Team Silver
Benjamin Trettin
Christian Shinkle
Kendall Berner
William Fuhrmann

Detailed Level Scenarios

Writing a test is typically a three-step process:

1. Write the business logic of a test and insert TestNG annotations in your code.
2. Add the information about a test (e.g. the class name, the groups you wish to run, etc...) in a testng.xml file or in build.xml.
3. Run TestNG.

1. User runs a testing suite through specifications in a xml file. The user can specify the package name instead of the class names. The user can also specify groups and methods to be included or excluded. The user can also define new groups inside testng.xml and specify additional details in attributes, such as whether to run the tests in parallel, how many threads to use, whether you are running JUnit tests, etc…

2. User runs a test file through ant. In this scenario, the user runs a test always through forked JVM. Many attributes can be assigned, but ant requires one of classpath, classpathref or nested <classpath> to be used to run a test. The user can specify the package name instead of the class names. The user can also specify groups and methods to be included or excluded. New groups and attributes can also be specified.

3. User runs a test through the command line. The user uses the command java org.testng.TestNG testng1.xml [testng2.xml testng3.xml ...] to invoke testng. The user needs to specify at least one XML file describing the TestNG suite you are trying to run. Additionally, many command-line switches are available, but the command line is the simplest way to invoke a test, because you are unable to specify parameters.

4. User runs a test through eclipse. The user creates a TestNG class with TestNG annotations and/or one or more testng.xml files and creates a TestNG Launch Configuration. The user then Selects the Run menu and create a new TestNG configuration. The user then launches the test in 3 different ways: from a class file, groups, definition file, or a method.

5. TestNG mode. The TestNG mode gets applied when tests are passed to TestNG using classfilesetref, methods or nested <classfileset> and tells TestNG what kind of tests it should look for and run:

- "testng": find and run TestNG tests.
- "junit": find and run JUnit tests.
- "mixed": run both TestNG and JUnit tests.

6. User converts Junit tests into TestNG tests. The user uses quick fix function to have TestNG automatically convert Junit tests to TestNG.

7. User uses Conversion refactoring. The refactoring wizard contains several pages that let the user generate a testng.xml automatically. The user can specify by packages or individual classes as well as decide to exclude certain files from the refactoring.

8. User runs a test suite. The user creates a series of test files and groups the files together into a suite. The user then runs the suite and receives that output grouped together.

9. User writes a method using the @Before annotation that runs before each test in their suite. The user then supplies multiple tests with the @Test annotation.

10. User runs a single test that calls a function and compares its output to the desired output using methods with the @Test annotation.

11. User organizes tests with a group. The user defines a set of tests to belong to the same group. The user then can run the all the tests in a group.

12. User defines parameters. The user defines parameters to be passes into a method in xml file. The user then uses @Parameter annotations to enforce parameters of specified methods.