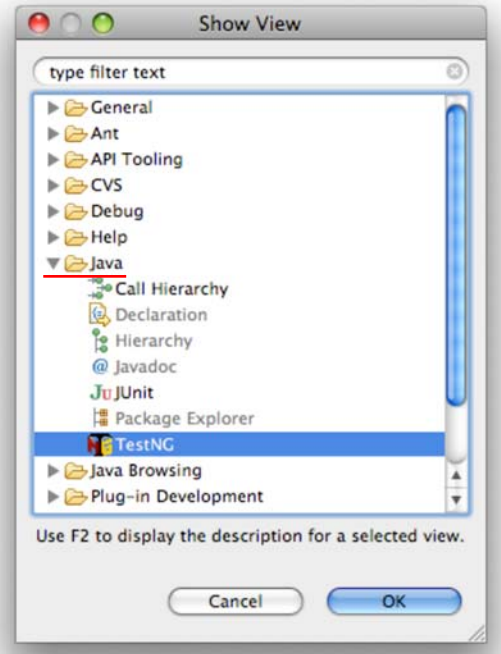
TestNG Eclipse plug-in

The TestNG Eclipse plug-in allows you to run your TestNG tests from Eclipse and easily monitor their execution and their output. It has its own project repository called `testng-eclipse`.

Table of Contents

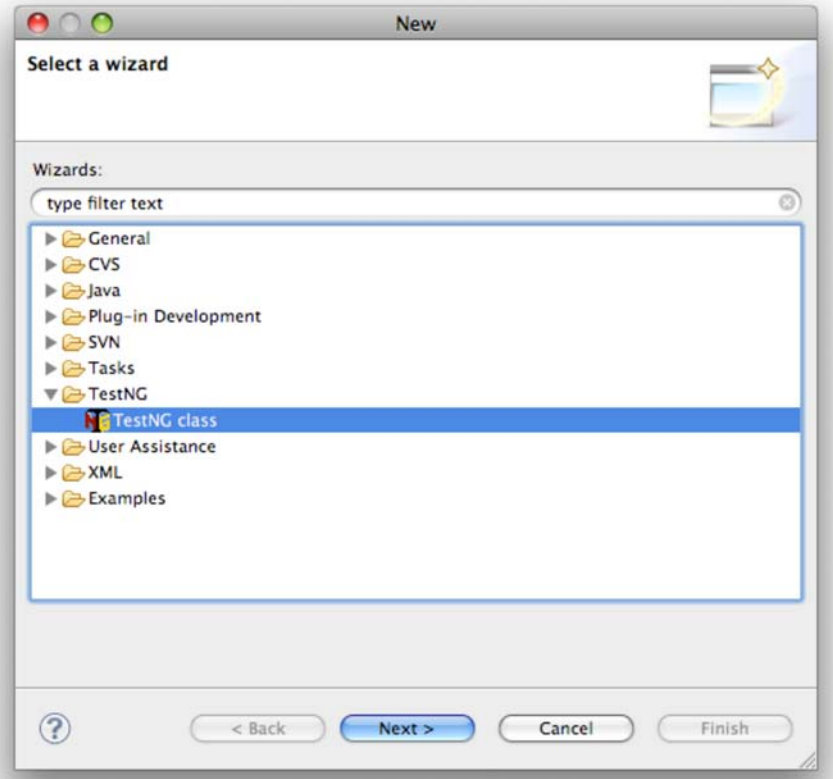
1 - Installation

Once you have installed the plug-in, restart Eclipse and select the menu `Window / Show View / Other...` and you should see the TestNG view listed in the Java category.

