Chapter No. 5
"Dependencies"
In this package, you will find:
A Biography of the author of the book
A preview chapter from the book, Chapter NO.5 "Dependencies"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Varun Menon is a QA consultant with several years of experience in developing automation frameworks on various technologies and languages such as Java, JavaScript, Ruby, and Groovy. He has worked on web and mobile applications for some of the leading clients in the field of supply chain management, online photo books, video analytics, and market research.

He blogs at http://blog.varunin.com and is active on Stack Overflow, Selenium, and robotium groups. He is also the author of an upcoming open source android automation tool Bot-bot, which has record and re-play features like Selenium.

He currently holds the position of QA Architect at Pramati Technologies Private Limited, Hyderabad, India.

For More Information:
TestNg Beginner's Guide

Currently, TestNG is the most widely used testing framework in the software industry. It provides a lot of features over the conventional JUnit framework and is used for different kinds of testing like unit, functional, integration testing, and so on. This book explains different features of TestNG with examples. You will learn about the basic features as well as some of the advanced features provided by TestNG.

What This Book Covers

Chapter 1, Getting Started, explains TestNG and its advantages over other existing frameworks. It also explains how to install and run your first TestNG test.

Chapter 2, Understanding testng.xml, explains the testng.xml file which is used to configure the TestNG tests. It also explains different ways to create test suites by adding test packages, test classes, and test methods to the respective test suite, according to test needs.

Chapter 3, Annotations, explains the various annotations in TestNG and the different features supported by using them.

Chapter 4, Groups, explains the grouping feature provided by TestNG and how you can use it to include or exclude a group of tests in test execution.

Chapter 5, Dependencies, explains the dependency feature provided by TestNG. You will learn how test methods can depend upon another method or a group of methods.

Chapter 6, The Factory Annotation, explains the Factory annotation and how tests can be created at runtime based on a set of data. You will also learn about the difference between Factory and DataProvider annotation and how they can be used together.

Chapter 7, Parallelism, explains a very important feature of TestNG which allows different configurations for different tests to be run in parallel.

Chapter 8, Using Build Tools, explains build automation and its advantages. It also explains the different build automation tools available and how TestNG can be used along with them.

Chapter 9, Logging and Reports, explains about the default logging and report options available with TestNG. It also explains how to extend and write your own logging and reporting framework above it.

Chapter 10, Creating a Test Suite through Code, explains the different ways to write and configure TestNG tests through code without the need of an XML configuration file.

Chapter 11, Migrating from JUnit, explains different ways to migrate to JUnit from TestNG and things that need to be taken care of while migrating.

For More Information:
Chapter 12, *Unit and Functional Testing*, explains the unit and functional testing usage of TestNG. It also explains the few mocking techniques to be used for Unit testing and covers the use of Selenium with TestNG for functional testing.

For More Information:

In the previous chapter we learned about grouping of tests which allows users to group the tests into specific named groups. In this chapter we will learn about the dependency feature in TestNG. Dependency allows users of TestNG to specify a dependency method or a dependency group for a test method.

In this chapter we'll cover the following topics:

- Dependency test
- Writing a multiple dependency test
- Dependency on group
- Using regular expressions
- Defining dependency through XML

**Dependency test**

As said earlier dependency is a feature in TestNG that allows a test method to depend on a single or a group of test methods. This will help in executing a set of tests to be executed before a test method. Method dependency only works if the depend-on-method is part of the same class or any of the inherited base class (that is, while extending a class).

**Test with single test method dependency**

To start with, dependency in TestNG lets you create a sample test method that depends on another test method of the same class.

For More Information:
**Time for action – creating a test that depends on another test**

Perform the following steps to create a test that depends on another test:

1. Open Eclipse and create a Java project with the following structure. Please make sure that the TestNG library is added to the build path of the project as mentioned in *Chapter 1, Getting Started*.

   ![Project Structure](image1.png)

2. Create a new package named `test.depends.method` under the existing `test.depends` package.

3. Create a new class named `SimpleDependencyTest` under the `test.depends.method` package and replace the following code in it:

   ```java
   package test.depends.method;

   import org.testng.annotations.Test;

   public class SimpleDependencyTest {
     @Test(dependsOnMethods={"testTwo"})
     public void testOne() {
       System.out.println("Test method one");
     }

     @Test
     public void testTwo() {
       System.out.println("Test method two");
     }
   }
   ```

   The preceding test class contains two test methods which print a message name onto the console when executed. Here, test method `testOne` depends on test method `testTwo`. This is configured by using the attribute `dependsOnMethods` while using the Test annotation as shown is the preceding code.

