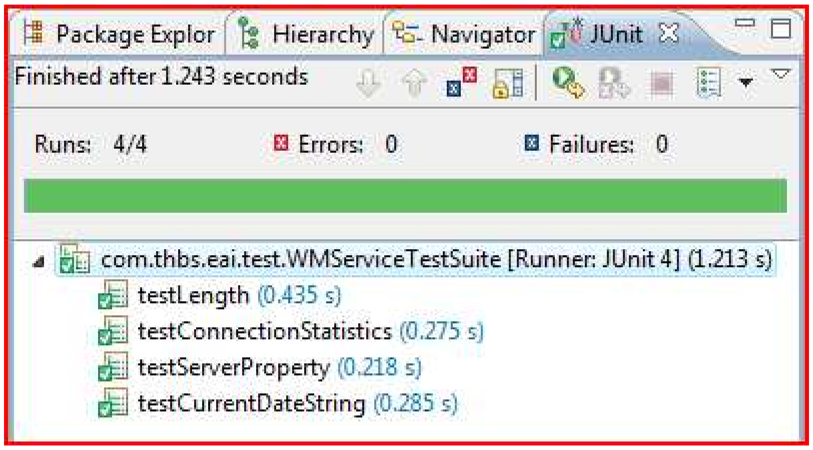
testConnectionStatistics.input	C:\\ZTestData\\getConnectionStatisticsInput.txt
testConnectionStatistics.output	C:\\ZTestData\\getConnectionStatisticsOutput.txt

The input and output file locations configured in the properties file.

The properties for the service under test and the server details are stored in a configuration file to allow changes to the inputs and server to test on without having to recompile the source. It also allows for easy customizations, for example, the variable to assert and the assertion value can also be made configurable for each service under test.

4. JUnit test suite for webMethods Services

A JUnit test suite was developed to test a few webMethods in-built services using the wrapper, the results as shown below:



Finished after 1.243 seconds

Runs: 4/4 Errors: 0 Failures: 0

com.thbs.eai.test.WMServiceTestSuite [Runner: JUnit 4] (1.213 s)
testLength (0.435 s)
testConnectionStatistics (0.275 s)
testServerProperty (0.218 s)
testCurrentDateString (0.285 s)

The result of a JUnit test suite execution that tests various webMethods built-in services.

With JUnit 4, creating a test suite is as easy as adding test methods to a source file. To create a new test method, any existing test method (like the one shown in the previous section) can be copied and renamed to reflect the new service under test. The input/output parameters are defined in the configuration file and the assert variable and value are updated to match the new service.

The unit test suites created as part of different projects can be invoked using 'ant' (http://ant.apache.org/) scripts from the command line as and when necessary (this will be mostly done when there are changes to dependant modules). Various JUnit ant tasks are available to generate reports on the tests executed. These reports can be used for verification of services tested and the coverage.

5. Limitations of the JUnit wrapper

The JUnit Wrapper for webMethods has the following limitations that JUnit has, it cannot:

- Automatically generate tests for a service under test
- Automatically generate test data for a service under test
- Provide coverage metrics for a webMethods flow/java service

6. Conclusion

Test-driven development is an important practice to delivering high quality code and to enable a flexible and result-oriented way of development. Apart from verification of code, it provides an automatic catalog of unit tests that can be regressed.

A test framework for webMethods integrations is even more crucial to simplify and automate unit tests as integrations work on heterogeneous environment. The framework presented in this paper is simple to setup and maintain, and enables TDD to be practiced for EAI projects. It uses the features made available by one of the most popular open-source testing solution and eliminates the need for developing a custom unit-testing solution for webMethods.