# TestNG Groups | MetaGroups with Example

**TestNG Groups** is one of the most popular features supported by [TestNG](https://www.scientecheasy.com/2019/04/testng-tutorial.html/) which is not available in the JUnit framework.

TestNG framework allows us to perform groupings of test methods.

Using TestNG, we can declare a set of test methods in a particular named group, or multiple groups and run multiple groups of tests at different times or in different places.

This feature provides us maximum flexibility in dividing tests and doesn’t require us to recompile anything if you want to run two different sets of tests back to back.

We can add a method or an entire class to a group by using groups parameter in the [@Test annotation](https://www.scientecheasy.com/2020/07/testng-test-annotation.html/).

The general syntax to add a class or method to one or several groups is as follows:

@Test(groups = {"GroupName"})

For example:

   @Test(groups = {"Chemistry"})

   public void atom()

  {

      . . . . . . . .

    }

   @Test(groups = {"Chemistry"})

   public void electron()

   {

    . . . . . . . .

   }

In the above example, both test methods will execute in one group named Chemistry.

**Test Method:** A method annotated with @Test annotation is called test method.

## Grouping of Test Methods in Single Group

Let’s take an example program where we will create a test class and execute certain test methods that belong to a single group. Look at the following source code.

**Program source code 1:**

package testngtests;

import org.testng.annotations.Test;

public class GroupingTestMethods

{

@Test(groups = {"Car"})

public void mahindra()

{

System.out.println("CAR 1: Mahindra");

}

@Test

public void sedan()

{

System.out.println("Sedan CAR");

}

@Test(groups = {"Car"})

public void Bajaj()

{

System.out.println("CAR 2: Bajaj Alto");

}

}

Output:

Sedan CAR

CAR 2: Bajaj Alto

CAR 1: Mahindra

In the above example program, the preceding class contains three test methods. Two of the test methods belong to one group whereas, one of the methods does not belong to any group.

When we will run the above test in eclipse normally, we will notice in the output that test execution has not considered the specified group for execution, and test methods are not executed in a group.

Therefore, if we want to execute test methods under a certain group, there are two ways by which we can execute in either one way as discussed in the following two sections.

## Running TestNG Group through Eclipse

TestNG plugin provides multiple options to run test classes. They are as follows.

**1. Class:** Using this option, we can run only a particular test class by a class name with the package.

**2. Method:** This option provides us to run only a specific method in a test class.

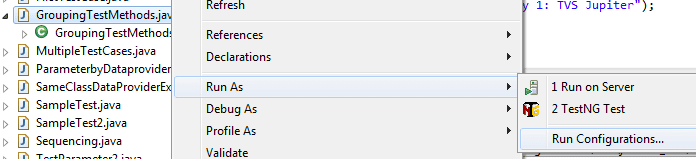
**3. Groups:** Using this option, we can run specific test methods belonging to a particular TestNG group.

**4. Package:** In case, we would want to run all the tests inside a package, we can select this option.

**5. Suite:** If we have suite files in the form of a [testng.xml file](https://www.scientecheasy.com/2020/07/create-run-testng-xml-file.html/), we can select it for execution.

Now, let us enter the earlier section. We had created a test class with certain test methods belonging to a test group. Let’s run the group of tests using eclipse step by step.

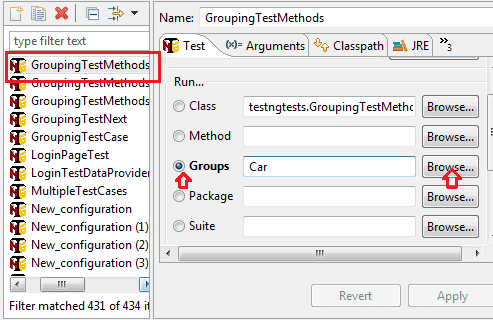
1. Right-click on test class “GroupingTestMethods” and go to Run option > Run Configurations.

[](https://www.scientecheasy.com/2020/07/testng-groups.html/)

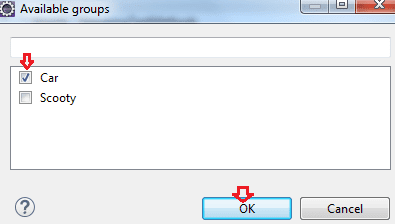
2. In the new dialog window, go to the Project section and click on the Browse button. Select the previously created project that is TestNGExamples.