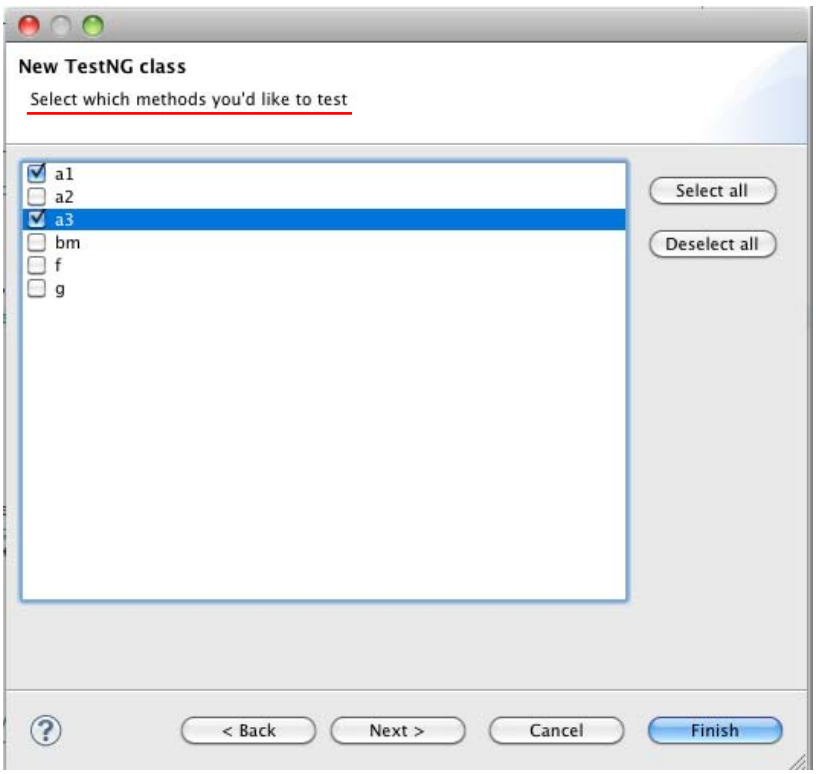


2 -Creating a TestNG class

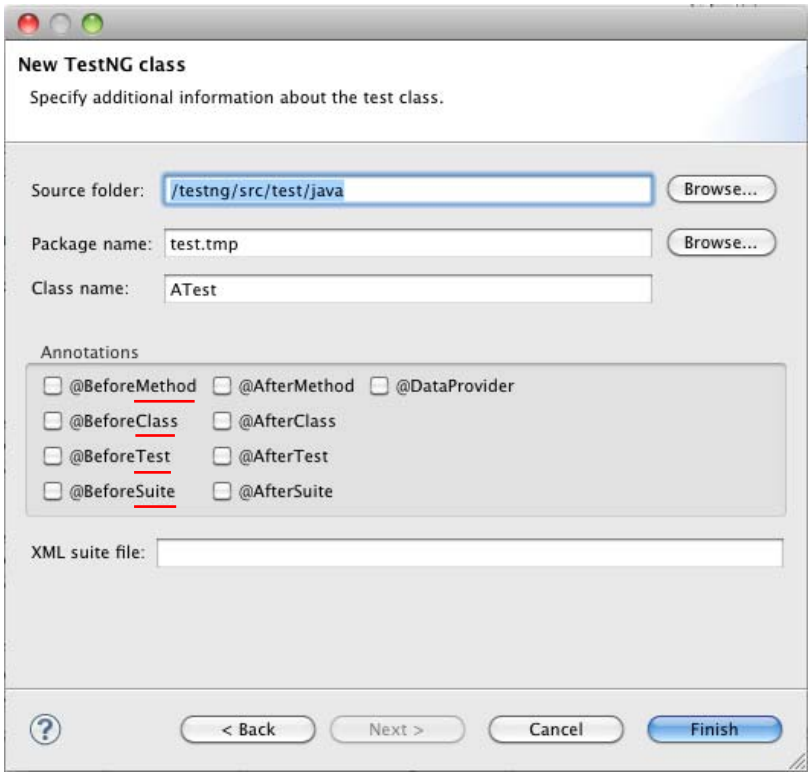
To create a new TestNG class, select the menu File / New / TestNG:



If you currently have a Java file open in the editor or if you have a Java file selected in the Navigator, the first page of the wizard will show you a list of all the public methods of that class and it will give you the option to select the ones you want to test. Each method you select on this page will be included in the new TestNG class with a default implementation that throws an exception, so you remember to implement it.



The next page lets you specify where that file will be created, whether it should contain default implementation for some configuration methods, if you'd like a data provider and finally, if a `testng.xml` file should be generated.

