



Documentation for Clover 4.0

# Contents

About Clover	7
About Code Coverage	8
About Distributed Per-Test Coverage	10
About Test Optimization	10
About Clover Editions	13
About Clover code metrics	13
Supported Platforms	16
End of Platform Support Announcements	17
Clover-for-Ant	18
Clover-for-Ant User's Guide	19
1. QuickStart Guide	20
Clover for Ant Best Practices	22
Clover-for-Ant Two Line Integration	24
Test Optimization Quick Start Guide for Ant	25
2. Using Clover Interactively	27
3. Using Clover in Automated Builds	29
4. Understanding Reports	30
'Current' Report	30
Coverage Legend	40
Dashboard Widgets	41
Source Cross-Referencing in Reports	42
Stack Trace Navigation	43
Tag Clouds	43
'Historical' Report	44
5. Configuring Reports	48
Unit Test Results and Per-Test Coverage	49
Using Coverage Contexts	51
Using Spans	55
Specifying an Interval	55
Sharing Report Formats	56
Extracting Coverage Data programmatically	57
6. Ant Task Reference	58
clover-check	59
clover-clean	62
clover-env	63
clover-historypoint	64
clover-html-report	66
clover-instr	68
clover-log	76
clover-merge	80
clover-pdf-report	81
clover-report	82
<added> element	97
<chart> element	98
<columns> element	98
<format> element	101
<movers> element	102
JSON reference	102
clover-setup	111
Clover test detection	121
methodContext	122
profiles	123
statementContext	124
clover-snapshot	125
7. Ant Type Reference	125
clover-columns	126

clover-format	126
clover-optimized-testset	126
8. Controlling Clover at Runtime	127
Clover Performance Tuning	127
Coverage Recorders	131
Managing the Coverage Database	135
Using a Flush Policy	136
Using Source Directives	137
Working with Distributed Applications	137
Using Distributed Per-test Coverage with Clover-for-Ant	144
Working with Restricted Security Environments	146
Working in OJVM	146
9. Clover Target Reference	147
A. Integrating Clover-for-Ant with other tools	150
Integrating Clover with JUnit4 Parameterized Tests	150
Using Clover-for-Ant with GWT	153
Clover-for-Ant Installation Guide	155
Adding to Ant's build.xml	156
Adding to Ant's Classpath	157
Clover-for-Ant Upgrade Guide	158
Clover-for-Ant Changelog	158
Changes in Clover-for-Ant 4.0.0	159
Changes in Clover-for-Ant 3.3.0	160
Clover-for-Maven 2 and 3	160
About Clover-for-Maven 2 and 3	162
Clover-for-Maven 2 and 3 Quick Start Guide	162
Clover-for-Maven 2 and 3 User's Guide	163
Basic usage	164
Configuring instrumentation	167
Configuring reports	170
Creating custom reports	173
Configuring a coverage goal	176
Using Test Optimization in Maven	176
Using Distributed Per-test Coverage	180
Using Clover in various environment configurations	183
Using Clover for web applications	203
Best Practices for Maven	205
Compiling Groovy with GMaven plugin	210
Compiling Groovy with Groovy Eclipse Plugin	212
Using with Surefire and Failsafe Plugins	218
Using Clover with the GWT-maven plugin	224
Using Clover with JAXB plugin	227
Using Clover with Maven + surefire-test + inner test classes	231
Using Clover with the maven-bundle-plugin	232
Using Clover via the maven-antrun-plugin	232
Using Clover with Maven Tycho Plugin	233
Clover-for-Maven 2 and 3 Installation Guide	238
Clover-for-Maven 2 and 3 Upgrade Guide	238
Clover-for-Maven 2 and 3 Changelog	239
Changes in Clover-for-Maven 4.0.0	240
Changes in Clover-for-Maven 3.3.0	240
Clover-for-Maven 2 and 3 FAQ	240
Clover-for-Eclipse	241
Clover-for-Eclipse User's Guide	242
1. Clover for Eclipse in 10 minutes	242
2. Exploration of coverage in Eclipse	253
3. Exploration of test results in Eclipse	263
4. Scope of instrumentation in Eclipse	266
5. Eclipse configuration options	271
6. Generating reports in Eclipse	281
7. Test Optimization for Eclipse	287
8. Launching an Ant build from Eclipse	294

9. Eclipse advanced topics .....	294
Instrumenting RCP Application .....	295
Performance Tuning in Clover for Eclipse .....	303
Clover-for-Eclipse Installation Guide .....	304
Installing Clover-for-Eclipse .....	306
Clover-for-Eclipse Upgrade Guide .....	307
Clover-for-Eclipse Changelog .....	307
Changes in Clover-for-Eclipse 4.0.0 .....	308
Changes in Clover-for-Eclipse 3.3.0 .....	309
Clover-for-Eclipse Glossary of Terms .....	309
Clover-for-Eclipse FAQ .....	311
Clover-for-IDEA .....	312
Clover-for-IDEA User's Guide .....	313
1. Clover for IDEA in 10 minutes .....	314
2. Exploration of coverage in IDEA .....	315
3. Exploration of test results in IDEA .....	319
4. Scope of instrumentation in IDEA .....	321
5. IDEA configuration options .....	326
Clover-for-IDEA Auto-Updates .....	331
6. Generating reports in IDEA .....	332
7. Test Optimization for IDEA .....	335
8. Launching Ant build from IDEA .....	340
9. IDEA Advanced topics .....	340
Performance Tuning in Clover for IDEA .....	340
Clover-for-IDEA Installation Guide .....	340
Clover-for-IDEA Upgrade Guide .....	343
Clover-for-IDEA Changelog .....	345
Changes in Clover-for-IDEA 4.0.0 .....	346
Changes in Clover-for-IDEA 3.3.0 .....	347
Clover-for-IDEA Glossary of Terms .....	347
Clover-for-IDEA FAQ .....	348
Clover-for-Grails .....	348
Clover-for-Grails Quick Start Guide .....	350
Clover-for-Grails User's Guide .....	351
Configuration options .....	351
Advanced report configuration .....	352
Advanced setup configuration .....	354
Configuring method context filters .....	356
Test Optimization with Clover-for-Grails .....	357
Clover-for-Grails Installation Guide .....	360
Clover-for-Grails Upgrade Guide .....	363
About Clover-for-Grails .....	364
Clover-for-Grails Changelog .....	364
Changes in Clover-for-Grails 4.0.0 .....	365
Changes in Clover-for-Grails 3.3.0 .....	365
Clover Command Line Tools .....	366
CloverInstr .....	367
CloverMerge .....	368
ConsoleReporter .....	370
HtmlReporter .....	371
JSONReporter .....	373
PDFReporter .....	374
XMLReporter .....	376
Bamboo Clover Plugin .....	378
Gradle Clover Plugin .....	378
Griffon Clover Plugin .....	379
Hudson Clover Plugin .....	379
Jenkins Clover Plugin .....	379
Sonar Clover Plugin .....	379
Clover Release Notes .....	379
Clover 4.0 Release Notes .....	382
A side-by-side comparison of the Classic and the ADG HTML report .....	387



Clover 3.3 Release Notes	393
Clover 3.2 Release Notes	397
Clover 3.1 Release Notes	400
Clover 3.0 Release Notes	404
Clover 2.6 Release Notes	409
Clover 2.5 Release Notes	411
Clover 2.4 Release Notes	412
Clover 2.3 Release Notes	414
Clover 2.2 Release Notes	415
Clover 2.1 Release Notes	417
Clover 2.0 Release Notes	418
Clover Release Summary	421
Clover Tutorials	421
Clover-for-Ant tutorial	422
Part 0 - Clover in 10 minutes	422
Part 1 - Measuring Coverage	424
Part 2 - Historical Reporting	430
Part 3 - Automating Coverage Checks	435
Part 4 - Test Optimization Tutorial	437
Clover-for-Maven tutorials	440
How to configure your Clover license	441
Hacking Clover	442
Clover-for-Android	442
Clover-for-Scala	454
Converting XML to database format	455
Measuring per-test coverage for manual tests	456
Updating optimization snapshot file	458
Using Clover for other programming languages	459
Instrumenting JSP files	459
Using Clover for PHP	461
Glossary	463
branch coverage	463
code coverage	463
coverage	464
coverage clouds	464
decision coverage	464
history point	464
interval	464
method coverage	464
span	464
statement coverage	464
test coverage	464
Clover FAQ	464
Concepts & Usage FAQ	466
Can I create a Clover Report on Server A if I have the clover.db which I generated on Server B?	467
Does Clover depend on JUnit?	468
Does Clover integrate with Maven?	468
Does Clover support the new language features in JDK1.5?	468
Does Clover work with JUnit4 and TestNG?	468
How are the Clover coverage percentages calculated?	468
How do I compare the code coverage between two releases of my code?	468
How do I get started with Clover?	468
How do I use Clover with NetBeans?	468
What are the limitations of Code Coverage?	470
What does the name "Clover" mean?	470
What is Code Coverage Analysis?	470
What is the coverage.db file and why am I seeing files like coverage.dbxxxxxxxx_xxxx_xxxx?	470
What third-party libraries does Clover utilise?	470
Where did Clover originally come from?	471
Why does Clover instrument classes I have excluded using the 'exclude' element of the 'clover-setup' task?	471
Why does Clover use source code instrumentation?	471

Will Clover integrate with my IDE? .....	472
Eclipse Plugin FAQ .....	472
I only need instrumented classes for unit testing and I don't want to risk publishing them to my production environment. How can I do this with Clover? .....	472
Is Clover supported on IBM's RAD? .....	473
I store my plugins and features in an Eclipse extension area. Does Clover support this? .....	473
Why can I only see coverage data for the last test case I executed? .....	473
IDEA Plugin FAQ .....	473
I've run my tests, but coverage information does not show in IDEA .....	473
What does enabling Instrument Test Source Folders do? .....	473
Where does IDEA write its log file? .....	473
Maven 2 and 3 Plugin FAQ .....	474
Deploying Instrumented Jars .....	474
How to keep Clover reports between builds? .....	475
How to remove -clover suffix from artifact name? .....	475
Is there an alternative to using the Maven Central repository? .....	476
Preparing multi-module projects for remote deployment with Clover-for-Maven 2 .....	477
Troubleshooting License problems .....	477
Troubleshooting problems with displaying characters .....	478
Support Policies .....	478
Bug Fixing Policy .....	479
How to Report a Security Issue .....	480
New Features Policy .....	480
Security Advisory Publishing Policy .....	481
Security Update Policy .....	482
Severity Levels for Security Issues .....	482
Update Policy .....	483
Troubleshooting .....	483
Compiling my instrumented sources fails with a 'code too large' error. ....	484
For some statements in my code Clover reports "No Coverage information gathered for this expression". What does that mean? .....	484
Hit count for multi-threaded test is incorrect in Clover's report. ....	485
I'm trying to get a coverage report mailed, but I keep getting "mail Failed to send email". How do I fix this? .....	485
I'm using the maven-clover-plugin version 2.4 with a license downloaded from Atlassian and get the message 'Invalid or missing License' .....	485
Tools for Troubleshooting Clover-for-Ant .....	486
Two questions to ask yourself first when troubleshooting Clover .....	486
When generating some report types on my UNIX server with no XServer, I get an exception "Can't connect to X11 server" or similar. ....	486
When using Clover, why do I get a java.lang.NoClassDefFoundError when I run my code? .....	487
When using Clover from Ant, why do I get "Compiler Adapter 'org.apache.tools.ant.taskdefs.CloverCompilerAdapter' can't be found." or similar? .....	487
Why does the 'Test Results' summary page report show that I have unique coverage, when the source page shows no unique coverage? .....	487
Why do I get 0% coverage when I run my tests and then a reporter from the same instance of Ant? .....	487
Why do I get a 'java.lang.OutOfMemoryError - PermGen space' error? .....	487
Why do I get an java.lang.OutOfMemoryError when compiling with Clover turned on? .....	488
Clover Resources .....	488
Clover Development Hub .....	489
Clover for Grails Developer Guide .....	490
Creating Grails plugins using Clover .....	492
Clover for Hudson Developer Guide .....	492
Clover for Jenkins Developer Guide .....	494
Clover for Maven 2 and 3 Developer Guide .....	496
Clover-for-Maven1 Developer Guide .....	497
Clover Road Map .....	498
Contributing to the Clover Documentation .....	499
Database Structure .....	500
Upgrading third party libraries .....	505

# About Clover

## Getting Started

If you are new to Clover and want to get it going quickly, try the following:

- The [Ant Quick Start Guide](#) and the [Maven 2 and 3 Quick Start Guide](#) will show you how to quickly set-up Clover in your builds.
- The [Clover for Eclipse in 10 minutes](#) and the [Clover for IDEA in 10 minutes](#) will show you how to use Clover in Eclipse/IDEA as well as briefly describe reports generated by Clover.
- The [Introduction to Code Coverage](#) section provides a brief background on the theory and motivation behind Code Coverage.
- The [Understanding Reports](#) page gives more details about an HTML report generated by Clover
- The [Clover Tutorial](#) provides a good alternative introduction to Clover.

For help with Ant, see the online Ant manual at <http://ant.apache.org/manual/index.html>. Also recommended reading is Eric Burke's [Top 15 Ant Best Practices](#).

For Clover troubleshooting information, see the [Atlassian Answers](#) and [Clover Knowledge Base](#). Still no luck? Raise an issue for [Atlassian Support](#) team.

## System Requirements and Supported Platforms

See the [Clover-for-Ant Installation Guide](#) and [Supported Platforms](#).

## What's New in Clover?

See the [Clover Release Notes](#).

## Acknowledgements

Clover makes use of the following excellent third-party libraries:

<a href="#">Apache Ant</a>	The Ant build system.
<a href="#">ANTLR</a>	A public domain parser generator.
<a href="#">Apache Commons</a>	A set of reusable Java components.
<a href="#">Apache Velocity</a>	Template engine used for HTML report generation.
<a href="#">Cajo</a>	A lightweight library for multi-machine communication.
<a href="#">FastUtil</a>	A library for high-performance operations on primitive types.
<a href="#">Groovy</a>	An agile and dynamic language for the Java Virtual Machine.
<a href="#">GSON</a>	A library converting Java objects into their JSON representation.
<a href="#">Guava</a>	Google's core libraries for collections, caching, primitives support, string processing, I/O etc.
<a href="#">iText (2.0.1)</a>	A library for generating PDF documents.

JCommon / JFreeChart	An open source library for generating charts.
JDOM	A library for accessing, manipulating, and outputting XML data from Java code.
overLIB	A JavaScript library for pop-ups and tool tips.
TheJIT	An open source toolkit for creating interactive data visualisations.
Utils.js	A JavaScript library.

**i** To prevent library version mismatches, all of these libraries have been obfuscated and/or repackaged and included in the Clover JAR. We do this to prevent pain for users who may use different versions of these libraries in their projects.

## About Code Coverage

### What is Code Coverage?

*Code coverage* is the percentage of code which is covered by automated tests. *Code coverage measurement* simply determines which statements in a body of code have been executed through a test run, and which statements have not. In general, a code coverage system collects information about the running program and then combines that with source information to generate a report on the test suite's code coverage.

Code coverage is part of a feedback loop in the development process. As tests are developed, code coverage highlights aspects of the code which may not be adequately tested and which require additional testing. This loop will continue until coverage meets some specified target.

### Why Measure Code Coverage?

It is well understood that unit testing improves the quality and predictability of your software releases. Do you know, however, how well your unit tests actually test your code? How many tests are enough? Do you need more tests? These are the questions code coverage measurement seeks to answer.

Coverage measurement also helps to avoid test entropy. As your code goes through multiple release cycles, there can be a tendency for unit tests to atrophy. As new code is added, it may not meet the same testing standards you put in place when the project was first released. Measuring code coverage can keep your testing up to the standards you require. You can be confident that when you go into production there will be minimal problems because you know the code not only passes its tests but that it is well tested.

In summary, we measure code coverage for the following reasons:

- To know how well our tests actually test our code
- To know whether we have enough testing in place
- To maintain the test quality over the lifecycle of a project

Code coverage is not a panacea. Coverage generally follows an 80-20 rule. Increasing coverage values becomes difficult, with new tests delivering less and less incrementally. If you follow defensive programming principles, where failure conditions are often checked at many levels in your software, some code can be very difficult to reach with practical levels of testing. Coverage measurement is not a replacement for good code review and good programming practices.

In general you should adopt a sensible coverage target and aim for even coverage across all of the modules that make up your code. Relying on a single overall coverage figure can hide large gaps in coverage.

### How Code Coverage Works

There are many approaches to code coverage measurement. Broadly there are three approaches, which may be used in combination:

<b>Source code instrumentation</b>	This approach adds instrumentation statements to the source code and compiles the code with the normal compile tool chain to produce an instrumented assembly.
------------------------------------	--

<b>Intermediate code instrumentation</b>	Here the compiled class files are instrumented by adding new bytecodes, and a new instrumented class is generated.
<b>Runtime information collection</b>	This approach collects information from the runtime environment as the code executes to determine coverage information

Clover uses source code instrumentation, because although it requires developers to perform an instrumented build, source code instrumentation produces the most accurate coverage measurement for the least runtime performance overhead.

Be aware that while Clover is capable of instrumenting both Java and Groovy source code, the instrumentation stage occurs prior to compilation with Java and during compilation with Groovy.

As the code under test executes, code coverage systems collect information about which statements have been executed. This information is then used as the basis of reports. In addition to these basic mechanisms, coverage approaches vary on what forms of coverage information they collect. There are many forms of coverage beyond basic statement coverage including conditional coverage, method entry and path coverage.

### Code Coverage with Clover

Clover is designed to measure code coverage in a way that fits seamlessly with your current development environment and practices, whatever they may be. Clover's IDE Plugins provide developers with a way to quickly measure code coverage without having to leave the IDE. Clover's Ant and Maven integrations allow coverage measurement to be performed in Automated Build and Continuous Integration systems, and reports generated to be shared by the team.

#### Types of Coverage measured

Clover measures three basic types of coverage analysis:

<b>Statement</b>	Statement coverage measures whether each statement is executed.
<b>Branch</b>	Branch coverage (sometimes called Decision Coverage) measures which possible branches in flow control structures are followed. Clover does this by recording if the boolean expression in the control structure evaluated to both true and false during execution.  In Groovy code, Clover also treats Elvis expressions ( <code>a :? b</code> ), safe method calls ( <code>a?.b()</code> ), safe property calls ( <code>a?.b</code> ) and safe attribute calls ( <code>a?.@b</code> ) as branches.
<b>Method</b>	Method coverage measures if a method was entered at all during execution.

Clover uses these measurements to produce a Total Coverage Percentage for each class, file, package and for the project as a whole. The Total Coverage Percentage allows entities to be ranked in reports. The Total Coverage Percentage (TPC) is calculated as follows:

$$\text{TPC} = (\text{BT} + \text{BF} + \text{SC} + \text{MC}) / (2 * \text{B} + \text{S} + \text{M}) * 100\%$$

where

BT - branches that evaluated to "true" at least once  
 BF - branches that evaluated to "false" at least once  
 SC - statements covered  
 MC - methods entered

B - total number of branches  
 S - total number of statements  
 M - total number of methods

#### What about Line Coverage metric?

A *Line Coverage* metric is a basic metric offered by bytecode instrumentation tools, such as Cobertura or Emma, and it's tightly related with a fact that inside compiled classes we can have only information about line numbers (instead of information about code elements, such as statements etc).

As Clover uses source code instrumentation, it actually "sees" a real code structure. Therefore, Clover offers a *Statement Coverage* metric, which is similar to a Line Coverage metric in terms of it's granularity and precision.

### Further reading

- [Why does Clover use source code instrumentation?](#)
- [About Clover code metrics](#)

## About Distributed Per-Test Coverage

With the Distributed Per-Test Coverage feature, Clover has the ability to record per-test coverage from tests that are running in separate test JVMs, which may be co-sited or distributed around a network. This allows you to roll together results from unit and functional tests, from JVMs running different test frameworks, possibly in remote locations, yet resulting in a single unified view of your project's per-test code coverage.

Measuring per-test coverage allows you to run Clover's new [Test Optimization](#) on your functional tests. A battery of functional tests (being generally more time-consuming than unit tests) strongly benefits from the ability to run only the tests on code which has changed.

### RELATED LINKS

- [Using Distributed Per-test Coverage](#)
- [Using Distributed Per-test Coverage with Clover-for-Ant](#)

## About Test Optimization

This page explains Clover's Test Optimization feature and gives a brief explanation of how it works. See the [Test Optimization Quick Start Guide for Ant](#) for practical instructions.

On this page:

- [What is Test Optimization?](#)
- [How Test Optimization Works](#)
- [Supported Test Environments for Test Optimization](#)
  - [Ant](#)
  - [Maven 2 & 3](#)
  - [Eclipse](#)
  - [IDEA](#)
- [Exemplary results](#)
- [Related Links](#)

### What is Test Optimization?

Test optimization will make a build potentially complete a lot faster than a full build and test run. It should do this without substantially compromising the quality of the feedback it gives; in other words a quicker pass or fail result, but a reasonably accurate pass or fail.

There are two ways of ensuring a build completes quickly:

1. Run only the tests required to confirm the validity of the changes that triggered the build.
2. Run all the tests but in an optimal order: any failed tests from the previous build, all tests covering modified code, then in ascending order by test invocation time.

Since Clover records which tests covered which lines of code, it can tell the build to only run tests that cover code modified since the last build.

### How Test Optimization Works

The following is a general outline of what is required to enable Clover to optimize the test and build process.



1. A full clean build is performed. Any existing Clover databases are removed.
2. Clover performs instrumentation on all Java source files. The Clover registry is created.
3. javac compiles the instrumented sources
4. All unit tests are run. Coverage data is stored next to the Clover registry.
5. A Clover snapshot is saved - this is essentially a mapping of application source files to the set of tests which hit each file.
6. **Zero or more Java source files are modified, added or removed**
7. Clover re-instruments either only the modified source files or all source files (depending on how Clover is invoked). The Clover registry is updated, and any files modified or added since the last snapshot are marked as such.
8. Clover uses this information and the snapshot from the previous test run to determine which tests need rerunning.
9. The test runner is invoked. Only tests obtained in #9 and any tests unknown to Clover (perhaps excluded from instrumentation) are run. Tests are ordered to encourage early failures.
10. A snapshot is saved.
11. **Go to step 7** unless a maximum number of optimised builds has reached, `clover.jar` has changed between builds or some other build-specific condition signifies a full rebuild is required (e.g. configuration file changes) in which case **go to step 1**.

## Supported Test Environments for Test Optimization

The following environments are supported for Test Optimization.

Your unit tests must be completely standalone and have no inter-test dependencies.

 Clover's Test Optimization features currently do not support Groovy.

### Ant

- Junit - using the `<batchset/>` element nested in the Ant `<junit/>` task to select tests to be run.
- Junit TestSuites are currently not supported.
- TestNG is unsupported by Test Optimization for Clover for Ant.
- If your tests run in a separate JVM to your application, you will need to [enable distributed coverage](#).

### Maven 2 & 3

- Maven version 2.0.8+
- maven-surefire-plugin.
- JUnit TestSuites are currently not supported.
- TestNG test suites are currently not supported. (**NB** Will not work in Maven 2.0.9)
- Parallel test execution is unsupported.
- If your tests run in a separate JVM to your application (e.g. in a forked web server), you will need to [enable distributed coverage](#).


### Eclipse

- JUnit test classes are supported but TestSuites are not currently supported.
- TestNG is not currently supported.
- Optimization of tests where they reside in different project to the application code is not currently supported (to be addressed in a future release).

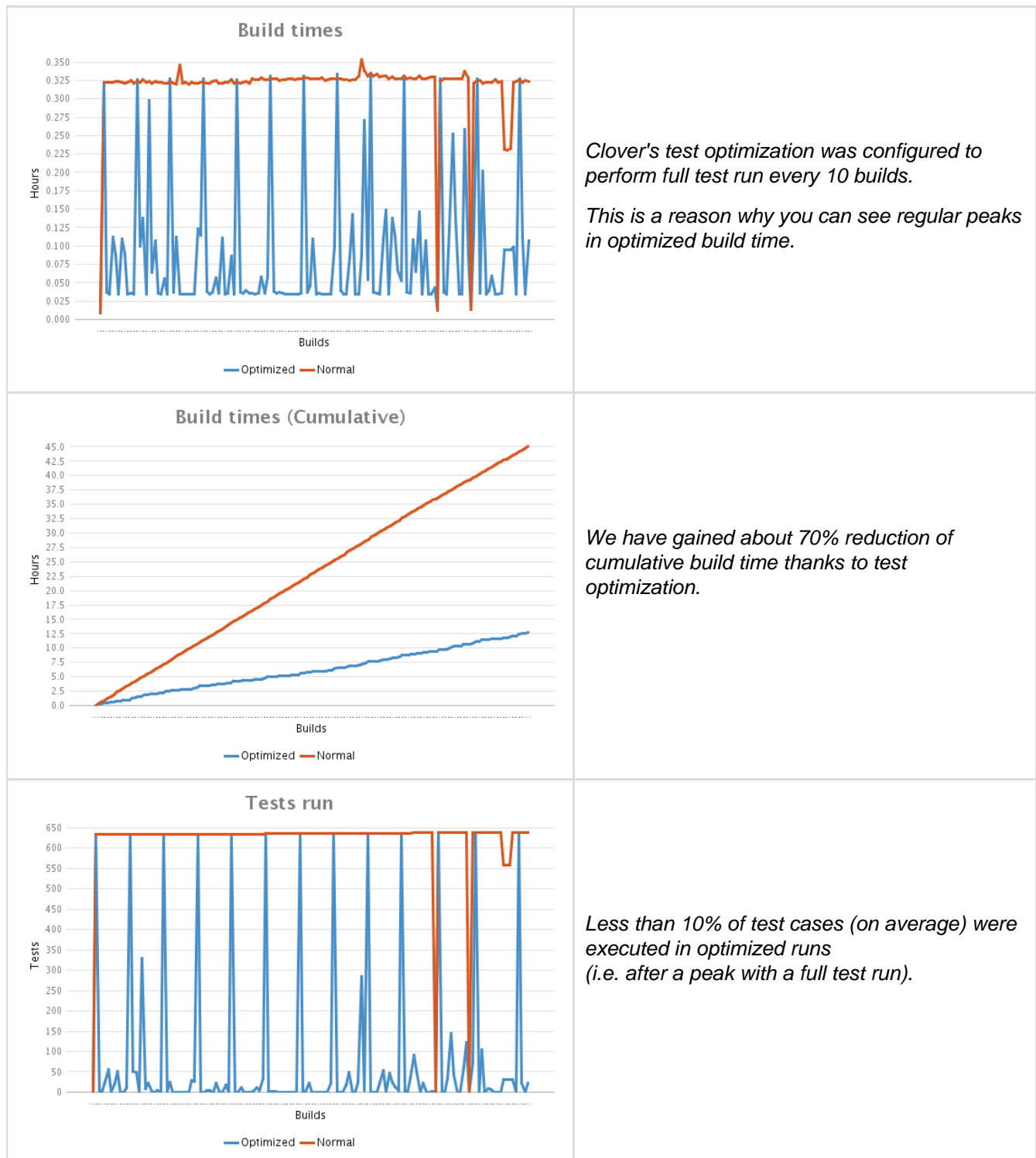
### IDEA

- JUnit test classes are supported but TestSuites are not currently supported.
- TestNG is not currently supported.

## Exemplary results

 **The following is a sample from development of one of Atlassian products. Optimization results in your project(s) may vary.**

To see the benefits of Clover's Test Optimisation, we have tracked build times on one Atlassian software development project. Over a 10 day period a development team committed 142 changesets as part of their ongoing development effort. For each changeset, two builds were triggered - a "normal" build, where all tests were executed, and a test-optimised build, where only relevant tests were executed. The following chart shows cumulative times for both the normal and test-optimised builds.



## Related Links

- [Overview of Test Optimization](#)
- [Test Optimization Quick Start for Ant](#)
- [Test Optimization Quick Start for Maven 2](#)



- [Clover for Maven 2 - Test Optimization Best Practices](#)
- [Test Optimization for IDEA](#)

## About Clover Editions

This page contains information about the different editions of Clover.

On this page:

- [Overview](#)
- [Clover Desktop Edition](#)
- [Clover Server Edition](#)

### Overview


Clover editions were introduced with the release of Clover 2.6, in September 2009. Previous versions of Clover do not belong to a particular edition. Editions only apply to the IDE plugin variants of Clover, i.e. Clover-for-Eclipse and Clover-for-IDEA.

### Clover Desktop Edition

Clover Desktop Edition is an affordable option for individual developers, providing code coverage analysis and also the use of test optimization for developers working in isolation. Clover Desktop Edition is only available for Clover's Eclipse and IDEA plugins. See the [pricing page](#) for more information.

### Clover Server Edition

Clover Server edition contains the full suite of report generation and Continuous Integration features for coding in a team environment. The Server Edition is essentially the full-featured product, simply renamed to Server Edition. Pricing for Server Edition remains the same.

 The Desktop Edition only applies to the IDE plugin variants of Clover, i.e. Clover-for-Eclipse and Clover-for-IDEA.

See the [pricing page](#) for more information.

## About Clover code metrics

### Code coverage metrics

#### Branch coverage

Branch coverage (or 'decision coverage') is a [code coverage](#) metric that measures which possible branches in flow control structures are followed. Clover does this by recording if the boolean expression in the control structure evaluated to both true and false during execution.

#### Statement coverage

Statement coverage is a [code coverage](#) metric that measures which statements in a body of code have been executed through a test run, and which statements have not.

#### Method coverage

Method coverage is a [code coverage](#) metric that measures whether a method was entered at all during execution.

#### Total Coverage Percentage (Coverage %, TPC)

The Total Coverage Percentage is calculated as follows:

$$\text{TPC} = (\text{BT} + \text{BF} + \text{SC} + \text{MC}) / (2 * \text{B} + \text{S} + \text{M}) * 100\%$$

where

BT - branches that evaluated to "true" at least once

BF - branches that evaluated to "false" at least once

SC - statements covered

MC - methods entered

B - total number of branches

S - total number of statements

M - total number of methods

## Code complexity metrics

### Average Method Complexity (Avg Method Cmp)

The average method complexity of code in the given context. In case of:

- classes: Average Method Complexity = sum of Method Complexity of all class' methods / number of methods in a class
- packages: Average Method Complexity = sum of Method Complexity of all package's methods / number of methods in a package
- project: Average Method Complexity = sum of Method Complexity of all project's methods / number of methods in a project

### Complexity (Cmp, Total Cmp)

Cyclomatic complexity of code in the given context.

### Method Complexity

Cyclomatic complexity of a single method. It's calculated as follows:

- empty method complexity == 1
- simple statement complexity == 0
- switch block complexity == number of case statements
- try catch block complexity == number of catch statements
- ternary expression complexity == 1
- boolean expression complexity == number of && or || in expression

### Complexity Density (Cmp Density)

Complexity Density is the average number of paths in your code per statement in given context (method, class, package). It's calculated as follows:

$$\text{Cmp Density} = \text{Complexity} / \text{number of statements}$$

### LOC

Lines Of Code (including comment lines).

### NC LOC

Non-Commented Lines Of Code. Lines of code that contain comments are not counted in this metric, leaving only the actual functioning code itself.

## Examples

```
// # of statements, # of branches, Method Cyclomatic Complexity

// 0, 0, 1
void A() {}

// 1, 0, 1
void A() { a(); }

// 1, 0, 1
void A() { a = (6 < 7); }

// 3, 0, 1
void A() { a(); b(); c(); }

// 3, 2, 2
void A() { if (a()) b(); else c(); }

// 2, 2, 3
void A() { if (a() || b()) c(); }

// 2, 0, 1
void A() { if (1 + 2 == 4) c(); }

// 2, 2, 2
void A() { for (; a(); ) b(); }

// 2, 2, 3
void A() { for (; a() || b(); ) c(); }

// 2, 0, 1
void A() { for (; 1 + 2 == 4; ) c(); }

// 2, 2, 2
void A() { while (a()) b(); }

// 2, 2, 3
void A() { while (a() || b()) c(); }

// 2, 0, 1
void A() { while (1 + 2 == 4) c(); }

// 3, 0, 2
void A() { switch (a()) { case 1: b(); } }

// 5, 0, 3
void A() { switch (a()) { case 1: b(); case 2: c(); } }

// 1, 2, 2
void A() { a() ? 1 : 2; }

// 1, 2, 3
void A() { a() || b() ? 1 : 2; }

// 1, 6, 4
void A() { a() ? b() ? c() ? 1 : 2 : 3 : 4; }
```



















## References

- [About Code Coverage](#)
- [Glossary](#)
- [clover-report#columns](#)

## Supported Platforms

This page shows the supported platforms for **Clover 4.0.x** and its minor releases.