If you do not get your test class name, you click on the new launch configuration and enter your test class name in the search box then go to the project section. You will get multiple options to run your test.

3. Go to the Groups option and click on the Browse button as shown in the below screenshot.

[](https://www.scientecheasy.com/2020/07/testng-groups.html/)

4. Now select the group which you would want to execute from the list. In this case, it is a test-group.

[](https://www.scientecheasy.com/2020/07/testng-groups.html/)

5. Click on the Apply button > Run. The following TestNG results will be shown on the console of Eclipse.

Output:

CAR 2: Bajaj Alto

CAR 1: Mahindra

PASSED: bajaj

PASSED: mahindra

===============================================

GRP-Car Tests run: 2, Failures: 0, Skips: 0

===============================================

Now you have successfully executed test methods belonging to a particular group using the TestNG runner configuration in Eclipse.

Let’s move to understand the second way where we will run TestNG group through testng.xml.

## Running TestNG Group through testng.xml file

This method is the most preferred and easy way to execute TestNG groups. It will run all the test methods in that group through the testng.xml file.

Follow all the steps below to run the TestNG group through testng.xml file.

**Step 1:** To create testng.xml file, right-click on the previously created test class “GroupingTestMethods.java”.

**Step 2:** Go to TestNG option and select Convert to TestNG option. A new dialog window will open in which you enter the test name “GroupingTestMethods”. Now click on the Finish button.

**Step 3:** You will notice that a new testng.xml file has created below in your project folder.

**Step 4:** Open it and modify it according to following below code.

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Suite"> <test name="GroupingTestMethods">

<groups>

<run>

<include name="Car"/>

</run>

</groups>

<classes>

<class name="testngtests.GroupingTestMethods"/>

</classes>

</test> <!-- GroupingTestMethods -->

</suite> <!-- Suite -->

**Step 5:** Now right-click on testng.xml file, go to Run As and click on TestNG Suite. You will get the following output.

**Explanation of testng.xml file:**

1. This TestNG XML file has only one test named GroupingTestMethods inside a suite.  
2. It contains a groups section that is defined by using groups tag like this <groups> & </groups> as shown in the above code.  
3. The run tag is used to run the group.  
4. The include tag represents the name of the group that needs to be executed.

## Grouping of Test Methods in Multiple Groups

In the earlier section, we have discussed the grouping of test methods that belonged to a single group but TestNG also allows grouping of test methods belonging to multiple groups.  
  
It can be done by providing the group names as an array in the groups attribute of the @Test annotation. The general syntax lets you add test methods in multiple groups.

@Test(groups = { "GroupName1", "GroupName2", "GroupName3" .... })

Let’s take an example program where we will group test methods in one or several groups.

**Program source code 2:**

package testngtests;

import org.testng.annotations.Test;

public class MultipleGroups

{

@Test(groups={"Group1"})

public void atom()

{

System.out.println("Smallest particle");

}

@Test(groups = {"Group1", "Group2"}) // The test method "electron" belongs to two groups, Group1 and Group2.

public void electron()

{

System.out.println("Negative charged particle");

}

@Test(groups = {"Group2"})

public void proton()

{

System.out.println("Positive charged particle");

}

@Test(groups = {"Group2"})

public void neutron()

{

System.out.println("Neutral particle");

}

}

The preceding class contains four test methods. Three of them belong to one group whereas, one of the methods belongs to two groups: Group1, and Group2 respectively.

Now, follow the same procedure to generate the testng.xml file for this code as discussed above.

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Suite">

<test name="Test one">

<groups>

<run>

<include name="Group1"/>

</run>

</groups>

<classes>

<class name = "testngtests.GroupingTestMethods"/>

</classes>

</test>

<test name = "Test two">

<groups>

<run>

<include name = "Group2"/>

</run>

</groups>

<classes>

<class name = "testngtests.GroupingTestMethods"/>

</classes>

</test> <!-- GroupingTestMethods -->

</suite> <!-- Suite -->

The preceding TestNG XML suite has two tests (Test one and Test two) in which each of them executing test methods belonging to a particular group.