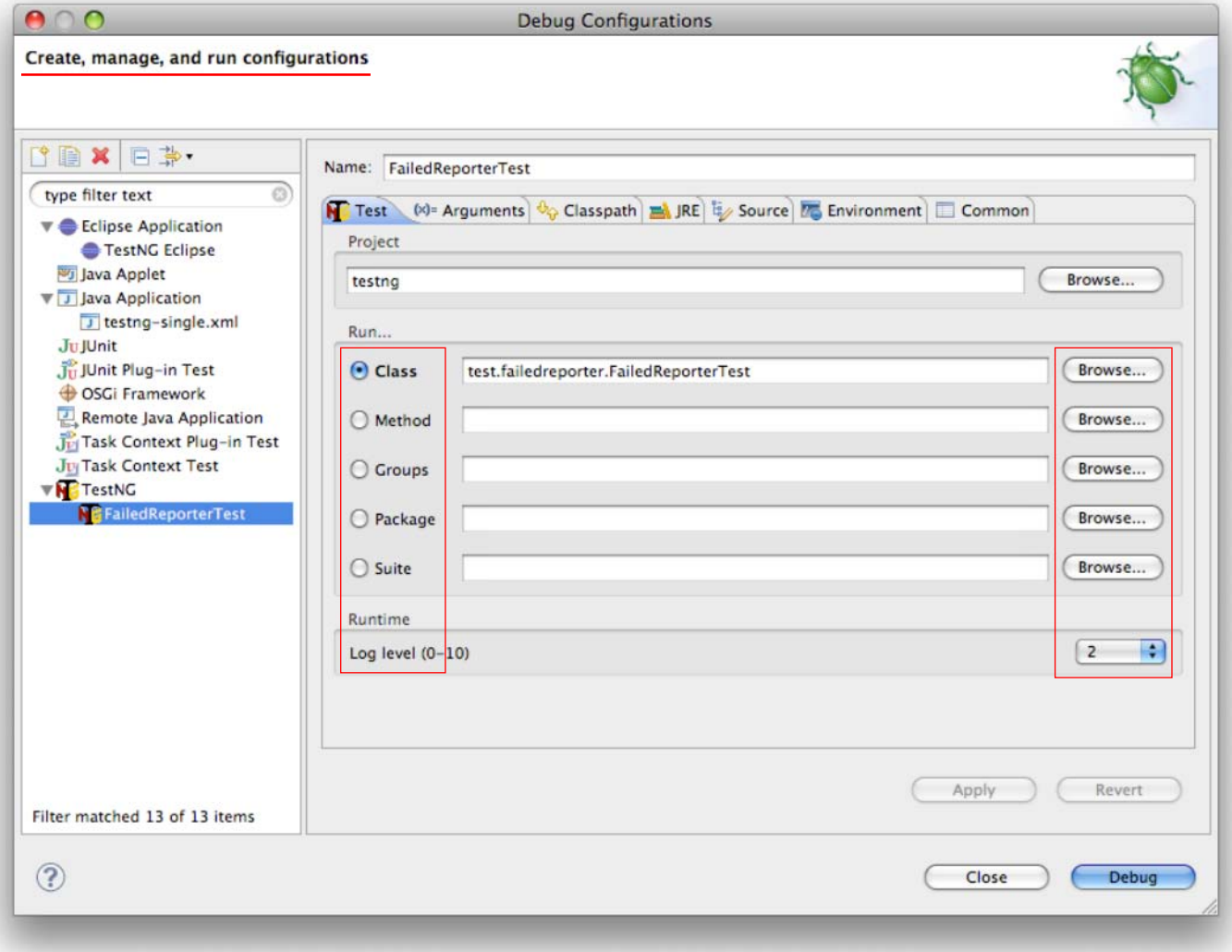


The plug-in will make a guess about the best location where this file should be created (for example, if you are using Maven, the default location will be under `src/test/java`).

3 - Launch configuration

Once you have created classes that contain TestNG annotations and/or one or more `testng.xml` files, you can create a TestNG Launch Configuration. Select the `Run / Run...` (or `Run / Debug...`) menu and create a new TestNG configuration:

从不同范围（一个类型、组、xml、单个方法）运行测试集

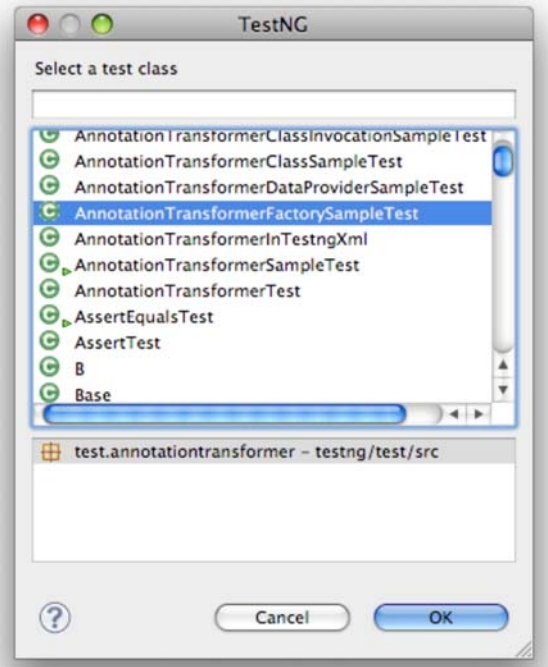


You should change the name of this configuration and pick a project, which can be selected by clicking on the `Browse...` button at the top of the window.

Then you choose to launch your TestNG tests in the following ways:

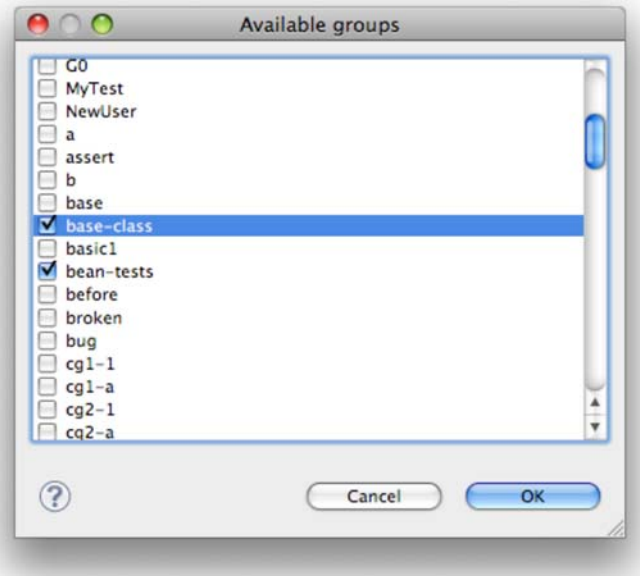
4.2.1 - From a class file

Make sure the box near `Class` is checked and then pick a class from your project. You can click on the `Browse...` button and pick it directly from a list. This list only contains classes that contain TestNG annotations:



