Maven Unit and Integration Test Guide

Karl Heinz Marbaise `<maui@soebes.de>`

Maven Unit and Integration Test Guide

by Karl Heinz Marbaise

Abstract

This is the `The Maven Unit and Integration Test Guide` (MaUI Test Guide) which describes the different options how to do unit- and integration testing with Maven[1] in different situations. For example if you are writing a Maven Plugin or just developing different other modules.

This is no plugin or development project it is `just` a guide how to do unit and or integration tests in relationship with Maven[2].

This guides requires that you are already familiar with Maven and it's life-cycles[3] etc. It will not explain how to configure a Maven Plugin etc. If you don't know Maven[4] or it's basic usage i recommend you to read Maven: The Complete Reference[5] or Maven by Example[6] other sources.

[1] http://maven.apache.org
[2] http://maven.apache.org
[3] http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html
[4] http://maven.apache.org
[5] http://www.sonatype.com/index.php/Support/Books/Maven-The-Complete-Reference
[6] http://www.sonatype.com/index.php/Support/Books/Maven-By-Example

List of Examples

Copyright

Online version published by ${project.organization.name}, ${organization.address}.

Chapter 1. Unit Testing

1.1. Overview

The execution of unit tests is controlled by the Maven Surefire Plugin[1] which handles the execution of the unit tests within the life-cycle.

If you already familiar with Maven[2] you know where to put the unit tests[3]. This means your unit tests are located in `src/test/java` and the appropriate resources are located in `src/test/resources`.

1.2. First Basic Unit Test Example

Assuming we have the following structure of a simple Java project currently consisting of a simple class `BitMask.java` and of course the unit test class `BitMaskTest.java`.

Example 1.1. Folder Strucutre of the basic unit test example

```
.
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- soebes
    |               `-- training
    |                   `-- maven
    |                       `-- simple
    |                           `-- BitMask.java
    `-- test
        `-- java
            `-- com
                `-- soebes
                    `-- training
                        `-- maven
                            `-- simple
                                `-- BitMaskTest.java
```

For the sake of clarity the `pom.xml` excerpt contains only the relevant things. If you don't know what a scope and/or a dependency is, i recommend to take a look into the documentation about dependency mechanism in Maven[4].

In this case we use JUnit[5] for writing the unit test. This will work as well with TestNG[6] if you prefer TestNG over JUnit. The following `pom.xml` snippet shows how to add the dependency of JUnit to your project.

Example 1.2. POM for Unit Tests

```
<project ...>
  [...]
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.1</version>
      <scope>test</scope>
    </dependency>
  [...]
  </dependencies>
  [...]
</project>
```

[1] http://maven.apache.org/plugins/maven-surefire-plugin/
[2] http://maven.apache.org
[3] http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html
[4] http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html
[5] http://www.junit.org
[6] http://www.testng.org

If you like to execute the unit tests of the project, which can simply be achieved by calling **`mvn test`**, you will get a result like the following

Example 1.3. Execution of the Unit Tests

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Unit Test :: example 0.1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.4.1:clean (default-clean) @ unit-test-example ---
[INFO] Deleting /maui/examples/unit-test-example/target
[INFO]
[INFO] --- maven-resources-plugin:2.4.3:resources (default-resources) @ unit-test-example ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /maui/examples/unit-test-example/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ unit-test-example ---
[INFO] Compiling 1 source file to /maui/examples/unit-test-example/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.4.3:testResources (default-testResources) @ unit-test-example ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /maui/examples/unit-test-example/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:testCompile (default-testCompile) @ unit-test-example ---
[INFO] Compiling 1 source file to /maui/examples/unit-test-example/target/test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.7.2:test (default-test) @ unit-test-example ---
[INFO] Surefire report directory: /maui/examples/unit-test-example/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.soebes.training.maven.simple.BitMaskTest
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.078 sec

Results :

Tests run: 5, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4.834s
[INFO] Finished at: Fri Apr 01 20:02:40 CEST 2011
[INFO] Final Memory: 9M/92M
[INFO] ------------------------------------------------------------------------
```

1.3. Interface Testing

A typical situation within a project is that you created an interface and having a number of implementations. Concluding the former idea the structure of the project follows:

Example 1.4. Interface Testing

```
.
|-- interface-impl-a
|   |-- pom.xml
|   ...
|-- interface-impl-b
|   |-- pom.xml
|   ...
|-- interface-interface
|   |-- pom.xml
|   ...
|-- interface-test
|   |-- pom.xml
|   ...
`-- pom.xml
```

So the question in this case is: How to test the different implementations? But is this true? No. We don't have to test the implementations we have to test the behavior of the interface for all implementations. This can be achieved by writing unit tests in both modules `interface-impl-a` and `interface-impl-b` which means in other words to copy&paste the unit test code but this is error prone and of course not effective.

The solution is to implement the unit test code in an abstract class in a separate module `interface-test`.

Example 1.5. Interface Test

```
public abstract class ImplementationUnitTest {

    protected static IFunction function;