**Key:**  = Supported;  = Not Supported

Java Version	
JRE / JDK <sup>(1)</sup>	 1.6 - 1.8
Groovy Version	
Groovy	 1.6.2 or later  1.6.8, 1.6.9 and 1.7.1 do not support <a href="#">Per-Test coverage</a> due to a Groovy bug  2.0.0 does not work with JDK1.5 due to a <a href="#">Groovy bug</a>  Eclipse and IDEA (No Groovy support in IDE)
JRE / JDK <sup>(1)</sup>	 1.6 or later
Operating Systems	
Microsoft Windows <sup>(2)</sup>	
Linux <sup>(2)</sup>	
Apple Mac OS X <sup>(2)</sup>	
Build Automation Tools	
Apache Ant	 1.7.0 or later
Apache Maven 2	 2.0.2 or later
Apache Maven 3	 3.0.0 or later
Application Development Frameworks	
Grails	 1.3 or later <sup>(6)</sup>
Integrated Development Environments (IDEs)	
JetBrains IntelliJ IDEA	 9.0 - 13.1 <sup>(5)</sup>
Eclipse <sup>(3)</sup>	 3.6 - 4.4, RAD 8 & 8.5 & 9 <sup>(4)</sup>
Web Browsers (for viewing Clover's HTML report)	
Internet Explorer	 9.0 or later
Firefox	
Chrome	

### Supported Platform Notes

1. The last Clover version supporting JDK/JRE 1.4 is 3.1.12. The last Clover version supporting JDK/JRE 1.5 is 3.3.x. Since Clover 4.0 you must use at least Java 1.6 for compilation and runtime. Please note that it's still possible to set `source level = 1.3` (no `assert` keyword) or 1.4 (no enums, generics) or 1.5 (no `@Override` on interfaces) for instrumentation. You can [download](#) a JRE or JDK for Windows/Linux. On Mac OS X, a JDK is bundled with the operating system. Once a JRE or JDK has been installed, you need to set the `JAVA_HOME` environment variable.
2. Clover is a pure Java application and should run on any operating system platform provided the requirements for the JRE or JDK are satisfied. Clover does not work on Solaris OS due to bugs in NIO implementation in Java on this platform.
3. Your Eclipse projects must use the built-in Java Builder for compilation of source code. Clover does not support AspectJ-based projects.
4. Support for Eclipse 3.2 and 3.3 and RAD 7.0 has ended with Clover 3.1.5. Support for Eclipse 3.4 and 3.5 as well as RAD 7.5 has ended with Clover 3.1.12.
5. Support for IntelliJ IDEA 7 and 8 has ended with Clover 3.1.5. The IDEA 12 is supported since Clover 3.1.8.
  - ⚠ If you're using IDEA 12.0.x and Clover 3.1.8-3.1.11 then you have to disable the "external build" feature.
  - The IDEA 13 is supported since Clover 3.2.1. The IDEA 13.1 is supported since Clover 3.3.0.
6. Support for Grails 1.2 has ended with Clover 3.1.5.

## End of Platform Support Announcements

## End of Platform Support Announcements

This page contains announcements of the end of support for various platforms and browsers when used with Clover. This is summarised in the table below. Please see the sections following for the full announcements.

### End of support matrix for upcoming versions of Clover

Platform	Announcement Date	Clover End of Technical Support
<a href="#">IntelliJ IDEA 9</a>	31 March 2014	As of Clover 4.1
<a href="#">IntelliJ IDEA 10.x, 10.5.x</a>	31 March 2014	As of Clover 4.1
<a href="#">Eclipse 3.6 and RAD 8.0</a>	31 March 2014	As of Clover 4.1

The table above summarises information regarding the end of support announcements for upcoming Clover releases. If a platform (version) has already reached its end of support date, it is not listed in the table.

### **i** Why is Atlassian ending support for these platforms?

Atlassian is committed to delivering improvements and bug fixes as fast as possible. We are also committed to providing world class support for all the platforms our customers run our software on. However, as new versions of IDEs, web browsers, build tools etc. are released, the cost of supporting multiple platforms grows exponentially, making it harder to provide the level of support our customers have come to expect from us. Therefore, we no longer support platform versions marked as end-of-life by the vendor, or very old versions that are no longer widely used.

On this page (most recent announcements first):

- [Deprecated Clover support for IntelliJ IDEA 9 \(announced 31 March 2014\)](#)
- [Deprecated Clover support for IntelliJ IDEA 10.0 and 10.5 \(announced 31 March 2014\)](#)

- [Deprecated Clover support for Eclipse 3.6 and RAD 8.0 \(announced 31 March 2014\)](#)

### Deprecated Clover support for IntelliJ IDEA 9 (announced 31 March 2014)

Atlassian announces the deprecation of Clover support for IntelliJ IDEA 9. We will no longer support IDEA 9 in Clover 4.1. Clover 4.1 is expected to be released in the first half of 2014.

If you have questions or concerns regarding this announcement, please email [eol-announcement at atlassian dot com](mailto:eol-announcement@atlassian.com).

### Deprecated Clover support for IntelliJ IDEA 10.0 and 10.5 (announced 31 March 2014)

Atlassian announces the deprecation of Clover support for IntelliJ IDEA 10.0.x and 10.5.x. We will no longer support these in Clover 4.1. Clover 4.1 is expected to be released in the first half of 2014.

If you have questions or concerns regarding this announcement, please email [eol-announcement at atlassian dot com](mailto:eol-announcement@atlassian.com).

### Deprecated Clover support for Eclipse 3.6 and RAD 8.0 (announced 31 March 2014)

Atlassian announces the deprecation of Clover support for Eclipse 3.6 and RAD 8.0. We will no longer support these in Clover 4.1. Clover 4.1 is expected to be released in the first half of 2014.

If you have questions or concerns regarding this announcement, please email [eol-announcement at atlassian dot com](mailto:eol-announcement@atlassian.com).

## Clover-for-Ant

### Clover-for-Ant Documentation

What is Clover-for-Ant?
Clover-for-Ant integrates the industry-leading code coverage tool, <a href="#">Atlassian Clover</a> with the Apache Ant build automation tool. Clover-for-Ant allows you to easily measure the coverage of your unit tests, enabling targeted work in unit testing — resulting in stability and enhanced quality code with maximal efficiency of effort.
Getting Started with Clover for Ant
<a href="#">Download Clover for Ant</a>
<a href="#">Quick Start Guide</a>
<a href="#">Installation Guide</a>
<a href="#">Changelog for latest version of Clover-for-Ant</a>

### Using Clover for Ant

User's Guide

Installation & Configuration Guide

Test Optimization Quick Start Guide for Ant

### Resources and Support

Atlassian Answers











Knowledge Base

Technical Support

### Offline Documentation

You can download the Clover documentation in PDF, HTML or XML format.

## Recently Updated

-  [Clover Road Map](#)  
Aug 12, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Upgrading third party libraries](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Updating optimization snapshot file](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Hacking Clover](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Part 4 - Test Optimization Tutorial](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Part 3 - Automating Coverage Checks](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Part 2 - Historical Reporting](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Part 1 - Measuring Coverage](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Clover 4.0 Release Notes](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [A side-by-side comparison of the Classic and the ADG HTML report](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)

## Clover-for-Ant User's Guide

- 1. QuickStart Guide
  - Clover for Ant Best Practices
  - Clover-for-Ant Two Line Integration
  - Test Optimization Quick Start Guide for Ant
- 2. Using Clover Interactively
- 3. Using Clover in Automated Builds
- 4. Understanding Reports
  - 'Current' Report
    - Coverage Legend
    - Dashboard Widgets
    - Source Cross-Referencing in Reports
    - Stack Trace Navigation

- Tag Clouds
  - 'Historical' Report
- 5. Configuring Reports
  - Unit Test Results and Per-Test Coverage
  - Using Coverage Contexts
  - Using Spans
    - Specifying an Interval
  - Sharing Report Formats
  - Extracting Coverage Data programmatically
- 6. Ant Task Reference
  - clover-check
  - clover-clean
  - clover-env
  - clover-historypoint
  - clover-html-report
  - clover-instr
  - clover-log
  - clover-merge
  - clover-pdf-report
  - clover-report
    - <added> element
    - <chart> element
    - <columns> element
    - <format> element
    - <movers> element
    - JSON reference
      - Basic Clover Confluence Integration
  - clover-setup
    - Clover test detection
    - methodContext
    - profiles
    - statementContext
  - clover-snapshot
- 7. Ant Type Reference
  - clover-columns
  - clover-format
  - clover-optimized-testset
- 8. Controlling Clover at Runtime
  - Clover Performance Tuning
  - Coverage Recorders
  - Managing the Coverage Database
  - Using a Flush Policy
  - Using Source Directives
  - Working with Distributed Applications
    - Using Distributed Per-test Coverage with Clover-for-Ant
  - Working with Restricted Security Environments
    - Working in OJVM
- 9. Clover Target Reference
- A. Integrating Clover-for-Ant with other tools
  - Integrating Clover with JUnit4 Parameterized Tests
  - Using Clover-for-Ant with GWT

## 1. QuickStart Guide

### Getting started with Clover-for-Ant

This section shows you how to quickly get Clover integrated into your build. Clover instrumentation and reporting are highly configurable so later sections of this manual will detail available configuration options and typical usage scenarios.





Do you want to type less code? See the fast [Clover Two Line Integration](#).

Do you want to try out a ready-to-use code sample? See the [Clover Tutorial, Part 0 - Clover in 10 minutes](#).

**i** Unless otherwise indicated, all configuration options throughout this User's Guide apply to both Java and Groovy.

## Follow these simple steps to integrate Clover with your build:

### 1. Install Clover

**1.1** Ensure you are using a recent version of Ant.

**1.2** Download Clover from <http://www.atlassian.com/software/clover/download>.

**1.3** Unzip the Clover distribution into a directory. This directory will be referred to as *CLOVER\_HOME* in this guide.

**1.4** Place your `clover.license` file in *CLOVER\_HOME/lib* (you can obtain evaluation license from <http://my.atlassian.com>).

### 2. Add Clover targets

Edit `build.xml` for your project:

**2.1** Add the Clover Ant tasks to your project:

```
<property name="clover.jar" location="CLOVER_HOME/lib/clover.jar"/>
<taskdef resource="cloverlib.xml" classpath="{clover.jar}"/>
```

**2.2** Add a target to switch on Clover:

```
<target name="with.clover">
  <clover-setup/>
</target>
```

**2.3** Add one or more targets to run Clover reports:

*For HTML reporting, use the following (but change the `outdir` to a directory path where Clover should put the generated HTML):*

```
<target name="clover.html">
  <clover-html-report outdir="coverage"/>
</target>
```

*OR, for PDF reporting, use the following (but change the `outfile` to a file where Clover should write the PDF file):*

```
<target name="clover.pdf">
  <clover-pdf-report outfile="coverage.pdf"/>
</target>
```

*OR, for XML reporting, use the following (but change the `outfile` to a file where Clover should write the XML file):*

```
<target name="clover.xml">
  <clover-report>
    <current outfile="coverage.xml">
      <format type="xml"/>
    </current>
  </clover-report>
</target>
```

OR, for simple emacs-style reporting to the console, try:

```
<target name="clover.log">
  <clover-log/>
</target>
```

**2.4** Add `clover.jar` to the runtime classpath for your tests. How you do this depends on how you run your tests. For tests executed via the `<junit>` task, add a `<classpath>` element:

```
<junit ...>
  ...
  <classpath>
    <pathelement path="{clover.jar}"/>
  </classpath>
  <formatter type="xml"/>
</junit>
```

### Compile and run with Clover


Now you can build your project with Clover turned on by adding the "with.clover" target to the list of targets to execute. For example (if your compile target is named 'build' and your unit test target is named 'test'):

```
ant with.clover build test
```

### Generate a Coverage Report

To generate a Clover coverage report:

```
ant clover.html (or clover.xml, clover.pdf etc)
```

 For a sample report, see 'Current' Report.

### NEXT STEPS

See the [Test Optimization Quick Start Guide for Ant](#), for how to set up Clover's Test Optimization feature to streamline your testing.

### FURTHER READING

See [Clover for Ant Best Practices](#).

### Clover for Ant Best Practices

This section describes some recommended practices when integrating Clover into your Ant build. For a great list of general Ant best practices, see [http://www.onjava.com/pub/a/onjava/2003/12/17/ant\\_bestpractices.html](http://www.onjava.com/pub/a/onjava/2003/12/17/ant_bestpractices.html).

### 1. Let Clover automatically choose the database location

The `initstring` attribute is optional. If not supplied, Clover will automatically create a special directory for the [Clover coverage database](#). There are several advantages in letting Clover use the default location. Clover tasks can find the database more easily, and build files become more portable. If left to the default setting, there is no need to have Clover reporting targets depend on the the Clover setup target.



#### Note

If you want to specify the `initstring` explicitly, it is strongly recommended that you give Clover its own direct directory, because a Coverage run can result in many files being written to the database.

### 2. Use the `<clover-clean>` task

Once you have generated the reports or history points you require from a Coverage run, use `<clover-clean>` to delete the database so that it will be freshly created for the next build. This is easily achieved by adding the `<clover-clean>` task to any existing clean target.

### 3. Avoid setting the compiler or executable attributes on the `<javac>` task

Setting either of these attributes makes your build less portable. It may also prevent Clover from installing correctly in your build.

### 4. Set the source attribute on the `<javac>` task

Setting the source attribute increases the portability of your build by explicitly defining the language level of the project. If you don't set it, the language level is determined by whatever underlying compiler is found by Ant.

### 5. Use Target dependencies in preference to `<ant>` and `<antcall>`

Ant's target dependencies are an efficient way to manage build dependencies. You should always strive to use this mechanism over the more 'procedural' style of explicitly calling targets. By explicitly calling Ant tasks, you miss out on Ant's powerful dependency management where up-to-date targets are skipped. You also introduce significant memory overhead (particularly if `fork="true"` is set). Excessive use of `<antcall>` can also make a build file less readable, because it can be difficult to trace which properties and references are valid for a given target.

**i** If you must use `<ant>` and `<antcall>`, be aware that you must set `inheritrefs` to "true" if you are calling `<clover-setup>` in an upper-level project.

Below, we demonstrate an alternative. Instead of this:

```
<!-- BAD. References wont be passed (References from <clover-setup/> would be
lost). -->
<target name="all">
  <antcall target="clean">
  <antcall target="compile">
  <antcall target="dist">
  <antcall target="test">
</target>
```

it is much better to use something like this:

```
<!-- GOOD -->
<target name="all" depends="clean, compile, dist, test"/>
```

### 6. If using the `<junit>` task, consider using `fork="true"` `forkmode="once"`

Setting these attributes means that your JUnit tests will run in a single, separate JVM. This isolates the unit tests from the Ant JVM, and means that no special flushing is required to have Clover coverage data

written to disk when the tests end.

## Clover-for-Ant Two Line Integration

To get Clover integrated into your build as quickly as possible, follow these simple steps.

1. Download [Clover-for-Ant](#), unzip it, take the lib/clover.jar and save it in your home directory.
2. Add the following lines to your build.xml file:

```
<taskdef resource="cloverlib.xml" classpath="${user.home}/clover.jar" />
<clover-env />
```

**i** Note that this will not work within an Ant target. It must be at the top level of the build file.

3. Add the clover.jar to your test classpath:

```
<junit fork="true" forkmode="once">
  <classpath>
    <pathelement location="${user.home}/clover.jar" />
  </classpath>
</junit>
```

4. If you have a target already called "test" you can simply run

```
ant clover.all
```

Otherwise, run the following:

```
ant with.clover your.test.target clover.report
```

Alternatively, define a property called "test.target" whose value is the name of your test target.


5. Complete! That concludes the Ant two-line integration. You should now be set up to run Clover on your Ant builds and start taking advantage of Clover's advanced code coverage analysis.

## Appendix


By calling `<clover-env/>`, the following targets becomes available to you:

Target Name	Description
clover.all	Runs clover.clean, with.clover, test, clover.report from a single target.
clover.clean	Deletes the clover database and the <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre>  \${clover.dest}</pre> </div> directory.

clover.current	Generates an HTML and XML report to <pre>       \${clover.dest}     </pre> using <pre>       \${project.title}     </pre> .
clover.report	Same as clover.current, however a history report will also be created, using the historypoints in <pre>       \${clover.project.historydir}     </pre> .
clover.save-history	Saves a history point to <pre>       \${clover.project.historydir}     </pre>
with.clover	Enables Clover on this build
clover.snapshot	Saves a snapshot file to assist with unit test optimization

 These are available also by running `ant -projecthelp`.

For more instructions about using targets, see the [Clover Target Reference](#). Any `${}` properties may be defined on the command line, for example: `-Dclover.project.historydir=/home/clover/historydir`

 Need more information? Find it in the [Clover QuickStart Guide](#).

## Test Optimization Quick Start Guide for Ant

This page contains the basic steps for adding Clover's Test Optimization to an existing Ant configuration.

Follow the steps in this document to set up Clover's Test Optimization, which allows targeted testing of only the code which has changed since the last build.

These steps assume your build is Clover-enabled already (in particular it has a `with.clover` task already set up and has a taskdef established for the Clover Ant tasks). You will have to complement this quick start guide with [basic Clover configuration information](#).

These steps also assume that your build file is currently used for a CI (Continuous Integration) build and possibly for general builds (e.g. On a developer's own machine). Below, we describe how you can take that build file and add sections to bake in Test Optimization. Adding optional support for Test Optimization (switching it on/off), specifying whether test minimisation is performed and test reordering other than the default 'failfast' are advanced options which are covered elsewhere.

**U** Clover's Test Optimization feature currently do not support optimization for test cases written in Groovy (CLOV-1152):

- test cases written in Groovy will be executed in each test run
- test cases written in Java will be optimized
- application code can be written in Java or Groovy in order to be optimized

**i** Clover's Test Optimization does not optimize execution of JUnit TestSuites (CLOV-616) and TestNG (CLOV-373).

- `<batchtest>` resource collection shall contain JUnit TestCases

## BEFORE YOU START

Try to ensure your unit tests do not have dependencies between them as this may cause optimized builds to fail more frequently than usual.

## BASIC STEPS

1. You will need to use this:

```
<taskdef resource="cloverlib.xml" classpath="{clover.jar}"/>
```

2. Choose a location for the test snapshot file that can survive clean builds. This location:

```
<PROJECT_DIR>/ .clover/coverage.db.snapshot
```

The default is not as good as manually deleting this directory each build, but it is workable if you only use `<clover-clean/>`as, by default, it won't delete snapshots. Add a property for this, as in the following example:

```
<property name="clover.snapshot.file" value="/path/to/clover.snapshot"/>
```

3. Add a target to generate the test snapshot:

```
<target name="clover.snapshot" depends="with.clover">
  <clover-snapshot file="{clover.snapshot.file}"/>
</target>
```

4. Modify the `<batchtest>` element of the `<junit/>` task used to test your application, so that the filesets are wrapped in the `clover-optimized-testset` element. See the following example:

```
<junit ...>
<batchtest fork="true" todir="{test.results.dir}/results">
  <fileset dir="src/tests" includes="{test.includes}"
excludes="{test.excludes}"/>
</batchtest>
</junit>
```

This becomes the following:

```

<junit ...>
<batchtest fork="true" todir="${test.results.dir}/results">
  <clover-optimized-testset snapshotfile="${clover.snapshot.file}">
    <fileset dir="src/tests" includes="${test.includes}"
excludes="${test.excludes}" />
  </clover-optimized-testset>
</batchtest>
</junit>

```

5. Run the optimized build (this will typically be run by their CI plan). Assuming a "test" target (with appropriate dependencies so that the code is instrumented/compiled):

```
ant with.clover clean test clover.snapshot
```

### Running Java and Groovy test cases

Please note that Ant's `<junit>/<batchtest>` collects the included resources from any number of nested resource collections and then generates a test class name for each resource that ends in `.java` or `.class`. It means that you cannot use `<include name="**/*Test.groovy"/>` because such files will be ignored. However, you can point to `*.class` files, for example:

```

<junit ...>
  <classpath refid="build.classpath"/>
  <batchtest fork="yes" todir="${test.result}">
    <clover-optimized-testset snapshotfile="${clover.snapshot.file}">
      <fileset dir="${build.dir}">
        <include name="**/*Test.class"/>
      </fileset>
    </clover-optimized-testset>
  </batchtest>
</junit>

```

Using the configuration above, Clover will optimize tests according to:

- changes in application code written in Java or Groovy
- changes in test code written in Java

### Related Links

- [Overview of Test Optimization](#)
- [Test Optimization Technical Details](#)
- [Test Optimization Quick Start for Maven 2](#)
- [Clover for Maven 2 - Test Optimization Best Practices](#)

## 2. Using Clover Interactively

In this scenario, a developer is responsible for obtaining a certain level of code coverage on her code before it is accepted into the base.

The typical cycle the developer follows is something like:

1. write code/tests
2. run tests
3. inspect test results and code coverage

This process is repeated until all tests pass and code coverage of the tests meets a certain level.

Clover provides the following features to support this development pattern:

- [Measuring coverage on a subset of source files](#)
- [Viewing source-level code coverage quickly](#)
- [Viewing summary coverage results quickly](#)
- [Incrementally building coverage results](#)

#### Measuring coverage on a subset of source files

The `<clover-setup>` task takes an optional nested `fileset` element that tells Clover which files should be included/excluded in coverage analysis:

```
<clover-setup>
  <files includes="**/plugins/cruncher/**, **/plugins/muncher/**"/>
</clover-setup>
```

The `includes` could be set using an Ant property so that individual developers can specify includes on the command line:

```
<property name="coverage.includes" value="**"/>
<clover-setup>
  <files includes="${coverage.includes}"/>
</clover-setup>
```

Developers can then use a command line like the following for Java code:

```
ant build -Dcoverage.includes=java/**/foo/*.java
```

And for Groovy code:

```
ant build -Dcoverage.includes=groovy/**/foo/*.groovy
```

#### Viewing source-level code coverage quickly

Clover provides two ways of quickly viewing coverage results. The `<clover-log>` task provides quick reporting to the console:

```
<clover-log/>
```

The output format from the `<clover-log>` task uses the `[file:line:column]` format that many IDEs can parse.

#### Viewing summary coverage results quickly

The `<clover-log>` task provides an option that will print a summary of coverage results to the console:

```
<clover-log level="summary"/>
```

#### Incrementally building coverage results

When iteratively improving coverage on a subset of your project, you may want to include coverage data from several iterations in coverage results. Clover supports this with the `span` attribute which works on current reports — see [Using Spans](#). This attribute can be used to tell Clover how far back in time to include coverage results (measured from the time of the last Clover build). To include results gathered over the last hour use:



```
<clover-log span="1h"/>
```

### 3. Using Clover in Automated Builds

In this scenario, the project is checked out, built and tested at regular intervals, usually by an automated process. Some tools that support this type of build are [AntHill](#), [Bamboo](#) and [CruiseControl](#).

Clover supports this scenario with the following features:

- Detailed coverage reports for the whole team
- Executive summary coverage reports
- Historical coverage and project metrics reporting
- Coverage criteria checking and triggers

#### Detailed coverage reports for the whole team

**Note**

The `<clover-html-report>` and the `<clover-pdf-report>` tasks used here are simplified versions of the `<clover-report>` task. If you require more control over your report formatting and structure, use the `<clover-report>` task.

In this example, the `<clover-html-report>` task is used to generate source-level coverage reports in HTML format that can be published for viewing by the whole team:

```
<target name="clover.report" depends="with.clover">
  <clover-html-report outdir="clover_html"/>
</target>
```

#### Executive summary coverage reports

In this example, the `<clover-pdf-report>` task is used to generate summary reports in PDF format, suitable for email or audit purposes.

```
<target name="clover.summary" depends="with.clover">
  <clover-pdf-report outfile="coverage.pdf"/>
</target>
```

#### Historical coverage and project metrics reporting

Clover can generate a historical snapshot of coverage and other metrics for your project using the `<clover-historypoint>` task. Historical data can then be collated into a historical report using the `<clover-report>` task:

```

<target name="clover.report" depends="with.clover">

  <!-- generate a historypoint for the current coverage -->
  <clover-historypoint historyDir="clover_hist" />

  <!-- generate a report with both current and historical data -->

  <clover-html-report outdir="clover_html"
                    historyDir="clover_hist" />
</target>

```

### Coverage criteria checking and triggers

The `<clover-check>` task can be used to monitor coverage criteria. If coverage does not meet the criteria, the build can be made to fail or an arbitrary activity can be triggered. In the example below, if project coverage is not 80%, an executive summary coverage report is generated and mailed to the team:

```

<target name="coverageAlert" depends="coverage.check"
        if="coverage_check_failure">

  <clover-pdf-report outfile="coverage.pdf" />

  <mail from="nightlybuild@somewhere.not"
        tolist="team@somewhere.not"
        subject="coverage criteria not met"
        message="${coverage_check_failure}"
        files="coverage.pdf" />
</target>

<target name="coverage.check" depends="with.clover">
  <clover-check target="80%"
                failureProperty="coverage_check_failure" />
</target>

```

## 4. Understanding Reports

- **'Current' Report** — 'Current' Clover reports display graphical and numerical data relating to the most recent coverage data collected for the project.
- **'Historical' Report** — 'Historical' reports display graphical and numerical data relating to sets of coverage data collected over time for the project.

### RELATED TOPICS

For information on generating reports, see the following [Ant Tasks](#):

- `<clover-html-report>` - HTML report with predefined settings
- `<clover-pdf-report>` - PDF report with predefined settings
- `<clover-report>` - highly customizable HTML, XML, PDF or JSON reports

Also see the tutorial '[Clover-for-Ant tutorial](#)'.

### 'Current' Report

'Current' Clover reports display graphical and numerical data relating to the most recent coverage data collected

for the project.

Clover 4 brings new HTML report developed according to the [Atlassian Design Guidelines](#); it's shortly named "ADG". Clover 3 offers an old-style Javadoc-like format, named "Classic". It is still possible to [fall-back to the "Classic" report](#) in Clover 4, however this report style is deprecated.

- [ADG report style \(Clover 4 only\)](#)
  - [Project overview page](#)
  - [Package-level and project-level overview of application and test code](#)
  - [Source file view](#)
  - [Test results](#)
  - ['Top risks' and 'Quick wins' tag clouds](#)
  - [Coverage tree map](#)
  - [See also: 'Historical' Report.](#)
- [Classic report style \(Clover 3 and Clover 4\)](#)
- [Appendix: naming convention of lambda functions](#)

### ADG report style (Clover 4 only)

#### Project overview page

##### General

On the project overview page you will find several tabs, thanks to which you can quickly learn about your project:

- [Dashboard](#) - contains several widgets with statistics and most critical issues
- [Application code](#) - browse through application classes
- [Test code](#) - browse through test classes
- [Test results](#) - contains results from your unit tests
- [Top risks](#) - the most complex and the least covered classes
- [Quick wins](#) - "low hanging fruits"
- [Coverage tree map](#)

##### Blue application header

- [Clover logo](#) - opens the Atlassian Clover home page
- [Linked reports](#) - shows reports linked with the current one (optional, see the `<clover-report>` task for more details)
- [Help icon](#) - opens the Clover documentation home page

##### Dashboard tab

It contains several useful widgets:

- [Code coverage](#) - shows Total Coverage Percentage metric of your application code (i.e. excluding test code)
- [Test results](#) - shows number of tests executed and their success ratio
- [Code metrics](#) - shows code metrics of your application code of the entire project (i.e. excluding test code)
- [Class coverage distribution](#) - histogram showing number of classes vs coverage
- [Class complexity](#) - dot chart showing class complexity vs coverage
- [Coverage tree map](#) - size of the rectangle represents package complexity, while colour shows its coverage
- [Top project risks](#) - the most complex and the least covered classes
- [Most complex packages](#) - packages with the highest cyclomatic complexity
- [Most complex classes](#) - classes with the highest cyclomatic complexity
- [Least tested methods](#) - methods having the lowest code coverage

##### Package tree view

You can use it to navigate through the project structure. You can also search for packages matching given

sub-string.

Clover Linked reports

**backport-util-concurrent**

**Project overview**

PACKAGES

Type and press enter to filter packages...

- default-pkg 88,3%
- edu.emory 0%
- sun.misc 0%

Project Clover database Sr lip 23 2014 10:49:52 CEST

**Project overview**

Dashboard
Application code
Test code
Test results
Top risks
Quick wins
Coverage tree map

**Code coverage** 215 classes, 10 139 / 14 415 elements

70,3%

[See more](#)

**Test results** 70 / 70 tests 13,87 secs

100%

[See more](#)

**Code metrics**

Branches: 3 784 Statements: 8 465 Methods: 2 166

Classes: 215 Files: 83 Packages: 6 LOC: 33 802

NCLOC: 15 283 Total complexity: 4 384

Complexity density: 0.52 Statements/Method: 3,91

Methods/Class: 10,07 Classes/Package: 35,83

Average method complexity: 2,02

**Class Coverage Distribution**

**Class Complexity**

**Coverage tree map**

[See more](#)

**Top 20 project risks**

**ThreadHelpers**

**CopyOnWriteArrayList.COWSubList**

Utils.Collections.CheckedMap

WaitQueue.WaitNode

Collections.CheckedMap.EntrySetView Arrays

TreeMap.ValueSet

ScheduledThreadPoolExecutor.DelayedW

Semaphore.NonfairSync

TreeMap.BaseEntryIterator

ConcurrentSkipListMap

ReentrantLock.NonfairSync

PriorityQueue.Itr

ReentrantReadWriteLock.ReadLock

Collections.CheckedMap.EntryView Exchanger

FIFOCondVar

ConcurrentHashMap.HashIterator

Collections.CheckedCollection

[See more](#)

**Most complex packages**

- 78,5% edu.emory.mathcs.backport.java.util.cc
- 49,1% edu.emory.mathcs.backport.java.util.cc
- 80,9% edu.emory.mathcs.backport.java.util.cc (289)
- 94,2% edu.emory.mathcs.backport.java.util.cc (152)
- 67,5% edu.emory.mathcs.backport.java.util.cc (83)

**Most complex classes**

- 77,5% ConcurrentSkipListMap (288)
- 2,6% Arrays (22)
- 87,6% TreeMap (215)
- 81,9% ThreadPoolExecutor (175)
- 87,8% ConcurrentSkipListMap.SubMap (158)

**Least tested methods**

- 0% Arrays.deepEquals(Object[], Object[]) : b
- 0% Arrays.deepToString(Object[], StringBuffer)
- 0% CopyOnWriteArrayList.COWSubList.rem
- 0% CopyOnWriteArrayList.COWSubList.reta
- 0% CopyOnWriteArrayList.COWSubList.add
- 0% ScheduledThreadPoolExecutor.DelayedE
- 0% ScheduledThreadPoolExecutor.DelayedE
- 0% Arrays.deepHashCode(Object[]) : int (13)
- 0% CopyOnWriteArrayList.COWSubList.equ
- 0% CopyOnWriteArrayList.COWSubList.rem
- 0% CopyOnWriteArrayList.COWSubList.rem
- 0% Collections.CheckedMap.putAll(Map) : v
- 0% CopyOnWriteArrayList.COWSubList.add
- 0% Collections.CheckedMap.EntrySetView.tr
- 0% CopyOnWriteArrayList.retainAll(Collection)
- 0% CopyOnWriteArrayList.COWSubList.toSI
- 0% Collections.frequency(Collection, Object)
- 0% LinkedList.indexOf(Object) : int (6)
- 0% LinkedList.lastIndexOf(Object) : int (6)
- 0% ReentrantLock.FairSync.tryLock(long) : l

Report generated by Atlassian Clover v 4.0.0 on Sr lip 23 2014 10:58:04 CEST using coverage data from Sr lip 23 2014 10:58:01 CEST.

Clover: Developer License registered to Atlassian.

Created in 2014 by Atlassian. Licensed under a Creative Commons Attribution 2.5 Australia License.

## Package-level and project-level overview of application and test code

The "Application code" and the "Test code" tabs show list of packages along with their metrics and their code coverage.

In case of a project-level overview, there is an additional toggle "Metrics from sub-packages: Separated / Aggregated", which allows to calculate an aggregated code coverage and code metrics, i.e. given package will include also metrics from its sub-packages.

Table columns are sortable and customizable.

The screenshot shows the Clover project overview for 'backport-util-concurrent'. The interface includes a sidebar with package filters (default-pkg: 88.3%, edu.emory, sun.misc: 0%) and a main content area with tabs for Dashboard, Application code, Test code, Test results, Top risks, Quick wins, and Coverage tree map. The 'Code metrics' section displays summary statistics: Branches: 3 784, Statements: 8 465, Methods: 2 166, Classes: 215, Files: 83, Packages: 6, LOC: 33 802, NLOC: 15 283, Total complexity: 4 384, Complexity density: 0.52, Statements/Method: 3.91, Methods/Class: 10.07, Classes/Package: 35.83, Average method complexity: 2.02. Below this, a table shows project-level metrics for 'Clover database Šr lip 23 2014 10:49:52 CEST' with 6 packages, an average method complexity of 2.02, 4 276 uncovered elements, and 70.3% total coverage. A second table, titled 'Metrics from sub-packages: Separated / Aggregated', shows detailed coverage for individual packages:

Package	Files	Average Method Complexity	Uncovered Elements	TOTAL Coverage
sun.misc	1	1	7	0%
edu.emory.mathcs.backport.java.util	17	1.75	2 135	49.1%
edu.emory.mathcs.backport.java.util.concurrent.helpers	5	2.77	91	67.5%
edu.emory.mathcs.backport.java.util.concurrent	44	2.27	1 827	78.5%
edu.emory.mathcs.backport.java.util.concurrent.locks	7	2.21	192	80.9%
edu.emory.mathcs.backport.java.util.concurrent.atomic	9	1.29	24	94.2%

At the bottom, a footer note states: 'Report generated by Atlassian Clover v 4.0.0 on Šr lip 23 2014 10:58:04 CEST using coverage data from Šr lip 23 2014 10:58:01 CEST. Clover: Developer License registered to Atlassian.'