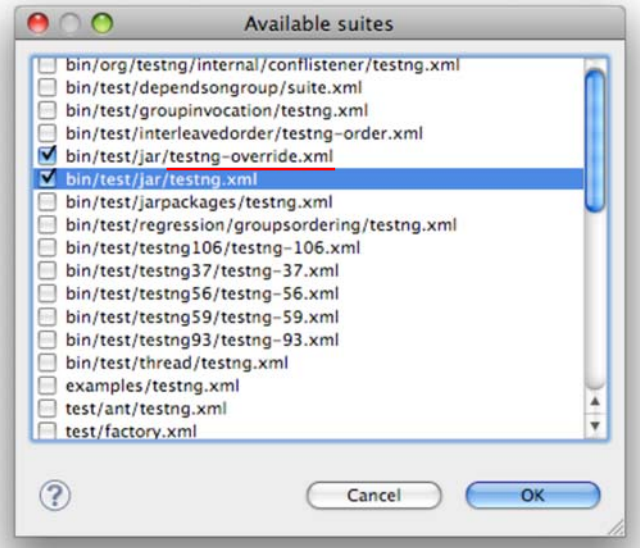
3.2 - From groups

If you only want to launch one or several groups, you can type them in the text field or pick them from a list by clicking on the `Browse...` button



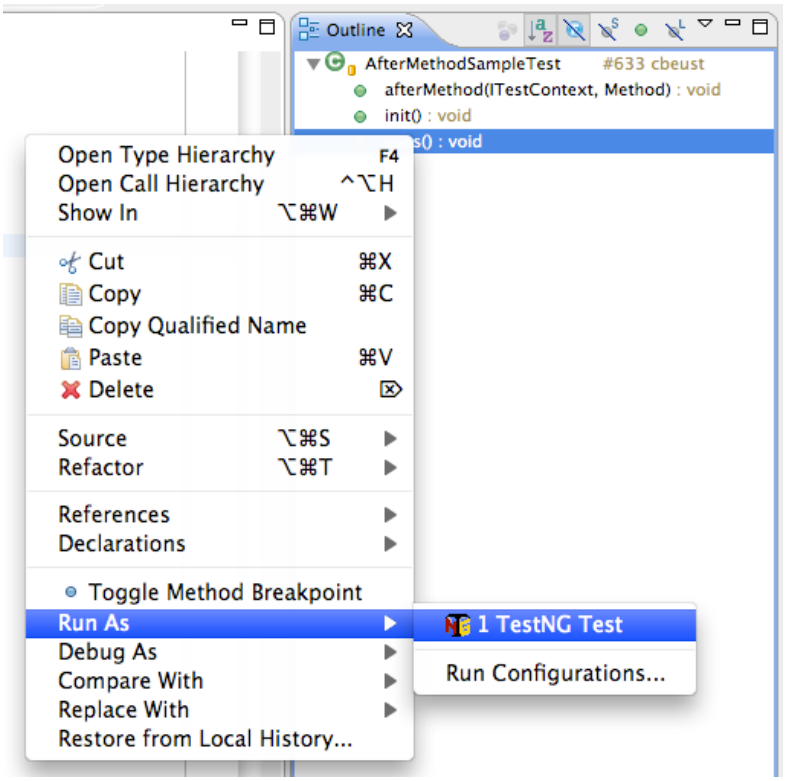
3.3 - From a definition file

Finally, you can select a suite definition from your project. It doesn't have to be named `testng.xml`, the plug-in will automatically identify all the applicable TestNG XML files in your project:



3.4 - From a method

This launch isn't accomplished from the Launch dialog but directly from your Outline view:



好用：也适用于JUnit！

You can right-click on any test methods and select `Run as... / TestNG test` and only the selected method will be run (not shown on the above screenshot because I couldn't find a way to capture a contextual menu).