    @Test
    public void firstTest() {
        assertTrue(function.function("function1"));
    }

    @Test
    public void secondTest() {
        assertFalse(function.function("function"));
    }
}
```

The more important part is the `pom.xml` file which is needed in this case. We have to make sure that the test code is made available for other modules as well. So we have to define the `pom` like the following:

Example 1.6. POM for Interface Testing

```
[...]
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <executions>
        <execution>
          <goals>
            <goal>test-jar</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

<dependencies>
  <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>interface-interface</artifactId>
    <version>${project.version}</version>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
[...]
```

Based on that we can now implement the unit test for implementation-a like this:

Example 1.7. Unit Test for Implementation A

```
public class ImplementationAUnitTest extends ImplementationUnitTest {
  @BeforeClass
  public static void beforeClass() {
    function = new ImplementationA();
  }
```

```
}
```

And for implementation-b like this:

Example 1.8. Unit Test for Implementation B

```
public class ImplementationBUnitTest extends ImplementationUnitTest {
  @BeforeClass
  public static void beforeClass() {
    function = new ImplementationB();
  }
}
```

The only thing we need to implement is the initialization code to create an instance of the particular class.

Based on the above steps the project layout [7]results into the following.

Example 1.9. Project Structure

```
|-- interface-impl-a
|   |-- pom.xml
|   `-- src
|       |-- main
|       |   `-- java
|       |       `-- com
|       |           `-- soebes
|       |                 `-- ImplementationA.java
|       `-- test
|           `-- java
|               `-- com
|                   `-- soebes
|                         `-- ImplementationAUnitTest.java
|-- interface-impl-b
|   |-- pom.xml
|   `-- src
|       |-- main
|       |   `-- java
|       |       `-- com
|       |           `-- soebes
|       |                `-- ImplementationB.java
|       `-- test
|           `-- java
|               `-- com
|                   `-- soebes
|                        `-- ImplementationBUnitTest.java
|-- interface-interface
|   |-- pom.xml
|   `-- src
|       `-- main
|           `-- java
|               `-- com
|                   `-- soebes
|                        `-- IFunction.java
|-- interface-test
|   |-- pom.xml
|   `-- src
|       `-- test
|           `-- java
|               `-- com
|                   `-- soebes
|                        `-- ImplementationUnitTest.java
`-- pom.xml
```

[7]The complete code for the example can be found on the web site [http://github.com/khmarbaise/maui/].

Chapter 2. Integration Testing

2.1. Overview

The execution of integration tests is handled by the Maven Failsafe Plugin[1] in contradiction to the Maven Surefire Plugin[2] which is responsible for the unit tests. By assuming that you are already familiar with Maven and it's directory layout[3] you know where to put unit tests but unfortunately there does not exist a definition or a convention like this for integration tests. There are some suggestions circulating around like `src/it/java`, but these are only rumors.

(Think about the following!) In this guide we suggest to put them into the same directory structure like the unit tests which means in other words `src/test/java` and make them distinguishable by the naming conventions[4] but in future Maven releases this might be changed.

2.2. First Basic Integration Test Example

AXXXX

Example 2.1. Folder Structure of basic integration test

```
.
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- soebes
    |               `-- maui
    |                   `-- it
    |                       `-- BitMask.java
    `-- test
        `-- java
            `-- com
                `-- soebes
                    `-- maui
                        `-- it
                            |-- BitMaskIT.java
                            `-- BitMaskTest.java
```

Example 2.2. POM for the integration test

```
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.soebes.maui</groupId>
  <artifactId>it-test-example</artifactId>
  <version>0.1.0-SNAPSHOT</version>

  <name>MaUI Test Guide :: Integration Test Example</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
  </properties>

  <build>
   <plugins>
      <plugin>
```

[1] http://maven.apache.org/plugins/maven-failsafe-plugin/
[2] http://maven.apache.org/plugins/maven-surefire-plugin/
[3] http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html
[4] http://maven.apache.org/plugins/maven-failsafe-plugin/integration-test-mojo.html#includes

```
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.3.2</version>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-failsafe-plugin</artifactId>
        <version>2.8</version>
        <executions>
          <execution>
            <id>integration-test</id>
            <goals>
              <goal>integration-test</goal>
            </goals>
          </execution>
          <execution>
            <id>verify</id>
            <goals>
              <goal>verify</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

2.3. Integration Test

Example 2.3.

```
        TEst
```

Chapter 3. Maven Plugin Integration Testing

3.1. Overview

If you are writing a Maven Plugin the time comes that you need to test your plugin in an real environment. The best thing you can do is to write integration test by the help of the maven-invoker-plugin.

3.2. First Basic Integration Test Example

AXXXX

Example 3.1. Folder Structure of basic integration test

```
.
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- soebes
    |               `-- maui
    |                   `-- it
    |                       `-- BitMask.java
    `-- test
        `-- java
            `-- com
                `-- soebes
                    `-- maui
                        `-- it
                            |-- BitMaskIT.java
                            `-- BitMaskTest.java
```