---

For More Information:
4. Select the above test class in Eclipse and run it as TestNG test. You will see the following test result in the Console window of Eclipse:

![Console output]

What just happened?

We successfully created a test class that contains a test method that depends upon another test method. In the above test result you can see the message Test method two printed before the Test method one message. This shows that the testOne method got executed after testTwo as it depends on testTwo.

The dependency on a test method is configured for a test by providing the dependent test method name to the attribute dependsOnMethods while using the Test annotation, as mentioned in the previous sample code.

Test that depends on multiple tests

Sometimes it may be required for a test method to depend upon multiple other methods. This feature is very well supported by TestNG as part of the dependency support. Let's create a sample program and see how to create a test with multiple dependency.

For More Information:  
Time for action – creating a test that depends on multiple tests

1. Create a new class named MultiDependencyTest under the test.depends.
method package and replace the following code in it:

```java
package test.depends.method;

import org.testng.annotations.Test;

public class MultiDependencyTest {
    @Test(dependsOnMethods={"testTwo","testThree"})
    public void testOne(){
        System.out.println("Test method one");
    }

    @Test
    public void testTwo(){
        System.out.println("Test method two");
    }

    @Test
    public void testThree(){
        System.out.println("Test method three");
    }
}
```

The preceding test class contains three test methods which print a
message name onto the console when executed. Here test method testOne
depends on test methods testTwo and testThree. This is configured by using
the attribute dependsOnMethods while using the Test annotation as shown in
the preceding code.
2. Select the above test class in Eclipse and run it as TestNG test. You will see the following test result in the Console window of Eclipse.

![Console](image)

What just happened?

We successfully created a test class that contains a test method that depends upon multiple test methods. By looking at the console message we can see that methods `testTwo` and `testThree` got executed before `testOne`.

The dependency on multiple test methods is configured for a test by providing comma separated dependent test method names to the attribute `dependsOnMethods` while using the `Test` annotation as mentioned in the preceding sample code.

Inherited dependency test

Till now we have seen samples in which the dependent test methods were part of the same class. As I said earlier, dependency on test methods can only be mentioned for test methods that belong to the same class or any of the inherited base classes. In this section we will see how TestNG executes the test methods when the dependent methods are part of the inherited base class.
Time for action – creating a test that depends on inherited tests

1. Create a new class named `InheritedTest` under the `test.depends.method` package and replace the following code in it:

   ```java
   package test.depends.method;

   import org.testng.annotations.Test;

   public class InheritedTest extends SimpleDependencyTest{
       @Test(dependsOnMethods={"testOne"})
       public void testThree(){
           System.out.println("Test three method in Inherited test");
       }

       @Test
       public void testFour(){
           System.out.println("Test four method in Inherited test");
       }
   }
   ```

   The preceding test class contains two test methods which print a message name onto the console when executed. Here test method `testThree` depends on test method `testOne`. This is configured by using the attribute `dependsOnMethods` while using the `Test` annotation as shown in the preceding code.

   Following is the code of the `SimpleDependencyTest` class:

   ```java
   package test.depends.method;

   import org.testng.annotations.Test;

   public class SimpleDependencyTest {
       @Test(dependsOnMethods={"testTwo"})
       public void testOne(){
           System.out.println("Test method one");
       }

       @Test
       public void testTwo(){
           System.out.println("Test method two");
       }
   }
   ```

For More Information:
The preceding class also contains two test methods which print a message to the console. As you can see, the `testOne` method in the preceding class depends on the `testTwo` method.

2. Select the preceding test class in Eclipse and run it as TestNG test. You will see the following test result in the Console window of Eclipse:

```
Test four method in Inherited test
Test method two
Test method one
Test three method in Inherited test
PASSED: testFour
PASSED: testTwo
PASSED: testOne
PASSED: testThree
```

What just happened?

We successfully created a test class that contains a test method that depends upon the parent class test method. As you can see from the test results the sequence of execution is `testFour`, `testTwo`, `testOne`, and lastly, `testThree`. As `testThree` depends on `testOne` and on `testTwo`, TestNG executes all the test methods based on the dependency and finally the respective test method. TestNG by default executes methods based on the ascending order of their names, so in this case, it executes `testFour` first because it comes at the top in the current list of the test methods, then when it encounters the dependency of `testThree`, it executes its dependent methods and then the said method itself.