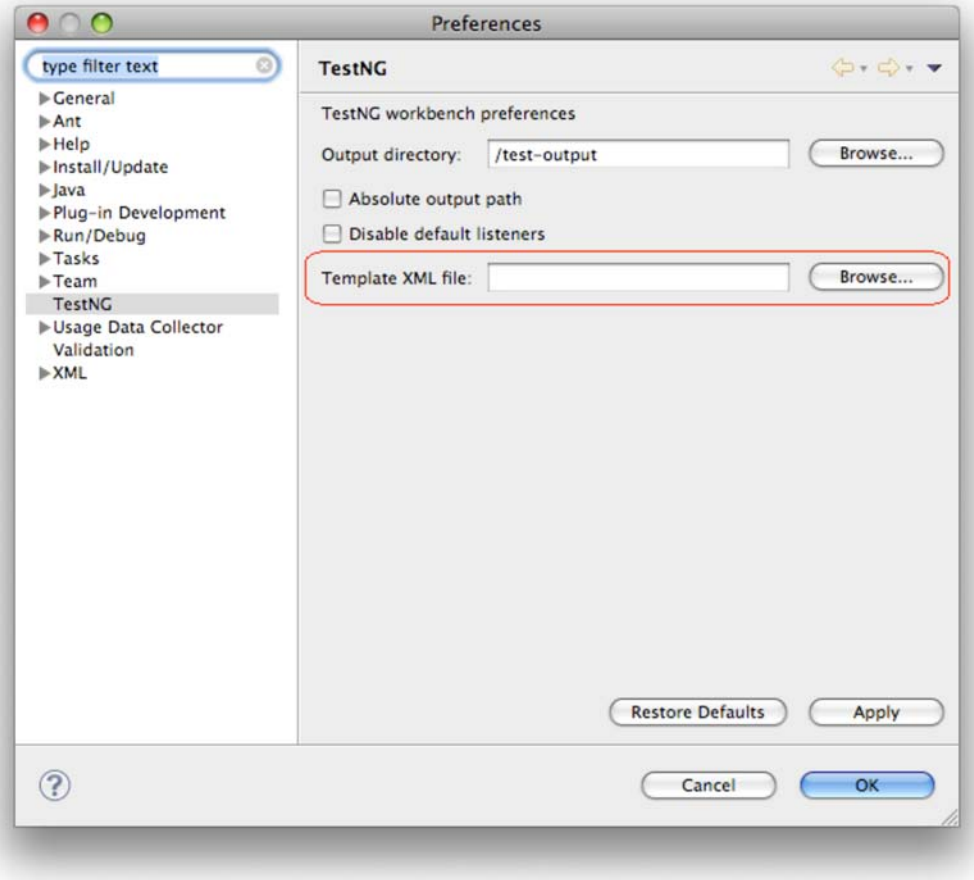
Method launching is also available from the Package Explorer view and from the Java Browser perspective.

Once you have selected one of these launches, you can also choose the logging of level. Then you can launch the tests by pressing the `Debug` (or `Run`) button, which will switch you to the Debug perspective and will open the main TestNG view.

3.5 -Specifying listeners and other settings

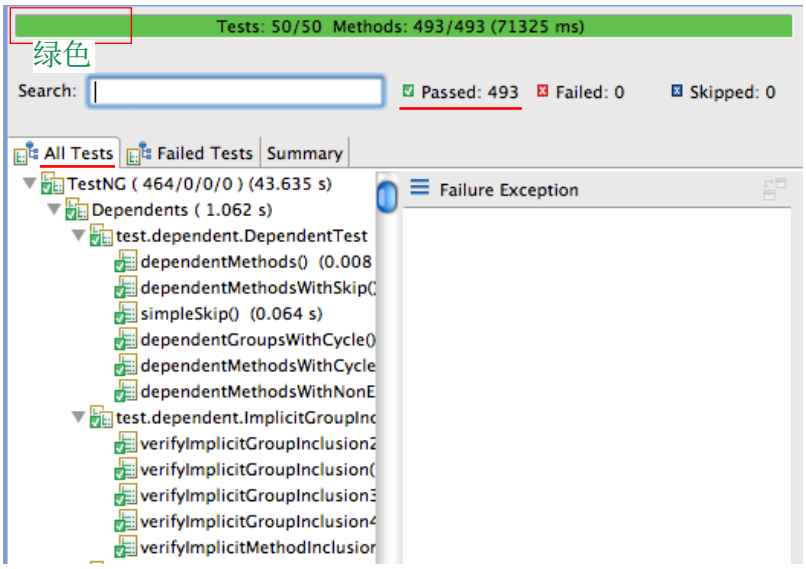
As you saw above, the plug-in will let you start tests in many different ways: from an XML file, from a method, a class, etc... When you are running an XML file, you can specify all the settings you want for this run in the XML file, but what if you want to run a package in parallel mode with specific listeners? How can you configure the settings for all the launches that are not done from an XML file?

In order to give you access to the most flexibility, TestNG lets you specify an XML suite file for all these launches, which you can find in the Preferences menu:



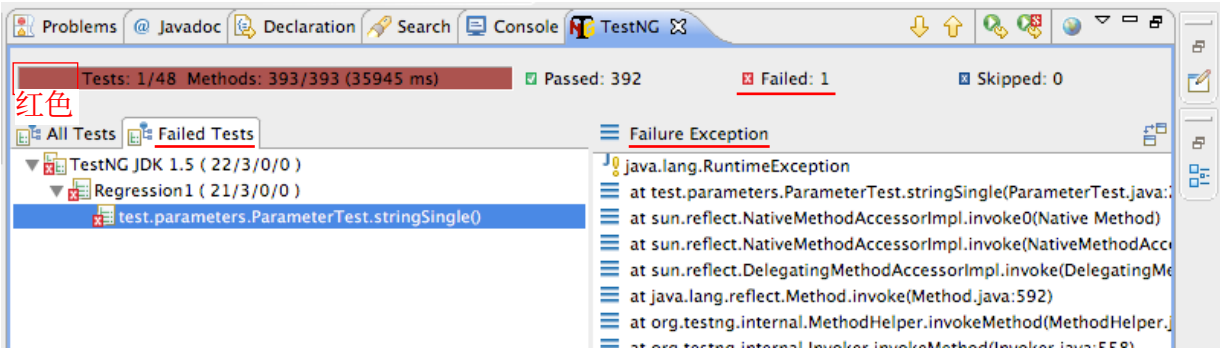
If you specify a valid suite file as "XML template file", TestNG will reuse all the settings found in this XML file, such as parallel, name, listeners, thread pool size, etc... Only the `<test>` tags in this file will be ignored since the plug-in will replace these by a generated `<test>` tag that represents the launch you chose.

