

TestNG XML Configuration

TestNG XML Configuration File – Complete Guide

In **TestNG**, an XML file is used to configure the test suite. This file helps you organize and control the execution of your test cases by defining:

- Which classes/methods/groups to run
- In what order they should run
- Parameters to pass
- Parallel execution

Basic Structure of testng.xml

```
xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="MySuite">
  <test name="MyTest">
    <classes>
      <class name="com.example.tests.LoginTest"/>
      <class name="com.example.tests.DashboardTest"/>
    </classes>
  </test>
</suite>
```

Components Explained

1. <suite>:

Defines the test suite (collection of tests).

- Attribute **name**: suite name.
- Optional: **parallel**, **thread-count**.

```
xml

<suite name="MySuite" parallel="tests" thread-count="2">
```

2. <test>:

Represents a group of test classes to execute.

- Attribute **name**: test group name.

- Can include **parameters**.

3. **<classes> and <class>:**

Defines the Java classes that contain the test methods.

```
xml

<classes>
  <class name="com.example.tests.LoginTest"/>
</classes>
```

4. **<methods> (optional):**

Lets you include or exclude specific test methods from a class.

```
xml

<class name="com.example.tests.LoginTest">
  <methods>
    <include name="validLogin"/>
    <exclude name="invalidLogin"/>
  </methods>
</class>
```

Example: Using Parameters

```
xml

<suite name="ParameterizedSuite">
  <test name="LoginTestWithParams">
    <parameter name="browser" value="chrome"/>
    <classes>
      <class name="com.example.tests.LoginTest"/>
    </classes>
  </test>
</suite>
```

In your Java code:

```
java

@Parameters("browser")
@Test
public void launchBrowser(String browser) {
    System.out.println("Browser is: " + browser);
}
```

Parallel Execution Example

```
xml

<suite name="ParallelSuite" parallel="classes" thread-count="3">
  <test name="ParallelTest">
    <classes>
      <class name="com.example.tests.LoginTest"/>
      <class name="com.example.tests.DashboardTest"/>
      <class name="com.example.tests.ProfileTest"/>
    </classes>
  </test>
</suite>
```

How to Run testng.xml from IntelliJ / Eclipse

1. Right-click the XML file.
2. Choose **Run** 'testng.xml'.
3. Output will show results in the console or TestNG window.

Advanced Options

◆ Grouping Tests:

```
xml

<groups>
  <run>
    <include name="smoke"/>
    <exclude name="regression"/>
  </run>
</groups>
```

Java code:

```
java

@Test(groups = {"smoke"})
public void test1() { }

@Test(groups = {"regression"})
public void test2() { }
```

◆ Multiple Test Tags (Test Different Browsers):

```
xml

<suite name="MultiBrowserSuite">
  <test name="ChromeTest">
    <parameter name="browser" value="chrome"/>
    <classes>
      <class name="com.example.tests.LoginTest"/>
    </classes>
  </test>
  <test name="FirefoxTest">
    <parameter name="browser" value="firefox"/>
    <classes>
```

```
<class name="com.example.tests.LoginTest"/>
</classes>
</test>
</suite>
```

Would you like a real-time project-based example or ready-made testng.xml files for Smoke, Regression, or E2E testing?

Great! Below are **ready-made testng.xml files** for common testing types — **Smoke, Regression,** and **End-to-End (E2E)** — based on a real-world project structure.

Assumed Java Project Structure

```
bash

src/
├── test/
│   └── java/
│       └── com/
│           └── demo/
│               ├── smoke/
│               │   └── SmokeTests.java
│               ├── regression/
│               │   └── RegressionTests.java
│               └── e2e/
│                   └── CheckoutFlowTest.java
```

Each class contains test methods grouped like this:

```
java

@Test(groups = {"smoke"})
public void loginTest() { ... }

@Test(groups = {"regression"})
public void validateCart() { ... }

@Test(groups = {"e2e"})
public void completeCheckoutFlow() { ... }
```