## Source file view

This page contains few sections:

- code metrics for current source file
- list of classes in the source file, the "Show methods" opens a dialog showing detailed data for a class
- number of tests "hitting" the source file, the "Select tests to highlight the test coverage" opens list of tests; it's possible to select them to see per-test coverage
- source code view page with highlighted code coverage

In the source view, the left-most column shows line numbers. The second column shows the number of times a particular line has been executed during the test run.

If a line is never executed or has only been partially executed, the entire line of code will be highlighted in red. You can hover the mouse over a line to get a pop-up describing in detail the coverage information for that line.

You can click on the 'Show legend' button to see more details about highlighting.

Rendered code provides also [source cross referencing](#) and [stack trace navigation](#).

The screenshot displays the Clover IDE interface for the project 'backport-util-concurrent'. On the left, the 'Project overview' section shows a package tree with 'default-pkg' at 88.3% coverage and 'sun.misc' at 0%. The main area is titled 'File SynchronousQueueTest.java' and has two tabs: 'Source file' and 'Test results'. Below the tabs, 'Code metrics' are shown: Branches: 6, Statements: 385, Methods: 72, Classes: 1, LOC: 899, NLOC: 650, Total complexity: 129, Complexity density: 0.34, Statements/Method: 5.35, Methods/Class: 72, Average method complexity: 1.79. A 'Classes' table lists 'SynchronousQueueTest' with 19 lines, 385 total statements, 129 complexity, 68 uncovered elements, and 85.3% total coverage. The 'Source view' section shows the Java code for 'SynchronousQueueTest' with line numbers 1 through 60. The code includes imports for JUnit and various utility classes, and defines a 'SynchronousQueueTest' class with several test methods like 'testEmptyFull()', 'testFairEmptyFull()', and 'testOfferNull()'.

### Test results

The "Test results" tab shows information about executed tests - for an entire project, a package, a class as well as for a single test case.

What is worth to note is that a class-level summary may contain multiple tests having the same name - it may happen in case you have executed test several times or you have used a test framework allowing test iterations (for instance, the JUnit4 with @Parameterized annotation or the Spock framework).

A test case page shows detailed information like test result, duration, stack traces (if any) as well as a list of classes which were covered by the test.

Project Clover database Sr lip 23 2014 10:49:52 CEST  
Package default-pkg

Test code | Test results

Package	Tests	Fail	Error	Time	% Pass
default-pkg	70	0	0	13,867	100%

Test Classes	Tests	Fail	Error	Time(secs)	% Pass
SynchronousQueueTest	51	0	0	12,974	100%
AtomicLongArrayTest	19	0	0	0,893	100%

Project Clover database Sr lip 23 2014 10:49:52 CEST / Package default-pkg  
Class SynchronousQueueTest

Source file | Test results

Class	Tests	Fail	Error	Time (secs)	% Tests Success
SynchronousQueueTest	51	0	0	12,974	100%

Tests	Started	Status	Time (secs)	Message
SynchronousQueueTest.testTimedPoll0	23 lip 10:55:14	PASS	0	
SynchronousQueueTest.testAddAllSelf	23 lip 10:55:09	PASS	0	
SynchronousQueueTest.testDrainToN	23 lip 10:55:22	PASS	0,301	
SynchronousQueueTest.testSerialization	23 lip 10:55:21	PASS	0,013	

Project Clover database Sr lip 23 2014 10:49:52 CEST / Package default-pkg / Class SynchronousQueueTest  
Test class testTimedPoll0

Test	Status	Start time	Time (seconds)	Message
testTimedPoll0	PASS	23 lip 10:55:14	0.0	

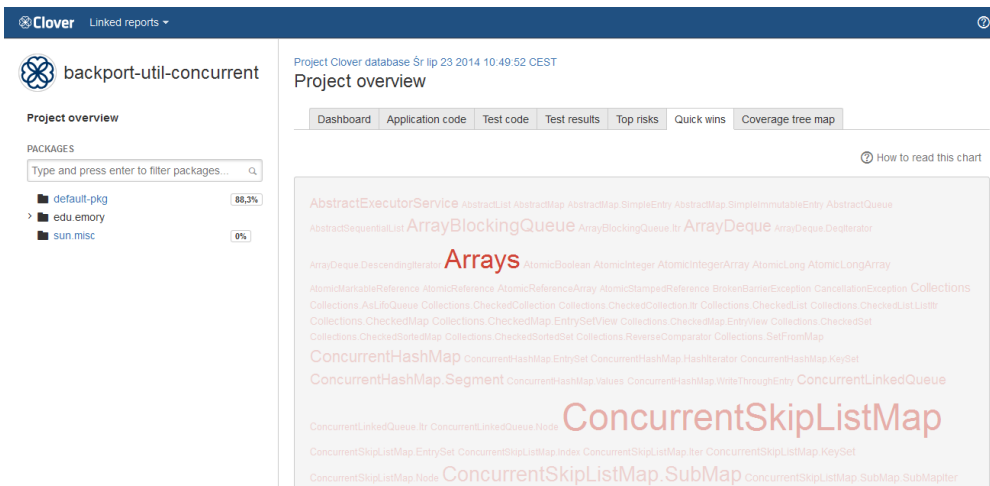
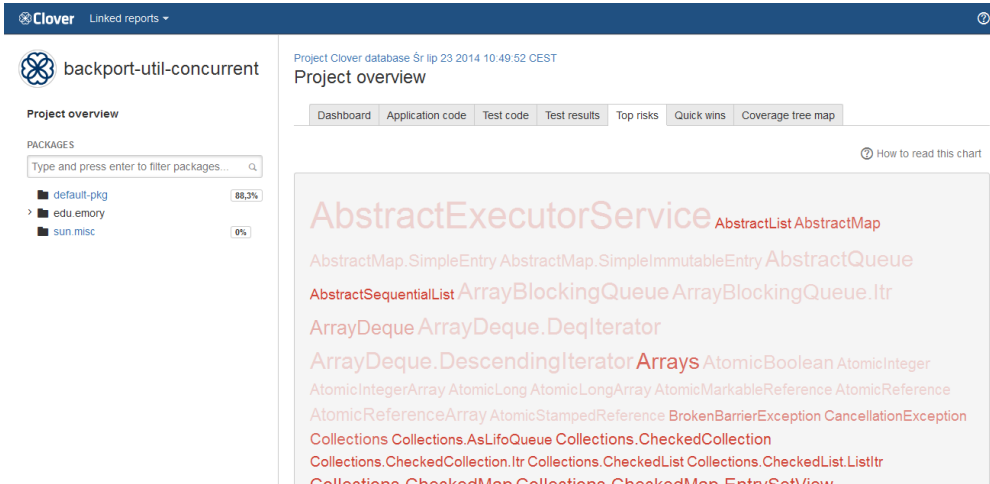
  

Target Class	Coverage contributed by testTimedPoll0
edu.emory.mathcs.backport.java.util.concurrent.SynchronousQueue.LifoWaitQueue	70%
edu.emory.mathcs.backport.java.util.AbstractCollection	33,3%
edu.emory.mathcs.backport.java.util.concurrent.SynchronousQueue.Node	17,8%
edu.emory.mathcs.backport.java.util.concurrent.SynchronousQueue	14,4%
edu.emory.mathcs.backport.java.util.concurrent.locks.ReentrantLock.NonfairSync	13,2%
edu.emory.mathcs.backport.java.util.concurrent.locks.ReentrantLock	9,4%
edu.emory.mathcs.backport.java.util.concurrent.TimeUnit	4,4%
edu.emory.mathcs.backport.java.util.concurrent.locks.ReentrantLock.Sync	2,4%
edu.emory.mathcs.backport.java.util.AbstractQueue	2,2%

### 'Top risks' and 'Quick wins' tag clouds

The Top Risks tag cloud highlights those classes that are the most complex, yet are the least covered by your tests. The larger and redder the class, the greater the risk that class poses for your project or package. Font size represents the Average Method Complexity metric, while the colour represents the Total Coverage metric (vivid red for 0%, pale red for 100%).

The Quick Wins tag cloud highlights the "low hanging coverage fruit" of your project or package. You will achieve the greatest increase in overall code coverage by covering the largest, reddest classes first. Big red classes contain the highest number of untested elements. Font size represents the Number of Elements metric, while the colour represents the Number of Elements Untested (vivid red means more untested).

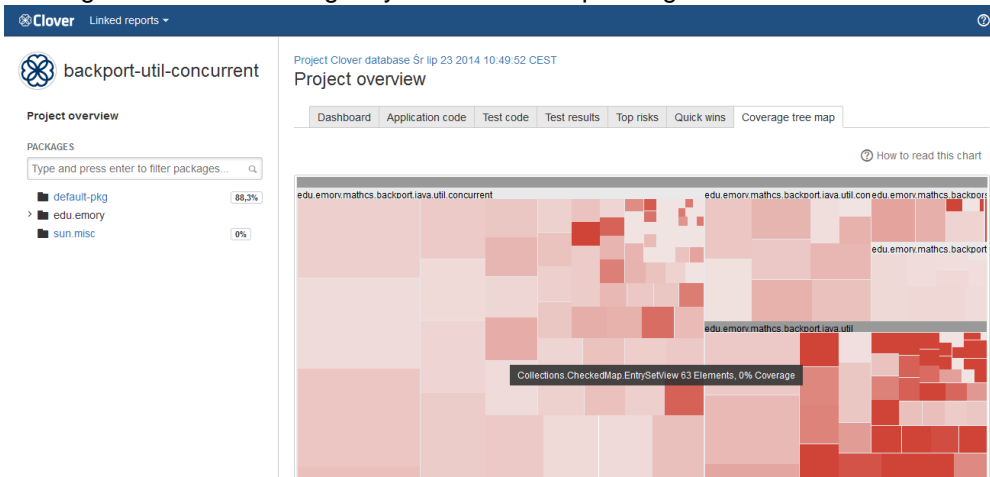


### Coverage tree map

The coverage tree map report allows simultaneous comparison of classes and packages by complexity and by code coverage. This is useful for spotting untested clusters of code. The tree map is divided by a package (labelled) and then further divided by a class (unlabelled). The size of the package or class indicates its complexity (larger squares indicate greater complexity, while smaller squares indicate less complexity). Colours indicate the level of coverage, as follows:

- pale red for most covered
- vivid red for least covered

Clicking on a class will navigate you to the corresponding source file view.




See also: 'Historical' Report.



## Classic report style (Clover 3 and Clover 4)

The screen shot below shows a generated Classic HTML report.

- **In the top left-hand corner is the list of packages.** You can view all classes in the project or select a particular package to view. Clicking on the name of a package will bring up the relevant classes in the frame below it. Selecting one of these classes will bring up the source code in the frame on the right.
- **The header provides summary information relating to the current project.** The left hand side displays the report title and the time of the coverage contained in the report. For current reports, the timestamp is the timestamp of the most recent run of tests. The right hand side of the header displays metrics for the package, file or project overview which is currently selected. Depending on the current selection, the metrics include all or a subset of:
  - Number of Lines of Code (LOC)
  - Number of Non-commented Lines of Code (NCLOC)
  -  This information is currently not available with Clover reports on Groovy code.
  - Number of Methods
  - Number of Classes
  - Number of Files
  - Number of Packages.
- **Test coverage is indicated by colour-coding.** For [details](#), click 'SHOW HELP' in the report header.
- **Inline help appears when you mouse-over any column header, button, etc.**

The screen shot shows the report for the `Money.java` source file with the green and red bar at the top showing the amount of code coverage on this class. The method, statement and conditional coverage percentages are beside this.

**Clover Demo  
Clover Coverage  
Report**

[Overview](#)  
[Aggregate](#)  
[Test Results](#)  
[Coverage Clouds](#)

**All Packages**

default-pkg (94.7%)

**Clover Coverage Report - Clover Demo**  
Coverage timestamp: Thu Nov 30 2006 14:45:18 EST

**Overview Package File**

[FRAMES](#) [NO FRAMES](#) [SHOW HELP](#)

Statistics for file Money

Stmts: 25	LOC:
Branches: 8	NCLOC:
Methods: 14	
Classes: 1	

<input checked="" type="checkbox"/> <b>Money</b>	Line # 7	Total Statements 25	Complexity 5	TOTAL Cover
--	----------	---------------------	--------------	-------------

Show Tests (22) Select All [Deselect All](#)

---

Collapse All

```

1 //package junit.samples.money;
2
3 /**
4  * A simple Money.
5  *
6  */
7
8 public class Money implements IMoney {
9
10     private int fAmount;
11     private String fCurrency;
12
13     /**
14      * Constructs a money from the given amount and cur
15      */
16     public Money(int amount, String currency) {
17         fAmount= amount;
18         fCurrency= currency;
19     }
20     /**
21      * Adds a money to this money. Forwards the request
22      */
23     public IMoney add(IMoney m) {
24         return m.addMoney(this);
25     }
26     public IMoney addMoney(Money m) {
27         if (m.currency().equals(currency() )
28             return new Money(amount()+m.amount(), curre
29             return MoneyBag.create(this, m);
30     }
31     public IMoney addMoneyBag(MoneyBag s) {
32         return s.addMoney(this);
33     }

```

	All	Targets	Tests		
Class				Coverage	
<input checked="" type="checkbox"/> Money				(89.4%)	
<input checked="" type="checkbox"/> MoneyBag				(94.4%)	
<input checked="" type="checkbox"/> MoneyTest				(97.8%)	

The left-most column shows line numbers. The second column shows the number of times a particular line has been executed during the test run. As you can see, lines 15-17 have been run 156 times by the JUnit tests, whereas line 28 has only been run twice.

If a line is never executed or has only been partially executed, the entire line of code will be highlighted in red. Depending on your browser, you can hover the mouse over a line to get a popup describing in detail the coverage information for that line. The following screenshot shows the coverage on a section of the MoneyBag.java source file:

```

44     }
45     13     fMonies.removeElement(old);
46     13     IMoney sum= old.add(aMoney);
47     13     if (sum.isZero())
48     5       return;
49     8     fMonies.addElement(sum);
50     }
51     14     public boolean equals(Object anObject) {
52     14         if (isZero())
53     0         if (anObject instanceof IMoney)
54     0         return ((IMoney)anObject).isZero();
55
56     14         if (anObject instanceof MoneyBag) {
57     12             MoneyBag aMoneyBag= (MoneyBag)anObject;
58     12             if (aMoneyBag.fMonies.size() != fMonies.size())
59     0             return false;
60
61     32         for (Enumeration e= fMonies.elements(); e.hasMoreElements(); ) {
62     22             Money m= (Money) e.nextElement();
63     22             if (!aMoneyBag.contains(m))
64     2         return false;
65         }
66     10         return true;
67     }
68     2     return false;
69     }

```

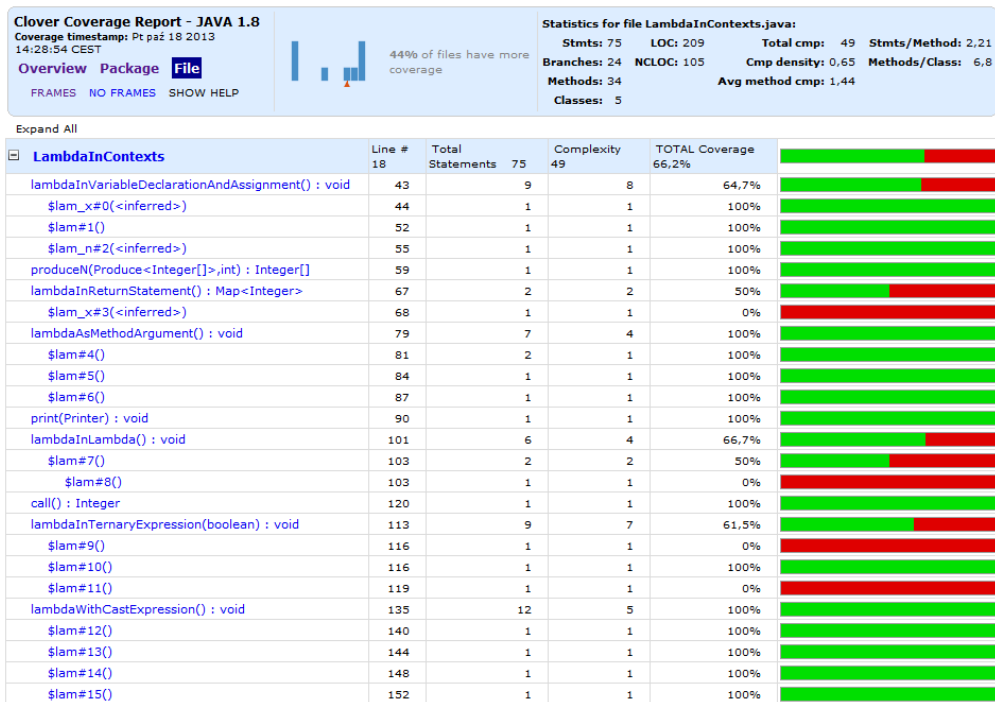
Although line 52 of the above `MoneyBag` class has been executed 14 times, the method `isZero()` has never evaluated to `true` so it has not been fully tested. Therefore it, and the following two lines, are highlighted. This is also the case with lines 58 and 59.

This highlighting feature makes it easy for you to see which parts of the code have not been fully exercised by your tests so that you can then improve testing to provide better code coverage.

**If any of the lines shaded red contained a bug, they may never be detected because the tests as they are don't test those parts of the code.**

If you click on a **[+]** button on the left side of the class name, table will expand and show methods declared in it. If the HTML report was generated with `showLambdaFunctions=true` and `showInnerFunctions=true` options, the table will also show lambda functions declared inside methods or assigned to a class field.

Screen shot: sample report for code containing lambda functions



## Appendix: naming convention of lambda functions

Naming convention for Java 8 lambda functions is as follows:  
 <lambda\_prefix><parameter\_list><sequence\_number> where:

- <lambda\_prefix> is **\$lam**
- <parameter\_list> is a list of argument names separated by underscore, like **x\_y\_z**, list can be empty
- <sequence\_number> is **#N** where N counts definitions of a lambda function in given source file (starting from 0)

For example: **\$lam\_x\_y\_z#6** means a sixth lambda in the source file having the **(x, y, z) =>** signature.

Why such naming convention?

- if you use meaningful names of lambda arguments, you'll be able to easily find such lambda function on a list
- sequence number helps to distinguish zero-argument lambda functions

## Coverage Legend

In Clover HTML reports, test coverage is indicated by the following colour-coded legend.

## Source view

[Collapse all methods](#)

**Legend**

**Left margin ruler**  
Left margin ruler shows information about global and per-test coverage

line#	hit count	description
1	17	line was covered, but not by a test (e.g. by main(), setUp() or tearDown())
2	86	line was covered by test(s) which passed
3	7	line was covered by test(s) which did not pass (includeFailedTests)
4	7	line was covered by test(s) which did not pass (includeFailedTests)
5	1	line is covered partially (i.e. some branch or a statement was not covered)
6	1	line is covered partially, but not by a test
7	0	line was not covered at all
8		line was filtered

**Source code highlighting**  
Highlighting shows information about contributing tests as well as about code which was not covered

- line not covered by any of the selected tests
- line hit by more than one of the selected tests
- line hit by one test only (unique per-test coverage)
- line hit by one or more of the selected tests that all failed
- line was not covered (partially or at all)
- line was filtered

The page margin has three colour indicators:

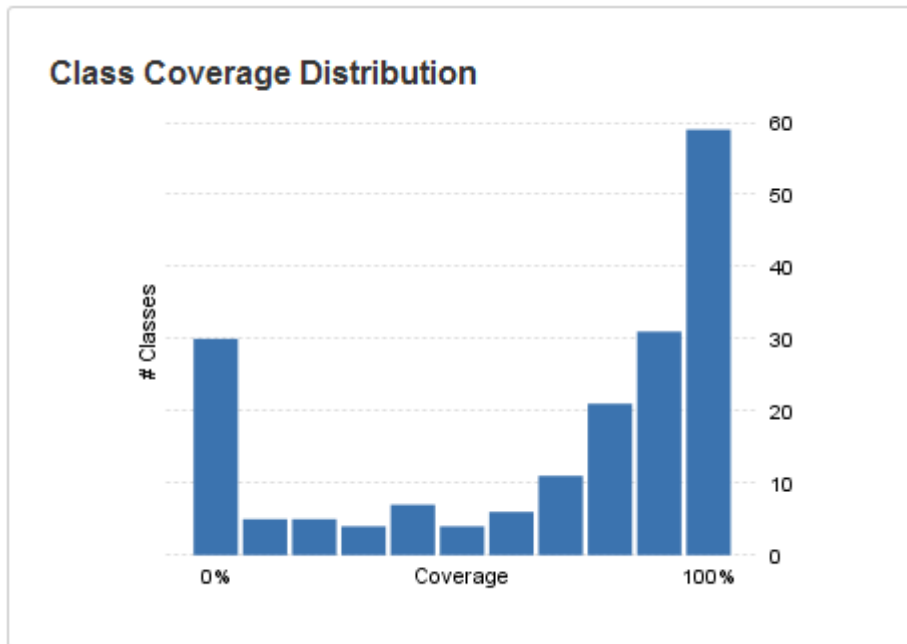
- left half with line number is the "optimistic marker", which is highlighted in:
  - green when at least part of the code line was covered during execution
  - otherwise red if it was not covered at all
- right half with number of hit counts (i.e. how many times given line was executed) is the "pessimistic marker" which is highlighted in
  - red if at least part of line was not executed or when branch expression was not evaluated to both true and false
  - otherwise green if the whole line was fully covered
- narrow strip related with per-test code coverage, which is:
  - dark green if given source line was executed at least once from a successful test case (i.e. test has passed), or
  - dark yellow if given source line was executed from one or more test cases, but all tests were failed or
  - empty if given code was not executed from a test case (for instance from "main()" method or called by JVM GC "finalize()" method)

### Dashboard Widgets

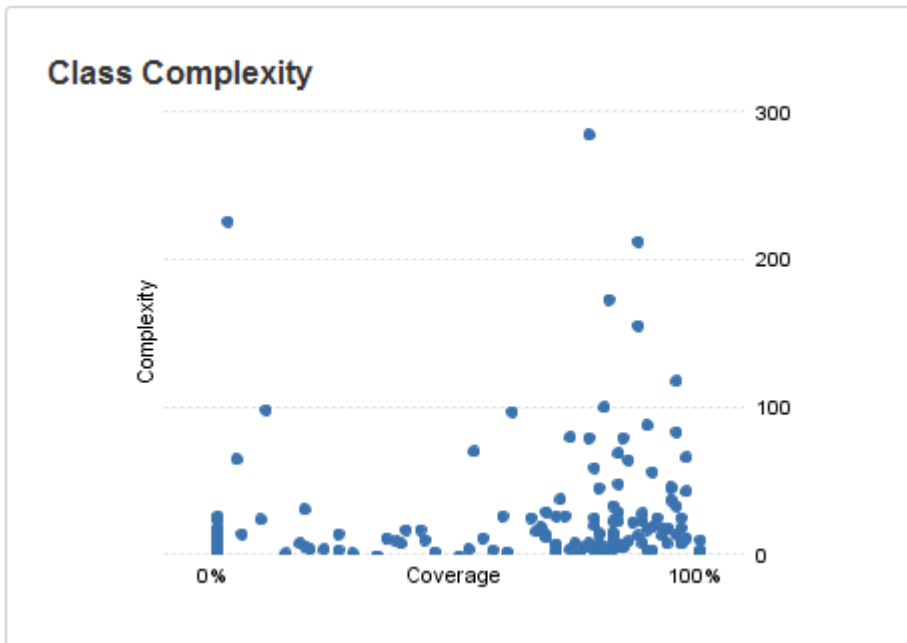
In Clover's HTML reports, the dashboard page provides a summary of the coverage and unit test results, as well as suggested entry points for the rest of the report.

Selected widgets:

- The **Class Coverage Distribution** chart shows a breakdown of how many classes have a given percentage of coverage. This offers a more granular insight into the nature of your project's coverage, compared to an aggregate overall percentage. You can mouse-over the chart's data points to see exactly how many classes are represented in each band.



- The **Class Complexity** chart allows you to quickly spot classes with high complexity, as well as low coverage. Expressed as a scatter plot, outlying classes are readily visible because they will rise prominently above the mean. You can mouse-over the chart's data points to see the names of the classes displayed and also click them, to access information about that class.



### Source Cross-Referencing in Reports

Clover provides source cross-referencing in HTML reports. Identifiers are looked up in the current scope and linked to the appropriate source where possible.




```

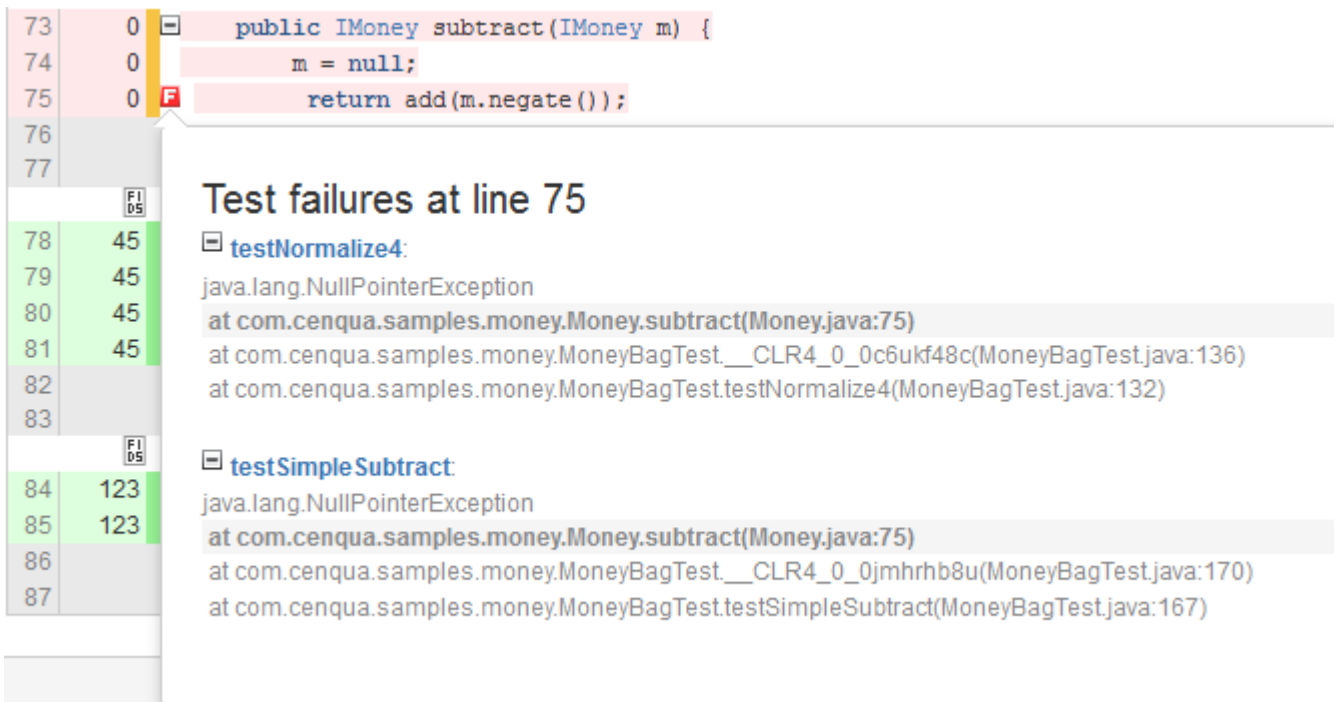
1109 273 public ThreadPoolExecutor(int corePoolSize,
1110                                int maximumPoolSize,
1111                                long keepAliveTime,
1112                                TimeUnit unit,
1113                                BlockingQueue workQueue) {
1114 273     this(corePoolSize, maximumPoolSize, keepAliveTime, unit, workQueue,
1115                Executors.defaultThreadFactory(), defaultHandler);
1116
1117 }
    
```

### Stack Trace Navigation

#### Stack Trace Navigation

Clover's Stack Trace Navigation feature provides quick, in-context information about unit test failures.

When a test failure occurs, the exception that caused the failure is captured and analysed by Clover. At each source line that propagated the exception, Clover places an  icon. Clicking on this icon provides a popup window showing all exceptions that were propagated from this line, and the full stack trace for that exception. This allows a developer to quickly navigate up or down the stack trace to investigate the cause of the failure.



```

73 0 public IMoney subtract(IMoney m) {
74 0     m = null;
75 0  return add(m.negate());
76
77
78 45  testNormalize4:
79 45 java.lang.NullPointerException
80 45 at com.cenqua.samples.money.Money.subtract(Money.java:75)
81 45 at com.cenqua.samples.money.MoneyBagTest__CLR4_0_0c6ukf48c(MoneyBagTest.java:136)
82 at com.cenqua.samples.money.MoneyBagTest.testNormalize4(MoneyBagTest.java:132)
83
84 123  testSimpleSubtract:
85 123 java.lang.NullPointerException
86 at com.cenqua.samples.money.Money.subtract(Money.java:75)
87 at com.cenqua.samples.money.MoneyBagTest__CLR4_0_0jmhrhb8u(MoneyBagTest.java:170)
    at com.cenqua.samples.money.MoneyBagTest.testSimpleSubtract(MoneyBagTest.java:167)
    
```

### Tag Clouds

A Tag Cloud or 'weighted list' is a way of visually representing information.

In Clover, '**Coverage Clouds**' provide an instant overview of your entire project and individual packages, enabling you to identify areas of your code that pose the highest risks or shortcomings. Each Coverage Cloud displays two metrics per Java or Groovy class. One metric is displayed via the font size, and the other via the font colour. Each attribute has relative weighting across the entire project. Classes are sorted alphabetically.

Tool tips on each class name provide you with the real values for each metric.

#### Risks

The Project Risks / Package Risks Cloud highlights those classes that are the **most complex, yet are the least covered** by your tests. The larger and redder the class, the greater the risk that class poses for your project or package. Package risk clouds can be toggled to include or exclude classes in sub-packages.

Metric	Attribute
--------	-----------

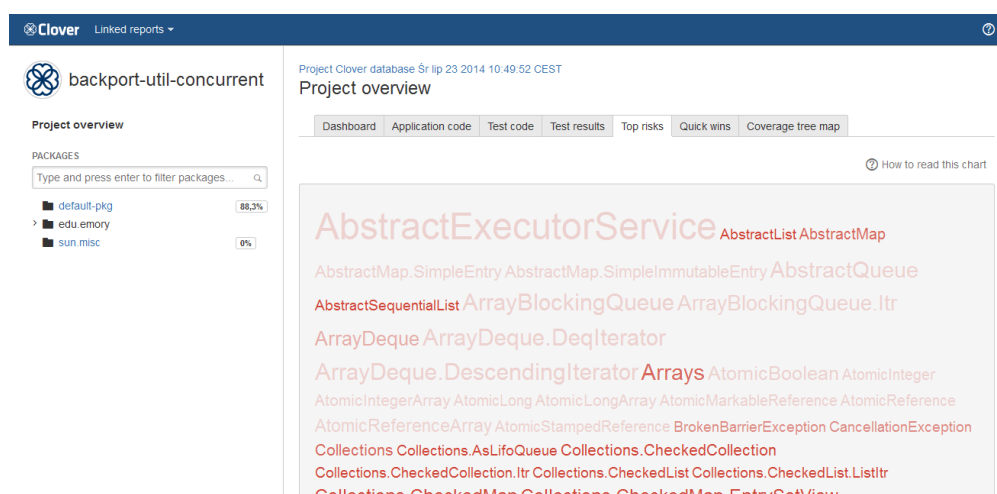
Average Method Complexity	Font Size
% Coverage	Font Color

### Quick Wins

This Cloud highlights the "**low hanging coverage fruit**" of your project or package. You will achieve the greatest increase in overall project coverage by covering the largest, reddest classes first. Package Quick Win clouds can be toggled to include or exclude classes in sub-packages.

Metric	Attribute
Number of Elements	Font Size
Number of Elements Untested	Font Color

### Screen shot: Top risks cloud



## 'Historical' Report

'Historical' reports display graphical and numerical data relating to *sets of coverage data collected over time* for the project.

Clover 4 brings new HTML report developed according to the [Atlassian Design Guidelines](#); it's shortly named "ADG". Clover 3 offers an old-style JavaDoc-like format, named "Classic". It is still possible to *fall-back to the "Classic" report* in Clover 4, however this report style is deprecated.