4 - Viewing the test results



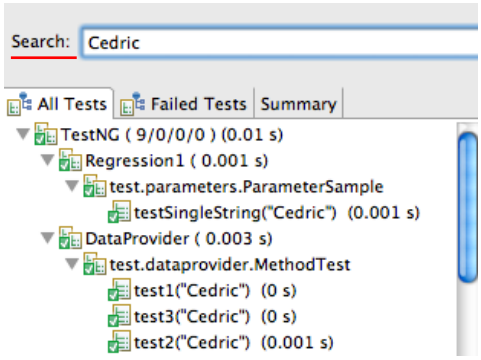
The above view shows a successful run of the tests: the bar is green and no failed tests are reported. The `All tests` tab shows you a list of all the classes and methods that were run.

If your test run contains failures, the view will look like this:



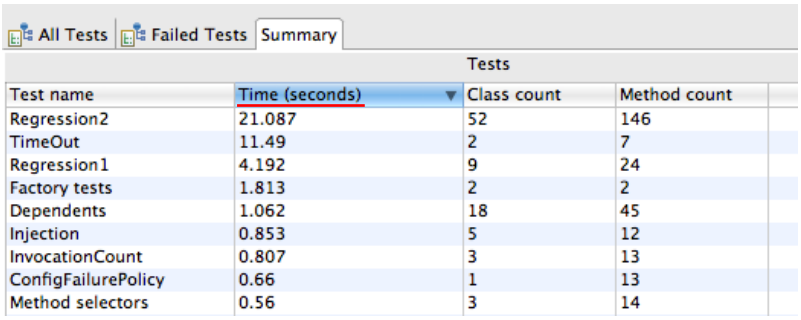
You can use the `Failed tests` tab to display only these tests that failed, and when you select such a test, the stack trace will be shown on the right-hand pane. You can double click on the offending line to be taken directly to the failure in your code.

5 - Search



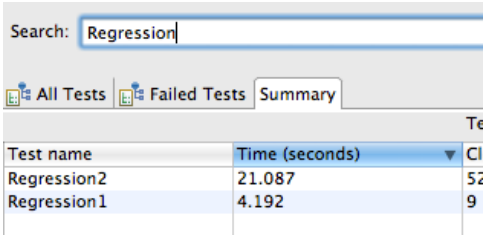
When you have hundreds of tests running, finding a specific one is not always easy, so you can type a few letters of the test method or its parameters in the Search box and the content of the tree will automatically narrow down to methods matching your search. Note in the screen shot above that the search also works on parameters provided by `@DataProvider`.

6 - Summary



The Summary tab gives you statistics on your test run, such as the timings, the test names, the number of methods and classes, etc... Since the results are shown in a table, you can also sort on any criterion you like for easier parsing. This is especially handy when you are trying to determine what tests take the longest time.

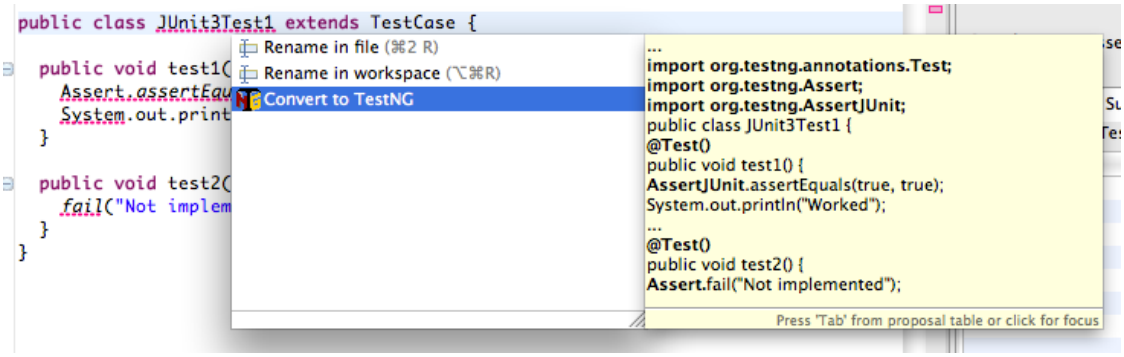
The search box works in this view as well, and note that in the screen shot below, the Time column is sorted in decreasing order:



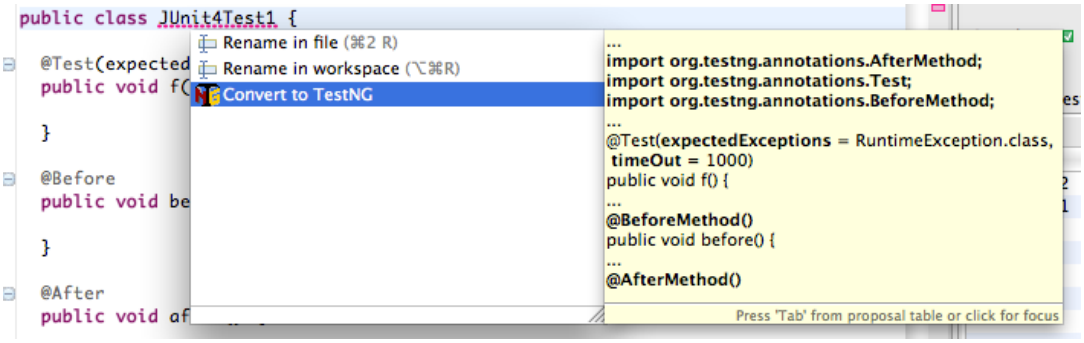
7 - Converting JUnit tests

You can easily convert JUnit 3 and JUnit 4 tests to TestNG.