1. Smoke Testing – testng-smoke.xml

```
xml

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="SmokeSuite">
  <test name="SmokeTests">
    <groups>
      <run>
        <include name="smoke"/>
      </run>
    </groups>
    <classes>
      <class name="com.demo.smoke.SmileTests"/>
    </classes>
  </test>
</suite>
```

✓ 2. Regression Testing – testng-regression.xml

```
xml

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="RegressionSuite">
  <test name="RegressionTests">
    <groups>
      <run>
        <include name="regression"/>
      </run>
    </groups>
    <classes>
      <class name="com.demo.regression.RegressionTests"/>
    </classes>
  </test>
</suite>
```

✓ 3. End-to-End Testing – testng-e2e.xml

```
xml

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="E2ESuite">
  <test name="CheckoutFlowTest">
    <groups>
      <run>
        <include name="e2e"/>
      </run>
    </groups>
    <classes>
      <class name="com.demo.e2e.CheckoutFlowTest"/>
    </classes>
  </test>
</suite>
```

Optional: Combined Suite for CI Pipelines – testng-full-suite.xml

```
xml

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="FullTestSuite" parallel="tests" thread-count="3">
  <test name="SmokeTests">
    <groups>
      <run>
        <include name="smoke"/>
      </run>
    </groups>
    <classes>
      <class name="com.demo.smoke.SmokeTests"/>
    </classes>
  </test>

  <test name="RegressionTests">
    <groups>
      <run>
        <include name="regression"/>
      </run>
    </groups>
  </test>
```

```
</groups>
<classes>
  <class name="com.demo.regression.RegressionTests"/>
</classes>
</test>

<test name="E2ETests">
  <groups>
    <run>
      <include name="e2e"/>
    </run>
  </groups>
  <classes>
    <class name="com.demo.e2e.CheckoutFlowTest"/>
  </classes>
</test>
</suite>
```

Would you like me to help you generate these files based on your actual Java class/package names or guide you on how to run these with Maven or from Jenkins?

To **create a testng.xml file** in your Java automation project (using TestNG), follow these **step-by-step instructions**:

✓ Step-by-Step: Creating testng.xml

🔧 Step 1: Know Your Test Classes

Assume your test classes are:

- `LoginTest.java`
 - `SearchTest.java`
 - `CheckoutTest.java`
- All inside the package: `com.demo.tests`

📄 Step 2: Create testng.xml File

In IntelliJ:

1. Right-click on your **project root** or `src/test/resources`.
2. Choose **New** → **File**.
3. Name it: `testng.xml`
4. Paste this content:

```
xml

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="DemoTestSuite" parallel="false">

  <test name="DemoTest">
    <classes>
      <class name="com.demo.tests.LoginTest"/>
      <class name="com.demo.tests.SearchTest"/>
      <class name="com.demo.tests.CheckoutTest"/>
    </classes>
  </test>
</suite>
```

💡 Make sure the class names (with full package path) are correct.

💻 Step 3: Run testng.xml

Option 1: From IntelliJ

- Right-click on `testng.xml`
- Select **Run 'testng.xml'**

Option 2: From Terminal (Maven)

If you're using Maven and TestNG is configured in `pom.xml`:

```
bash

mvn test -DsuiteXmlFile=testng.xml
```

Bonus: Group-wise Execution Example

Java Test Methods:

```
java

@Test(groups = {"smoke"})
public void loginTest() {}

@Test(groups = {"regression"})
public void checkoutTest() {}
```

testng.xml:

```
xml

<suite name="GroupedSuite">
  <test name="SmokeGroup">
    <groups>
      <run>
        <include name="smoke"/>
      </run>
    </groups>
    <classes>
      <class name="com.demo.tests.LoginTest"/>
      <class name="com.demo.tests.CheckoutTest"/>
    </classes>
  </test>
</suite>
```

Maven Dependency (If not added):

```
xml

<dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>7.10.2</version> <!-- Use latest -->
  <scope>test</scope>
</dependency>
```

Would you like me to generate a custom `testng.xml` for your project if you share your package names and class names?

