Cobertura Usage

**Plugin Information**

|  |  |
| --- | --- |
| **Plugin ID** | cobertura |
| **Latest Release** | 1.6-h-3 |
| **Latest Release Date** | Jan 14, 2013 |
| **Plugin Central** | [Plugin Central 3.2](http://hudson-ci.org/PluginCentral/site/3.2/) |
| **Sources** | [Github](https://github.com/hudson3-plugins/cobertura-plugin) |
| **Support** | [Eclipse Jenkins Forum](http://www.eclipse.org/forums/index.php?t=thread&frm_id=229) |
| **Issue Tracking** | [Eclipse Bugzilla](https://bugs.eclipse.org/bugs/enter_bug.cgi?product=Hudson) |
| **Jenkins Core (latest)** | [3.2.2](https://www.eclipse.org/hudson/download.php) |

This plugin allows you to capture code coverage report from [Cobertura](http://cobertura.sourceforge.net/). Jenkins will generate the trend report of coverage.  
The Cobertura plug in can be [downloaded here](http://hudson-ci.org/download/plugins/cobertura/).

**Configuring the Cobertura Plugin**

1. Install the cobertura plugin (via Manage Jenkins -> Manage Plugins)
2. Configure your project's build script to generate cobertura XML reports (See below for examples with Ant and Maven2)
3. Enable the "Publish Cobertura Coverage Report" publisher
4. Specify the directory where the coverage.xml report is generated.
5. (Optional) Configure the coverage metric targets to reflect your goals.

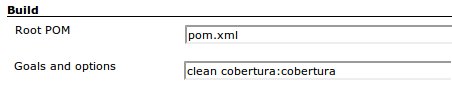
**Configuring build tools**

Here are the configuration details for common build tools. Please feel free to update this with corrections or additions.

**Maven 2**

**Quick configuration**

You can either, enable "cobertura" analysis in your 'pom.xml' files or just tell Jenkins to run "cobertura" goal.

If you don't want to change your pom files, just add the goal 'cobertura:cobertura' to your Maven project in Jenkins.  


**Single Project**

If you are using a single module configuration, add the following into your pom.xml. This will cause cobertura to be called each time you run "mvn package".

<project ...>

...

<build>

...

<plugins>

...

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>cobertura-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<formats>

<format>xml</format>

</formats>

</configuration>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>cobertura</goal>

</goals>

</execution>

</executions>

</plugin>

...

</plugins>

...

</build>

...

</project>

**Execute cobertura only from Jenkins using profiles**

You can reduce the load of your developer machine by using maven profiles to execute the plugin only if you are running within Jenkins. Theconfiguration below shows how to enable the plugin based on the information if maven was started by Jenkins.

<project ...>

...

<profiles>

<!-- Jenkins by default defines a property BUILD\_NUMBER which is used to enable the profile. -->

<profile>

<id>Jenkins</id>

<activation>

<property>

<name>BUILD\_NUMBER</name>

</property>

</activation>

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>cobertura-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<formats>

<format>xml</format>

</formats>

</configuration>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>cobertura</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

</profile>

</profiles>

...

</project>

**Project hierarchies**

If you are using a common parent for all Maven2 modules you can move the plugin configuration to the pluginManagement section of the common *parent*...

<project ...>

...

<build>

...

<pluginManagement>

<plugins>

...

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>cobertura-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<formats>

<format>xml</format>

</formats>

</configuration>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>cobertura</goal>

</goals>

</execution>

</executions>

</plugin>

...

</plugins>

</pluginManagement>

...

</build>

...

</project>

 And add the plugin group and artifact to the children

<project ...>

...

<build>

...

<plugins>

...

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>cobertura-maven-plugin</artifactId>

</plugin>

...

</plugins>

...

</build>

...

</project>

**Execute cobertura only from Jenkins using profiles**

It is highly recommend to reduce the workload of the developers machines by disabling the cobertura plugin and only using it from within Jenkins. The following excerpt from the *parent* shows how to do so:

<project ...>

...

<profiles>

<!-- Jenkins by default defines a property BUILD\_NUMBER which is used to enable the profile. -->

<profile>

<id>Jenkins</id>

<activation>

<property>

<name>BUILD\_NUMBER</name>

</property>

</activation>

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>cobertura-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<formats>

<format>xml</format>

</formats>

</configuration>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>cobertura</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</pluginManagement>

</build>

</profile>

</profiles>

...

</project>

 Now that your parent is only using the plugin management section if it is running from within Jenkins, you need the childern poms adapted as well:

<project ...>

...

<!-- If we are running in Jenkins use cobertura. -->

    <profiles>

        <profile>

            <id>Jenkins</id>

            <activation>

                <property>

                    <name>BUILD\_NUMBER</name>

                </property>

            </activation>

            <build>

                <plugins>

                    <plugin>

                        <groupId>org.codehaus.mojo</groupId>

                        <artifactId>cobertura-maven-plugin</artifactId>

                    </plugin>

                </plugins>

            </build>

        </profile>

    </profiles>

...

</project>

**Ant**

You must first tell Ant about the Cobertura Ant tasks using a taskdef statement. The best place to do this is near the top of your build.xml script, before any target statements.

<property name="cobertura.dir" value="C:/javastuff/cobertura" />

<path id="cobertura.classpath">

<fileset dir="${cobertura.dir}">

<include name="cobertura.jar" />

<include name="lib/\*\*/\*.jar" />

</fileset>

</path>

<taskdef classpathref="cobertura.classpath" resource="tasks.properties" />

You'll need to instrument the classes that JUnit will be testing (not the test classes themselves) as such:

<cobertura-instrument todir="${instrumented.dir}">

<ignore regex="org.apache.log4j.\*" />

<fileset dir="${classes.dir}">

<include name="\*\*/\*.class" />

<exclude name="\*\*/\*Test.class" />

</fileset>

<fileset dir="${guiclasses.dir}">

<include name="\*\*/\*.class" />

<exclude name="\*\*/\*Test.class" />

</fileset>

<fileset dir="${jars.dir}">

<include name="my-simple-plugin.jar" />

</fileset>

</cobertura-instrument>

Here's an example call to the JUnit ant task that has been modified to work with Cobertura.

<junit fork="yes" dir="${basedir}" failureProperty="test.failed">

<!--

Specify the name of the coverage data file to use.

The value specified below is the default.

-->

<sysproperty key="net.sourceforge.cobertura.datafile"

file="${basedir}/cobertura.ser" />

<!--

Note the classpath order: instrumented classes are before the

original (uninstrumented) classes. This is important.

-->

<classpath location="${instrumented.dir}" />

<classpath location="${classes.dir}" />

<!--

The instrumented classes reference classes used by the

Cobertura runtime, so Cobertura and its dependencies

must be on your classpath.

-->

<classpath refid="cobertura.classpath" />

<formatter type="xml" />

<test name="${testcase}" todir="${reports.xml.dir}" if="testcase" />

<batchtest todir="${reports.xml.dir}" unless="testcase">

<fileset dir="${src.dir}">

<include name="\*\*/\*Test.java" />

</fileset>

</batchtest>

</junit>

Finally, you need a task to generate the xml report, where 'destdir' is where you want the report (coverage.xml) generated.

Your cobertura.ser is generated to your module root.

srcdir is where your \*.java files are located. If you use multiple modules in one build process you need to include the module name, if you use the simple srcdir parameter. It is not required to include module name if you use fileset.

<cobertura-report format="xml" destdir="${coveragereport.dir}" srcdir="${src.dir}" />

You can use multiple source directories this way:

<cobertura-report format="xml" destdir="${coveragereport.dir}" >

<fileset dir="${src.dir.java}">

<include name="\*\*/\*.java" />

</fileset>

<fileset dir="${src.dir.main}">

<include name="\*\*/\*.java" />

</fileset>

</cobertura-report>