Your first option is to use the Quick Fix function:

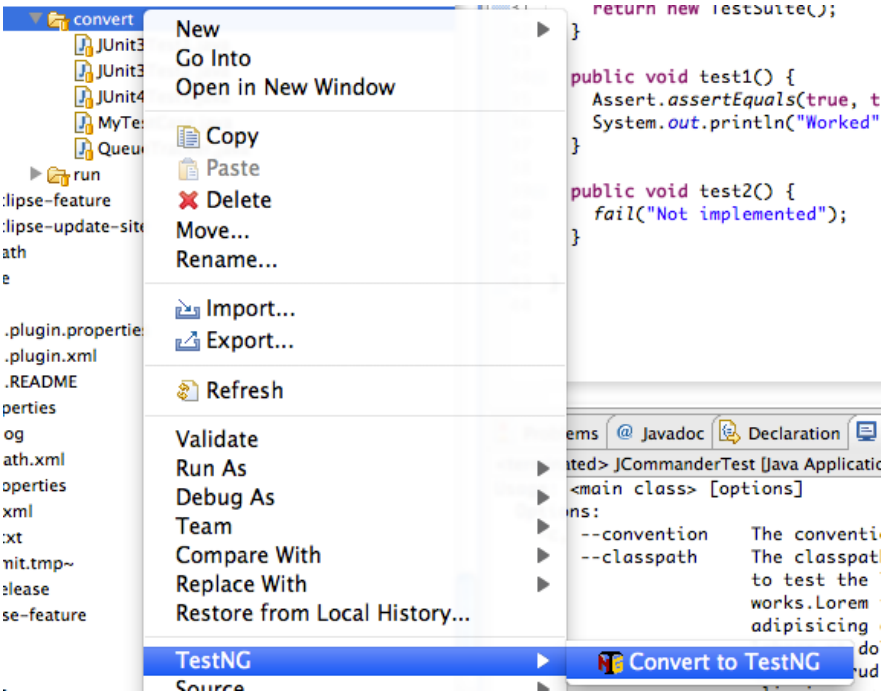


Convert from JUnit 3

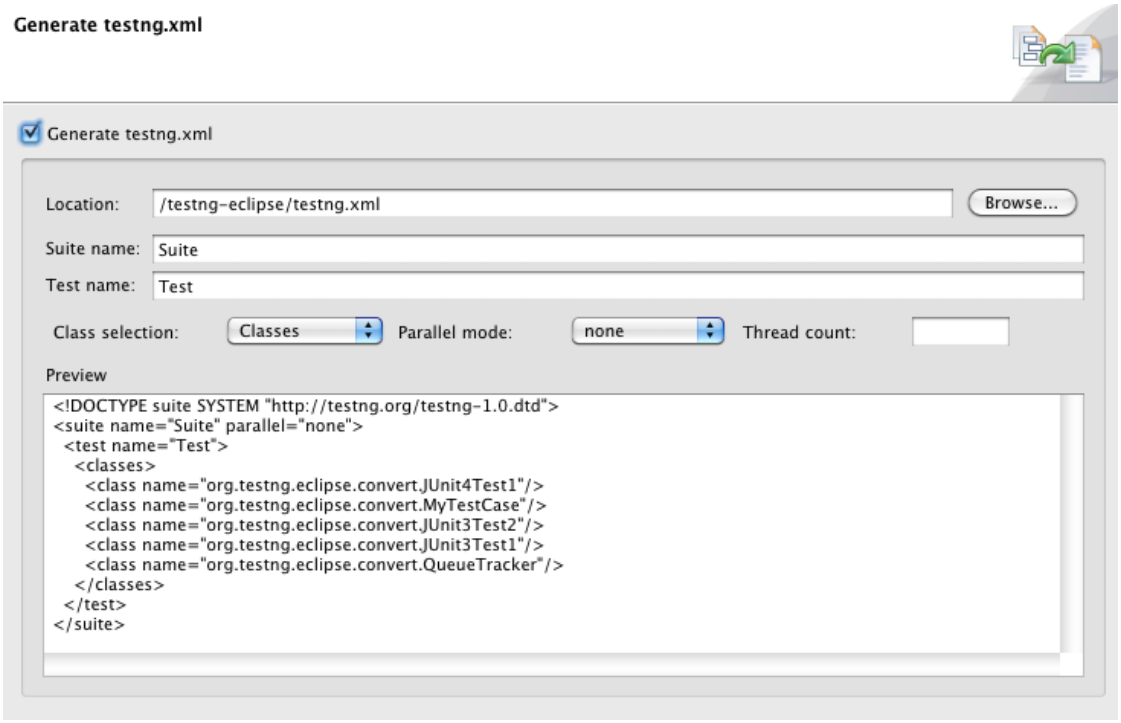


Convert from JUnit 4

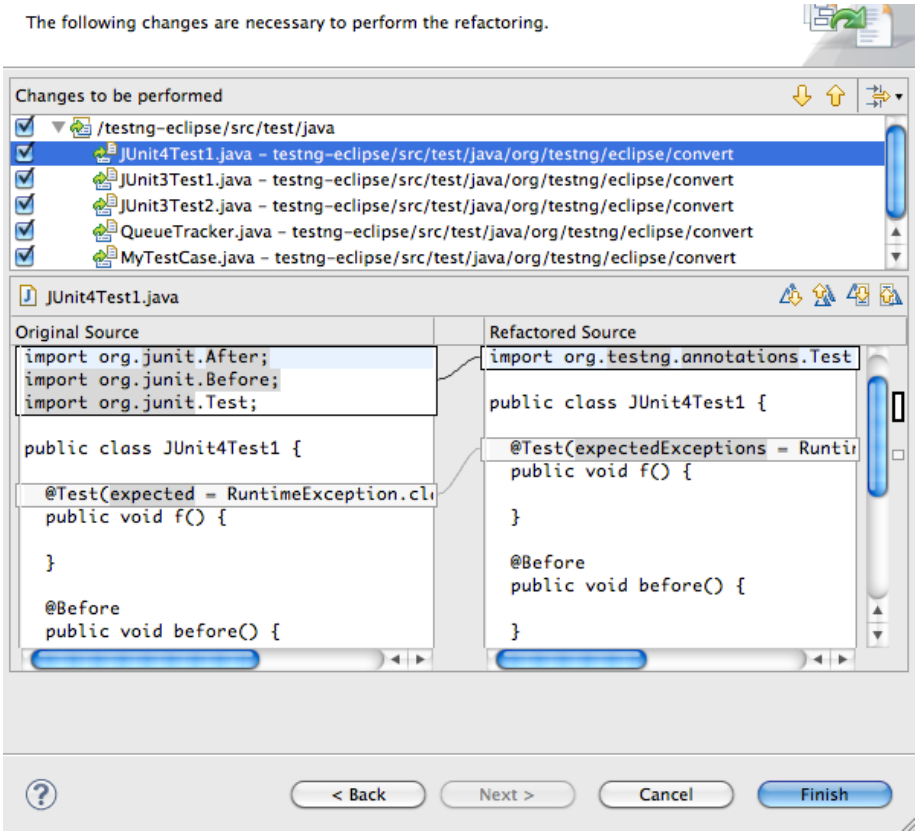
You can also convert packages or entire source folders with the conversion refactoring:



The refactoring wizard contains several pages:

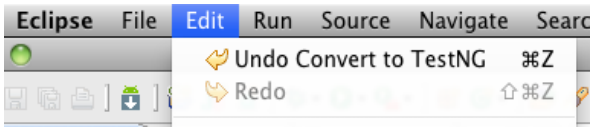


This page lets you generate a testng.xml automatically. You can configure whether to include your test classes individually or by package, the suite and test name and also whether these tests should run in parallel.



This page gives you an overview of the changes that are about to be performed. You can also decide to exclude certain files from the refactoring.