- ADG report style (Clover 4 only)
- Classic report style (Clover 3 and Clover 4)
  - Custom charts

### ADG report style (Clover 4 only)

#### General

Historical report consists of several sections:

- Date range and Code metrics widgets
- Coverage overview - shows coverage from the last [history point](#)
- Added classes - classes which have been added in the specified time span
- Changed classes - classes for which metrics have changed above specified thresholds in given time span
- Charts - set of charts (they are [customizable](#))



### Blue application header

- Clover logo - opens the Atlassian Clover home page
- Linked reports - shows reports linked with the current one (optional)
- Help icon - opens the Clover documentation home page

Clover Linked reports ▾

backport-util-concurrent

- [Coverage overview](#)
- [Added classes](#)
- [Changed classes](#)
- [Charts](#)

#### Historical coverage report

**Date range**

From: Wt cze 24 2014 22:19:17 CEST

To: Šr lip 23 2014 10:58:01 CEST

**Code metrics**

Branches: 3 784    Statements: 8 465

Methods: 2 166    Classes: 215    Files: 83

Packages: 6    LOC: 33 802    NCLoc: 15 283

Total complexity: 4 384    Complexity density: 0.52

Statements/Method: 3.91    Methods/Class: 10.07

Classes/Package: 35.83

Average method complexity: 2.02

**Coverage overview**

	Conditionals	Statements	Methods	TOTAL
Project	62,8%	73,7%	70,1%	70,3%

**Added classes**

Table shows classes which have been added over the last 4 weeks. Actual interval (based on timestamps of history points) is 4 weeks. Showing maximum 5 classes. Metric used: % TOTAL Coverage, max value is 100.

ⓘ No new classes.

**Changed classes**

Table shows classes for which metric has changed (increased or decreased) above the threshold (+/-1) over the last 4 weeks. Actual interval (based on timestamps of history points) is 4 weeks. Showing maximum 5 classes. Metric used: % TOTAL Coverage, max value is 100.

**Gainners**

Class	% TOTAL Coverage
edu.emory.mathcs.backport.java.util.concurrent.ScheduledThreadPoolExecutor.ScheduledFutureTask	+2,5 (78,5%)

**Losers**

% TOTAL Coverage	Class
(67,1%) -13,2	edu.emory.mathcs.backport.java.util.concurrent.helpers.WaitQueue.WaitNode
(82%) -2,2	edu.emory.mathcs.backport.java.util.concurrent.locks.CondVar
(87,5%) -1,8	edu.emory.mathcs.backport.java.util.concurrent.CountDownLatch

**Charts**

**Metrics**

**Coverage**

Report generated by Atlassian Clover v 4.0.0 on Šr lip 23 2014 10:58:12 CEST using coverage data from Wt cze 24 2014 22:19:17 CEST.

Clover: Developer License registered to Atlassian.

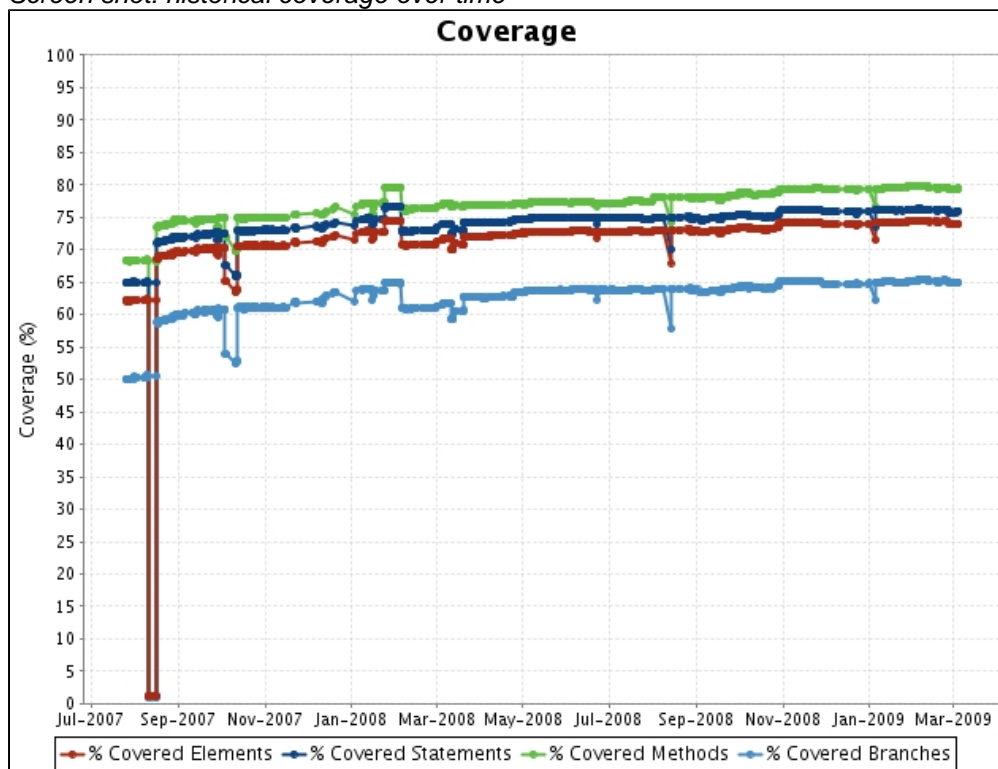
Created in 2014 by Atlassian. Licensed under a [Creative Commons Attribution 2.5 Australia License](#).

### Classic report style (Clover 3 and Clover 4)

When you view the report you should see a picture similar to the screenshot below, although it is likely that the graphs that you produce will contain different values.

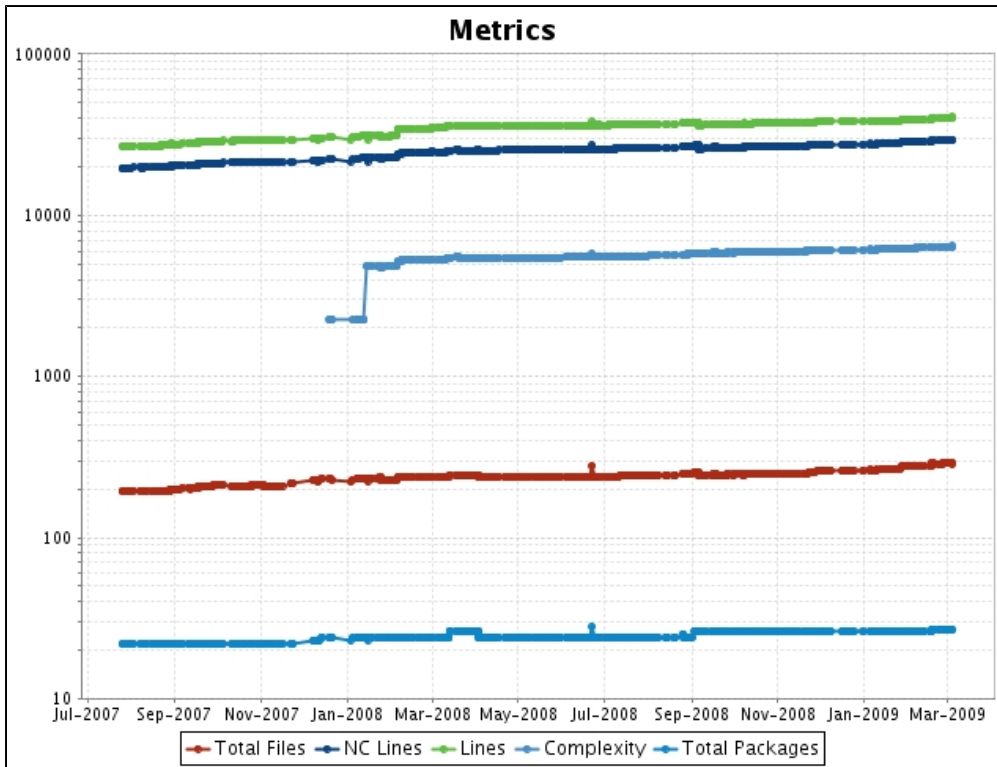
- Like the 'current' report, the historical report begins with a header containing relevant project information. This includes the report title, the project metrics and the period for which history points are included in the report. (A *history point* is a snapshot of code coverage and metrics data for the project at a particular point in time.)
- Below this header is the **Project Overview Chart** which shows the branch, statement, method and total coverage percentages for the project for the **most recent** history point included in the report.
- The '**Coverage over time**' graph shows the percentage values of branch, statement, method and total coverage for each history point and plots them against time in an easy-to-read chart.

Screen shot: *historical coverage over time*



- The '**Metrics over time**' graph shows the project statistics for each history point plotted against time. It is therefore possible to observe changes in metrics such as the number of methods. In the example, the number of methods can be seen shown in green.

Screenshot: *Historical Metrics Over Time*



- The section '**Classes added**' allows you to monitor the coverage of newly added classes.
- The final section, '**Movers**', displays classes that have increased or decreased in coverage by more than a specified percentage point threshold over a particular time interval, the default being one percentage point over the two latest history points.

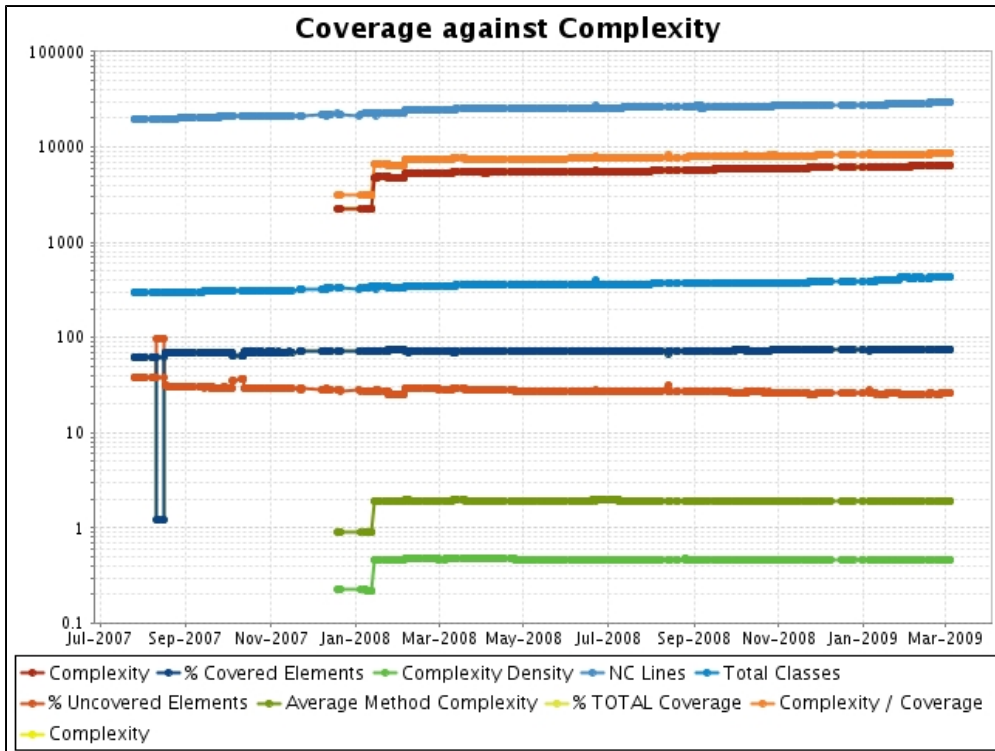
Screen shot: *Classes Added and Movers*

Classes added over the last 1 week (Column: % TOTAL Coverage) (Actual interval: 1 week, Range: 5, Max %)	
<a href="#">com.googlecode.guice.StrictContainerTestSuiteBuilder.SecurityManagedTest</a>	<div style="width: 100%; height: 10px; background-color: green;"></div>
<a href="#">com.googlecode.guice.StrictContainerTestSuite</a>	<div style="width: 100%; height: 10px; background-color: green;"></div>
<a href="#">com.googlecode.guice.StrictContainerTestSuiteBuilder.NonSystemClassLoader</a>	<div style="width: 100%; height: 10px; background-color: green;"></div>
<a href="#">com.googlecode.guice.StrictContainerTestSuiteBuilder</a>	<div style="width: 100%; height: 10px; background-color: green;"></div>
<a href="#">com.googlecode.guice.StrictContainerTestSuiteBuilder.SuiteConstructionError</a>	<div style="width: 100%; height: 10px; background-color: red;"></div>
Movers over the last 1 week (Column: % TOTAL Coverage) (Actual interval: 1 week, Range: 5, Threshold: +/- 1)	
<a href="#">com.google.inject.internal.MapMaker.LinkedSoftEntry</a>	<div style="width: 100%; height: 10px; background-color: green;"></div>
<a href="#">com.google.inject.internal.FinalizableReferenceQueue.SystemLoader</a>	<div style="width: 80%; height: 10px; background-color: green;"></div>
<a href="#">com.google.inject.internal.FinalizableReferenceQueue.DecoupledLoader</a>	<div style="width: 60%; height: 10px; background-color: green;"></div>
<a href="#">com.google.inject.internal.FinalizableReferenceQueue</a>	<div style="width: 40%; height: 10px; background-color: green;"></div>
<a href="#">com.google.inject.internal.BytecodeGen</a>	<div style="width: 20%; height: 10px; background-color: green;"></div>
(77.8%) -22.2	<div style="width: 100%; height: 10px; background-color: red;"></div> <a href="#">com.google.inject.in</a>

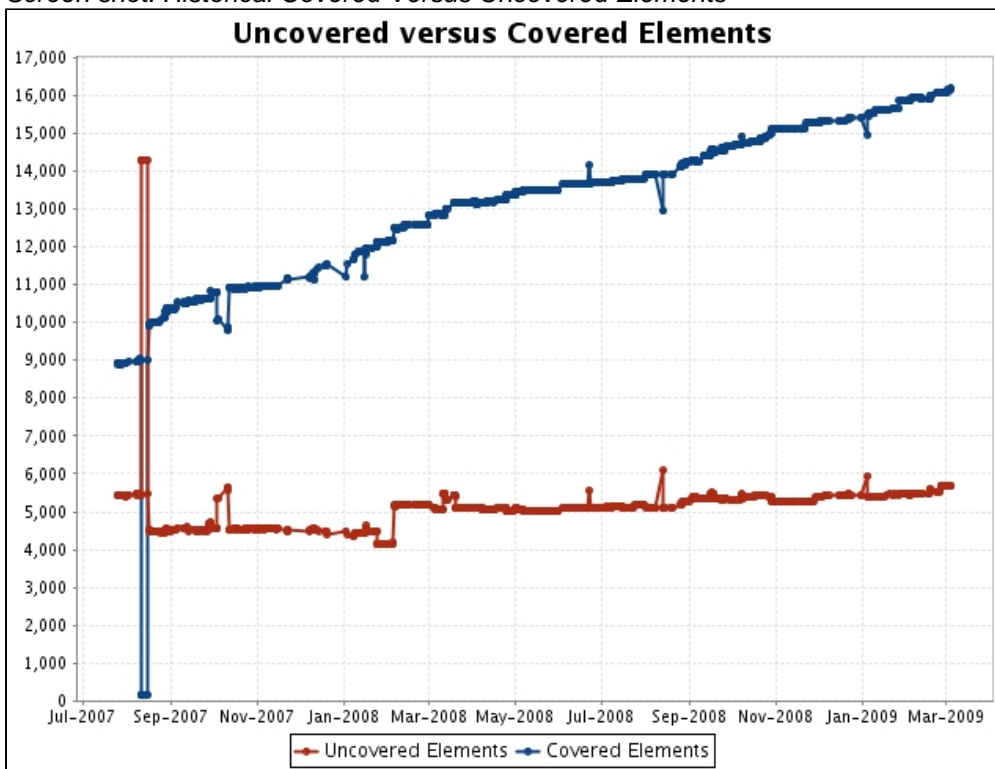
Custom charts

You can also view charts for coverage against complexity and covered versus uncovered elements (see below). These are just two examples. Any metrics can be charted, including user-defined metrics, customised for your needs.

Screen shot: *Historical Coverage Against Complexity*



Screen shot: Historical Covered Versus Uncovered Elements



✔ The tutorial discusses how you can customise many aspects of the historical report.

## 5. Configuring Reports

For information on generating reports, see the following Ant Tasks:

- `<clover-html-report>`
- `<clover-pdf-report>`
- `<clover-report>`

Also see the tutorial '[Clover-for-Ant tutorial](#)'.

Further reading:

- [Unit Test Results and Per-Test Coverage](#) — Clover has the ability to integrate Unit Test results and per-test coverage information into Coverage reports. This unique feature gives you a powerful insight into how well tested your covered code actually is, by showing you exactly which statements each of your tests cover.
- [Using Coverage Contexts](#)
- [Using Spans](#)
  - [Specifying an Interval](#)
- [Sharing Report Formats](#) — You can share report formats across a number of reports. This allows you to standardise on a set of report formats and use these for all of your reports.
- [Extracting Coverage Data programmatically](#)

## Unit Test Results and Per-Test Coverage

### *Per-Test Coverage*

Clover has the ability to integrate Unit Test results and per-test coverage information into Coverage reports. This unique feature gives you a powerful insight into how well tested your covered code actually is, by showing you exactly which statements each of your tests cover.

To take advantage of this feature all you need to do is:

- **Allow Clover to instrument your test classes.**

You must ensure that all your test classes are included for Clover instrumentation. To enable Clover instrumentation of your source code, use the `<clover-setup>` task.

This allows Clover to specially mark test methods to enable per-test coverage reporting. The Clover instrumenter [automatically detects test methods](#) for Junit 3.x, Junit 4.x and TestNG. If you are using another testing framework, see "Advanced Test Case Detection" below.

### *Advanced Test Case Detection*

If you are using a testing framework which does not use Junit or TestNG conventions for determining test methods, the nested `<testsources/>` element of `<clover-setup>` and `<clover-instr>` will allow you to specify the convention used by your framework.

The [Instinct](#) framework for example uses the following:

```
<clover-setup>
  <testsources dir="tests">
    <testclass>
      <testmethod annotation="Specification"/>
      <testmethod name="^should.*"/>
      <testmethod name="^must.*"/>
    </testclass>
  </testsources>
</clover-setup>
```

You can optionally group `<testclass/>` definitions in `<or/>` or `<and/>` elements. Each `<testmethod/>` element is automatically grouped by an OR.

```
<clover-setup>
  <testsources dir="tests">
    <or>
      <testclass name=".*Context">
        <testmethod annotation="Specification"/>
      </testclass>
      <testclass name=".*Test">
        <testmethod name="test.*"/>
      </testclass>
    </or>
  </testsources>
</clover-setup>
```

### **i** Differences between Groovy and Java with respect to **<clover-setup>** tasks.

- In Java, the names of annotations and classes that you use to define your test code could be either fully-qualified or not fully-qualified, and depend on how you declared these annotations and classes in your actual Java source code.
- In Groovy, fully qualified annotation and class names are required in your test code, regardless of how you have declared these annotations and classes in your actual Groovy source code.

### **Advanced Test Result Configuration**

Clover does its best to record your test results in the Clover database. In some instances however, Clover can not always do so. Although unit tests using `@Test(expected=Exception.class)` annotations will be marked as passed, more novel JUnit constructs such as [rules](#) may not be recognized and Clover will flag those tests as failed when in fact they passed. To integrate these test results into your Clover reports follow these steps:

- **Enable XML reports from your Unit Test execution.**

In an Ant build, if using the `<junit>` task, you need to add an XML result formatter:

```
<property name="testreport.dir" value="build/test-reports">

  <junit ...>
    ...
    <formatter type="xml"/>
    <batchtest todir="{testreport.dir}">
      ...
    </batchtest>
  </junit>
```

Such results can be similarly produced by the Ant TestNG tasks.

- **Supply the `<testresults>` element at report time.**

If the `<testresults>` element is specified, `<clover-report>` and similar tasks will use these results instead of those collected by Clover. Clover's test result collection may also be switched off via the `dontRecordTestResults` attribute on `<clover-setup>` or `<clover-instr>`.

e.g.

```
<testresults dir="test-results" include="TEST-*.xml"/>
```

### **Limitations**


Clover's per-test coverage collection does not support parallel test execution.

## Using Coverage Contexts

Clover defines a *Context* as a part of source code that matches a specified structure or pattern. Contexts are either pre-defined or user-defined at instrumentation time. Each context must have a unique name. At report time, you can specify which contexts you would like to exclude in the coverage report.

### Block Contexts

Block Contexts are pre-defined by Clover. They represent 'block' syntactic constructs in the Java language. A full list of supported Block Contexts is shown below.

 Clover's block context feature currently does not support Groovy.

Name	Description
static	Static initializer block
instance	Instance initializer block
constructor	Constructor body
method	Method body
switch	Switch statement body
while	While loop body
do	do-while loop body
for	For loop body
if	if body
else	else body
try	try body
catch	catch body
finally	finally body
sync	synchronized block
assert	assert statement
@deprecated	a deprecated block

### Method Contexts

A Method Context represents the set of methods whose signature matches a given pattern. Clover provides several pre-defined method contexts:

Name	Regexp	Description
private	<code>(.* )?private .*</code>	matches all private methods
property	<code>(.* )?public .*(get set is)[A-Z0-9].*</code>	matches all property getters/setters

A method signature includes all annotations, modifiers (public, static, final etc), the return type, the method

name, parameter types and names, the throws clause and exceptions.



#### Note

When matching method signatures against context regexps, whitespace is normalised and comments are ignored.

You can define your own method contexts via the `<methodContext>` sub-element of `<clover-setup>`, or via the configuration panel of your Clover IDE Plugin.



#### Note

Contexts are matched against your source at **instrumentation-time**. This means you need to **re-instrument your code** after defining a new context.

#### Method Contexts with Groovy code

While [Groovy syntax](#) is flexible in nature, the regular expressions defined in the `regexp` parameters of `<methodContext>` elements must match a 'normalised' method signature.

Bear in mind that this is not necessarily how you would define the method in your Groovy source code.

For example, in Groovy code, a method defined via the 'def' keyword is always 'public'. This means that your `regexp` must actually match "public def". Hence, if you wanted to create a `regexp` that matched the following Groovy method:

```
def void foo()
```

Your `regexp` must assume a match against:

```
public def void foo()
```

#### Normalised method signature rules for defining `regexp` parameters:

The following list illustrates the normalised form of the method signature (and hence, order) in which your `regexp` must be defined to match specific methods in your Groovy source code:

1. **Modifiers**– in the following order:
  - a. public
  - b. protected
  - c. private
  - d. abstract
  - e. static
  - f. final
  - g. transient
  - h. volatile
  - i. synchronized
  - j. native
  - k. strictfp
  - l. interface

(Refer to [Sun Java's documentation](#) for more information.)
2. **Type Parameters** (optional) – for example, `<T>`, `<E extends Object>`
3. **Return Type** – for example, `void`, `int`, `String`, `Object[]`
4. **Name** – for example, `myMethod`
5. **Parameter List** – for example, `(String arg1, int arg2)`
6. **Throws** – for example, `throws Exception1, Exception2`

#### Examples of normalized signatures for Groovy



```
// normalized signature is: "private void <init>()"
private PrivateConstructor() {

}

// normalized signature is: "public void <init>(int i)"
public PublicConstructor(int i) {

}

// normalized signature is: "private def implicitVoidMethod()"
private implicitVoidMethod() {

}

// normalized signature is: "private void explicitVoidMethod()"
private void explicitVoidMethod() {

}

// normalized signature is: "private int someNonVoidMethod()"
private int someNonVoidMethod() {
    return 0
}

// normalized signature is: "public static synchronized Object[]
complexSignature(String i, List<Integer> j)"
synchronized static public Object [ ]    complexSignature(    String i, List <
Integer > j) {
    return new Object[0]
}
```

### Statement Contexts

 Clover's statement context feature currently does not support Groovy.

A Statement Context represents the set of statements that match a given pattern. For example, you might want to set up a statement context to allow you to filter out 'noisy' statements (such as logging calls) by defining a statement context regexp `LOG\.debug.*`.

A regular expression defined in a statement context will be matched against a *normalized form* of the statement:

- any white space characters before and after the statement are removed
- any newline characters are removed
- single space character is used to separate code tokens

When writing a regular expression you should take into account that in case of nested statements, the outer statement will contain inner statements as well. Consider the following example:

```
void testStatementContext() {
    while (true) {
        if (logger.isDebugEnabled()) {
            logger.debug("abc");
        }
    }
}
```

in this case, method body has three statements:

a) the while loop

```
while (true) {{ if (logger.isDebugEnabled()) {{ logger.debug("abc"); } }}
```

b) the if condition

```
if (logger.isDebugEnabled()) {{ logger.debug("abc"); }
```

c) the `logger.debug()` method call

```
logger.debug("abc");
```

Assuming that you'd like to filter-out "logger.debug" calls, the regular expression should look like this:

```
<statementcontext name="logger" regexp="^logger\.debug\.*/>
```

Note that if the expression would be written as `^.*logger\.debug\.*`, then it would match also outer statements.

### Using Context Filters



#### Note

This section describes using context filters with Ant. For details of using filters with the IDE plugins, see the individual documentation for the plugin.

#### Filtering catch blocks

In some cases you may not be interested in the coverage of statements inside catch blocks. To filter them, you can use Clover's predefined `catch` context to exclude statements inside catch blocks from a coverage report:

```
<clover-report>
  <current outfile="clover_html">
    <format type="html" filter="catch"/>
  </current>
</clover-report>
```

This generates a source-level HTML report that excludes coverage from statements inside catch blocks.

#### Filtering simple methods

In order to filter-out simple getters and setters you can use built-in "property" method context.

You can define also own filter, based on cyclomatic complexity and/or number of statements. For example:

```
<clover-setup ...>
  <!-- Don't instrument methods which have cyclomatic complexity <= 1 or <= 3
statements -->
  <methodContext name="simple_method" regexp=".*" maxComplexity="1"
maxStatements="3"/>
</clover-setup>
```

### Filtering logging statements

To remove logging statements for coverage reports, you will need to define one or more statement contexts that match logging statements in your source:

```
<clover-setup ...>
  <statementContext name="log" regexp="^LOG\..*" />
  <statementContext name="iflog" regexp="^if \(LOG\.is.*" />
  <methodContext name="main" regexp="public static void main\(String
args\[\\]\).*" />
  ...
</clover-setup>
```

This defines two statement contexts and one method context. The first matches statements that start with 'LOG.' while the second matches statements that start with 'if (LOG.', which is designed to match conditional logging statements such as:

```
if (LOG.isDebugEnabled()) {
    // do some expensive debug logging
}
```

The third matches all 'main' methods that have a String Array named 'args' in the constructor:

```
public static void main(String args[]) throws Exception
```

After defining these contexts, you now need to re-compile with Clover and then re-run your tests. You can then generate a report that excludes logging statements:

```
<clover-report>
  <current outfile="clover_html" title="My Coverage">
    <format type="html" filter="log,iflog" />
  </current>
</clover-report>
```

This generates a source-level HTML report that excludes coverage from logging statements.

## Using Spans

The `span` attribute allows you to control which coverage recordings are merged to form a current coverage report. By default, Clover includes all coverage data found. You can configure it to include a different span of coverage recordings. The `span` attribute lets you do this.

The `span` attribute takes an [Interval](#) which tells Clover how far back in time since the last Clover compilation that coverage recordings should be merged to build the report.

The `span` attribute applies to the following Ant tasks or sub-elements thereof:

- `<clover-check>`
- `<clover-historypoint>`
- `<clover-log>`
- `<clover-merge>`
- `<clover-report>`

### Specifying an Interval

The `interval` type is used to specify a period of time. It consists of a value and a unit specifier, eg. "3 days". The interval type is very flexible about how it interprets the time unit. In general, the first letter is sufficient to indicate the interval unit. For example, the previous example could be written as "3 d". The time ranges

supported are specified in the following table:

Unit specifier	Abbreviation	Example Values
second	s	3 seconds 20s
minute	m	5 minute 7 min, 11m
hour	h	4 hours 2h
day	d	7 days 365d
week	w	4 weeks 10w
month	mo	5.6 months 24mo
year	y	100 years 5y

If no time unit is provided, the default unit of "days" is used. A numeric value must always be provided or an exception will be thrown. Numeric values may be fractional (e.g. 5.6).



#### Note

Due to the variable lengths of months and years, approximations are used for these values within Clover. A month is considered to be 30.346 days and a year is considered to be 365.232 days. All other units are exact.

## Sharing Report Formats

You can share report formats across a number of reports. This allows you to standardise on a set of report formats and use these for all of your reports.

Standalone format elements are created using the `<clover-format>` type. These standalone types support the same attributes and elements as the internal `<format>` elements of the `<clover-report>` task. To name the format, use the standard Ant "id" attribute.

The following code declares two report formats:

```
<clover-format id="std.format" srclevel="true" type="pdf"/>
<clover-format id="bw.format" bw="true" srclevel="true" type="pdf"/>
```

In this example, the first format is for source level, PDF reports. It is named "std.format". The second format, "bw.format", is essentially the same except that it specifies black-and-white output.

Once the format is declared with an identifier, it can be used by reference with a "refid" attribute. This is shown in the following report example:

```
<clover-report>
  <current summary="yes" outfile="report-current.pdf"
    title="Ant Coverage">
    <format refid="std.format"/>
  </current>
</clover-report>
```

This report, a summary report, uses the "std.format" format defined above. The `refid` values in the `<format>` elements can be an Ant property allowing selection of the report format at build time. The following is a

complete example:

```
<target name="report">
  <clover-format id="std.format" srclevel="true" type="pdf"/>
  <clover-format id="bw.format" bw="true" srclevel="true" type="pdf"/>
  <property name="format" value="std.format"/>
  <clover-report>
    <current summary="yes" outfile="report-current.pdf"
              title="Ant Coverage">
      <format refid="{format}"/>
    </current>
    <historical historydir="clover-hist" outfile="report-history.pdf"
               title="Ant Historical Coverage">
      <format refid="{format}"/>
    </historical>
  </clover-report>
</target>
```

This example generated two reports, which share a format. The format defaults to the standard format, a colour report. This default can be overridden from the command line. To generate black-and-white reports, use:

```
ant report \-Dformat=bw.format
```

## Extracting Coverage Data programmatically

### Using XPath with Clover's XML reports

Clover's XML reports provide detailed coverage data in a format that is easy to access programmatically using XPath. XML coverage reports can be generated by the `<clover-report>` or `<clover-historypoint>` Ant tasks. The following example XPath expressions show how to extract data from a Clover XML coverage report:

```
/coverage/project/metrics[@statements]
```

The above statement extracts the total number of statements in the project.

```
/coverage/project/metrics[@coveredstatements]
```

The above extracts the total number of covered statements in the project.

```
/coverage/project/package[name='com.foo.bar']/metrics[@statements]
```

The above extracts the total number of statements in the package `com.foo.bar`.

```
/coverage/project/package[name='com.foo.bar']/metrics[@coveredstatements]
```

The above extracts the total number of covered statements in the package `com.foo.bar`.

An XPath implementation is shipped with the JDK1.5 distribution. Third party implementations that work with JDK1.4 and below include [Jaxen](#), [Dom4j](#) and [JXP](#).

The following code example (using the JDK1.5 implementation of XPath) demonstrates simple extraction of coverage data from a Clover XML report: `import javax.xml.xpath.*;`

```
...

XPath xpath = XPathFactory.newInstance().newXPath();
String stmtExpr = "/coverage/project/metrics[@statements]";
String coveredStmtExpr = "/coverage/project/metrics[@coveredstatements]";
InputStream inputSource = new InputStream("coverage.xml");
Double projectStatements = (Double) xpath.evaluate(expression, inputSource,
                                                    XPathConstants.NUMBER);

Double projectCoveredStatements = (Double) xpath.evaluate(expression, inputSource,
                                                         XPathConstants.NUMBER);

...
```

## 6. Ant Task Reference

### Installing the Clover Ant Tasks

Clover provides a set of Ant tasks to make project integration easy. To make these tasks available in your project build file, you need to:

- load the 'cloverlib.xml' antlib by adding the following line to your build file:

```
<taskdef resource="cloverlib.xml" classpath="/path/to/clover.jar"/>
```

Make sure you change the above `"/path/to/clover.jar"` to point directly to your `clover.jar`.

For further options, see also [Ant Installation Options](#).

### The tasks

<code>&lt;clover-setup&gt;</code>	Installs Clover as the Ant <code>build.compiler</code> . This means that Clover will be invoked whenever the Ant <code>&lt;javac&gt;</code> is used, resulting in instrumented compilation.
<code>&lt;clover-instr&gt;</code>	Allows manual instrumentation of source files, for cases where the normal <code>&lt;clover-setup&gt;</code> integration approach can't be used.
<code>&lt;clover-report&gt;</code>	Produces coverage reports in different formats.
<code>&lt;clover-html-report&gt;</code>	Generates a HTML report with default settings.
<code>&lt;clover-pdf-report&gt;</code>	Generates a PDF report with default settings.

<code>&lt;clover-historypoint&gt;</code>	Records a coverage history point for use in historical coverage reports.
<code>&lt;clover-snapshot&gt;</code>	Generates a snapshot file used to assist Clover in optimizing the tests run in subsequent build.
<code>&lt;clover-merge&gt;</code>	Merges several Clover databases into one, to allow for combined reports to be generated.
<code>&lt;clover-check&gt;</code>	Tests project/package code coverage against criteria, optionally failing the build if the criteria are not met.
<code>&lt;clover-log&gt;</code>	Reports coverage results to the console at various levels.
<code>&lt;clover-clean&gt;</code>	Deletes the coverage database and/or associated coverage records.
<code>&lt;clover-env&gt;</code>	This task imports several useful Ant targets that can help you quickly integrate Clover into many common Ant builds.

## clover-check

### Description

The `<clover-check>` task tests project/package code coverage against criteria, optionally failing the build if the criteria are not met. This task needs to be run after coverage has been recorded.