Right-click on the testng.xml file and run it as a TestNG suite. You will get the following test results:

Output:

Groups "Group1": Smallest particle

Negative charged particle

===============================================

GRP-Particle Tests run: 2, Failures: 0, Skips: 0

===============================================

Groups "Group2": Negative charged particle

Neutral particle

Positive charged particle

===============================================

GRP-Subparticle Tests run: 3, Failures: 0, Skips: 0

===============================================

As you can observe in the previous test result, test method electron() has been executed in both tests of the test suite. This is because it belongs to both of the groups “Particle” and “Subparticle” whose test methods are executed by TestNG.

## Grouping of Test methods with Priority

You will observe in all the above output that test methods in groups have been executed in alphabetical order. If you want to execute test methods in your order then you will have to set priority with parameters like this:

@Test(groups = {“GroupName”}, priority = 0) // The test method annotated with this group will execute first.

@Test(groups = {“GroupName”}, priority = 1) // The test method annotated with this group will execute after executing the first group.

Let’s understand grouping of test methods with priority with the help of an example program.

**Program source code 3:**

package testngtests;

import org.testng.annotations.Test;

public class GroupsPriority

{

@Test(groups = {"Largest Asian Country"}, priority = 2)

public void India()

{

System.out.println("India: Delhi");

}

@Test(groups = {"Largest Asian Country"}, priority = 1)

public void china()

{

System.out.println("China: Beijing");

}

@Test(groups = {"Largest Asian Country"}, priority = 3)

public void Nepal()

{

System.out.println("Nepal: Kathmandu");

}

@Test(groups = {"Largest Asian Country"}, priority = 0)

public void Russia()

{

System.out.println("Russia: Moscow");

}

}

Output:

Russia: Moscow

China: Beijing

India: Delhi

Nepal: Kathmandu

## Inclusion & Exclusion Groups in TestNG

A group that is included in test execution is called **inclusion group**.

A group that is excluded from test execution is called **exclusion group**.

TestNG also provides features of inclusion and exclusion of groups, you can include and exclude certain groups from test execution. It helps to execute only a particular set of tests and to exclude certain tests.

For example, suppose a feature is temporarily broken during execution due to a recent change and you do not have time to fix the breakage yet but you want to have running your functional test.

In this situation, you will need to exclude these tests from execution since these tests will fail during execution.

A simple way to solve this problem is to create a group called “exclude group” and make these test methods belong to it.

Let us consider that a test method called testMethod() is now broken and we want to disable it. Then you can do like this:

@Test(groups = { "include-group",  "exclude group" })

If a test method belongs to both included and excluded group, the excluded group takes priority first and the test method excludes it from the test execution.

Once the feature is fixed, you can then reactivate the feature by just running the respective group of tests.

Now we can also set up test groups in our test suite with including or excluding. The syntax to include or exclude groups is given below:

Syntax for exclude tag:

   <exclude name = "include-group-name"/>

Syntax for include tag:

   <include name = "exclude-group-name"/>

Let’s understand it with the help of an example program and learn how to exclude a group of tests.

**Program source code 4:**

package testngtests;

import org.testng.annotations.Test;

public class ExcludeGroupTest

{

@Test(groups = {"Cricket Player"})

public void player1()

{

System.out.println("Sachin Tendulkar");

}

@Test(groups = {"Cricket Player"})

public void player2()

{

System.out.println("Virat Kohli");

}

@Test(groups = {"Cricket Player"})

public void player3()

{

System.out.println("Anil Kumble");

}

@Test(groups = {"Cricket Player", "exclude-group"})

public void player4()

{

System.out.println("Rohit Sharma");

}

}

The preceding class has four test methods. All four methods belong to the group “Cricket Player” whereas the player4() method also belongs to the group exclude-group.

Let’s set up test groups with include and exclude tags in the test suite.

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Suite">

<test name="Exclude Groups Test">

<groups>

<run>

<include name="Cricket Player"/>

<exclude name="exclude-group"/>

</run>

</groups>

<classes>

<class name="testngtests.ExcludeGroupTest"/>

</classes>

</test> <!-- Exclude Groups Test -->

</suite> <!-- Suite -->

When you will run the above testng.xml file, TestNG will execute three methods from the group “Cricket Player” and exclude the fourth method that belongs to the group exclude-group.