Using the dependency feature, you can also make certain methods run sequentially by configuring the dependency accordingly.

For More Information:
Dependencies

Have a go hero

Having gone through the chapter, feel free to attempt the following:

- Create a test class which contains three test methods and configure dependency for these test methods in such a way that all of them get executed in a particular sequential order.
- Create a test class that is inherited from another test class and has overriding test methods in it. Configure one of the test methods such that it depends on the overriding test method.

Dependent groups

Similar to dependent methods TestNG also allows test methods to depend on groups. This makes sure that a group of test methods get executed before the dependent test method.

Time for action – creating a test that depends on a group

1. Create a new package named `groups` under the `test.depends` package in the earlier project.
2. Create a new class named `SimpleGroupDependency` under the `test.depends.groups` package and replace the following code in it:

```java
package test.depends.groups;

import org.testng.annotations.Test;

public class SimpleGroupDependency {
    @Test(dependsOnGroups={"test-group"})
    public void groupTestOne(){
        System.out.println("Group Test method one");
    }

    @Test(groups={"test-group"})
    public void groupTestTwo(){
        System.out.println("Group test method two");
    }

    @Test(groups={"test-group"})
    public void groupTestThree(){
        System.out.println("Group Test method three");
    }
}
```

For More Information:

The preceding test class contains three test methods which print a message onto the console when executed. Two of the test methods belong to a group named test group whereas the third method named groupTestOne depends on the group test group. The dependency on the group is configured using the attribute dependsOnGroups while using the Test annotation, as shown in the preceding code.

3. Select the preceding test class in Eclipse and run it as TestNG test. You will see the following test result in the Console window of Eclipse.

```
<terminated> SimpleGroupDependency [TestNG] / 
[TestNG] Running: 
/tmp/testng-eclipse-2131325914/testng-c
Group test method two
Group test method three
Group Test method one
PASSED: groupTestTwo
PASSED: groupTestThree
PASSED: groupTestOne
=======================================================
Default test
Tests run: 3, Failures: 0, Skips: 0
```

**What just happened?**

We have successfully created a test class that contains a test method that depends upon a test group. The dependency is configured by providing the dependent group names to the attribute dependsOnGroups, which uses the Test annotation.

Like method dependency, group dependency also supports configuration for a method to depend on multiple groups. All dependent group names have to be provided as array names to the attribute dependsOnGroups.

**Depending on methods from different classes**

As explained in the earlier examples, method dependency only works with other methods that belong to the same class or in one of the inherited classes but not across different classes. In case you need a test method that exists in a separate class; you can achieve this by assigning the said test method to a group and configuring the dependent test method to be dependent on the said group.

Let’s create a sample test to learn how to create dependency across multiple classes.

---

For More Information:
**Dependencies**

**Time for action – depending on a method from a different class**

1. Create a new class named `DifferentClassDependency` under the `test.depends.groups` package and replace the following code in it:

   ```java
   package test.depends.groups;

   import org.testng.annotations.Test;

   public class DifferentClassDependency {
     @Test(dependsOnGroups={"test-group","same-class"})
     public void testOne(){
       System.out.println("Different class test method one");
     }

     @Test(groups={"same-class"})
     public void testTwo(){
       System.out.println("Different class test method two");
     }

     @Test(groups={"same-class"})
     public void testThree(){
       System.out.println("Different class test method three");
     }
   }
   
   The preceding test class contains three test methods which print a message onto the console when executed. Two of the test methods belong to group named `same-class` whereas the third method, named `testOne`, depends on the groups named `test-group` and `same-class`. The group `test-group` refers to the test that belongs to the earlier created class named `SimpleGroupTest`.

2. Create a new file named `diff-class-testng.xml` under the current project and replace the existing code with the following code:

   ```xml
   <suite name="Different class dependency Suite" verbose="1">
     <test name="Different class dependency Test">
       <packages>
         <package name="test.depends.groups" />
       </packages>
     </test>
   </suite>
   
   The preceding testng XML contains a test that considers all the test class under the package named `test.depends.groups`.

3. Select the preceding testng XML in Eclipse and run it as TestNG suite. You will see the following test result in the Console window of Eclipse.

What just happened?
We successfully created a test class that contains a test method which depends upon a test method that belongs to another class. This is achieved using the dependsOnGroup feature supported by TestNG. The previous results show that all the dependent methods were executed prior to the execution of the depending method.