When you are done, press the "Finish" button. Like all Eclipse refactorings, you can undo all these changes in one click:



8 - Quick fixes

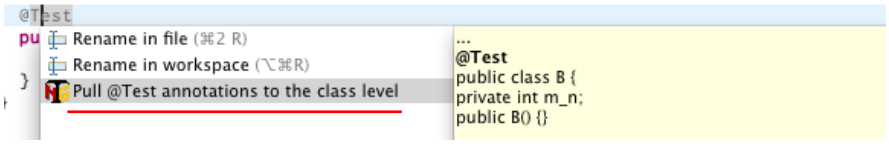
The TestNG Eclipse plug-in offers several quick fixes while you are editing a TestNG class (accessible with `Ctrl-1` on Windows/Linux and `⌘-1` on Mac OS):

Convert to JUnit

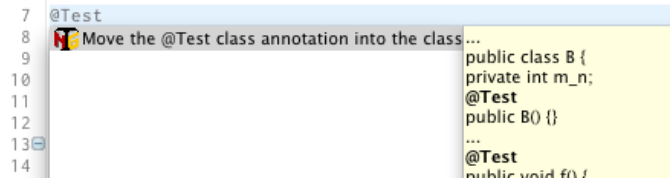
This was covered in the previous section.

Pushing and pulling `@Test` annotations

If you have several test methods annotated with `@Test` and you'd like to replace them all with a single `@Test` annotation at the class level, choose the "Pull annotation" quick fix:



Reciprocally, you can move a class level `@Test` annotation onto all your public methods:



Automatically importing `asserts`

Apply a quick fix on an assert method to automatically import it: