# What is Junco?

*Junco* (JUN-int + CO-verage) is a Surefire provider that executes JUnit tests cases and calculate coverage for each test case using Jacoco. Junco simply connects to the Jacoco agent , dump the current coverage information and resets the hit counters. Doing this each time a test case is executed, we may expect to obtain the coverage information solely for the executed test case.

The previous information is obtained in a first test run. In posterior runs, Junco execute test in order so the test cases covering most a certain source location passes first. The location is described in a .JSON file passed as parameter. Such location is called Transplantation point.

# Usage

To use Junco in our project, Jacoco and Surefire must be present in our POM and properly setup. This sections describes how to configure Jacoco and Surefire to allow Junco to do its work.

Follows the Jacoco configuration:

<plugin>

<groupId>org.jacoco</groupId>

<artifactId>jacoco-maven-plugin</artifactId>

<version>0.6.3.201306030806</version>

<executions>

<execution>

<id>pre-unit-test</id>

<goals> <goal>prepare-agent</goal> </goals>

<configuration>

*<!--*

*Sets the name of the property containing the settings*

*for JaCoCo runtime agent, which is passed as VM argument.*

*-->*

<propertyName>surefireArgLine</propertyName>

*<!-- This is needed in order to Junco connect to Jacoco-->*

<output>tcpserver</output>

</configuration>

</execution>

</executions>

</plugin>

Code Snippet 1

At this point, we have declared that we need Jacoco. Nothing unusual for regular users of the library. The only interesting thing is that the ***output*** property value must be set to ***tcpserver***.This allows Junco to connect to the Jacoco agent and obtain the coverage information. Since Junco create the reports, the configuration of Jacoco's report section is not needed.

Now the Surefire configuration:

<!-- Used for unit tests -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.15</version>

<!--

Declare Junco as Surefire provider.

Force Surefire to use Junco as the only provider

-->

<dependencies>

<dependency>

<groupId>fr.inria.juncoprovider</groupId>

<artifactId>junco-provider</artifactId>

<version>0.1</version>

</dependency>

</dependencies>

<configuration>

<!-- Properties of the Junco provider -->

<properties>

<property>

<name>html:report</name>

<value>true</value>

</property>

<!-- Transplantation point -->

<property>

<name>transplant:point</name>

<value>${basedir}/resources/transplant.json</value>

</property>

</properties>

<!-- Sets the VM argument line used when unit tests are run. -->

<argLine>${surefireArgLine}</argLine>

</configuration>

</plugin>

</plugins>

</build>

Code Snippet 2

As said before, Junco is *NOT*  a Maven plug-in: it is a Surefire provider. Surefire itself does not execute the tests, it delegate such task to their providers. Surefire find the tests, fetch dependencies and do some other dirty work like class loading. Finally, it hand over the tests to the provider, which in turn, execute them.

To force Surefire to use Junco, is first needed to declare Junco as a Surefire provider and later as a normal project dependency. In the *Code Snippet 2* is shown how to declare Junco as a Surefire provider. In the following snippet, Junco is declared as a project dependency:

<dependencies>

<dependency>

<groupId>fr.inria.juncoprovider</groupId>

<artifactId>junco-provider</artifactId>

<version>0.1</version>

</dependency>

</dependencies>

Code Snippet 3

Declaring Junco as a provider has the side effect that Surefire uses ONLY Junco as Provider. The disadvantage of this is that only the test cases that Junco may handle (JUnit 4.x) are executed.

Finally, as it may be already known, setting ***argLine*** to ***${surefireArgLine}*** is the way to connect Jacoco with Surefire. The Jacoco ***prepare-agent*** goal writes a proper VM argument string to this property, containing all the information for the VM to start the Jacoco agent. When Surefire starts, the property value is handed to the VM, making the Jacoco agent ready before test execution.

## Junco Properties

In Code Snippet 2, is shown how to set Surefire provider properties. As an example , the Junco property ***html:report*** is set to ***true.*** The complete set of Junco properties is listed in Table 1:

|  |  |  |
| --- | --- | --- |
| **Property name** | **Default value** | **Description** |
| classes:dir | "target/classes" | Indicates where to find the built classes. Needed for the report. |
| report:dir | "target/site/junco" | Sets the report directory path |
| sources:dir | "src/main" | Indicates where to find the sources. Needed for the report |
| jacoco:address | "localhost" | Indicates the TCP/IP address of the Jacoco agent. Default value is good in most cases. (See Jacoco documentation) |
| jacoco:port | "6300" | Indicates the TCP/IP port of Jacoco. Default value is good in most cases. (See Jacoco documentation) |
| html:report | "false" | True to generate the html report. By default Junco only generates the xml report. |
| transplant:point | N/A (MUST BE SET) | Path for file containing the transplantation point (source location) |

Table 1

## Transplantation point file format

The JSON file is as simple as:

{

"Position": "<*class canonical name*>:<*line number*>"

}

For example, in the test project coming along with Junco, the transplantation point is the line 15 of the fr.inria.testproject.Arithmetic class. So the file contain is as follows:

{

"Position": "fr.inria.testproject.Arithmetic:15"

}