Using regular expressions
Regular expressions feature as discussed in earlier chapters of this book is also supported while using the dependsOnGroups feature in TestNG. This helps users to do name based search on groups and add dependency onto the said groups for a test method.

In this section we will create a sample program that does a regular expression search to look for dependOnGroups.

Time for action – using regular expressions
1. Create a new package named regularexp under the existing package test.depends in the earlier project.
2. Create a new class named RegularExpressionTest under the test.depends.regularexp package and replace the following code in it:
package test.depends.regularexp;

For More Information:
Dependencies

```java
import org.testng.annotations.Test;

public class RegularExpressionTest {
    @Test(dependsOnGroups="starts-with.*")
    public void regularExpMethod(){
        System.out.println("Dependent method");
    }

    @Test(groups="starts-with-one")
    public void startsWithMethodOne(){
        System.out.println("Starts with method one");
    }

    @Test(groups="starts-with-two")
    public void startsWithMethodTwo(){
        System.out.println("Starts with method two");
    }
}
```

The preceding test class contains three test methods which print a message onto the console when executed. Two of the test methods belong to different groups named `starts-with-one` and `starts-with-two`, respectively, whereas the third method named `regularExpMethod` depends on all the groups whose names start with the text `starts-with`.

3. Select the above test class in Eclipse and run it as TestNG test. You will see the following test result in the Console window of Eclipse.

![Console output of TestNG test](image_url)

For More Information:
What just happened?
We successfully created a test class which contains a test method that depends on groups whose names start with the text starts-with. As you can see from the test results, all the test methods that belonged to the particular group got executed before the dependent method. Regular expression-based search can be done on a text by using .* as discussed in earlier chapters. Other than the preceding example of a search based on starts-with we can easily do contains and ends-with search too.

Have a go hero
Create a sample program where a test method depends on all the groups whose name contains a specific text in it.

XML-based dependency configuration
TestNG also allows group-based dependency to be defined inside the testng XML configuration file. This can be done when defining a test inside a suite. We write some similar dependency programs that we have written earlier using XML configuration.

Simple group dependency
In this section we will write a simple XML configuration file to define a simple group dependency for a test method.

Time for action – using simple dependency in XML
1. Create a new package named xml under the existing package test.depends in the earlier project.
2. Create a new class named SimpleXmlDependency under the test.depends.xml package and replace the following code in it:
   ```java
   package test.depends.xml;

   import org.testng.annotations.Test;

   public class SimpleXmlDependency {
     @Test(groups={"dependent-group"})
     public void groupTestOne() {
       System.out.println("Group Test method one");
     }
   }
   ```
Dependencies

```java
@Test(groups={"test-group"})
public void groupTestTwo(){
    System.out.println("Group test method two");
}

@Test(groups={"test-group"})
public void groupTestThree(){
    System.out.println("Group Test method three");
}
}
```

The preceding test class contains three test methods which print a message onto the console when executed. Two of the test methods belong to a different group named `test-group`, whereas the third method named `groupTestOne` belongs to group a named `dependent-group`.

3. Create a new XML named `simple-xml-dependency.xml` and add the following code to it:

```xml
<suite name="Simple xml dependency Suite" verbose="1">
    <test name="Simple xml dependency Test">
        <groups>
            <dependencies>
                <group name="dependent-group" depends-on="test-group" />
            </dependencies>
            <run>
                <include name="dependent-group" />
            </run>
        </groups>
        <classes>
            <class name="test.depends.xml.SimpleXmlDependency" />
        </classes>
    </test>
</suite>
```

The preceding testNG XML configuration file contains a single test inside the suite. Group dependency is defined using the `dependencies` attribute under the `groups` block. The `group` tag is used with the group name and the names of the group that the said group depends on, as shown in the previous XML file.

For More Information:
4. Select the previous XML file in Eclipse and run it as TestNG suite. You will see the following test result in the Console window of Eclipse:

![Console output](image)

**What just happened?**

We successfully created a test class which contains a test method that depends on groups, and the dependency configuration was done using the testng XML file. As you can see from the results, the test methods from the group `test-group` got executed before the dependent test method.

**Multigroup dependency**

In this section we will create a sample XML configuration for defining a multigroup dependency for a particular group.