The following test result is generated after the execution of the above test.

Output:

Sachin Tendulkar

Virat Kohli

Anil Kumble

===============================================

Suite Total tests run: 3, Failures: 0, Skips: 0

===============================================

We can also disable tests on an individual basis by using the “enabled” property available on both @Test and @Before/After annotations.

The syntax can be like this:

@Test(groups = {"Cricket Player"}, enabled = false)

In this method, you have not to write exclude tag in the TestNG XML file.

## Default Group for all Test methods

When an entire class is added to a group, it is called **default group**. It is a good way of defining a default group for all unit tests within a class.

It saves time and typing. It can be achieved by using the @Test annotation at the class level and defining the default group in the said @Test annotation.

**Partial groups:** When you define groups at the class level and then add groups at the method level, it is called partial groups.

Let’s take an example program based on the default group.

**Program source code 5:**

package testngtests;

import org.testng.annotations.Test;

@Test(groups = {"default-group"}) // All test methods in this class should be considered as default group.

public class DefaultGroup

{

public void m1()

{

System.out.println("m1 method");

}

public void m2()

{

System.out.println("m2 method");

}

@Test(groups = {"test-group"}) // m3() belongs to both "default-group" and "test-group" (Partial group).

public void m3()

{

System.out.println("m3 method");

}

@Test

public void m4()

{

System.out.println("m4 method");

}

}

In the above class, all four test methods are the part of default group which is defined at class level, while the test method m3() belongs to both groups “default-group” and “test-group”.

When you will run default group, the following test result will be obtained.

Output:

m1 method

m2 method

m3 method

m4 method

===============================================

GRP-Default-group Tests run: 4, Failures: 0, Skips: 0

===============================================

## Groups of groups or MetaGroups

When groups include other groups, these groups are called **metagroups**. TestNG allows the users to create new groups by including and excluding certain groups and then can use them during the creation of the test suite.

Let’s understand this concept with the help of a sample test program and learn how groups can also include other groups which are called MetaGroups.

**Program source code 6:**

package testngtests;

import org.testng.annotations.Test;

public class MetaGroupsTest

{

@Test(groups = {"Group1"})

public void m1()

{

System.out.println("m1-Group1");

}

@Test(groups = {"Group1"})

public void m2()

{

System.out.println("m2-Group1");

}

@Test(groups = {"Group2"})

public void m3()

{

System.out.println("m3-Group2");

}

@Test(groups = {"Group3"})

public void m4()

{

System.out.println("m4-Group3");

}

}

Now let’s create testng.xml file and modify it.

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Suite">

<test name="MetaGroupsTest">

<groups>

<define name="metagroups1"> // define tag is used to create metagroups inside groups tag.

<include name="Group1"/>

<include name="Group2"/>

</define>

<define name="metagroups2"> // New group

<include name="metagroups1"/>

<include name="Group3"/>

</define>

<run>

<include name="metagroups2"></include>

</run>

</groups>

<classes>

<class name = "testngtests.MetaGroupsTest"/>

</classes>

</test>

</suite>

In the above test class, two groups of groups have been defined inside the test, and then these groups are used for test execution.

For example, a group “metagroups2” includes “metagroups1” and “Group3”. This group is called metagroups.

While “metagroups1” itself contains the groups “Group1” and “Group2”. So, metagroups1 is also called metagroups because it contains two groups.

**define tag** is used to create meta groups inside groups tag. The name of the meta group is defined using the name attribute under the defined tag.

Groups can be included and excluded from the new group (metagroup) by using the include and exclude tags.

Now run the testng.xml test and observe the test result on the console.

Output:

m1-Group1

m2-Group1

m3-Group2

m4-Group3

===============================================

Suite Total tests run: 4, Failures: 0, Skips: 0

===============================================

Here, TestNG executed four methods, as mentioned in the metagroups2.

We can define multiple groups of groups as you want. This feature is useful in creating specific groups for module-wise testing, regression, and sanity.

\