Running tests through a **TestNG XML configuration file** is one of the most powerful and flexible ways to control test execution in TestNG. Here's a full breakdown:

✓ How to Run Tests Through testng.xml

🔧 Step 1: Create the testng.xml File

Basic Example:

```
xml

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="MyTestSuite">
  <test name="MyTest">
    <classes>
      <class name="com.demo.tests.LoginTest"/>
      <class name="com.demo.tests.SearchTest"/>
    </classes>
  </test>
</suite>
```

- ◆ Make sure the **package + class names** match your project structure.

▶ Step 2: Run testng.xml

✓ Option 1: From IntelliJ IDEA or Eclipse

1. Right-click on **testng.xml**
2. Select **Run 'testng.xml'**

TestNG will automatically pick up the classes and execute the tests.

✓ Option 2: From Maven

If you're using Maven, and TestNG is set in your **pom.xml**, run:

```
bash

mvn test -DsuiteXmlFile=testng.xml
```

This will trigger only the tests defined in that XML file.

✓ Option 3: From Command Line (Using testng.jar)

If you're using a standalone JAR setup:

```
bash

java -cp "path/to/testng.jar;path/to/your/classes" org.testng.TestNG testng.xml
```

Replace paths accordingly.

Example Scenarios Using testng.xml

✓ Running Specific Methods

```
xml

<class name="com.demo.tests.LoginTest">
  <methods>
    <include name="validLogin"/>
    <include name="invalidLogin"/>
  </methods>
</class>
```

✓ Running Groups (like Smoke, Regression)

In your Java code:

```
java

@Test(groups = {"smoke"})
public void loginTest() {}
```

In testng.xml:

```
xml

<groups>
  <run>
    <include name="smoke"/>
  </run>
</groups>
```

✓ Running Tests in Parallel

```
xml

<suite name="ParallelSuite" parallel="classes" thread-count="3">
  <test name="ParallelTests">
    <classes>
      <class name="com.demo.tests.LoginTest"/>
      <class name="com.demo.tests.SearchTest"/>
      <class name="com.demo.tests.CheckoutTest"/>
    </classes>
  </test>
</suite>
```

✓ Using Parameters

```
xml

<parameter name="browser" value="chrome"/>
```

Java code:

```
java

@Parameters("browser")
@Test
public void launchBrowser(String browser) {
    System.out.println("Launching: " + browser);
}
```

Sample Maven pom.xml with TestNG

```
xml

<dependencies>
  <dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>7.10.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Summary

Step	Description
1	Create testng.xml file
2	Add suite/test/class info
3	Run via IDE, Maven, or CLI
4	(Optional) Use groups, parameters, or parallel execution

Would you like me to create a complete working mini project with **testng.xml**, test classes, and Maven setup that you can import into your IDE?

Absolutely! In **TestNG**, you can **include or exclude specific test classes and methods** directly from the **testng.xml** file. This is useful when you want to run only selected tests without modifying your Java code.

✓ Include/Exclude Classes and Methods in testng.xml

Project Example

Assume you have these classes in package `com.demo.tests`:

```
java

public class LoginTest {
    @Test
    public void validLogin() { }

    @Test
    public void invalidLogin() { }
}

public class DashboardTest {
    @Test
    public void viewDashboard() { }

    @Test
    public void logout() { }
}
```

◆ 1. Including Specific Classes Only

```
xml

<suite name="IncludeClassesSuite">
  <test name="IncludeOnlyLoginTest">
    <classes>
      <class name="com.demo.tests.LoginTest"/>
    </classes>
  </test>
</suite>
```

◆ 2. Excluding a Class

You can't directly exclude classes, but you can include only the classes you want to run, leaving others out.

To truly exclude, use method-level exclusion inside a class:

◆ 3. Including Specific Methods in a Class

```
xml

<suite name="IncludeMethodsSuite">
  <test name="LoginMethods">
    <classes>
```

```
<class name="com.demo.tests.LoginTest">
  <methods>
    <include name="validLogin"/>
  </methods>
</class>
</classes>
</test>
</suite>
```

- ◆ Only `validLogin()` will run, `invalidLogin()` will be skipped.

◆ 4. Excluding Specific Methods in a Class

```
xml

<suite name="ExcludeMethodSuite">
  <test name="LoginMethodsExclude">
    <classes>
      <class name="com.demo.tests.LoginTest">
        <methods>
          <exclude name="invalidLogin"/>
        </methods>
      </class>
    </classes>
  </test>
</suite>
```

- ◆ All methods in `LoginTest` will run except `invalidLogin()`.

◆ 5. Include & Exclude Together

```
xml

<class name="com.demo.tests.DashboardTest">
  <methods>
    <include name="viewDashboard"/>
    <exclude name="logout"/>
  </methods>
</class>
```

💡 Best practice: Use `include` if you want to run a **specific set**, use `exclude` if you want to **skip certain methods** but run the rest.



Optional: Combine Multiple Classes and Filters

```
xml

<suite name="CustomSuite">
  <test name="SelectedTests">
    <classes>
      <class name="com.demo.tests.LoginTest">
        <methods>
          <include name="validLogin"/>
        </methods>
      </class>
```

```
<class name="com.demo.tests.DashboardTest">
  <methods>
    <exclude name="logout"/>
  </methods>
</class>

</classes>
</test>
</suite>
```

Summary Table

Goal	What to Use
Run specific class	<class name="..." />
Run only some methods	<include name="methodName" />
Exclude methods from a class	<exclude name="methodName" />
Run all except one class	Just don't include that class

Would you like me to generate a **testng.xml** for your own test class names and methods if you provide them?

Running **multiple test classes** in TestNG using `testng.xml` is **simple and powerful** — you define all the classes you want to execute under the `<classes>` tag within a `<test>` block.

✓ Example: Running Multiple Test Classes

 Suppose you have these classes:

- `LoginTest.java`
- `SearchTest.java`
- `CheckoutTest.java`

All inside the package: `com.demo.tests`

 Your `testng.xml` File:


```
xml
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="MultiClassSuite">
  <test name="RunMultipleClasses">
    <classes>
      <class name="com.demo.tests.LoginTest"/>
      <class name="com.demo.tests.SearchTest"/>
      <class name="com.demo.tests.CheckoutTest"/>
    </classes>
  </test>
</suite>
```

✓ This will run all test methods in all three classes in the listed order.

✓ Optional: Run Multiple Classes in Parallel

If you want to execute these classes in parallel (multithreaded):

```
xml
<suite name="ParallelClassSuite" parallel="classes" thread-count="3">
  <test name="ParallelTests">
    <classes>
      <class name="com.demo.tests.LoginTest"/>
      <class name="com.demo.tests.SearchTest"/>
      <class name="com.demo.tests.CheckoutTest"/>
    </classes>
  </test>
</suite>
```

 This executes the classes in parallel using 3 threads.

✓ Run Specific Methods From Multiple Classes


```
xml

<suite name="PartialMethodSuite">
  <test name="RunSelectiveMethods">
    <classes>
      <class name="com.demo.tests.LoginTest">
        <methods>
          <include name="validLogin"/>
        </methods>
      </class>

      <class name="com.demo.tests.CheckoutTest">
        <methods>
          <include name="verifyCart"/>
        </methods>
      </class>
    </classes>
  </test>
</suite>
```

Running the Suite

From IntelliJ or Eclipse:

- Right-click on `testng.xml`
- Select **Run 'testng.xml'**

From Maven:

```
bash

mvn test -DsuiteXmlFile=testng.xml
```

Summary

Feature	What You Use
Run multiple classes	List under <code><classes></code>
Parallel class execution	<code>parallel="classes" thread-count="X"</code>
Run selected methods	<code><include name="methodName"/></code>

Would you like a downloadable or copy-ready sample TestNG project that includes multiple test classes and a configured `testng.xml`?

In **TestNG**, the `<suite>` and `<test>` tags in the `testng.xml` file are **essential building blocks** used to define how your tests are organized and executed.

Let's break them down in detail with examples:

Structure of testng.xml

```
xml

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="SuiteName" parallel="none" thread-count="1">
  <test name="TestName">
    <classes>
      <class name="your.package.YourTestClass"/>
    </classes>
  </test>
</suite>
```

◆ `<suite>` Tag

Purpose:

Defines the **entire test suite**. It can contain one or more `<test>` tags.

Attributes:

Attribute	Description
name	Name of the suite
parallel	Optional — values: " false " (default), " tests ", " classes ", " methods "
thread-count	Number of threads to run in parallel (used with parallel execution)
verbose	Level of logging detail (0 to 10)

✓ Example:

```
xml

<suite name="MyAutomationSuite" parallel="classes" thread-count="3">
  ...
</suite>
```

◆ `<test>` Tag

Purpose:

Defines a **set of classes** to run as part of the suite.

Each `<test>` is executed in the order it's defined (unless parallel is set).

Attributes:

Attribute	Description
name	Name of the test set (for reporting)
verbose	Optional — sets logging level

✓ Example:

```
xml

<test name="SmokeTests">
  <classes>
    <class name="com.demo.tests.LoginTest"/>
    <class name="com.demo.tests.DashboardTest"/>
  </classes>
</test>
```

You can have **multiple** `<test>` tags within a single suite.

💡 Real-Life Example: Multiple Tests in One Suite

```
xml

<suite name="EcommerceSuite" parallel="tests" thread-count="2">
  <test name="SmokeSuite">
    <classes>
      <class name="com.demo.tests.LoginTest"/>
    </classes>
  </test>

  <test name="RegressionSuite">
    <classes>
      <class name="com.demo.tests.SearchTest"/>
      <class name="com.demo.tests.CheckoutTest"/>
    </classes>
  </test>
</suite>
```

This runs **SmokeSuite** and **RegressionSuite** in parallel using 2 threads.

✓ Summary

Tag	What It Represents	Contains
<suite>	The whole test execution suite	One or more <test>
<test>	A single test group/set	One or more <classes> or <packages>

Would you like a live simulation example or a `testng.xml` that includes groups, parameters, and parallel execution together?

Great! TestNG allows you to pass **parameters** from the **testng.xml** file into your test methods using the **@Parameters** annotation. This is especially useful for things like browser name, environment, or credentials.

✓ Purpose of Parameters in testng.xml

- Customize test execution without changing code
- Externalize values like browser name, URL, user credentials, etc.

🔧 Java Code Example (Using @Parameters)

```
java

import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class LoginTest {

    @Test
    @Parameters({"username", "password"})
    public void login(String user, String pass) {
        System.out.println("Username: " + user);
        System.out.println("Password: " + pass);
    }
}
```

📄 Corresponding testng.xml File

```
xml

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="ParameterizedSuite">
  <test name="LoginTestWithParams">

    <!-- Define parameters here -->
    <parameter name="username" value="admin"/>
    <parameter name="password" value="admin123"/>

    <classes>
      <class name="com.demo.tests.LoginTest"/>
    </classes>
  </test>
</suite>
```

- ◆ Parameters must be defined inside the <test> tag
- ◆ The names must match exactly with the ones used in @Parameters("...")

🧠 What Happens If a Parameter Is Missing?

- TestNG will throw a `org.testng.TestNGException` saying the parameter is not found.
- You can avoid this using `@Optional`:

```
java

@Parameters("browser")
@Test
public void setup(@Optional("chrome") String browser) {
```

```
System.out.println("Running in browser: " + browser);
}
```

✓ Multiple Parameters

```
xml

<parameter name="env" value="staging"/>
<parameter name="browser" value="firefox"/>
```

```
java

@Parameters({"env", "browser"})
@Test
public void runTest(String env, String browser) {
    System.out.println("Environment: " + env);
    System.out.println("Browser: " + browser);
}
```

✓ Use Case: Passing Parameter to Multiple Classes

```
xml

<parameter name="browser" value="chrome"/>
<classes>
  <class name="com.demo.tests.LoginTest"/>
  <class name="com.demo.tests.SearchTest"/>
</classes>
```

Both classes will receive the **browser** parameter if they declare it.

Summary

Feature	Usage
Define in XML	<parameter name="..." value="..." /> inside <test>
Access in Java	@Parameters({}) above method
Optional fallback value	@Optional("defaultValue")
Common use cases	browser, env, user credentials, URLs

Would you like a real-world project-style example where parameters are used for cross-browser testing or test environment selection?