**Time for action – defining multigroup dependency in XML**

1. Create a new class named `MultiGrpXmlDependency` under the `test.depends.xml` package and replace the following code in it:

   ```java
   package test.depends.xml;

   import org.testng.annotations.Test;

   public class MultiGrpXmlDependency {
       @Test(groups={"dependent-group"})
       public void groupTestOne(){
           System.out.println("Group Test method one");
       }
   }
   ```

   For More Information:  
The preceding test class contains three test methods which print a message onto the console when executed. Each of the test methods belong to a different group.

2. Create a new XML with the name `multigroup-xml-dependency.xml` and add the following code to it:

```xml
<suite name="Multi group xml dependency Suite" verbose="1">
    <test name="Multi group xml dependency Test">
        <groups>
            <dependencies>
                <group name="dependent-group" depends-on="test-group-one test-group-two" />
            </dependencies>
            <run>
                <include name="dependent-group" />
            </run>
        </groups>
        <classes>
            <class name="test.depends.xml.MultiGrpXmlDependency" />
        </classes>
    </test>
</suite>
```

The preceding testng XML configuration contains a single test inside the suite. Group dependency is defined using the `dependencies` attribute under the `groups` block. The `group` tag is used with the group name and the names of the groups that the said group depends on, as shown in the preceding XML file. In case the group is dependent upon multiple groups, each group is separated by a space, as shown in the preceding XML file.
3. Select the preceding XML file in Eclipse and run it as TestNG suite. You will see the following test result in the Console window of Eclipse:

![Console output]

**What just happened?**

We successfully created a test class which contains a test method that depends on multiple groups, and where the dependency configuration was done using the testng XML file. You can see from the results that the test methods from the group `test-group-one` and `test-group-two` got executed before the dependent test method which belongs to the `dependent-group`. Multiple group dependency is defined in XML by providing all the group names separated by a space.

**Using regular expressions for defining dependency**

In this section we will create a sample XML configuration file that defines a group dependency using regular expressions.

Time for action – using regular expressions for dependency

1. Create a new class named `RegularExpressionXmlTest` under the `test.depends.xml` package and replace the following code in it:

```java
package test.depends.xml;

import org.testng.annotations.Test;

public class RegularExpressionXmlTest {
    @Test(groups={"test"})
```

---

For More Information:  
public void regularExpMethod(){
    System.out.println("Dependent method");
}

@Test(groups="starts-with-one")
public void startsWithMethodOne(){
    System.out.println("Starts with method one");
}

@Test(groups="starts-with-two")
public void startsWithMethodTwo(){
    System.out.println("Starts with method two");
}

The preceding test class contains three test methods which print a message onto the console when executed. Each of the test methods belongs to a different group.

2. Create a new XML file named regexp-xml-dependency.xml and add the following code to it:

```xml
<suite name="Regexpxmldependency Suite" verbose="1">
    <test name="Regexp xml dependency Test">
        <groups>
            <dependencies>
                <group name="test" depends-on="starts-with.*" />
            </dependencies>
            <run>
                <include name="test" />
            </run>
        </groups>
        <classes>
            <class name="test.depends.xml.RegularExpressionXmlTest" />
        </classes>
    </test>
</suite>
```

The preceding testng XML configuration contains a single test inside the suite. Group dependency is defined using the dependencies attribute under the groups block. The group tag is used with the group name. Dependent group is defined using the regular expressions.

For More Information:
3. Select the preceding XML file in Eclipse and run it as TestNG suite. You will see the following test result in the Console window of Eclipse:

What just happened?

We successfully created a test class which contains a test method for which the dependency was configured using a regular expression. In the preceding sample program dependent groups were added based on the names that start with starts with. For regular expressions, search is done using the expression .*. This can also be used for ends with and contains text-based search.

Pop quiz – dependencies

Q1. Which attribute has to be used with annotation Test to define a method dependency?

1. dependsOnMethods
2. dependsONMethod
3. methodsDependent

Q2. What kind of dependency can be configured using the testng XML configuration file?

1. Methods
2. Groups
3. Both of the above

For More Information:
Dependencies

Summary

In this chapter we have learned about dependency feature in TestNG. We have learned multiple ways to define/configure test methods to be dependent on other test methods.

The dependency can be configured for both test methods and also for groups. We can even configure/define a dependency through the testng XML suite file. Use of regular expressions is supported for groups and can be used for name-based search to add groups to dependency.

In the next chapter we will cover the Factory annotation provided by TestNG. We will learn about its usage and how it's different from theDataProvider annotation.

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.