### Parameters

Attribute	Description	Required
codeType	<b>Since 3.1.6:</b> Specifies which sources shall be taken for calculation.  Valid values are: APPLICATION (business code), TEST (test code), ALL (business + test code).	No; Default value: APPLICATION.
conditionalTarget	The target percentage conditional coverage for the project.	At least one of <code>target</code> , <code>methodTarget</code> , <code>statementTarget</code> or <code>conditionalTarget</code> is required, unless nested <code>clover-check</code> elements are specified.
failureProperty	Specifies the name of a property to be set if the target is not met. If the target is not met, the property will contain a text description of the failure(s).	No.
filter	comma or space separated list of contexts to exclude when calculating coverage. See <a href="#">Using Coverage Contexts</a> .	No.
haltOnFailure	Specifies if the build should be halted if the target is not met.	No; default is "false".

historydir	Allows you to specify a location for historical build data, along with a configurable threshold expressed as a percentage – used to cause the build to fail if coverage has dropped. This attribute is passed down to specified packages, then the same test is done for these at the package level.	No. Example: historyDir="/history" threshold="1%"
includeFailedTestCoverage	Specifies whether to include failed test coverage when calculating the total coverage percentage.	No; defaults to "false".
initstring	The <code>initstring</code> of the coverage database.	No; If not specified here, Clover will look in the default location ( <code>\${basedir}/.clover</code> ). If you have specified an <code>initstring</code> on the <code>&lt;clover-setup&gt;</code> task, you must ensure that <code>&lt;clover-setup&gt;</code> is called prior to the execution of this task.
methodTarget	The target percentage method coverage for the project.	At least one of <code>target</code> , <code>methodTarget</code> , <code>statementTarget</code> or <code>conditionalTarget</code> is required, unless nested <code>clover-check</code> elements are specified.
span	Specifies how far back in time to include coverage recordings from since the last Clover build. See <a href="#">Using Spans</a> .	No; default includes "all coverage data found".
statementTarget	The target percentage statement coverage for the project.	At least one of <code>target</code> , <code>methodTarget</code> , <code>statementTarget</code> or <code>conditionalTarget</code> is required, unless nested <code>clover-check</code> elements are specified.
target	The target percentage total coverage for the project. e.g. "10%"	At least one of <code>target</code> , <code>methodTarget</code> , <code>statementTarget</code> or <code>conditionalTarget</code> is required, unless nested <code>clover-check</code> elements are specified.



#### Important note on `target`, `methodTarget`, `statementTarget`, `conditionalTarget`

Comparison of actual value with a target percentage is performed with such numerical precision as number of fractional digits set for a target percentage. A standard rounding is used (`BigDecimal.ROUND_HALF_EVEN`).

For example, if actual coverage value is 99.9% then for the `target="100%"` a build will pass, whereas for the `target="100.000000%"` a build will fail.

#### Nested elements of `<clover-check>`

`<package>`

Specifies a target for a named package.

#### Parameters



Attribute	Description	Required
conditionalTarget	The target percentage conditional coverage for the package.	At least one of target, methodTarget, statementTarget or conditionalTarget is required.
methodTarget	The target percentage method coverage for the package.	At least one of target, methodTarget, statementTarget or conditionalTarget is required.
name	The name of the package.	Either (but not both) of name or regex is required.
regex	Regular expression to match package names.	Either (but not both) of name or regex is required.
statementTarget	The target percentage statement coverage for the package.	At least one of target, methodTarget, statementTarget or conditionalTarget is required.
target	The target percentage total coverage for the package. e.g. "10%"	At least one of target, methodTarget, statementTarget or conditionalTarget is required.

#### <testsources>

<testsources> is an [Ant fileset](#) that can be used to distinguish test source code from application source code. All files included in the fileset will be displayed in the separate 'Test' node of the coverage tree. If omitted, Clover's default test detection algorithm will be used to distinguish test sources.

#### <testResults>

An optional Ant [fileset](#) containing a list of test result XML files.

<testresults> is generally not required by most users, as the built-in test results will provide all required information in the majority of cases. For more details please see ['Advanced Usage'](#).

#### Examples

```
<clover-check target="80%" />
```

Tests if total percentage coverage is at least 80%. If not, a message is logged and the build continues.

```
<clover-check target="80%"
  haltOnFailure="true" />
```

Tests if total percentage coverage is at least 80%. If not, a message is logged and the build fails.

```
<clover-check target="80%"
  failureProperty="coverageFailed" />
```

Tests if total percentage coverage is at least 80%. If not, a message is logged and the project property coverageFailed is set.

```
<clover-check target="80%"
  <package name="com.acme.killerapp.core" target="70%" />
  <package name="com.acme.killerapp.ai" target="40%" />
</clover-check>
```

Tests if:

- total percentage coverage for project is at least 80%.
- total percentage coverage for package `com.acme.killerapp.core` is at least 70%.
- total percentage coverage for package `com.acme.killerapp.ai` is at least 40%.

If any of these criteria are not met, a message is logged and the build continues.

```
<clover-check target="80%"
  filter="catch">
  <package name="com.acme.killerapp.core" target="70%"/>
  <package name="com.acme.killerapp.ai" target="40%"/>
</clover-check>
```

As above, but doesn't include coverage of catch blocks when measuring criteria.

```
<clover-check target="80%" conditionalTarget="90%"
  filter="catch">
  <package name="com.acme.killerapp.core" target="70%"/>
  <package name="com.acme.killerapp.ai" target="40%"/>
</clover-check>
```

As previous example, but also ensures that the project conditional coverage is at least 90%.

```
<clover-check>
  <package regex="com.acme.killerapp.core.*" target="70%"/>
</clover-check>
```

Tests if coverage for `com.acme.killerapp.core` and all subpackages is at least 70%.

## clover-clean

### Description

The `<clover-clean>` task deletes the [coverage database](#) and associated coverage recording files.

### Parameters

Attribute	Description	Required
haltOnError	Controls whether an error (such as a failure to delete a file) stops the build or is merely reported to the screen ("true"/"false").	No; defaults to "false".
initstring	The initstring of the database to clean.	No; if not specified here, Clover will use the default location ( <code>\${basedir}/.clover</code> ). If you have specified an initstring on the <code>&lt;clover-setup&gt;</code> task, you must ensure that <code>&lt;clover-setup&gt;</code> is called prior to the execution of this task.
keepdb	Controls whether to keep the coverage database file ("true"/"false"). If "false", the coverage database will be deleted.	No; defaults to "false".

keepTestSnapshot	Specifies whether the test snapshot file should be kept or deleted; defaults to true. ("true"/"false")	No; if not specified here, the last test snapshot will be stored. \(\i\) This is not deleted between builds (unlike the .db file and the coverage files).
verbose	Controls whether to show the name of each deleted file ("true"/"false").	No; defaults to "false".

**Examples**

```
<clover-clean/>
```

Deletes the coverage database and all of the coverage recordings.

```
<clover-clean verbose="true"/>
```

Deletes the coverage database and all of the coverage recordings, printing out a log statement for each file deleted.

```
<clover-clean keepdb="true"/>
```

Deletes the coverage recordings but keeps the coverage database.

**clover-env****Description**

The `<clover-env>` task imports a set of standard Ant targets into the current project.

The following targets will be available:

Target Name	Description
clover.all	Runs clover.clean, with.clover, test, clover.report from a single target.
clover.clean	Deletes the clover database and the <div data-bbox="434 1541 1410 1641" style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre> \${clover.dest} </pre> </div> directory.

clover.current	Generates an HTML and XML report to <pre>       \${clover.dest}     </pre> using <pre>       \${project.title}     </pre> .
clover.report	Same as clover.current, however a history report will also be created, using the historypoints in <pre>       \${clover.project.historydir}     </pre> .
clover.save-history	Saves a history point to <pre>       \${clover.project.historydir}     </pre>
with.clover	Enables Clover on this build
clover.snapshot	Saves a snapshot file to assist with unit test optimization

Alternatively, running: `$ ant -projecthelp` will display the list of targets available.

#### Parameters

This task has no parameters.

### clover-historypoint

#### Description



The `<clover-historypoint>` task records a coverage history point for use in [historical coverage reports](#). The basic nesting of elements within the `<clover-historypoint>` task is as follows:

```

<clover-historypoint>
  <fileset/>
  <testsources/>
  <testresults/>
</clover-historypoint>
  
```

#### Parameters

Attribute	Description	Required
date	Specifies an override date for this history point. This allows for generation of past historical data for a project.	No; defaults to the timestamp of the current coverage data.

dateFormat	Specifies a date format string for parsing the "date" attribute. The string must contain a valid <code>java.text.SimpleDateFormat</code> pattern.	No; defaults to <code>java.text.SimpleDateFormat</code> using the default pattern and date format symbols for the default locale.
filter	A comma or space separated list of contexts to exclude when generating the historypoint. See <a href="#">Using Coverage Contexts</a> .	No.
historyDir	The directory where historical data is stored.	Yes.
includeFailedTestCoverage	If true, all coverage attributed to a test that failed will be included.  If history-point generation is taking a long time, you can speed up the performance of your tests by setting this value to 'true'.	No; defaults to "false"
initstring	The <code>initstring</code> of the coverage database.	No; if not specified here, Clover will look in the default location ( <code>\${basedir}/.clover</code> ). If you have specified an <code>initstring</code> on the <code>&lt;clover-setup&gt;</code> task, you must ensure that <code>&lt;clover-setup&gt;</code> is called prior to the execution of this task.
overwrite	If true, existing history point for the same date will be automatically overwritten.	No; defaults to "false".
property	If set, the name of the property to hold the absolute path name of the history point file that was created by this task.	No.
span	Specifies how far back in time to include coverage recordings from since the last Clover build. See <a href="#">Using Spans</a> .	No; default includes "all coverage data found".
srcLevel	Prevents excessive Clover data from being loaded when generating a history point. This should be used for large projects, where saving a full source level history point is taking a long time.  Clover's history reports currently do not use line level information. This means there is no loss of functionality when <code>srcLevel="false"</code> . In the future however, line level information may be used.	No; defaults to "false"

#### **Nested elements of `<clover-historypoint>`**

##### **`<fileset>`**

`<clover-historypoint>` supports nested filesets which control which source files are to be included in a historypoint. Only classes which are from the source files in the fileset are included in the history point. This allows history points to focus on certain packages or particular classes. By using Ant's fileset selectors, more complicated selections are possible, such as the files which have recently changed, or files written by a particular author.

**<testsources>**

<testsources> is an [Ant fileset](#) that can be used to distinguish test source code from application source code. All files included in the fileset will be displayed in the separate 'Test' node of the coverage tree. If omitted, Clover's default test detection algorithm will be used to distinguish test sources.

**<testresults>**

<testresults> is an optional [Ant fileset](#) that defines the location of all test result XML files for your project. Currently, these will be used by <clover-historypoint> to determine coverage data if the `includeFailedTestCoverage` flag is set to false.

<testresults> is generally not required by most users, as the built-in test results will provide all required information in the majority of cases. For more details please see '[Advanced Usage](#)'.

**Examples**

```
<clover-historypoint historyDir="clover-historical"/>
```

Records a history point into the directory `PROJECT_DIR/clover-historical`.

```
<clover-historypoint historyDir="clover-historical"
    date="010724120856"
    dateFormat="yyMMddHHmmss"/>
```

Records a history point, with the effective date of 24/07/01 12:08:56.

```
<clover-historypoint historydir="history"
    filter="toString"
    includeFailedTestCoverage="false"
    property="clover.historypoint.path">
</clover-historypoint>
```

Records a history point that excludes all test coverage that is attributed to a failed test; `clover.historypoint.path` will contain the absolute file path of the history point that was created.

```
<clover-historypoint historyDir="clover-historical">
  <testsources dir="src" include name="**/*Test.java"/>
</clover-historypoint>
```

Records a history point that recognises classes in the "src" directory as tests.

**clover-html-report****Description**

The <clover-html-report> task generates a full HTML report with sensible default settings. If configured, a [history point](#) is also generated prior to generation of the full report.

 If you need more options, please use the <clover-report> task, which provides more functionality,

such as:

- generating reports in HTML, PDF, JSON, XML formats
- generating multiple reports linked with each other
- generating custom charts, defining metrics, 'movers', overviews etc
- pointing to source location different than during compilation

### Parameters

Attribute	Description	Required
coverageCacheSize	This is a performance tuning option, which is used to specify the maximum size of coverage data kept in memory when generating an HTML report. Accepts storage values such as 256m (256 megabytes) or 1g (1 gigabyte). Also accepts the value "nocache" to force all coverage data to be loaded into memory.	No. Default is 256m.
historydir	The directory for Clover history points. <b>If this attribute is set, a new history point will be generated prior to the generation of the full report.</b> For more information, see <code>&lt;clover-historypoint&gt;</code> .	No.
historyIncludes	An <a href="#">Ant GLOB</a> to select specific history point files within the <code>historyDir</code> directory	No. Default is <code>clover-*.xml.gz</code>
initstring	The path to the Clover database. If not specified, Clover will use the <code>initstring</code> set by a previous execution of <code>&lt;clover-setup/&gt;</code> in the current build sequence. Otherwise, the default database location will be used.	No.
maxtestsperfile	This limits the number of tests displayed for each file.	No.
numThreads	The number of threads to start when generating a HTML report. A value of 0 will disable multi-threading for report generation. This is a performance tuning option.	No. Default is 2.
<b>outdir</b>	The directory to write the report to.	<b>Yes.</b>
projectName	Overrides the project name set in the Ant build file. This is used for display purposes only.	No; defaults to the project name of the Ant build file.
showUniqueCoverage	Calculate and show unique per-test coverage (for large projects, this can take a significant amount of time). Defaults to <ul style="list-style-type: none"> <li>- <code>true</code> for Clover 3.1.4 and older</li> <li>- <code>false</code> for Clover 3.1.5 and newer.</li> </ul>	No.
testresultsdir	The directory containing the XML results of the unit tests. Clover will look for all <code>TEST*.xml</code> files in this directory. <b>This attribute is generally not required by most users</b> , as the built-in test results will provide all required information in the majority of cases. For more details please see ' <a href="#">Advanced Usage</a> '.	No.
title	The title to use in the report.	No.

### Examples

```
<clover-html-report outdir="clover/report"/>
```

This is the simplest way to generate an HTML report. It will be written to the directory "clover/report".

```
<clover-html-report outdir="build/clover/report"
  historydir="clover/historypoints"
  title="MyProject Coverage"/>
```

This will generate a report in the "build/clover/report" directory. A history point will be created in the "clover/historypoints" directory, and all history points in that directory will be used to generate the historical section of the report. The report will be titled "MyProject Coverage".

## clover-instr

### Description

The `<clover-instr>` task produces instrumented versions of sets of Java source files. These can then be compiled in place of the original source to produce an instrumented Java build.

**i** The `<clover-instr>` task is provided for users who can't make use of the standard `<clover-setup>` integration task. The `<clover-setup>` task offers a simpler and less intrusive integration option for most users.

Be aware that this element does not support Groovy code.


The basic nesting of elements within the `<clover-instr>` task is as follows:

```
<clover-instr>
  <distributedCoverage/>
  <fileset/>
  <methodcontext/>
  <statementcontext/>
  <profiles/>
  <testsources>
    <testclass>
      <testmethod/>
    </testclass>
  </testsources>
</clover-instr>
```

### Parameters

Attribute	Description	Required
destdir	The directory into which Clover will write an instrumented copy of the source code.	Yes.
initstring	The Clover <code>initString</code> describes the location of the Clover coverage database. Typically this is a relative or absolute file reference, e.g. <code>\${basedir}/build/clover.db</code> . If not specified it defaults to <code>.clover</code> , relative to the project's base directory.	No.
flushinterval	When the <code>flushpolicy</code> is set to <code>interval</code> or <code>threaded</code> this value is the minimum period between flush operations (in milliseconds).	No.



flushpolicy	<p>This attribute controls how Clover flushes coverage data during a test run. Valid values are <code>directed</code>, <code>interval</code> or <code>threaded</code>.</p> <p><code>directed</code> — Coverage data is flushed at JVM shutdown, and after an inline flush directive.</p> <p><code>interval</code> — Coverage data is flushed as for <code>directed</code>, as well as periodically at a maximum rate based on the value of <code>flushinterval</code>. This is a 'passive' mode in that flushing potentially occurs as long as instrumented code is being executed.</p> <p><code>threaded</code> — Coverage data is flushed as for <code>directed</code>, as well as periodically at a rate based on the value of <code>flushinterval</code>. This is an 'active' mode in that flushing occurs on a separate thread and is not dependent on the execution of instrumented code.</p> <p>For more information, see <a href="#">Using a Flush Policy</a>.</p>	No; defaults to <code>directed</code> .
fullyQualifyJavaLang	This should only be set to <code>false</code> if you have defined a variable called <code>'java'</code> in your source files. If <code>false</code> , Clover will instrument source files without using fully qualified <code>java.lang</code> names.	No; defaults to <code>'true'</code> .
instrumentationLevel	This setting can reduce accuracy to method level, to enhance the speed of instrumentation, compilation & test execution. Valid values are <code>'method'</code> and <code>'statement'</code> .	No; defaults to <code>statement</code> .
instrumentLambda	<p><b>Since 3.2.2.</b> Whether Java 8 lambda functions shall be instrumented. If instrumented, they're treated like normal methods (and can be shown in HTML report and considered in code metrics, for example). Possible values:</p> <ul style="list-style-type: none"> <li><code>none</code> - do not instrument lambda functions,</li> <li><code>expression</code> - instrument lambdas in expression-like form, e.g. <code>"(a, b) -&gt; a + b"</code>,</li> <li><code>block</code> - instrument lambdas in code blocks, e.g. <code>"(a, b) -&gt; { return a + b; }"</code></li> <li><code>all</code> - instrument all lambda functions.</li> </ul> <p> Due to Clover's restrictions related with code instrumentation and javac compiler's type inference capabilities, you may get compilation errors when expression-like lambda functions are passed to generic methods or types. In such case disable instrumentation of expression-like form (i.e. use the <code>none</code> or <code>block</code> setting). See the <a href="#">Java 8 code instrumented by Clover fails to compile</a> Knowledge Base article for more details.</p>	No; defaults to <code>"all"</code> .
recordTestResults	If set to <code>false</code> , test results will not be recorded; instead, results can be added via the <code>&lt;testResults&gt;</code> fileset at report time. For more details please see <a href="#">Advanced Usage</a> .	No; defaults to <code>'true'</code> .
relative	This controls whether the <code>initstring</code> parameter is treated as a relative path or not.	No; defaults to <code>'false'</code> .
srcdir	The directory of source code to instrument.	Yes, unless a nested <code>clover-instrument</code> element is used.

source	The source level to process source files at. <b>It is recommended that you set this parameter, either here or on &lt;javac&gt; invocations.</b>	No; defaults to the Java version detected at runtime.
--------	---	---

### Nested Elements of <clover-instr>

#### <distributedCoverage>

This element turns on Clover's distributed coverage feature, enabling the collection of per-test coverage data, when your test environment requires more than one JVM ([Java Virtual Machine](#)).

#### Parameters

Attribute name	Description	Required
name	The name of this configuration.	No; defaults to 'tcp-config'
port	The port the test JVM should listen on.	No; defaults to '1198'
host	The hostname the test JVM should bind to.	No; defaults to 'localhost'
timeout	(a number) The amount of time (in milliseconds) to wait before a connection attempt will fail.	No; defaults to '5000'
numClients	(a number) The number of clients that need to connect to the test server before the tests will continue.	No; defaults to '0'
retryPeriod	(a number) The amount of time (in milliseconds) to wait before attempting to reconnect in the event of a network failure.	No; defaults to '1000'

 All attributes are optional.

#### <fileset>

Specifies a set of source files to instrument.

#### <methodContext>

Specifies a method Context definition. See [Using Coverage Contexts](#) for more information.

#### Parameters

Attribute	Description	Required
name	The name for this context. Must be unique, and not be one of the reserved context names (see <a href="#">Using Coverage Contexts</a> ).	Yes.
regexp	A Perl 5 Regexp that defines the context. This regexp should match the method signatures of methods you wish to include in this context. Note that when method signatures are tested against this regexp, whitespace is normalised and comments are ignored.	Yes.

maxComplexity	Match a method to this pattern if its cyclomatic complexity is not greater than maxComplexity. In other words - all methods with complexity <= maxComplexity will be filtered out.	No.
maxStatements	Match a method to this pattern if its number of statements is not greater than maxStatements. In other words - all methods with statements <= maxStaments will be filtered out.	No.
maxAggregatedComplexity	<b>Since 3.1.10.</b> Match a method to this pattern if its aggregated cyclomatic complexity is not greater than maxAggregatedComplexity. In other words - all methods with aggregated complexity <= maxAggregatedComplexity will be filtered out. Aggregated complexity metric is a sum of the method complexity and complexity of all anonymous inline classes declared in the method.	No.
maxAggregatedStatements	<b>Since 3.1.10.</b> Match a method to this pattern if its number of aggregated statements is not greater than maxAggregatedStatements. In other words - all methods with aggregated statements <= maxAggregatedStaments will be filtered out. Aggregated statements metric is a sum of the method statements and statements of all anonymous inline classes declared in the method.	No.

### What is the difference between maxComplexity and maxAggregatedComplexity or maxStatements and maxAggregatedStatements?

Aggregated metrics calculate method statements/complexity including the code of all anonymous inline classes declared inside the method. Thanks to this, it is possible to distinguish between a trivial single-statement method like:

```
int getNumber() {
    return number;
}
```

and a single-statement method which actually returns more complex data, like:

```
ActionListener getListener() {
    return new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            System.out.println("statement #1");
            System.out.println("statement #2");
            System.out.println("statement #3");
        }
    };
}
```

If you would use a method context filter with *maxStatements* attribute, like the following:

```
<methodContext name="trivial" regexp=".*" maxStatements="1">
```


then both *getNumber()* and *getListener()* methods would be filtered-out, because each of them contains only one statement: "return <xxx>".

If you would use new *maxAggregatedStatements*, for instance:

```
<methodContext name="trivial" regexp=".*" maxAggregatedStatements="1">
```

then the *getNumber()* would be filtered-out and the *getListener()* method would **not** be filtered out (because it contains 4 statements in total - 1 "return" statement from the method itself and 3 "System.out.println()" statements from anonymous class).

### Regular expression tip:

 If you would like to filter-out all methods, except those having a specific name, you could write a negative-look-ahead regular expression. For example:

```
<methodContext name="trivial" regexp="^(?!.*(getRunnable|getListener)).*$"
maxStatements="1"/>
```

will filter-out all methods having not more than one statement, except those which are named *getRunnable* or *getListener*.

### <statementContext>

Specifies a statement Context definition. See [Using Coverage Contexts](#) for more information.

 This element does not support Groovy.

#### Parameters

Attribute	Description	Required
name	The name for this context. Must be unique, and not be one of the reserved context names (see <a href="#">Using Coverage Contexts</a> ).	Yes.
regexp	A Perl 5 Regexp that defines the context. This regexp should match statements you wish to include in this context. Note that when statements are tested against this regexp, whitespace is normalised and comments are ignored.	Yes.

### <profiles>

**Since 3.1.11.** Optional element. Defines a list of Clover profiles, which can be selected at runtime by providing a **clover.profile=<name>** system property. Thanks to this you can change some of Clover's behaviour without code recompilation.

```
<profiles>
  <profile name="default" coverageRecorder="FIXED|GROWABLE|SHARED">
    <distributedCoverage/> <!-- optional -->
  </profile>
  <profile ../>
    <!-- more profiles -->
</profile>
```

`<profile>`

**Since 3.1.11.** Contains a definition of a single runtime profile.

#### Parameters

Attribute	Description	Required
name	The name for this profile; name must be unique among profiles. There must be one profile named "default".	No. Defaults to "default".
coverageRecorder	Type of coverage recorder which will be used for gathering coverage data at runtime. Possible values: FIXED, GROWABLE, SHARED (case insensitive).  ⚠ Warning: we strongly recommend using the default setting. Do not change until you deeply understand <a href="#">how it works</a> .	No. Defaults to "FIXED".

#### Nested elements

`<distributedCoverage/>`

💡 Note: a definition in `<profile>/<distributedCoverage>` element has priority over the `<clover-setup/clover-instr>/<distributedCoverage>` element.

#### Selecting clover.profile at runtime

Clover profile is being selected at runtime using the following algorithm:

- Are there any profiles defined in compiled code?
  - **yes** -
    - 1. read the **clover.profile** system property. is it defined?
      - **yes** - use the value as profile name
      - **no** - use the "default" profile name
    - 2. is the profile name found on list of defined profiles?
      - **yes** - **use settings from this profile**
      - **no** - **use system settings** (default coverage recorder etc...)
  - **no** - **use system settings** (default coverage recorder etc...)

So it fall-backs to default system settings in case of missing profile.

`<testsources>`

`<testsources>` is an [Ant fileset](#) which should only be used if Clover's [default test detection](#) is not adequate. Clover's default test detection algorithm is used to distinguish test cases if this element is omitted.

⚠ To have test sources reported in a separate tree to your application code, use the `<testsources/>` element in the `<clover-report/>` task.

#### Nested elements of <testsources>

`<testclass>`

`<testclass>` can be used to include only specific test classes.

#### Parameters

Attribute	Description	Required
-----------	-------------	----------

name	A regex on which to match the test class's name.	No.
super	A regex on which to match the test class's superclass.	No.
annotation	A regex on which to match the test class's annotation.	No.
package	A regex on which to match the test class's package.	No.
tag	A regex on which to match the test class's javadoc tags.	No.

**i** For more information about regular expressions, please visit <http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html#sum>.

<and>

<and> can be used to specify multiple instances of <testclass>, all of which must be matched for a class to be detected as a test, e.g.:

```
<testsources dir="tests">
<and>
  <testclass annotation="Specification" />
  <testclass annotation="Test" />
</and>
</testsources>
```

In this example, a class will only be recognised as a test if it has "Specification" **and** "Test" annotations.

<or>

<or> can be used to specify multiple instances of <testclass>, any of which must be matched for a class to be detected as a test, e.g.:

```
<testsources dir="tests">
<or>
  <testclass name=".*Spec" />
  <testclass name=".*Test" />
</or>
</testsources>
```

In this example, a class will be recognised as a test if its name matches ".\*Spec", **or** its name matches ".\*Test".

#### **Nested elements of <testclass>**

<testmethod>

<testmethod> can be used to perform more fine grained detection of test methods.

**!** Clover matches methods only; it does not match constructors (CLOV-1339).

Parameters

Attribute	Description	Required
name	A regex on which to match the test method's name.	No.
annotation	A regex on which to match the test method's annotation.	No.
tag	A regex on which to match the test method's javadoc tags.	No.
returntype	A regex on which to match the return type of the method, e.g.: <ul style="list-style-type: none"> <li>".*" will match any return type.</li> <li>"void" will match methods with no return type.</li> </ul>	No.

Note that you can include multiple instances of `<testmethod>`, in which case they will be treated as 'or' clauses, e.g.:

```
<testsources dir="tests">
  <testclass>
    <testmethod annotation="Specification"/>
    <testmethod name="^should.*"/>
    <testmethod name="^must.*"/>
  </testclass>
</testsources>
```

In this example, a method will be recognised as a test if its annotation is "Specification", **or** its name matches "^should\*", **or** its name matches "^must\*".

## Examples

```
<clover-instr srcdir="src" destdir="instr"/>
```

Produce an instrumented copy of all source files in the `srcdir` into the `destdir`. The Clover registry is at the default location.

```
<clover-instr destdir="instr"/>
  <fileset dir="src">
    <include name="**/*.java"/>
  </fileset>
</clover-instr>
```

This example achieves the same as the first example, but using an embedded fileset.

```
<clover-instr destdir="instr"/>
  <testSources dir="src">
    <include name="**/*Test.java"/>
    <testclass name=".*Test">
      <testmethod name=".*Bag.*"/> <!-- only the Bag related tests -->
    </testclass>
  </testSources>
</clover-instr>
```

This example produces an instrumented copy which recognises all of the following as tests: classes in the directory "src"; classes in files whose names end with "Test"; methods whose names contain with "Bag".

## Interval Flushing

By default Clover will write coverage data to disk when the hosting JVM exits, via a shutdown hook. This is not always practical, particularly when the application you are testing runs in an Application Server. In this situation, you can configure Clover to use 'interval' flushing, where coverage data is written out periodically during execution:

```
<clover-instr flushpolicy="interval" flushinterval="5000" srcdir="src"
destdir="instr" />
```

The "flushinterval" defines in milliseconds the minimum interval between coverage writes.

#### Specifying the location of the Clover Database.

By default, Clover writes its [internal database](#) to the `.clover` directory relative to the project's `basedir`. To override this location, use the `initstring` attribute.

```
<clover-instr initstring="clover-db/coverage.db" srcdir="src" destdir="instr"/>
```

This example will use `clover-db/coverage.db` as the location for the Clover database. Note that the directory `clover-db` should exist before running this task.

## Troubleshooting

#### Clover does not support parallel instrumentation

You cannot use `<parallel/>` task for code instrumentation, for instance:

```
<target name="instrument">
  <parallel>
    <clover-instr srcdir="module1" destdir="module1-instr" ../>
    <clover-instr srcdir="module2" destdir="module1-instr" ../>
  </parallel>
</target>
```

will produce error message like:

```
[clover] Error finalising instrumentation:
[clover] java.io.IOException: Failed to move tmp registry file
/myproject/.clover/clover3_1_6.db.tmp to final registry file
```

## clover-log

#### Description

The `<clover-log>` task reports coverage information to the console at different levels.

#### Parameters

Attribute	Description	Required
initstring	The <code>initstring</code> of the coverage database.	No; if not specified here, Clover will use the default location ( <code>\${basedir}/.clover</code> ). If you have specified an <code>initstring</code> on the <code>&lt;clover-setup&gt;</code> task, you must ensure that <code>&lt;clover-setup&gt;</code> is called prior the execution of this task.



level	Controls the level of detail included in the report. Valid values are <code>summary</code> , <code>class</code> , <code>method</code> , <code>statement</code>	No; defaults to "summary".
filter	comma or space separated list of contexts to ignore when calculating coverage. See <a href="#">Using Coverage Contexts</a> .	No.
span	Specifies how far back in time to include coverage recordings from since the last Clover build. See <a href="#">Using Spans</a> .	No; defaults includes "all coverage data found".
codeType	<b>Since 3.1.6:</b> Specifies which sources shall be taken for calculation. This attribute should be used together with <code>&lt;testSources&gt;</code> nested element.  Valid values are: APPLICATION (business code), TEST (test code), ALL (business + test code).	No; Default value: APPLICATION.
showUnitTests	<b>Since 3.1.6:</b> Show unit tests summary in the report. Must be used with <code>codeType=ALL</code> or <code>codeType=TEST</code> .	No; Default value: false
outputProperty	<b>Since 3.1.6:</b> Name of the Ant property in which a clover-log report will be stored, instead of printing it to console.	No;

### Nested elements

#### `<fileset>`

Specifies an Ant [fileset](#). Only these files will be used when generating the clover-log messages and coverage data.

#### `<package>`

Specifies a named package to restrict the report to. Multiple `<package>` elements can be specified.

#### Parameters

Attribute	Description	Required
name	The name of the package to include.	Yes.

#### `<sourcepath>`

Specifies an Ant path that Clover should use when looking for source files.

<testSources>

**Since 3.1.6:** Specifies an Ant [fileset](#). These files will be treated as test code (codeType=TEST), all others will be treated as business code (codeType=APPLICATION).

### Examples

```
<clover-log/>
```

Prints a summary of code coverage to the console.

```
<clover-log>
  <package name="com.acme.killerapp.core"/>
</clover-log>
```

Prints a summary of code coverage for the package `com.acme.killerapp.core` to the console.

```
<clover-log level="statement">
  <package name="com.acme.killerapp.core"/>
</clover-log>
```

Prints detailed (source-level) code coverage information for the package `com.acme.killerapp.core` to the console.

```
<clover-log level="statement"
  filter="catch">
  <package name="com.acme.killerapp.core"/>
</clover-log>
```

As above, but catch blocks will not be considered in coverage reporting.

```
<clover-log level="statement">
  <sourcepath>
    <pathelement path="/some/other/location"/>
  </sourcepath>
</clover-log>
```

Prints source-level coverage report to the console. Clover will look for source files in the directory `/some/other/location`.

```
<clover-log codeType="APPLICATION">
  <fileset dir="src/main/java"/>
  <fileset dir="src/main/groovy"/>
  <fileset dir="src/test/java"/>
  <testSources dir="src/test/java"/>
</clover-log>
```

Collect coverage data from three directories: `src/main/java`, `src/main/groovy` and `src/test/java`. Test sources are located in: `src/test/java`. Report coverage for business code only (`codeType = APPLICATION`), i.e. skip files from `src/test/java`.

### Sample output

Example for `<clover-log showUnitTests="true" code="ALL".../>`

```
Clover Coverage Report
Coverage Timestamp: Tue May 22 13:21:44 CEST 2012
Report for code    : ALL

Coverage Overview -
Coverage:-
  Methods: 4/7 (57,1%)
  Statements: 8/14 (57,1%)
  Branches: 0/0 ( - )
  Total: 57,1%
Complexity:-
  Avg Method: 1.0
  Density: 0.5
  Total: 7
Tests:-
  Tests number: 2
  Tests run: 2
  Tests passed: 2
  Tests failed: 0
  Tests errors: 0
```

By default, log output is written in plain text, for instance:

```
[INFO] Clover Coverage Report
Coverage Timestamp: Mon Jan 07 10:54:27 CET 2013
Report for code   : APPLICATION

Coverage Overview -
Coverage:-
  Methods: 31/31 (100%)
  Statements: 90/95 (94.7%)
  Branches: 39/46 (84.8%)
  Total: 93%
Complexity:-
  Avg Method: 1.742
  Density: 0.568
  Total: 54
```

but if you have a terminal supporting ANSI colors (Linux, for example) you can use `-Dansi.color=true` property to get a report like this:

```
[INFO] Clover Coverage Report
Coverage Timestamp: Mon Jan 07 10:53:00 CET 2013
Report for code   : APPLICATION

Coverage Overview -
Coverage:-
  Methods: 31/31 (100%)
  Statements: 90/95 (94.7%)
  Branches: 39/46 (84.8%)
  Total: 93%
Complexity:-
  Avg Method: 1.742
  Density: 0.568
  Total: 54
```

## clover-merge

### Description

The `<clover-merge>` task merges several Clover databases into one, to allow for combined reports to be generated. The resultant database can be used with reporting tasks, such as `<clover-report>` or `<clover-log>`, using the `initstring` parameter.

### Parameters

Attribute	Description	Required
<code>initstring</code>	The location to write the Clover coverage database resulting from the merge.	Yes.
<code>update</code>	If set to true and there is an existing database at <code>initString</code> , that database will be included in the merge.	No. Defaults to ' <b>fa lse</b> '.
<code>updateSpan</code>	If <code>update</code> is true, this span is used when merging any existing database at <code>i nitString</code> .	No. Defaults to 0 seconds.

### Nested Elements of `<clover-merge>`

#### `<cloverDb>`

Specifies a Clover database to merge.

**Parameters**

Attribute	Description	Required
initstring	The <code>initString</code> of the database to merge.	Yes.
span	Specifies how far back in time to include coverage recordings from since the last Clover build for this database. See <a href="#">Using Spans</a> .	No; default includes 'all coverage data found'.

**<cloverDbSet>**

Specifies an Ant FileSet of Clover databases to merge. Apart from those shown below, parameters and sub-elements are the same as for an [Ant FileSet](#).

**Parameters**

See the [Ant FileSet](#) documentation for parameters and sub-elements available on FileSets.

Attribute	Description	Required
refid	Lets you specify a <a href="#">fileset</a> that references a group of Clover database files. This references a fileset defined elsewhere.	No; defaults to none.
span	Specifies how far back in time to include coverage recordings from since the last Clover build for this database. See <a href="#">Using Spans</a> .	No; defaults to '0 seconds'.

**Examples**

This example produces a merged database containing the measured coverage of project A and project B:

```
<clover-merge initString="mergedcoverage.db">
  <cloverDb initString="projectAcoverage.db" />
  <cloverDb initString="projectBcoverage.db" span="30 mins" />
</clover-merge>
```

This example produces a merged database containing the measured coverage of all databases found under `/home/projects`:

```
<clover-merge initString="mergedcoverage.db">
  <cloverDbSet dir="/home/projects" span="30 mins">
    <include name="**/coverage.db" />
  </cloverDbSet>
</clover-merge>
```

**clover-pdf-report****Description**

The `<clover-pdf-report>` task generates a PDF report with sensible default settings. If configured, a history point is also generated prior to generation of the full report. For more configuration options, use the `<clover-report>` task.

**Parameters**

Attribute	Description	Required
outfile	The filename to write the report to.	Yes.

initstring	The path to the Clover database. If not specified, Clover will use the <code>initstring</code> set by a previous execution of <code>&lt;clover-setup&gt;</code> in the current build sequence. Otherwise, the default database location will be used.	No.
historydir	The directory that contains any clover history points. <b>If this attribute is set, a new history point will be generated prior to the generation of the full report.</b> For more information, see <code>&lt;clover-historypoint&gt;</code> .	No.
title	The title to use in the report.	No.

### Examples

```
<clover-pdf-report outfile="clover_coverage.pdf"/>
```

This is the simplest way to generate a pdf report. It will be written to the file "clover\_coverage.pdf".

```
<clover-pdf-report outfile="clover_coverage.pdf" historydir="clover/historypoints"
title="MyProject Coverage"/>
```

This will generate a report in the current directory called "clover\_coverage.pdf". A history point will be created in the "clover/historypoints" directory, and all history points in that directory will be used to generate the historical section of the report. The report will be titled "MyProject Coverage".

## clover-report

### Introduction

The `<clover-report>` task generates [current](#) and [historical](#) reports in multiple formats.

**i** If you do not require fine-grained control over your Clover reports, use the `<clover-html-report>` or `<clover-pdf-report>` tasks.

On this page:

- [Introduction](#)
- [Overview](#)
- [Parameters](#)
- [Nested elements](#)
  - [<current>](#)
  - [<historical>](#)
- [Examples](#)
  - [Examples of Current Report configurations](#)
  - [Example of customising columns](#)
  - [Example of linked reports](#)
  - [Examples of Historical Report Configurations](#)
- [References](#)
  - [Column Name Reference Table](#)
  - [Clover Expression Language](#)

### Overview

The basic nesting of elements within the `<clover-report>` task is as follows: (click any item for detailed user documentation).

```

<clover-report>
  <current>
    <columns>
      (one or more columns)
    </columns/>
    <fileset/>
    <format/>
    <sourcepath/>
    <testsources/>
    <testresults/>
  </current>
  <historical>
    <added>
      <columns>
        (a single column)
      </columns/>
    </added>
    <chart>
      (one or more columns)
    </chart/>
    <coverage/>
    <format/>
    <metrics/>
    <movers>
      <columns>
        (a single column)
      </columns/>
    </movers/>
    <overview/>
  </historical>
</clover-report>

```

## Parameters

The `<clover-report>` task generates `current` and `historical` reports in multiple formats. Parameters for `<clover-report>`:

Attribute	Description	Required
coverageCacheSize	This is a performance tuning option, which is used to specify the maximum size of coverage data kept in memory when generating a report. Accepts storage values such as 256m (256 megabytes) or 1g (1 gigabyte). Also accepts the value "nocache" to force all coverage data to be loaded into memory.	No. Default is 256m.
initstring	The initstring of the coverage database.	No; if not specified here, Clover will look in the default location ( <code>\${basedir}.clover</code> ). If you have specified an <code>initstring</code> on the <code>&lt;clover-setup&gt;</code> task, you must ensure <code>&lt;clover-setup&gt;</code> is called prior the execution of this task.
failOnError	If true, failure to generate a report causes a build failure.	No; defaults to "true".
projectName	Overrides the project name set in the Ant build file. This is used for display purposes only.	No; defaults to the project name of the Ant build file.

## Nested elements

These elements represent the actual reports to be generated. You can generate multiple reports by specifying more than one of these inside a `<clover-report>` element. Each report will contain links to the other reports. See [clover-report](#).

### `<current>`

Generates a current coverage report. Specify the report format using a nested [Format](#) element. Valid formats are XML, HTML, PDF and JSON, although not all configurations support all formats. The default format is PDF if `summary="true"`, or XML if not. See [clover-report](#).

### Parameters for `<current>`


Attribute	Description	Required
<code>alwaysReport</code>	If set to true, a report will be generated even in the absence of coverage data.	No; defaults to "false".
<code>charset</code>	The character set to use in the HTML reports.	No. Default is UTF-8
<code>homepage</code>	Specifies the start page to use. This can be one of the predefined pages: <code>dashboard</code> , <code>overview</code> , <code>aggregate</code> , <code>test results</code> , <code>quickwins</code> , <code>projectrisks</code> or an arbitrary URL.	No; defaults to "dashboard".
<code>includeFailedTestCoverage</code>	Specifies whether or not to include coverage attributed to a test that has failed.	No; default is "false".
<code>maxTestsPerFile</code>	Specifies the maximum number of tests (ranked by coverage contribution) to display on a source file report page. This parameter can be used to reduce the size of reports for projects with very large numbers of tests.	No; unlimited if not specified or -1.
<code>numThreads</code>	The number of threads to start when generating an HTML report. A value of 0 will disable multi-threading for report generation.	No. Default is 2.
<code>outfile</code>	The outfile to write output to. If it does not exist, it is created. Depending on the specified format, this either represents a regular file (PDF, XML) or a directory (HTML, JSON).	Yes.
<code>span</code>	Specifies how far back in time to include coverage recordings from since the last Clover build. See <a href="#">Using Spans</a> .	No; default includes "all coverage data found".
<code>summary</code>	Specifies whether to generate a summary report or detailed report. Currently this applies for XML and PDF reports. See <code>srcLevel</code> on the format element for HTML.	No; defaults to "false".
<code>showLambdaFunctions</code>	Whether to present lambda functions in the report. If set to <i>false</i> , they are hidden on the methods' list, but code metrics still include them.	No; defaults to "false".
<code>showInnerFunctions</code>	Whether to show inner functions, i.e. functions declared inside methods in the report. This applies to Java8 lambda functions for instance. If set to <i>false</i> , then they are hidden on the methods' list, but code metrics still include them.	No; defaults to "false".



showUniqueCoverage	Calculate and show unique per-test coverage (for large projects, this can take a significant amount of time). Defaults to - <code>true</code> for Clover 3.1.4 and older - <code>false</code> for Clover 3.1.5 and newer.	No.
timeout	The amount of time to wait for report generation to finish before failing the build. This value must be an <a href="#">Interval</a> .	No. Default is no timeout.
title	Specifies a title for the report.	No.
titleAnchor	If specified, the report title will be rendered as a hyperlink to this href.	No; default is to not render the report title as a hyperlink.
titleTarget	Specifies the href target if the title is to be rendered as a hyperlink (see <code>titleAnchor</code> above). <i>HTML format only</i> .	No; default is <code>"_top"</code> .

### Nested elements of <current>

These elements represent individual sections of the 'current' report. If you do not specify any of these elements, all the sections will be included in the report. If you specify more one or more of these elements, only the specified sections will be included.

 You may specify multiple <overview> and <coverage> elements in the 'current' report. These may have different properties and include different elements. The charts will appear in the report in the same order they appear in the <current> element. The <movers> element always appears at the end of the report following these charts regardless of its location in the <current> element.

### <columns>

Specifies the data columns to be included on summary pages. If not specified, default columns will be output.

Specific columns are defined as sub-elements to this one. See the [clover-report](#).

Columns can be defined in a <clover-columns/> Ant type for [referencing](#) elsewhere in the build file.

Each column element takes an optional `format` attribute which determines how the column's value is rendered. The `format` attribute may be one of the following:


- `raw` — the actual value. Always used for total columns
- `bar` — render a bar chart (40px wide) showing the coverage percentage
- `longbar` — same as `bar` above, except 200px wide
- `%` — The coverage percentage value

Note that `bar` and `%` are not valid formats for total columns.

All column elements also take `max` and/or `min` threshold attributes. If the value for the column is outside the threshold, the value will be highlighted.

### Table of Column Names

Column	Description	Valid Format Attributes
avgClassesPerFile	The average number of classes per file.	<code>raw</code>
avgMethodComplexity	The average number of paths per method.	<code>raw</code>

avgMethodsPerClass	The average number of methods per class.	raw
avgStatementsPerMethod	The average number of statements per method.	raw
complexity	Cyclomatic Complexity is a measure of the number of paths in your code.	raw
complexityDensity	The average complexity per statement.	raw
coveredBranches	The amount of covered branches.	raw;bar;%;longbar
coveredElements	The total number of covered elements (branches + statements) in the project.	raw;bar;%;longbar
coveredMethods	The amount of covered methods.	raw;bar;%;longbar
coveredStatements	The amount of covered statements.	raw;bar;%;longbar
expression	The body of this element will be evaluated as an arithmetic expression. All other column names can be referenced. See Clover EL. This column takes an optional title attribute.	raw
filteredElements	The amount of elements that have been filtered out of the report.	raw;bar;%;longbar
ncLineCount	The total number of non-comment lines.  When using Clover on Groovy source code, this column consistently reports '0' at the moment.	raw
lineCount	The total number of lines.	raw
SUM	Scientifically Untested Metric. This is very similar to <code>crap4j</code> and is defined by this expression: <div style="border: 1px solid #ccc; padding: 10px; margin: 10px 0;"><pre>complexity^2 * ((1 - %coveredElements/100)^3) + complexity</pre></div>	raw
percentageCoveredContribution	Helps you to work out how much an individual package, file or class contributes (percentage-wise) to the overall number of covered elements in the project. Useful for spotting quick wins.	raw;bar;%;longbar
percentageUncoveredContribution	Helps you to work out how much an individual package, file or class contributes (percentage-wise) to the overall number of uncovered elements in the project. Useful for spotting quick wins.	raw;bar;%;longbar
totalBranches	The total number of branches in the project.	raw
totalChildren	The number of lower order elements. The order of elements is: Project, Package, File, Class, Method, Statement	raw
totalClasses	The total number of classes below the package, project or file.	raw

totalElements	The total number of elements (branches + statements) in the project.	raw
totalFiles	The total number of files below the package or project.	raw
totalMethods	The total number of methods in the project.	raw
totalPercentageCovered	The total coverage.	raw;bar;%;longbar
totalStatements	The total number of statements in the project.	raw
uncoveredBranches	Branches that were not executed.	raw;bar;%;longbar
uncoveredElements	Elements that were not executed.	raw;bar;%;longbar
uncoveredMethods	Methods that were not executed.	raw;bar;%;longbar
uncoveredStatements	Statements that were not executed.	raw;bar;%;longbar

### Column Attributes

Each of the above column elements can take the following attributes:

Attribute	Description	Required
format	Determines how the value is rendered. Depending on the column, this may be one of raw, bar, % or longbar.	No.
min	Sets a minimum threshold on the value of the column. If the value is less than this it will be highlighted.	No.
max	Sets a maximum threshold on the value of the column. If the value is greater than this it will be highlighted.	No.
scope	Controls at which level in the report the column will appear. The scope attribute can be one of: "package", "class" or "method". If omitted, the column will be used at every level in the report. Note that only the following columns support the scope attribute: <b>expression</b> , <b>complexity</b> , <b>complexityDensity</b> , <b>coveredXXX</b> , <b>uncoveredXXX</b> and <b>totalXXX</b> .	No.

### Clover Expression Language

Clover Expression Language enables you to combine any of Clover's built-in column types to produce a custom column. The following arithmetic operators are available: +, -, \*, /, ^, (). Any of Clover's columns may be referenced.

A percentage sign, '%', before a column identifier will evaluate to the percentage of that columns data, rather than its raw value. e.g. %CoveredElements == (CoveredElements/TotalElements) \* 100

*Example:*

```
<columns>
  <expression title="SUM">complexity^2 * ((1 - %coveredElements/100)^3) +
  complexity</expression>
</columns>
```

**<fileset>**

<current> supports nested filesets which control which source files are to be included in a report. Only classes which are from the source files in the fileset are included in the report. This allows reports to focus on certain packages or particular classes. By using [Ant fileset](#) selectors, more complicated selections are possible, such as the files which have recently changed, or files written by a particular author.

**<format>**

Specifies the output format and various options controlling the rendering of a report.

**Parameters for <format>**

Attribute	Description	Required
type	The output format in which to render the report. Valid values are pdf, xml, html and json. Note that not all report formats support all other attributes.	Yes, unless refid is set.
refid	The id of another format element that will be used for this report. See <a href="#">Sharing Report Formats</a> .	No.
id	The id of this format element.	No.
bw	Specify that the report should be black-and-white. Supported by PDF reports only.	No; defaults to "false".
orderBy	Specify how to order coverage tables. This attribute has no effect on XML format. Valid values are:  Alpha — Alphabetical. PcCoveredAsc — Percent total coverage, ascending. PcCoveredDesc — Percent total coverage, descending. ElementsCoveredAsc — Total elements covered, ascending. ElementsCoveredDesc — Total elements covered, descending. ElementsUncoveredAsc — Total elements uncovered, ascending. ElementsUncoveredDesc — Total elements uncovered, descending.	No; defaults to PcCoveredAsc.
noCache	<i>(HTML only)</i> If true, insert nocache directives in HTML output.	No; defaults to "false".
srcLevel	If true, include source-level coverage information in the report.	No; defaults to "true".
filter	comma or space separated list of contexts to exclude when generating coverage reports. See <a href="#">Using Coverage Contexts</a> .	No.
pageSize	<i>(PDF only)</i> Specify the page size to use. Valid values are A4, LETTER.	No; defaults to "A4".
showEmpty	If true, classes, files and packages that do not contain any executable code (i.e. methods, statements, or branches) are included in reports. These are normally not shown.	No; defaults to "false".
reportStyle	<b>Since Clover 4.0.</b> Style of the HTML report: <ul style="list-style-type: none"> <li>"adg" - new <a href="#">ADG</a> look &amp; feel</li> <li>"classic" - old JavaDoc-like report (deprecated)</li> </ul>	No; defaults to "adg".
tabWidth	<i>(Source level reports only)</i> The number of space chars to replace TAB characters with.	No; defaults to 4.

maxNameLength	The maximum length in chars of package or classnames in the report. Longer names will be truncated. A value < 0 indicates no limit.	No; defaults to no limit.
callback	The name of the callback function to wrap the JSON. If set to an empty string, "", then the JSON will not be wrapped.	No; default is 'processClover'.

**<sourcepath>**

Specifies an Ant path that Clover should use when looking for source files.

**<testsources>**

<testsources> is an [Ant fileset](#) that can be used to distinguish test source code from application source code. All files included in the fileset will be displayed in the separate 'Test' node of the coverage tree. If omitted, Clover's [default test detection algorithm](#) will be used to distinguish test sources.

**<testresults>**

<testresults> is an optional [Ant fileset](#) that Clover uses to integrate the results of your unit tests into the report. The results should be generated using the [Ant junit](#) task with an XML formatter.

<testresults> is generally not required by most users, as the built-in test results will provide all required information in the majority of cases. For more details please see '[Advanced Usage](#)'.

**<historical>**

Generates a historical coverage report. Specify the report format using a nested Format element (see below). Valid formats are HTML or PDF. The default format is HTML. Contents of the historical report are optionally controlled by nested elements. See [clover-report](#).

**Parameters for <historical>**

Attribute	Description	Required
title	Specifies a title for the report.	No.
titleAnchor	if specified, the report title will be rendered as a hyperlink to this href.	No; default is to not render the report title as a hyperlink.
titleTarget	Specifies the href target if the title is to be rendered as a hyperlink (see <code>titleAnchor</code> above). <i>HTML format only</i>	No; default is "_top".
charset	The character set to use in the HTML reports.	No. Default is UTF-8
dateFormat	Specifies a date format string for parsing the "from" and "to" fields. The string must contain a valid <a href="#">java.text.SimpleDateFormat</a> pattern.	No; default is set to <a href="#">java.text.SimpleDateFormat</a> using the default pattern and date format symbols for the default locale.
from	Specifies the date before which data points will be ignored. The date must be specified either using the default <a href="#">java.text.SimpleDateFormat</a> for your locale or using the pattern defined in the "dateFormat" attribute.	No.
historyDir	The directory containing Clover historical data as produced by the <a href="#">&lt;clover-historypoint&gt;</a> task.	Yes.

historyIncludes	An <a href="#">Ant GLOB</a> to select specific history point files within the <code>historyDir</code> directory	No; default is <code>clover-*.xml.gz</code>
json	A true or false value. If set to true, then a file called <code>historical-json.js</code> will be created in the top level directory containing some <a href="#">JSONP</a> . The callback method will be: <code>processHistoricalCloverData</code> and takes a single <a href="#">JSON</a> object parameter. The structure of the JSON is the same as used by the <a href="#">Google Visualization API</a> . <a href="#">More information</a> .	No. Default is "false".
outfile	The outfile to write output to. If it does not exist, it is created. Depending on the specified format, this either represents a regular file (PDF) or a directory (HTML).	Yes; default format is HTML.
package	Restricts the report to a particular package.	No.
to	Specifies the date after which data points will be ignored. The date must be specified either using the default <code>java.text.SimpleDateFormat</code> for your locale or using the pattern defined in the "dateFormat" attribute.	No.

#### Nested elements of `<historical>`

These elements represent individual sections of the historical report. If you do not specify any of these elements, all the sections will be included in the report. If you specify more one or more of these elements, only the specified sections will be included. You may specify multiple `<overview>` and `<coverage>` elements in the historical report. These may have different properties and include different elements. The charts will appear in the report in the same order they appear in the `<historical>` element. The `<movers>` element always appears at the end of the report following these charts regardless of its location in the `<historical>` element.

#### `<added>`

`<added>` displays new classes for the given column.

#### Parameters for `<added>`

Attribute	Description	Required
range	The maximum number of classes to show. If the value is 5, then a maximum of 5 "gainers" and 5 "losers" will be shown.	No; defaults to 5.
interval	The time interval over which the delta should be calculated (from the last history point). Uses the <a href="#">Interval</a> format. The range is automatically adjusted to the closest smaller interval available.	No; the default is to take the delta of the last two history points.

#### Nested elements of `<added>`

The `<added>` element can take a single [column](#) element, allowing you to add one additional metric to the data shown in `<added>`.

#### Example: `<totalStatements>`

You could add `totalStatements` to `<added>` with the following code.

```
<added>
  <columns>
    <totalStatements/>
  </columns>
</added>
```

**i** No more than one additional column can be attached to `<added>`. The 'added' classes table will always appear above the Movers section in the report - regardless of its order in the XML.

### `<chart>`

A custom chart.

#### Parameters for `<chart>`

Attribute	Description	Required
title	The title to use for the chart.	No.
logscale	Use a log scale for the y axis.	No; default is set to 'true'.
xLabel	The x label to use.	No.
yLabel	The x label to use.	No.
width	The width of the chart image.	No; default is set to 640 pixels (640px).
height	The height of the chart image.	No; default is set to 480 pixels (480px).
upperBound	The maximum y value to display.	No; default is set to -1 (unbounded).
autoRange	If set to 'true', the y-axis will be chosen automatically to fit the data best.	No; default is 'false'.

#### Nested elements of `<chart>`

The `<chart>` element can take arbitrary columns, but the format of each column can be only 'raw', or '%'. The supported columns are identical to [columns in the `<current>` element](#). If no columns nested element is supplied, then the default columns from the `<coverage>` element are used.

#### `<coverage>`

- Note that the 'include' attribute from Clover 2.0 has been replaced by the `<columns>` element.

Specifies a chart showing percentage coverage over time.

This element does not support any attributes. The default behaviour is that everything is included.

#### `<format>`

Specifies the output format and various options controlling the rendering of a report.

#### Parameters for `<format>`

Attribute	Description	Required
-----------	-------------	----------

type	The output format in which to render the report. Valid values are <code>pdf</code> , <code>xml</code> , <code>html</code> and <code>json</code> . Note that not all report formats support all other attributes.	Yes, unless <code>refid</code> is set.
refid	The id of another format element that will be used for this report. See <a href="#">Sharing Report Formats</a> .	No.
id	The id of this format element.	No.
bw	Specify that the report should be black-and-white. Supported by PDF reports only.	No; defaults to "false".
orderBy	Specify how to order coverage tables. This attribute has no effect on XML format. Valid values are:  Alpha — Alphabetical. PcCoveredAsc — Percent total coverage, ascending. PcCoveredDesc — Percent total coverage, descending. ElementsCoveredAsc — Total elements covered, ascending. ElementsCoveredDesc — Total elements covered, descending. ElementsUncoveredAsc — Total elements uncovered, ascending. ElementsUncoveredDesc — Total elements uncovered, descending.	No; defaults to <code>PcCoveredAsc</code> .
noCache	<i>(HTML only)</i> If true, insert <code>nocache</code> directives in HTML output.	No; defaults to "false".
srcLevel	If true, include source-level coverage information in the report.	No; defaults to "true".
filter	comma or space separated list of contexts to exclude when generating coverage reports. See <a href="#">Using Coverage Contexts</a> .	No.
pageSize	<i>(PDF only)</i> Specify the page size to use. Valid values are <code>A4</code> , <code>LETTER</code> .	No; defaults to "A4".
showEmpty	If true, classes, files and packages that do not contain any executable code (i.e. methods, statements, or branches) are included in reports. These are normally not shown.	No; defaults to "false".
reportStyle	<b>Since Clover 4.0.</b> Style of the HTML report: <ul style="list-style-type: none"> <li>"adg" - new <a href="#">ADG</a> look &amp; feel</li> <li>"classic" - old JavaDoc-like report (deprecated)</li> </ul>	No; defaults to "adg".
tabWidth	<i>(Source level reports only)</i> The number of space chars to replace TAB characters with.	No; defaults to 4.
maxNameLength	The maximum length in chars of package or classnames in the report. Longer names will be truncated. A value < 0 indicates no limit.	No; defaults to no limit.
callback	The name of the callback function to wrap the JSON. If set to an empty string, "", then the JSON will not be wrapped.	No; default is 'processClover'.

### <metrics>

- Note that the 'include' attribute from Clover 2.0 has been replaced by the <columns> element.

Specifies a chart showing other metrics over time.

Parameters for <metrics>:



Attribute	Description	Required
logscale	Specifies that a log scale be used on the Range Axis. This can be useful if you are including, for example, LOC and packages in the same chart.	No; defaults to "true".

The default metrics included in the chart are `loc`, `ncloc`, `methods` and `classes`.

#### <movers>

Specifies a table that shows those classes that have a coverage delta higher than a specified threshold over a specified time period. This can be specified multiple times, to track project movers over a given time frame, for example, weeks and months.

Parameters for <movers>:

Attribute	Description	Required
threshold	The absolute point change in percent coverage that class must have changed by for inclusion. e.g "10%".	No; defaults to 1% .
range	The maximum number of classes to show. If the value is 5, then a maximum of 5 "gainers" and 5 "losers" will be shown.	No; defaults to 5 .
interval	The time interval over which the delta should be calculated (from the last history point). Uses the <a href="#">Interval</a> format. The range is automatically adjusted to the closest smaller interval available.	No; the default is to take the delta of the last two history points.

#### Nested elements of <movers>

The <movers> element can take a single [column](#) element, allowing you to add one additional metric to the data shown in <movers>.

#### Example: <totalStatements>

You could add `totalStatements` to <movers> with the following code.

```
<movers>
  <columns>
    <totalStatements/>
  </columns>
</movers>
```

 No more than one additional column can be added to <movers>.

#### <overview>

Specifies a section that provides summary of the total percentage coverage at the last history point. This element does not support any attributes.

#### Examples

##### Examples of Current Report configurations

```
<clover-report>
  <current outfile="current.xml"/>
</clover-report>
```

Generates an XML report of the current coverage.

```
<clover-report>
  <current outfile="current.pdf" summary="true">
    <format type="pdf"/>
  </current>
</clover-report>
```

Generates a PDF report of the current coverage.

```
<target name="report.json" depends="with.clover">
<clover-report>
  <current outfile="clover_json">
    <format type="json"/>
    <columns>
      <lineCount/>
      <ncLineCount/>
    </columns>
  </current>
</clover-report>
</target>
```

Generates a JSON report, where the chart elements are customised by specifying them with `<columns>`. See the [JSON reference page](#).

```
<clover-report>
  <current outfile="clover_html" title="My Project" summary="true">
    <format type="html"/>
  </current>
</clover-report>
```

Generates a summary report, in HTML with a custom title. Note that the "outfile" argument requires a directory instead of a filename.

```
<clover-report>
  <current outfile="report-current" title="Coverage">
    <fileset dir="src/main/java">
      <exclude name="**/*Blah.java"/>
    </fileset>
    <format srclevel="true" type="html"/>
  </current>
</clover-report>
```

Generates a HTML clover report and excludes all \*Blah.java classes from the report.

```
<clover-report>
  <current outfile="clover_html" title="Util Coverage">
    <format type="html" orderBy="ElementsCoveredAsc"/>
    <testsources dir="src/test" includes="**/*.java"/>
  </current>
</clover-report>
```

Generates a detailed coverage report in HTML with output ordered by total number of covered elements, rather than percentage coverage. All source files under `src/test` will be in the separate 'Test' coverage node in the report.

```
<clover-report>
  <current outfile="clover_html" title="My Project">
    <format type="html"/>
    <sourcepath>
      <pathelement path="/some/other/location"/>
    </sourcepath>
  </current>
</clover-report>
```

Generates a source-level report in HTML. Clover will search for source files in the directory `/some/other/location`.

```
<tstamp>
  <format property="report.limit" pattern="MM/dd/yyyy hh:mm aa"
    offset="-1" unit="month"/>
</tstamp>
<clover-report>
  <current outfile="report-current"
    title="Coverage since ${report.limit}">
    <fileset dir="src/main">
      <date datetime="${report.limit}" when="after"/>
    </fileset>
    <format srclevel="true" type="html"/>
  </current>
</clover-report>
```

This example generates a current coverage report for all files in the project that have changed in the last month. Replacing the `<date>` selector with `<contains text="@author John Doe"/>` would generate a coverage report for all code where John Doe is the author.

```
<clover-report>
  <current outfile="report-current" title="Coverage">
    <fileset dir="src">
      <patternset refid="clover.files"/>
    </fileset>
    <format srclevel="true" type="html"/>
  </current>
</clover-report>
```

In this example the standard Clover patternset is used to restrict the report to the currently included source files. You could use this if you have changed the `exclude` or `include` definitions in the `<clover-setup>` task and you have not removed the coverage database. It will prevent classes, currently in the database but now excluded, from being included in the report. It is prudent, however, to delete the coverage database, coverage information and recompile when you change these settings.

#### Example of customising columns

```
<clover-report>
  <current outfile="report-current" title="Coverage">
    <format type="html"/>
    <columns>
      <coveredMethods format="bar" min="75"/>
      <coveredStatements format="%" />
      <coveredBranches format="raw" />
    </columns>
  </current>
</clover-report>
```

Generates a HTML report that will only include a bar chart showing the percentage of methods covered, the actual percentage of statements covered and the actual number of branches covered. If less than 75% of methods are covered, those values will be highlighted.

#### Example of linked reports

```
<clover-report>
  <current outfile="report1" title="Coverage Report 1">
    <format type="html"/>
    <fileset dir="src">
      <patternset refid="clover.files"/>
    </fileset>
  </current>

  <current outfile="report2" title="Coverage Report 2">
    <format type="html"/>
    <fileset dir="othersrc">
      <patternset refid="other.clover.files"/>
    </fileset>
  </current>
</clover-report>
```

Generates two HTML reports. Each of these reports will contain a link to the other.

#### Examples of Historical Report Configurations

```
<clover-report>
  <historical outfile="historical.pdf"
             historyDir="clover_history">
    <format type="pdf"/>
  </historical>
</clover-report>
```

Generates a historical report in PDF. Assumes that `<clover-historypoint>` has generated more than one history

file in the directory "clover\_history". Writes the output to the file specified in the `outfile` parameter.

```
<clover-report>
  <historical outfile="two_months" title="My Project"
    from="020101" to="020301" dateFormat="yyMMdd"
    historyDir="clover_history">
    <format type="html"/>
  </historical>
</clover-report>
```

Generates a basic historical report in HTML for a certain time period. Clover will scan the `historyDir` and use any history points that fall within the requested time period. The `outfile` attribute will be treated as a **directory**; a file `historical.html` will be written into this directory. If the directory doesn't exist, it will be created.

```
<clover-report>
  <historical outfile="report.pdf" title="My Project"
    historyDir="clover_history">
    <overview/>
    <movers threshold="5%" range="20" interval="2w"/>
    <format type="pdf"/>
  </historical>
</clover-report>
```

Generates a PDF historical report that only includes an overview section (showing summary coverage at the last history point) and a movers table showing classes that have a code coverage delta of greater than +/- 5% over the two weeks prior to the last history point. Will include at most 20 gainers and 20 losers.

## References

[Column Name Reference Table](#)

[Clover Expression Language](#)

### <added> element

<added> displays new classes for the given column.

### Parameters for <added>

Attribute	Description	Required
range	The maximum number of classes to show. If the value is 5, then a maximum of 5 "gainers" and 5 "losers" will be shown.	No; defaults to 5.
interval	The time interval over which the delta should be calculated (from the last history point). Uses the <a href="#">Interval</a> format. The range is automatically adjusted to the closest smaller interval available.	No; the default is to take the delta of the last two history points.

### Nested elements of <added>

The <added> element can take a single [column](#) element, allowing you to add one additional metric to the data shown in <added>.

**Example: <totalStatements>**

You could add totalStatements to <added> with the following code.

```
<added>
  <columns>
    <totalStatements/>
  </columns>
</added>
```

**i** No more than one additional column can be attached to <added>. The 'added' classes table will always appear above the Movers section in the report - regardless of its order in the XML.

**<chart> element**

A custom chart.

**Parameters for <chart>**

Attribute	Description	Required
title	The title to use for the chart.	No.
logscale	Use a log scale for the y axis.	No; default is set to 'true'.
xLabel	The x label to use.	No.
yLabel	The x label to use.	No.
width	The width of the chart image.	No; default is set to 640 pixels (640px).
height	The height of the chart image.	No; default is set to 480 pixels (480px).
upperBound	The maximum y value to display.	No; default is set to -1 (unbounded).
autoRange	If set to 'true', the y-axis will be chosen automatically to fit the data best.	No; default is 'false'.

**Nested elements of <chart>**

The <chart> element can take arbitrary columns, but the format of each column can be only 'raw', or '%'. The supported columns are identical to [columns in the <current>element](#). If no columns nested element is supplied, then the default columns from the <coverage> element are used.

**<columns> element**

Specifies the data columns to be included on summary pages. If not specified, default columns will be output.

Specific columns are defined as sub-elements to this one. See the [clover-report](#).

Columns can be defined in a <clover-columns/> Ant type for [referencing](#) elsewhere in the build file.


Each column element takes an optional *format* attribute which determines how the column's value is rendered. The *format* attribute may be one of the following:

- *raw* — the actual value. Always used for total columns
- *bar* — render a bar chart (40px wide) showing the coverage percentage
- *longbar* — same as *bar* above, except 200px wide
- *%* — The coverage percentage value

Note that `bar` and `%` are not valid formats for total columns.

All column elements also take `max` and/or `min` threshold attributes. If the value for the column is outside the threshold, the value will be highlighted.

### Table of Column Names

Column	Description	Valid Format Attributes
<code>avgClassesPerFile</code>	The average number of classes per file.	<code>raw</code>
<code>avgMethodComplexity</code>	The average number of paths per method.	<code>raw</code>
<code>avgMethodsPerClass</code>	The average number of methods per class.	<code>raw</code>
<code>avgStatementsPerMethod</code>	The average number of statements per method.	<code>raw</code>
<code>complexity</code>	Cyclomatic Complexity is a measure of the number of paths in your code.	<code>raw</code>
<code>complexityDensity</code>	The average complexity per statement.	<code>raw</code>
<code>coveredBranches</code>	The amount of covered branches.	<code>raw;bar;%;longbar</code>
<code>coveredElements</code>	The total number of covered elements (branches + statements) in the project.	<code>raw;bar;%;longbar</code>
<code>coveredMethods</code>	The amount of covered methods.	<code>raw;bar;%;longbar</code>
<code>coveredStatements</code>	The amount of covered statements.	<code>raw;bar;%;longbar</code>
<code>expression</code>	The body of this element will be evaluated as an arithmetic expression. All other column names can be referenced. See Clover EL. This column takes an optional title attribute.	<code>raw</code>
<code>filteredElements</code>	The amount of elements that have been filtered out of the report.	<code>raw;bar;%;longbar</code>
<code>ncLineCount</code>	The total number of non-comment lines.  When using Clover on Groovy source code, this column consistently reports '0' at the moment.	<code>raw</code>
<code>lineCount</code>	The total number of lines.	<code>raw</code>
<code>SUM</code>	Scientifically Untested Metric. This is very similar to <code>crap4j</code> and is defined by this expression: <div style="border: 1px solid #ccc; padding: 10px; margin: 10px 0;"><pre>complexity^2 * ((1 - %coveredElements/100)^3) + complexity</pre></div>	<code>raw</code>
<code>percentageCoveredContribution</code>	Helps you to work out how much an individual package, file or class contributes (percentage-wise) to the overall number of covered elements in the project. Useful for spotting quick wins.	<code>raw;bar;%;longbar</code>

percentageUncoveredContribution	Helps you to work out how much an individual package, file or class contributes (percentage-wise) to the overall number of uncovered elements in the project. Useful for spotting quick wins.	raw;bar;%;longbar
totalBranches	The total number of branches in the project.	raw
totalChildren	The number of lower order elements. The order of elements is: Project, Package, File, Class, Method, Statement	raw
totalClasses	The total number of classes below the package, project or file.	raw
totalElements	The total number of elements (branches + statements) in the project.	raw
totalFiles	The total number of files below the package or project.	raw
totalMethods	The total number of methods in the project.	raw
totalPercentageCovered	The total coverage.	raw;bar;%;longbar
totalStatements	The total number of statements in the project.	raw
uncoveredBranches	Branches that were not executed.	raw;bar;%;longbar
uncoveredElements	Elements that were not executed.	raw;bar;%;longbar
uncoveredMethods	Methods that were not executed.	raw;bar;%;longbar
uncoveredStatements	Statements that were not executed.	raw;bar;%;longbar

### Column Attributes

Each of the above column elements can take the following attributes:

Attribute	Description	Required
format	Determines how the value is rendered. Depending on the column, this may be one of raw, bar, % or longbar.	No.
min	Sets a minimum threshold on the value of the column. If the value is less than this it will be highlighted.	No.
max	Sets a maximum threshold on the value of the column. If the value is greater than this it will be highlighted.	No.
scope	Controls at which level in the report the column will appear. The scope attribute can be one of: "package", "class" or "method". If omitted, the column will be used at every level in the report. Note that only the following columns support the scope attribute: <b>expression</b> , <b>complexity</b> , <b>complexityDensity</b> , <b>coveredXXX</b> , <b>uncoveredXXX</b> and <b>totalXXX</b> .	No.

### Clover Expression Language

Clover Expression Language enables you to combine any of Clover's built-in column types to produce a custom



column. The following arithmetic operators are available: +, -, \*, /, ^, (). Any of Clover's columns may be referenced.

A percentage sign, '%', before a column identifier will evaluate to the percentage of that columns data, rather than its raw value. e.g. `%CoveredElements == (CoveredElements/TotalElements) * 100`

*Example:*

```
<columns>
  <expression title="SUM">complexity^2 * ((1 - %coveredElements/100)^3) +
  complexity</expression>
</columns>
```

### <format> element

Specifies the output format and various options controlling the rendering of a report.

#### Parameters for <format>

Attribute	Description	Required
type	The output format in which to render the report. Valid values are <code>pdf</code> , <code>xml</code> , <code>html</code> and <code>json</code> . Note that not all report formats support all other attributes.	Yes, unless <code>refid</code> is set.
refid	The id of another format element that will be used for this report. See <a href="#">Sharing Report Formats</a> .	No.
id	The id of this format element.	No.
bw	Specify that the report should be black-and-white. Supported by PDF reports only.	No; defaults to "false".
orderBy	Specify how to order coverage tables. This attribute has no effect on XML format. Valid values are:  Alpha — Alphabetical. PcCoveredAsc — Percent total coverage, ascending. PcCoveredDesc — Percent total coverage, descending. ElementsCoveredAsc — Total elements covered, ascending. ElementsCoveredDesc — Total elements covered, descending. ElementsUncoveredAsc — Total elements uncovered, ascending. ElementsUncoveredDesc — Total elements uncovered, descending.	No; defaults to <code>PcCoveredAsc</code> .
noCache	<i>(HTML only)</i> If true, insert <code>nocache</code> directives in HTML output.	No; defaults to "false".
srcLevel	If true, include source-level coverage information in the report.	No; defaults to "true".
filter	comma or space separated list of contexts to exclude when generating coverage reports. See <a href="#">Using Coverage Contexts</a> .	No.
pageSize	<i>(PDF only)</i> Specify the page size to use. Valid values are <code>A4</code> , <code>LETTER</code> .	No; defaults to "A4".
showEmpty	If true, classes, files and packages that do not contain any executable code (i.e. methods, statements, or branches) are included in reports. These are normally not shown.	No; defaults to "false".

reportStyle	<b>Since Clover 4.0.</b> Style of the HTML report: <ul style="list-style-type: none"> <li>"adg" - new <a href="#">ADG</a> look &amp; feel</li> <li>"classic" - old JavaDoc-like report (deprecated)</li> </ul>	No; defaults to "adg".
tabWidth	<i>(Source level reports only)</i> The number of space chars to replace TAB characters with.	No; defaults to 4.
maxNameLength	The maximum length in chars of package or classnames in the report. Longer names will be truncated. A value < 0 indicates no limit.	No; defaults to no limit.
callback	The name of the callback function to wrap the JSON. If set to an empty string, "", then the JSON will not be wrapped.	No; default is 'processClover'.

### <movers> element

Specifies a table that shows those classes that have a coverage delta higher than a specified threshold over a specified time period. This can be specified multiple times, to track project movers over a given time frame, for example, weeks and months.

Parameters for <movers>:

Attribute	Description	Required
threshold	The absolute point change in percent coverage that class must have changed by for inclusion. e.g "10%".	No; defaults to 1%.
range	The maximum number of classes to show. If the value is 5, then a maximum of 5 "gainers" and 5 "losers" will be shown.	No; defaults to 5.
interval	The time interval over which the delta should be calculated (from the last history point). Uses the <a href="#">Interval</a> format. The range is automatically adjusted to the closest smaller interval available.	No; the default is to take the delta of the last two history points.

### Nested elements of <movers>

The <movers> element can take a single [column](#) element, allowing you to add one additional metric to the data shown in <movers>.

### Example: <totalStatements>

You could add totalStatements to <movers> with the following code.

```
<movers>
  <columns>
    <totalStatements/>
  </columns>
</movers>
```

**i** No more than one additional column can be added to <movers>.

### JSON reference

This page documents Clover-for-Ant's implementation of JSON data and how to make use of it.

On this page:

- [Why JSON?](#)
- [JSON data format](#)
  - [Current report](#)
  - [Historical report](#)

- [Examples of JSON report usage](#)
  - [Clover Historical Data in JSONP](#)

### Why JSON?

The **JSON format** is supported as an output type in Clover specifically to create integration opportunities with other applications. The JSON data from Clover is easy to manipulate programmatically, allowing innovative developers to use it for displaying or processing their coverage data in novel ways.

### Clover-Ant code for JSON output

```
<target name="report.json" depends="with.clover">
<clover-report>
  <current outfile="clover_json">
    <format type="json"/>
    <columns>
      <lineCount/>
      <ncLineCount/>
    </columns>
  </current>
</clover-report>
</target>
```

### JSON data format

#### Current report

#### Code snippet

```
<clover-report>
  <current outfile="${clover.report}">
    <format type="json"/>
  </current>
</clover-report>
```

### Files generated for a current report

```

<json_report_directory>
  <package/name/directory>
    aggregate-pkg-risks.js - the aggregated "Top Risks" cloud map
for a package (including sub-packages)
    aggregate-quick-wins.js - the aggregated "Quick Wins" cloud map
for a package (including sub-packages)
    <source file>.js      - detail data for a source file (hit
counts, statistics)
    package.js            - package summary (list of classes +
package statistics)
    pkg-risks.js         - the "Package Risks" cloud map for a
package (without sub-packages)
    quick-wins.js       - the "Quick Wins" cloud map for a
package (without sub-packages)
    colophon.js         - build time stamp etc
    project.js          - project summary (list of packages +
projects statistics)
    proj-risks.js      - the "Project Risks" cloud map for a
project
    quick-wins.js     - the "Quick Wins" cloud map for a
project

```

#### Format of a source file summary ("**<source file>.js**")

```

processClover ( {
  "id": "com_cenqua_samples_money_Money_java", // name of source
files, dots and slashes replaced by underscores
  "lines": [ "", "", "157", "157" ], // hit counts for each
source line
  "stats": { // statistics, -1 means
no data or not applicable
    "Complexity": 18,
    "CoveredElements": 42,
    "ErroneousTests": 0,
    "FailingTests": 0,
    "PassingTests": 0,
    "PcErroneousTests": -1,
    "PcFailingTests": -1,
    "PcPassingTests": -1,
    "TestExecutionTime": 0,
    "Tests": 0,
    "TotalPercentageCovered": 89.3617,
    "UncoveredElements": 5
  }
} );

```

#### Format of a package summary (**package.js**)

```

processClover ( {
  "children":["IMoney.java", "MoneyBag.java", "Money.java"], // list of
files in the packages
  "name":"com.cenqua.samples.money", // name of
the package
  "stats": { //
statistics
  "AvgClassesPerFile":1,
  "AvgMethodComplexity":1.7419355,
  "AvgMethodsPerClass":10.333333,
  "AvgStatementsPerMethod":3.064516,
  "ComplexityDensity":0.56842107,
  "ComplexityToCoverage":58,
  "CoveredBranches":39,
  "CoveredMethods":31,
  "CoveredStatements":90,
  "FilteredElements":0,
  "LineCount":286,
  "NcLineCount":183,
  "PercentageCoveredContribution":100,
  "PercentageUncoveredContribution":100,
  "TotalBranches":46,
  "TotalChildren":3,
  "TotalClasses":3,
  "TotalElements":172,
  "TotalFiles":3,
  "TotalMethods":31,
  "TotalPackages":0,
  "TotalStatements":95,
  "UncoveredBranches":7,
  "UncoveredMethods":0,
  "UncoveredStatements":5
  // ... plus attributes as for a source file
}
} );

```

### Format of a project summary (*project.js*)

```

processClover ( {
  "children":["com/cenqua/samples/money/"], // list of
packages
  "name":"Clover database Pt gru 21 2012 09:22:03 CET", // name of
the report
  "stats": { // statistics
  // ... attributes as for a package
}
} );

```

### Format of top risks (*proj-risks.js*, *pkg-risks.js*, *aggregate-pkg-risks.js*)

```
processClover ( {
  "axis": { // X-Y axis description
    "x": {
      "max":211,"min":128,"title":"Average Method Complexity"
    },
    "y": {
      "max":10,"min":5,"title":"Coverage"
    }
  },
  "classes":[ // values for a chart

{"name":"Money","path":"com/cenqua/samples/money/Money","x":128,"y":10},

{"name":"MoneyBag","path":"com/cenqua/samples/money/MoneyBag","x":211,"y":5}]
} );
```

#### Format of quick wins (*quick-wins.js*, *aggregate-quick-wins.js*)

```
processClover ( {
  "axis": { // X-Y axis description
    "x": {
      "max":125,"min":47,"title":"# Elements"
    },
    "y": {
      "max":7,"min":5,"title":"# Elements Untested"
    }
  },
  "classes":[ // values for a chart

{"name":"Money","path":"com/cenqua/samples/money/Money","x":47,"y":5},

{"name":"MoneyBag","path":"com/cenqua/samples/money/MoneyBag","x":125,"y":7}]
} );
```

Historical report

Code snippet

```

<clover-report>
  <historical outfile="${clover.report}" historydir="${historydir}" json="true">
<!-- json="true" generates historical-json.js file -->
  <format type="html"/> <!-- Note that historical report does not handle format
type="json" -->
  <!-- ... -->
</historical>
</clover-report>

```

### Format of historical-json.js file

```

processHistoricalCloverData ( {
  "name": "money_demo", // name of the project
  "table": [ {
    "cols": [ // list of available metrics and their data format
      {
        "id": "timestamp",
        "label": "Date",
        "type": "date"
      },
      {
        "id": "FilteredElements",
        "label": "% Filtered",
        "type": "number"
      },
      // ...
    ],
    "rows": [ // data
      { "c": [ {
        "f": "21.12.12 09:02",
        "v": new Date(1356076936171)
      }, {
        "f": "0%",
        "v": 0
      },
      // ...
    ] }
  ]
} );

```

### Examples of JSON report usage

#### JSON for a file page:

```

processClover ( {
  "lines": [ "", "", "10", "10", "10", "", "", "16", "1", "1", "1", "", "", "", "",
"22", "22", "5678", "5678", "1", "1"],
  "stats": {
    "Complexity": 22,
    "TotalPercentageCovered": 100,
    "CoveredElements": 63,
    "UncoveredElements": 0
  },
  "id": "org_apache_commons_codec_net_BCodec_java"
}
);

```

The "lines" array contains the hit counts for each line in the BCodec.java file.

### JSON in an HTML page:

```

<!-- Define the callback function before including the java.js for the file you
wish to process. -->
  <script type="text/javascript">
    function processClover(obj) {
      alert(obj.id);
      alert(obj.stats.CoveredElements);
      alert(obj.stats.Complexity);
      alert(obj.lines);
    }
  </script>
<!-- Now, include as many java.js files as you wish. Each will call your
"processClover" callback function above. -->
  <script type="text/javascript"

src="http://downloads.atlassian.com/software/clover/samples/codec/org/apache/common
s/codec/StringEncoderAbstractTest.java.js">
  </script>
  <script type="text/javascript"

src="http://downloads.atlassian.com/software/clover/samples/codec/org/apache/common
s/codec/net/BCode.java.js">
  </script>

```

### JSON live demo

This example uses JavaScript alerts to display the JSON data. [Click here](#) to run the live demo.

### Clover Historical Data in JSONP

You can also access Clover historical data using JSON or [JSONP](#). If your coverage data is available online, you could view your coverage inside a Google Gadget.

Using this data, you can create your own visualisations in Javascript or HTML. The data from Clover is compatible with the [Google Visualisation API](#).

### Basic example:



```
table: {
  cols: [{id: 'timestamp', label: 'Date', type: 'date'}, ...],
  rows: [{c:[{v: new Date(0), f: '1 January 1970 00:00'}, ...]}]
}
```

This is the basic JSON object produced by Clover. The above JSON object may be passed directly to a `google.visualization.DataTable` constructor like so:

```
new google.visualization.DataTable(json.table[0], 0.5);
```

#### Example for use with Javascript:

```
var data = new google.visualization.DataTable(json.table[0], 0.5);
var chart = new
google.visualization.AreaChart(document.getElementById('chart_div'));
chart.draw(data, {width: 800, height: 400, legend: 'bottom', title:
'Clover Historical Chart'});
```

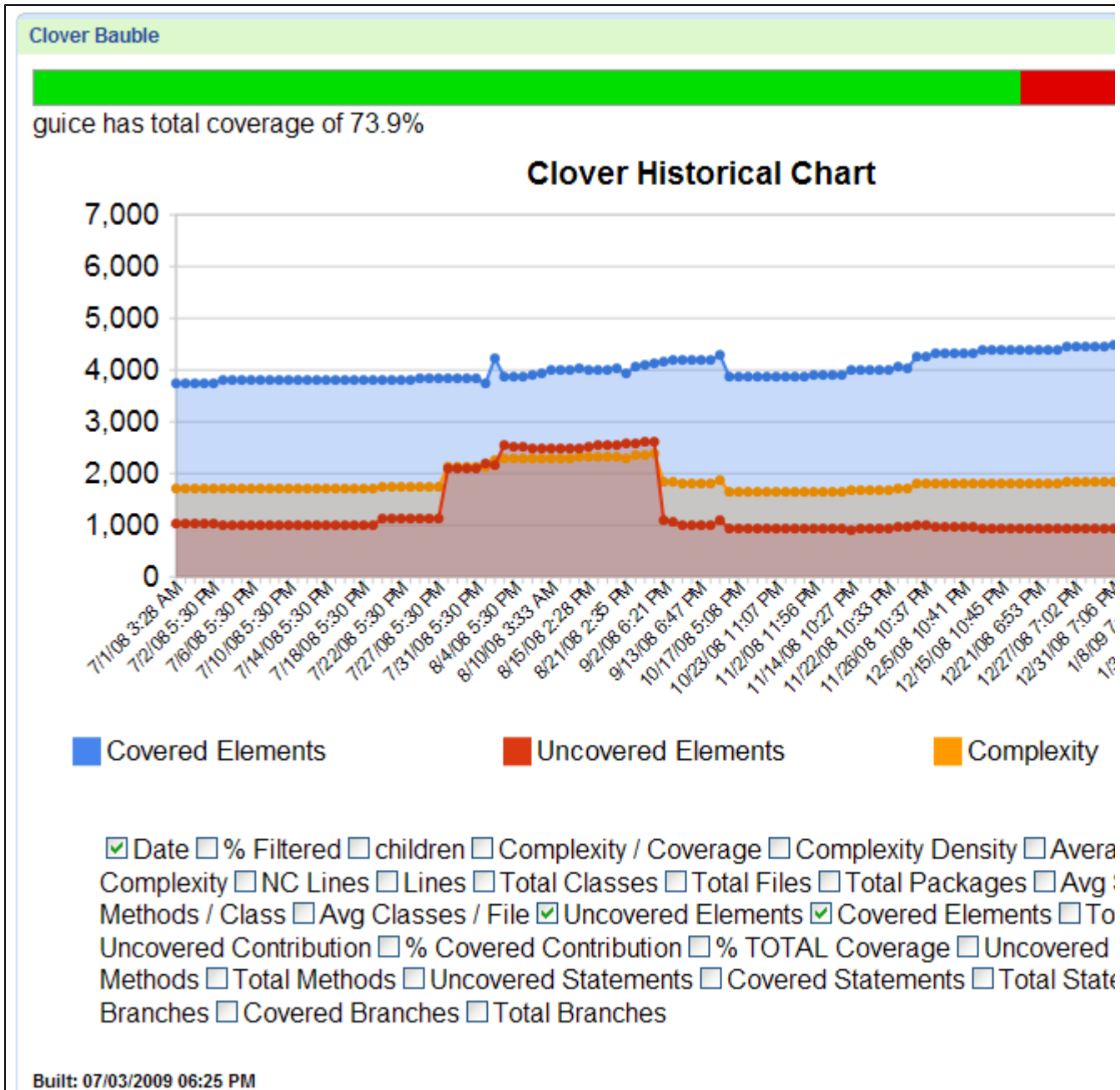
#### Example Clover Gadget:

The screenshot below shows Clover historical data in a Gadget:

The XML source for this gadget is as follows:

```
http://labs.atlassian.com/gadgets/npellow/src/clover-gadget.xml
```

*Screenshot: Clover in a Google Gadget*



#### Basic Clover Confluence Integration

#### Embedding Clover Data into a Confluence Page

There is currently no native support for embedding Clover in Confluence. Many use cases can be achieved however by using the `{html}` macro.

The following are examples of displaying live Clover data directly in Confluence.

#### Clover Dashboard

```
{html}
<iframe
src="http://downloads.atlassian.com/software/clover/samples/lucene/dashboard.html"
width="100%" height="600px"/>
{html}
```

**Project Risks Coverage Cloud**

```
{html}
<iframe
src="http://downloads.atlassian.com/software/clover/samples/lucene/proj-risks.html"
width="100%" height="600px"/>
{html}
```

**clover-setup**

On this page:

- [Description](#)
- [Parameters](#)
- [Nested Elements](#)
  - [<distributedCoverage>](#)
  - [<files>](#)
  - [<fileset>](#)
  - [<methodContext>](#)
  - [<statementContext>](#)
  - [<profiles>](#)
  - [<testsources>](#)
- [Examples](#)
- [Troubleshooting](#)

**Description**

The `<clover-setup>` task initialises Clover for use with your project.








The basic nesting of elements within the `<clover-setup>` task is as follows:








```
<clover-setup>
  <distributedCoverage/>
  <files/>
  <fileset/>
  <methodcontext/>
  <statementcontext/>
  <profiles/>
  <testsources> advanced users only
    <testclass>
      <testmethod/>
    </testclass>
  </testsources>
</clover-setup>
```

**Parameters**


All attributes of the `<clover-setup>` element support Java. However, as indicated in the last column of the following table, some of these attributes do not support Groovy.

Attribute	Description	Required	Groovy Support

clovercompiler	After instrumentation, Clover hands off compilation to the standard Ant compiler adapter (or the compiler specified by the <code>build.compiler</code> Ant property). This attribute specifies the adapter to use. It takes the same values as the standard Ant <code>build.compiler</code> property. If you wish to specify an alternative compiler, you can either set the <code>build.compiler</code> property or use this attribute.	No.	
enabled	This controls whether Clover will instrument code during code compilation. This attribute provides a convenient control point to enable or disable Clover from the command line.	No; defaults to 'true'.	
flushinterval	When the flushpolicy is set to <code>interval</code> or <code>threaded</code> this value is the minimum period between flush operations (in milliseconds)	No.	
flushpolicy	<p>This attribute controls how Clover flushes coverage data during a test run. Valid values are <code>directed</code>, <code>interval</code>, or <code>threaded</code>.</p> <p><code>directed</code> — Coverage data is flushed at JVM shutdown, and after an inline flush directive.</p> <p><code>interval</code> — Coverage data is flushed as for <code>directed</code>, as well as periodically at a <i>maximum rate</i> based on the value of <code>flushinterval</code>. This is a 'passive' mode in that flushing potentially occurs as long as instrumented code is being executed.</p> <p><code>threaded</code> — Coverage data is flushed as for <code>directed</code>, as well as periodically at a rate based on the value of <code>flushinterval</code>. This is an 'active' mode in that flushing occurs on a separate thread and is not dependent on the execution of instrumented code.</p> <p>For more information, see <a href="#">Using a Flush Policy</a>.</p>	No; defaults to 'directed'.	
fullyQualifyJavaLang	This should only be set to 'false' if you have defined a variable called 'java' in your source files. If false, Clover will instrument source files without using fully qualified <code>java.lang</code> names.	No; defaults to 'true'.	
initstring	The Clover <code>initString</code> describes the location of the <a href="#">Clover coverage database</a> . Typically this is a relative or absolute file reference, e.g. <code>\${basedir}/build/clover.db</code> . If not specified it defaults to <code>.clover/clover.db</code> , relative to the project's base directory.	No.	
instrumentationLevel	This setting can reduce accuracy to method level, to enhance the speed of instrumentation, compilation & test execution. Valid values are 'method' and 'statement'.	No; defaults to <code>statement</code> .	

instrumentLambda	<p><b>Since 3.2.2.</b> Whether Java 8 lambda functions shall be instrumented. If instrumented, they're treated like normal methods (and can be shown in HTML report and considered in code metrics, for example). Possible values:</p> <ul style="list-style-type: none"> <li>• <i>none</i> - do not instrument lambda functions,</li> <li>• <i>expression</i> - instrument lambdas in expression-like form, e.g. "(a, b) -&gt; a + b",</li> <li>• <i>block</i> - instrument lambdas in code blocks, e.g. "(a, b) -&gt; { return a + b; }"</li> <li>• <i>all</i> - instrument all lambda functions.</li> </ul> <p> Due to Clover's restrictions related with code instrumentation and javac compiler's type inference capabilities, you may get compilation errors when expression-like lambda functions are passed to generic methods or types. In such case disable instrumentation of expression-like form (i.e. use the <i>none</i> or <i>block</i> setting). See the <a href="#">Java 8 code instrumented by Clover fails to compile</a> Know ledge Base article for more details.</p>	No; defaults to "all".	
preserve	A boolean attribute which controls whether the instrumented source will be retained after compilation.	No; defaults to 'false'.	
recordTestResults	If set to 'false', test results will not be recorded; instead, results can be added via the <testResults> fileset at report time. For more details please see <a href="#">Advanced Usage</a> .	No; defaults to 'true'.	
relative	This controls whether the initstring parameter is treated as a relative path or not.	No; defaults to 'false'.	
source	The default source level to process source files at. Note that setting the source attribute on the <javac> target will override this setting.	No.	
tmpdir	The directory into which Clover will write an instrumented copy of the source code.	No.	

It is important to note that the Clover compiler adapter still picks up its settings from the set of Clover Ant properties. The <clover-setup> task provides a convenient method to set these properties. This means that builds that use the Clover 1.0 property set will continue to operate as expected.

 Do not set the 'compiler' attribute on the <javac/> task as this overrides the Clover compiler set up by <lover-setup>. Use the 'clovercompiler' attribute instead.

## Nested Elements

### <distributedCoverage>

This element, which supports both Java and Groovy, turns on Clover's distributed coverage feature, enabling the collection of per-test coverage data, when your test environment requires more than one JVM ([Java Virtual Machine](#)).

#### Parameters

Attribute name	Description	Required
name	The name of this configuration.	No; defaults to 'tcp-config'
port	The port the test JVM should listen on.	No; defaults to '1198'

host	The hostname the test JVM should bind to.	No; defaults to 'localhost'
timeout	(a number) The amount of time (in milliseconds) to wait before a connection attempt will fail.	No; defaults to '5000'
numClients	(a number) The number of clients that need to connect to the test server before the tests will continue.	No; defaults to '0'
retryPeriod	(a number) The amount of time (in milliseconds) to wait before attempting to reconnect in the event of a network failure.	No; defaults to '1000'

 All attributes are optional.

#### <files>

An Ant patternset, relative to the top level package (e.g. com/atlassian/clovertest), element which controls which files are included or excluded from Clover instrumentation. Use this when you wish to exclude files based on packages.



#### Note

The <useclass> sub-element has been deprecated and has no effect.

#### <fileset>

As of Clover 1.2, <clover-setup> also supports multiple Ant <filesets>. These give greater flexibility in specifying which source files are to be instrumented by Clover. This is useful when you have more than one source base and only want some of those source bases to be instrumented. This can be difficult to setup with patterns. Filesets also allow much greater flexibility in specifying which files to instrument by facilitating the use of Ant's fileset selectors. Use this when you wish to exclude files based on directory structure.



The <fileset> tag has a "dir" attribute and uses standard Ant directory scanner to find sources for exclusion. It means that at the time when <clover-setup> is called the directory must be present and files to be excluded/included too.

#### <methodContext>

Specifies a method Context definition. See [Using Coverage Contexts](#) for more information.

#### Parameters

Attribute	Description	Required
name	The name for this context. Must be unique, and not be one of the reserved context names (see <a href="#">Using Coverage Contexts</a> ).	Yes.
regex	A Perl 5 Regexp that defines the context. This regexp should match the method signatures of methods you wish to include in this context. Note that when method signatures are tested against this regexp, whitespace is normalised and comments are ignored.	Yes.
maxComplexity	Match a method to this pattern if its cyclomatic complexity is not greater than maxComplexity. In other words - all methods with complexity <= maxComplexity will be filtered out.	No.
maxStatements	Match a method to this pattern if its number of statements is not greater than maxStatements. In other words - all methods with statements <= maxStatements will be filtered out.	No.

maxAggregatedComplexity	<b>Since 3.1.10.</b> Match a method to this pattern if its aggregated cyclomatic complexity is not greater than maxAggregatedComplexity. In other words - all methods with aggregated complexity $\leq$ maxAggregatedComplexity will be filtered out. Aggregated complexity metric is a sum of the method complexity and complexity of all anonymous inline classes declared in the method.	No.
maxAggregatedStatements	<b>Since 3.1.10.</b> Match a method to this pattern if its number of aggregated statements is not greater than maxAggregatedStatements. In other words - all methods with aggregated statements $\leq$ maxAggregatedStatements will be filtered out. Aggregated statements metric is a sum of the method statements and statements of all anonymous inline classes declared in the method.	No.

### What is the difference between maxComplexity and maxAggregatedComplexity or maxStatements and maxAggregatedStatements?

Aggregated metrics calculate method statements/complexity including the code of all anonymous inline classes declared inside the method. Thanks to this, it is possible to distinguish between a trivial single-statement method like:

```
int getNumber() {
    return number;
}
```

and a single-statement method which actually returns more complex data, like:

```
ActionListener getListener() {
    return new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            System.out.println("statement #1");
            System.out.println("statement #2");
            System.out.println("statement #3");
        }
    };
}
```

If you would use a method context filter with *maxStatements* attribute, like the following:

```
<methodContext name="trivial" regexp=".*" maxStatements="1">
```

then both *getNumber()* and *getListener()* methods would be filtered-out, because each of them contains only one statement: "return <xxx>".


If you would use new *maxAggregatedStatements*, for instance:

```
<methodContext name="trivial" regexp=".*" maxAggregatedStatements="1">
```

then the *getNumber()* would be filtered-out and the *getListener()* method would **not** be filtered out (because it

contains 4 statements in total - 1 "return" statement from the method itself and 3 "System.out.println()" statements from anonymous class).

### Regular expression tip:

 If you would like to filter-out all methods, except those having a specific name, you could write a negative-look-ahead regular expression. For example:

```
<methodContext name="trivial" regexp="^(?!.*(getRunnable|getListener)).*$"
maxStatements="1"/>
```

will filter-out all methods having not more than one statement, except those which are named *getRunnable* or *getListener*.

### <statementContext>

Specifies a statement Context definition. See [Using Coverage Contexts](#) for more information.

 This element does not support Groovy.

#### Parameters

Attribute	Description	Required
name	The name for this context. Must be unique, and not be one of the reserved context names (see <a href="#">Using Coverage Contexts</a> ).	Yes.
regexp	A Perl 5 Regexp that defines the context. This regexp should match statements you wish to include in this context. Note that when statements are tested against this regexp, whitespace is normalised and comments are ignored.	Yes.

### <profiles>

**Since 3.1.11.** Optional element. Defines a list of Clover profiles, which can be selected at runtime by providing a **clover.profile=<name>** system property. Thanks to this you can change some of Clover's behaviour without code recompilation.

```
<profiles>
  <profile name="default" coverageRecorder="FIXED|GROWABLE|SHARED">
    <distributedCoverage/> <!-- optional -->
  </profile>
  <profile .../>
  <!-- more profiles -->
</profiles>
```

### <profile>

**Since 3.1.11.** Contains a definition of a single runtime profile.



**Parameters**

Attribute	Description	Required
name	The name for this profile; name must be unique among profiles. There must be one profile named "default".	No. Defaults to "default".
coverageRecorder	Type of coverage recorder which will be used for gathering coverage data at runtime. Possible values: FIXED, GROWABLE, SHARED (case insensitive).  ⚠ Warning: we strongly recommend using the default setting. Do not change until you deeply understand <a href="#">how it works</a> .	No. Defaults to "FIXED".

**Nested elements****<distributedCoverage/>**

📌 Note: a definition in **<profile>/<distributedCoverage>** element has priority over the **<clover-setup/clover-instr>/<distributedCoverage>** element.

**Selecting clover.profile at runtime**

Clover profile is being selected at runtime using the following algorithm:

- Are there any profiles defined in compiled code?
  - **yes** -
    - 1. read the **clover.profile** system property. is it defined?
      - **yes** - use the value as profile name
      - **no** - use the "default" profile name
    - 2. is the profile name found on list of defined profiles?
      - **yes** - **use settings from this profile**
      - **no** - **use system settings** (default coverage recorder etc...)
  - **no** - **use system settings** (default coverage recorder etc...)

So it fall-backs to default system settings in case of missing profile.

**<testsources>**

**<testsources>** is an [Ant fileset](#) which should only be used if Clover's [default test detection](#) is not adequate. Clover's default test detection algorithm is used to distinguish test cases if this element is omitted.

⚠ To have test sources reported in a separate tree to your application code, use the **<testsources/>** element in the **<clover-report/>** task.

**Nested elements of <testsources>****<testclass>**

**<testclass>** can be used to include only specific test classes.

**Parameters**

Attribute	Description	Required
name	A regex on which to match the test class's name.	No.
super	A regex on which to match the test class's superclass.	No.
annotation	A regex on which to match the test class's annotation.	No.

package	A regex on which to match the test class's package.	No.
tag	A regex on which to match the test class's javadoc tags.	No.

**i** For more information about regular expressions, please visit <http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html#sum>.

<and>

<and> can be used to specify multiple instances of <testclass>, all of which must be matched for a class to be detected as a test, e.g.:

```
<testsources dir="tests">
<and>
  <testclass annotation="Specification" />
  <testclass annotation="Test" />
</and>
</testsources>
```

In this example, a class will only be recognised as a test if it has "Specification" **and** "Test" annotations.

<or>

<or> can be used to specify multiple instances of <testclass>, any of which must be matched for a class to be detected as a test, e.g.:

```
<testsources dir="tests">
<or>
  <testclass name=".*Spec" />
  <testclass name=".*Test" />
</or>
</testsources>
```

In this example, a class will be recognised as a test if its name matches ". \*Spec", **or** its name matches ". \*Test".

#### **Nested elements of <testclass>**

<testmethod>

<testmethod> can be used to perform more fine grained detection of test methods.

**!** Clover matches methods only; it does not match constructors (CLOV-1339).

Parameters

Attribute	Description	Required
name	A regex on which to match the test method's name.	No.
annotation	A regex on which to match the test method's annotation.	No.
tag	A regex on which to match the test method's javadoc tags.	No.
returntype	A regex on which to match the return type of the method, e.g.: <ul style="list-style-type: none"> <li>".*" will match any return type.</li> <li>"void" will match methods with no return type.</li> </ul>	No.

Note that you can include multiple instances of <testmethod>, in which case they will be treated as 'or' clauses, e.g.:

```
<testsources dir="tests">
  <testclass>
    <testmethod annotation="Specification"/>
    <testmethod name="^should.*"/>
    <testmethod name="^must.*"/>
  </testclass>
</testsources>
```

In this example, a method will be recognised as a test if its annotation is "Specification", **or** its name matches "^should\*", **or** its name matches "^must\*".

## Examples

```
<clover-setup/>
```

This example is the minimal setup to use Clover. In this case, the [Clover coverage database](#) is located in the .clover relative directory.

```
<clover-setup enabled="${enable}">
  <files>
    <exclude name="**/atlassian/clover/**/*.*.java"/>
  </files>
</clover-setup>
```

This example shows the use of a property, "enable", to control whether Clover instrumentation is enabled. Additionally, the instrumentation will exclude all Java source files in trees belonging to the com.atlassian.clover.\* packages (please note, that even if the files belong in the src/main or src/test directory, you cannot specify src, main or test as these are directories and do not belong to the package structure. When using files, you need to filter by files or the packages as in the example above). Note that the fileset can also be referenced using a refid attribute.

```
<clover-setup enabled="${coverage.enable}">
  <fileset dir="src/main">
    <contains text="Joe Bloggs"/>
  </fileset>
</clover-setup>
```

This example instruments all source files in the src/main directory tree that contain the string "Joe Bloggs". Ant's filesets supports a number of these selectors. Please refer to the Ant manual for information on these selectors.

### Interval Flushing

By default Clover will write coverage data to disk when the hosting JVM exits, via a shutdown hook. This is not always practical, particularly when the application you are testing runs in an Application Server. In this situation, you can configure Clover to use "interval" flushing, where coverage data is written out periodically during execution:

```
<clover-setup flushpolicy="interval"
  flushinterval="5000"/>
```

The "flushinterval" defines in milliseconds the minimum interval between coverage data writes.

### Specifying a delegate compiler

Clover provides the optional "clovercompiler" attribute to allow specification of the java compiler to delegate to once instrumentation is completed. The attribute accepts the same values "compiler" attribute of the [Ant Javac Task](#).

```
<clover-setup clovercompiler="jikes"/>
```

This example will pass compilation to the "jikes" compiler once instrumentation is complete.

#### Specifying the location of the Clover Coverage database

By default, Clover writes its [internal database](#) to the `.clover/clover.db` file relative to the project's base directory. To override this location, use the `initstring` attribute, e.g.:

```
<clover-setup initstring="clover-db/coverage.db" />
```

This example will use `clover-db/coverage.db` as the location for the [Clover database](#). Note that the directory `clover-db` should exist before running this task.

#### Specifying a custom test matcher

By default, Clover attempts to detect your test classes and methods. Clover's default behaviour may be overridden via the following:

```
<clover-setup>
  <testsources dir="src">
    <include name="**/*Test.java"/>
    <testclass name=".*Test">
      <testmethod name=".*Bag.*"/> <!-- only the Bag related tests -->
    </testclass>
  </testsources>
</clover-setup>
```

This example tells Clover to recognise all of the following as tests: classes in the directory "src"; classes in files whose names end with "Test"; methods whose names contain with "Bag".

## Troubleshooting

### Clover does not support parallel compilation

You cannot use `<parallel/>` task for code compilation, for instance:

```
<target name="init">
  <clover-setup/>
</target>

<target name="compile" depends="init">
  <parallel>
    <javac srcdir="module1" .../>
    <javac srcdir="module2" .../>
  </parallel>
</target>
```

will produce error message like:

```
[clover] Error finalising instrumentation:
[clover] java.io.IOException: Failed to move tmp registry file
/myproject/.clover/clover3_1_6.db.tmp to final registry file
```

## Clover test detection

### Detection of test methods

Clover is able to detect test methods for following test frameworks and code patterns.

#### JavaDoc tags

This approach can be used for Java 1.4, which does not support annotations. In such case, test methods can be marked using JavaDoc tags:

#### TestNG style

```
/** @testng.test */
class MyTest {
    /** @testng.test */
    void myTestMethod() { }
}
```

#### JUnit style

```
/** @test */
void myTestMethod() { }
```

#### JUnit3

Methods with a following signature:

```
public void test***()
```

#### JUnit4

Methods annotated with one of the following:

```
@junit.org.Test(expected={Foo.class})
@junit.org.Test(expected=Foo.class)
@Test(expected=Foo.class)
```

#### JUnit4+Spring

Methods annotated with:

```
@Test
@org.springframework.test.annotation.ExpectedException({Bar.class})
```

or

```
@Test
@ExpectedException(value={Bar.class})
```

#### TestNG

Methods annotated with one of the following:

```
@org.testng.annotations.ExpectedExceptions(Foo.class)
@ExpectedExceptions(org.bar.Foo.class)
```

```
@org.testng.annotations.Test(expectedExceptions={Foo.class})
@Test(expectedExceptions={Foo.class})
@org.testng.annotations.Test(expectedExceptions=Foo.class)
@Test(expectedExceptions=Foo.class)
```

**Instinct**

Methods annotated with one of the following:

```
@com.googlecode.instinct.marker.annotate.Specification(expectedException=Foo.class)
@Specification(expectedException=Foo.class)
```

**Usage context**

These patterns are being used by Clover for per-test coverage, test optimization and reporting.

**methodContext****<methodContext>**

Specifies a method Context definition. See [Using Coverage Contexts](#) for more information.

Parameters

Attribute	Description	Required
name	The name for this context. Must be unique, and not be one of the reserved context names (see <a href="#">Using Coverage Contexts</a> ).	Yes.
regex	A Perl 5 Regexp that defines the context. This regexp should match the method signatures of methods you wish to include in this context. Note that when method signatures are tested against this regexp, whitespace is normalised and comments are ignored.	Yes.
maxComplexity	Match a method to this pattern if its cyclomatic complexity is not greater than maxComplexity. In other words - all methods with complexity <= maxComplexity will be filtered out.	No.
maxStatements	Match a method to this pattern if its number of statements is not greater than maxStatements. In other words - all methods with statements <= maxStaments will be filtered out.	No.
maxAggregatedComplexity	<b>Since 3.1.10.</b> Match a method to this pattern if its aggregated cyclomatic complexity is not greater than maxAggregatedComplexity. In other words - all methods with aggregated complexity <= maxAggregatedComplexity will be filtered out. Aggregated complexity metric is a sum of the method complexity and complexity of all anonymous inline classes declared in the method.	No.
maxAggregatedStatements	<b>Since 3.1.10.</b> Match a method to this pattern if its number of aggregated statements is not greater than maxAggregatedStatements. In other words - all methods with aggregated statements <= maxAggregatedStaments will be filtered out. Aggregated statements metric is a sum of the method statements and statements of all anonymous inline classes declared in the method.	No.

**What is the difference between maxComplexity and maxAggregatedComplexity or maxStatements and**

### maxAggregatedStatements?

Aggregated metrics calculate method statements/complexity including the code of all anonymous inline classes declared inside the method. Thanks to this, it is possible to distinguish between a trivial single-statement method like:

```
int getNumber() {
    return number;
}
```

and a single-statement method which actually returns more complex data, like:

```
ActionListener getListener() {
    return new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            System.out.println("statement #1");
            System.out.println("statement #2");
            System.out.println("statement #3");
        }
    };
}
```

If you would use a method context filter with *maxStatements* attribute, like the following:

```
<methodContext name="trivial" regexp=".*" maxStatements="1">
```


then both *getNumber()* and *getListener()* methods would be filtered-out, because each of them contains only one statement: "return <xxx>".

If you would use new *maxAggregatedStatements*, for instance:

```
<methodContext name="trivial" regexp=".*" maxAggregatedStatements="1">
```

then the *getNumber()* would be filtered-out and the *getListener()* method would **not** be filtered out (because it contains 4 statements in total - 1 "return" statement from the method itself and 3 "System.out.println()" statements from anonymous class).

### Regular expression tip:

 If you would like to filter-out all methods, except those having a specific name, you could write a negative-look-ahead regular expression. For example:

```
<methodContext name="trivial" regexp="^(?!.*(getRunnable|getListener)).*$"
maxStatements="1"/>
```

will filter-out all methods having not more than one statement, except those which are named *getRunnable* or *getListener*.

### profiles

**<profiles>**

**Since 3.1.11.** Optional element. Defines a list of Clover profiles, which can be selected at runtime by providing a **clover.profile=<name>** system property. Thanks to this you can change some of Clover's behaviour without code recompilation.

```
<profiles>
  <profile name="default" coverageRecorder="FIXED|GROWABLE|SHARED">
    <distributedCoverage/> <!-- optional -->
  </profile>
  <profile .../>
  <!-- more profiles -->
</profiles>
```

**<profile>**

**Since 3.1.11.** Contains a definition of a single runtime profile.

Parameters

Attribute	Description	Required
name	The name for this profile; name must be unique among profiles. There must be one profile named "default".	No. Defaults to "default".
coverageRecorder	Type of coverage recorder which will be used for gathering coverage data at runtime. Possible values: FIXED, GROWABLE, SHARED (case insensitive).  ⚠ Warning: we strongly recommend using the default setting. Do not change until you deeply understand <a href="#">how it works</a> .	No. Defaults to "FIXED".

Nested elements

**<distributedCoverage/>**

📌 Note: a definition in **<profile>/<distributedCoverage>** element has priority over the **<clover-setup/clover-instr>/<distributedCoverage>** element.

Selecting clover.profile at runtime

Clover profile is being selected at runtime using the following algorithm:

- Are there any profiles defined in compiled code?
  - **yes** -
    - 1. read the **clover.profile** system property. is it defined?
      - **yes** - use the value as profile name
      - **no** - use the "default" profile name
    - 2. is the profile name found on list of defined profiles?
      - **yes** - **use settings from this profile**
      - **no** - **use system settings** (default coverage recorder etc...)
  - **no** - **use system settings** (default coverage recorder etc...)

So it fall-backs to default system settings in case of missing profile.

**statementContext****<statementContext>**

Specifies a statement Context definition. See [Using Coverage Contexts](#) for more information.



! This element does not support Groovy.

#### Parameters

Attribute	Description	Required
name	The name for this context. Must be unique, and not be one of the reserved context names (see <a href="#">Using Coverage Contexts</a> ).	Yes.
regexp	A Perl 5 Regexp that defines the context. This regexp should match statements you wish to include in this context. Note that when statements are tested against this regexp, whitespace is normalised and comments are ignored.	Yes.

## clover-snapshot

### Description

The `<clover-snapshot>` task generates a snapshot file used to assist Clover in optimizing the tests run in subsequent. This task should be run at the end of a build (i.e. after all unit tests have run).

### Parameters

Attribute	Description	Required
file	Specifies an alternative location for the snapshot file.	No; defaults to the <code>initstring + ".snapshot"</code> .
span	Clover interval (e.g. "2m" or "3h"); specifies, for the initial build cycle, the span for coverage used in generating the snapshot file used for test optimization; normally only specified when multiple interleaved compiles & test runs happen during a build cycle	No; defaults to "0s" (zero seconds).

### Examples

```
<clover-snapshot />
```

Generates a snapshot file used to assist Clover.

```
<clover-snapshot file="C:\My Documents\clover.snapshot" />
```

Specifies an alternative location of "C:\My Documents\clover.snapshot" for the snapshot file.

```
<clover-snapshot span="3m" />
```

Defines a custom span of "3m" (three minutes) used in generating the snapshot file used for Test Optimization.

## 7. Ant Type Reference

### Clover Ant Types

<code>&lt;clover-format&gt;</code>	Creates standalone format elements which can then be used across a number of reports.
------------------------------------	---

<code>&lt;clover-columns&gt;</code>	Defines a set of custom columns to be used by tasks which take a <code>&lt;columns/&gt;</code> element, such as historical charts, JSON report, and HTML current report.
<code>&lt;clover-optimized-tests et&gt;</code>	Test optimization for the <code>&lt;junit&gt;</code> task.

## clover-columns

### Description

Defines a set of custom columns to be used by tasks which take a `<columns/>` element, such as historical charts, JSON report, and HTML current report.

**Example: Creating a custom chart with `<clover-columns>`**

```
<clover-columns id="my.columns">
  <totalChildren/>
  <avgMethodComplexity/>
  <totalPercentageCovered format="bar"/>
</clover-columns>
```

**Example: Using the custom chart elsewhere**

```
<clover-report>
  <current>
    <columns refid="my.columns"/>
  </current>
</clover-report>
```

## clover-format

### Description

The `<clover-format>` type creates standalone format elements which can then be used across a number of reports.

These standalone types support the same attributes and elements as the internal `<format>` elements of the `<clover-report>` task. To name the format, use the standard Ant "id" attribute.

### Parameters

### Examples

See [Sharing Report Formats](#) for some usage examples.

## clover-optimized-testset

### Description

The `<clover-optimized-testset>` type is designed to be used within JUnit and Ant's `<batchtest/>` task (which is used to feed JUnit a list of .java files that map to the test classes to be run). This type accepts other resource collection elements (e.g. `<fileset/>`), filters and reorders their resources for Test Optimization. To use this type, you must first include `cloverlib.xml`.

### Parameters

Attribute	Description	Required
<b>snapshotfile</b>	Snapshot file path if not in the default location.	No; defaults to (initstring + ".snapshot").
<b>enabled</b>	Boolean value that specifies whether the selector should optimize. If set to false no optimization occurs, if true, then optimization occurs as per the other attributes.	No; defaults to "true".
<b>minimize</b>	Boolean value that (when true) tells Clover to restrict the set of tests, running only those that caused coverage in a changed file. When set to false, this will cause Clover to run all tests. this attribute can be used to force a full test run if, for instance, some important configuration files change and the build system decides that a full test run should be executed	No; defaults to "true".
<b>fullrunevery</b>	Specifies how many optimized builds can run before a full run should be performed (to re-calibrate the optimization).	No; defaults to "10".
<b>ordering</b>	Specifies how tests should be ordered during an optimized build. Accepts values "failfast", "original" or "random". The "failfast" setting runs the previously failed test first then shortest to longest test. The "original" setting orders the tests as they were found by the underlying fileset. The "random" setting applies a random ordering to the tests.	No; defaults to "failfast".

**Elements:**

`<*/>` - any [Ant resource collection](#): anything that implements `org.apache.tools.ant.types.ResourceCollection`.

**Example**

```
<junit ...>
  <batchtest todir="${outdir}/${testresultsprefix}" fork="true">
    <clover-optimized-testset fullrunevery="${max.optimized.builds}">
      <fileset dir="${test.location}" includes="**/*Test.java"/>
    </clover-optimized-testset>
    <formatter type="xml"/>
    <formatter type="plain"/>
  </batchtest>
</junit>
```

## 8. Controlling Clover at Runtime

- [Clover Performance Tuning](#)
- [Coverage Recorders](#)
- [Managing the Coverage Database](#)
- [Using a Flush Policy](#)
- [Using Source Directives](#)
- [Working with Distributed Applications](#)
- [Working with Restricted Security Environments](#)

### Clover Performance Tuning

*This page contains instructions on how to tune Clover's performance when running your builds and measuring code coverage.*

On this page:

- [Tips for improving performance](#)
  - [Configure 'Unique per-test coverage' tracking in HTML report](#)
  - [Set Instrumentation to "method level" \(when using Test Optimization\)](#)

- [Select per-test coverage recording strategy](#)
- [Related Links](#)
- [Appendix - sample performance data](#)
  - [Comparison of statement and method instrumentation level](#)
  - [Comparison of different per-test coverage recording strategies](#)

## Tips for improving performance

### Configure 'Unique per-test coverage' tracking in HTML report

Unique coverage relates to a line of code that was hit by only one test. Unique coverage tracking can be switched off to reduce CPU & memory usage when running Clover. You can configure unique coverage reporting in the following Clover components:

- [Clover Command-Line Interface HTML Reporter](#)
- [The HTML report in Clover-for-Ant](#)
- [The Current report in Clover-for-Ant](#)

### Set Instrumentation to "method level" (when using Test Optimization)

If you use Clover in your build purely for Test Optimization purposes and not for coverage reporting, you can reduce the granularity of Clover instrumentation from statement to method level. The 'instrumentationLevel' attribute set to method level allows for speedier instrumentation, compilation & test execution.

This speeds up the build at the loss of some accuracy. This is the setting to use if you want to improve Clover's performance. When this attribute is set to '**statement**' (the default), the builds will take longer but the optimization intelligence will also be stronger.

You can configure instrumentation level in the following Clover-for-Ant tasks:

- [clover-setup](#)
- [clover-instr](#) (Clover instrumentation)

See the [Ant Task Reference](#) for more information.

### Select per-test coverage recording strategy

During your test runs, Clover tries to record total code coverage and per-test code coverage as efficiently as possible but defaults to settings best for applications which are not highly CPU intensive. If your application is highly CPU intensive and code coverage recording is causing slow running tests, the following options may assist:

- Supply this option to the JVM running your tests:

```
-Dclover.pertest.coverage=diff
```

This changes the way per-test coverage is recorded at runtime to work faster for CPU intensive applications.

- Supply this option to the JVM running your tests:

```
-Dclover.pertest.coverage=off
```

This tells Clover to not record any per-test coverage data at runtime. With this you gain a faster running time for CPU intensive applications, although you lose per-test coverage information.

- If you fork your unit tests, this must be passed to the forked JVM as a command line argument in Ant, Maven or the Eclipse or IDEA IntelliJ unit test launchers through their respective dialogs; if you don't fork your tests, this must be supplied to Ant through the ANT\_OPTS environment variable or to Maven through the MAVEN\_OPTS variable.

### Related Links

- Performance Tuning in Clover for Eclipse

## Appendix - sample performance data

This appendix contains few sample performance results based on a synthetic and a real code. Results in your project may be different.

### Comparison of statement and method instrumentation level

#### Sample code

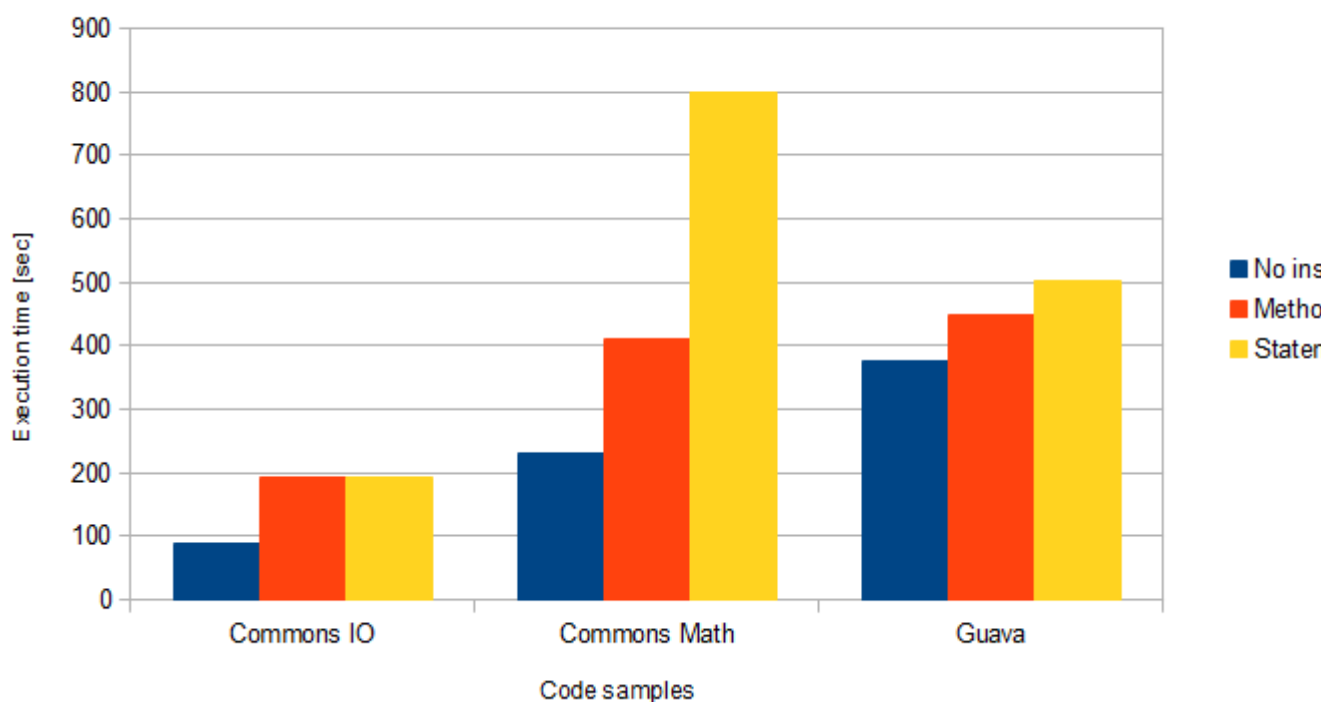
A following open source libraries were tested using statement- and method-level instrumentation.

- [Apache Commons IO](#) version 2.4 (I/O operations)
- [Apache Commons Math](#) version 3.2 (CPU-intensive calculations)
- [Guava](#) version 14.0.1 (data collections)

Standard test of unit tests was executed and a total time of test execution was measured.

#### Results

### Comparison of method-level and statement-level instrumentation



(\*) only 200 test classes were executed for Commons Math

#### Conclusion

Performance penalty for a method and statement instrumentation level may vary significantly, especially for CPU-intensive applications.

### Comparison of different per-test coverage recording strategies

Strategy	System properties	Comment
disabled	-Dclover.pertest.coverage=off	Use this if to disable per-test coverage recording.

diffing	-Dclover.pertest.coverage=diff	Theoretically it is good for highly CPU-intensive code (lot of hit counts to be recorded) and a relatively small code base (smaller hit count arrays to be compared).
single threaded	(nothing)	Default strategy. Designed for single-threaded applications. Per-test coverage might be inaccurate if used with multi-threaded application. Very good performance.
synchronized	-Dclover.pertest.coverage.threading=synchronized	Safe for multi-threaded applications. Performance penalty as recording of every hit count is encapsulated in synchronized block.
volatile	-Dclover.pertest.coverage.threading=volatile	Safe for multi-threaded applications, but requires at least JRE 1.5.

#### Sample code

The following class:

```
import junit.framework.TestCase;
public class PerformanceTest extends TestCase {
    static int hitsPerTest;
    static int numberOfTests;

    public void testPerformance() {
        for (int i = 0; i < hitsPerTest; i++); // empty loop, one R.inc(...) call
per loop
    }

    public static void main(String[] args) {
        numberOfTests = Integer.valueOf(args[0]);
        hitsPerTest = Integer.valueOf(args[1]);

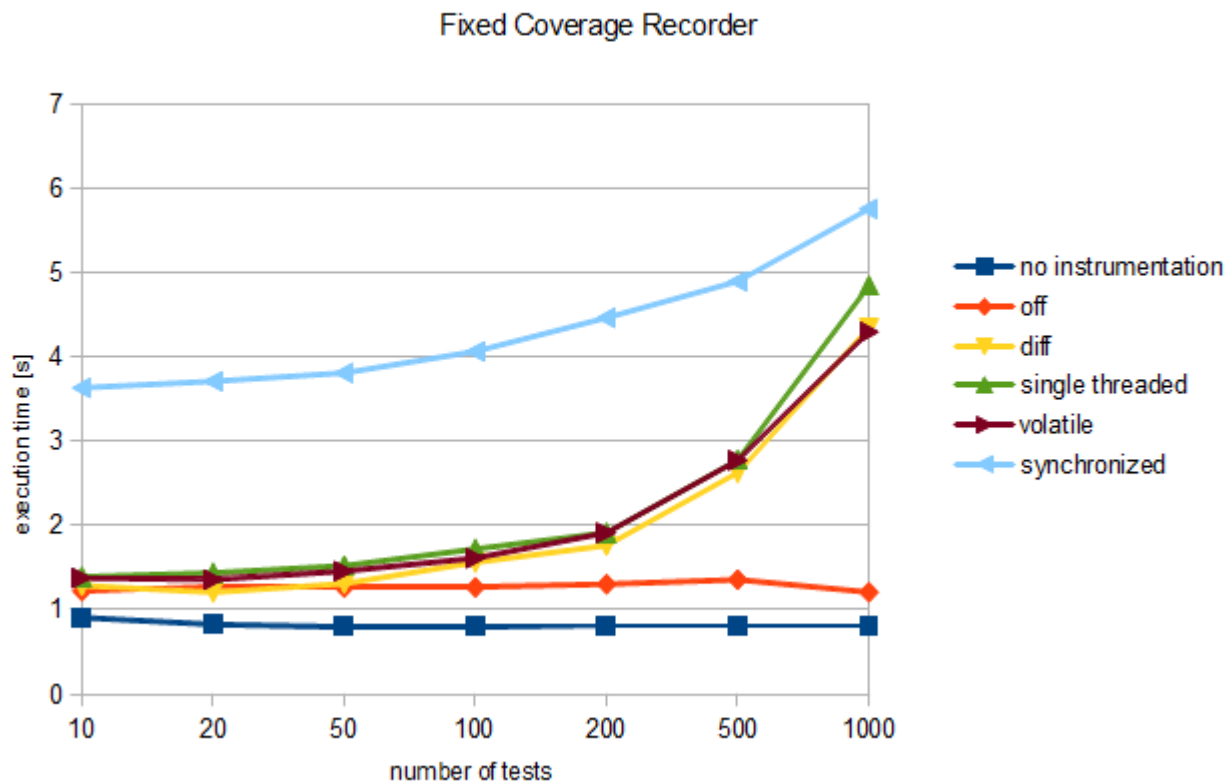
        PerformanceTest pt = new PerformanceTest();
        for (int i = 0; i < numberOfTests; i++) {
            pt.testPerformance();
        }
    }
}
```

was instrumented using Fixed Coverage Recorder and executed with different per-test recording strategies. In order to have roughly 10'000'000 hits recorded by Clover, application was executed with following arguments:

```
PerformanceTest 10 1000000
PerformanceTest 20 500000
PerformanceTest 50 200000
PerformanceTest 100 100000
PerformanceTest 200 500000
PerformanceTest 500 200000
PerformanceTest 1000 10000
PerformanceTest 10000 1000
```

#### Results

## Performance comparison of different per-test recording strategies



Test environment: JDK 1.5, Windows 7; Core i7 2670QM 2.2 GHz; 8GB RAM; HDD 750GB 7200RPM.

#### Conclusion

For a large number of tests, performance is mainly affected by a fact that the coverage recording file is being created on a hard disk. If you have more than 1000 tests there is practically no difference which strategy is used.

For CPU intensive applications the "diffing" strategy is slightly faster than the "single-threaded" or "volatile".

## Coverage Recorders

This page explains few details how Clover code instrumentation works and how the coverage data is being collected at runtime.

This article might be helpful for you in case you have significant performance problems or your application runs in a restricted environment.

#### Instrumenting code

Every time Clover instruments the code (via `<clover-instr/>` or `<clover-setup/>` - both Java and Groovy) it records information about the code structure (packages, files, classes, method, statements, branches, test methods etc) into the Clover database. Because of fact that the same database can be used multiple times (for instance in case of incremental compilation; or when project has several modules compiled separately), it maintains history of changes in blocks named 'instrumentation session'.

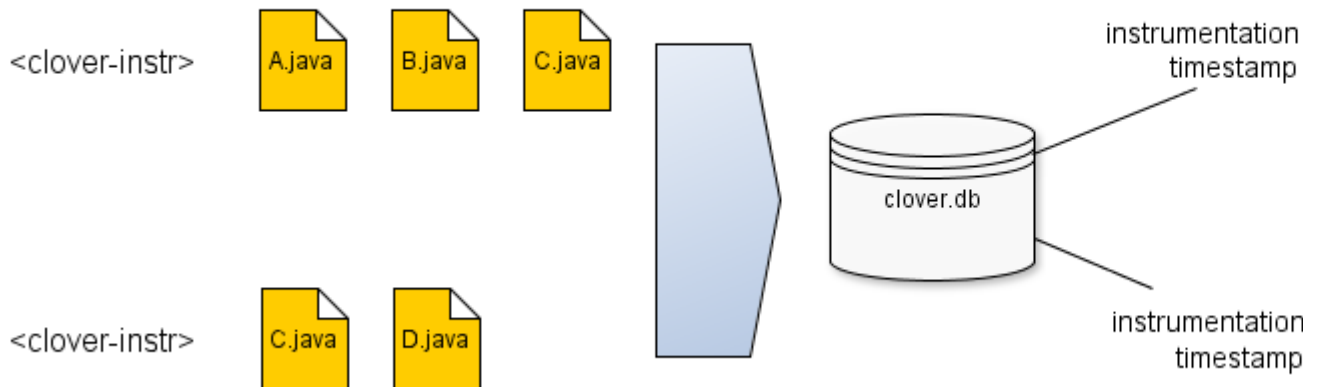
Every instrumented class is enhanced by adding code which requests a Clover's coverage recorder instance. Such coverage recorder getter is called with some arguments, which allows to determine which Clover database file and which instrumentation session in this file contains information about the class structure (like indexes of statements, methods, branches etc).

As project contains many classes, Clover has an optimization so that some of these classes will share the same coverage recorder instance - exact strategy depends on selected recorder type - see below.

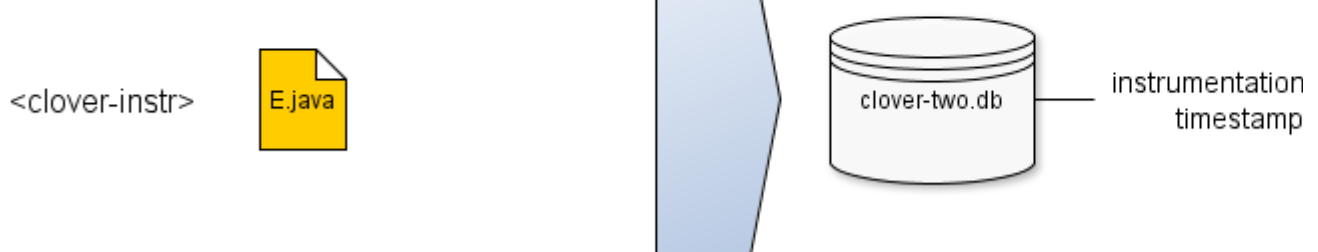
*Diagram: code instrumentation process*

Information about code structured is stored in Clover database. Every instrumented class contains information about database (*initstring*) and the instrumentation session time stamp. Thanks to this it's possible to map compiled class file to a corresponding source file (also in correct version - see *C.java*).

## MODULE ONE




## MODULE TWO



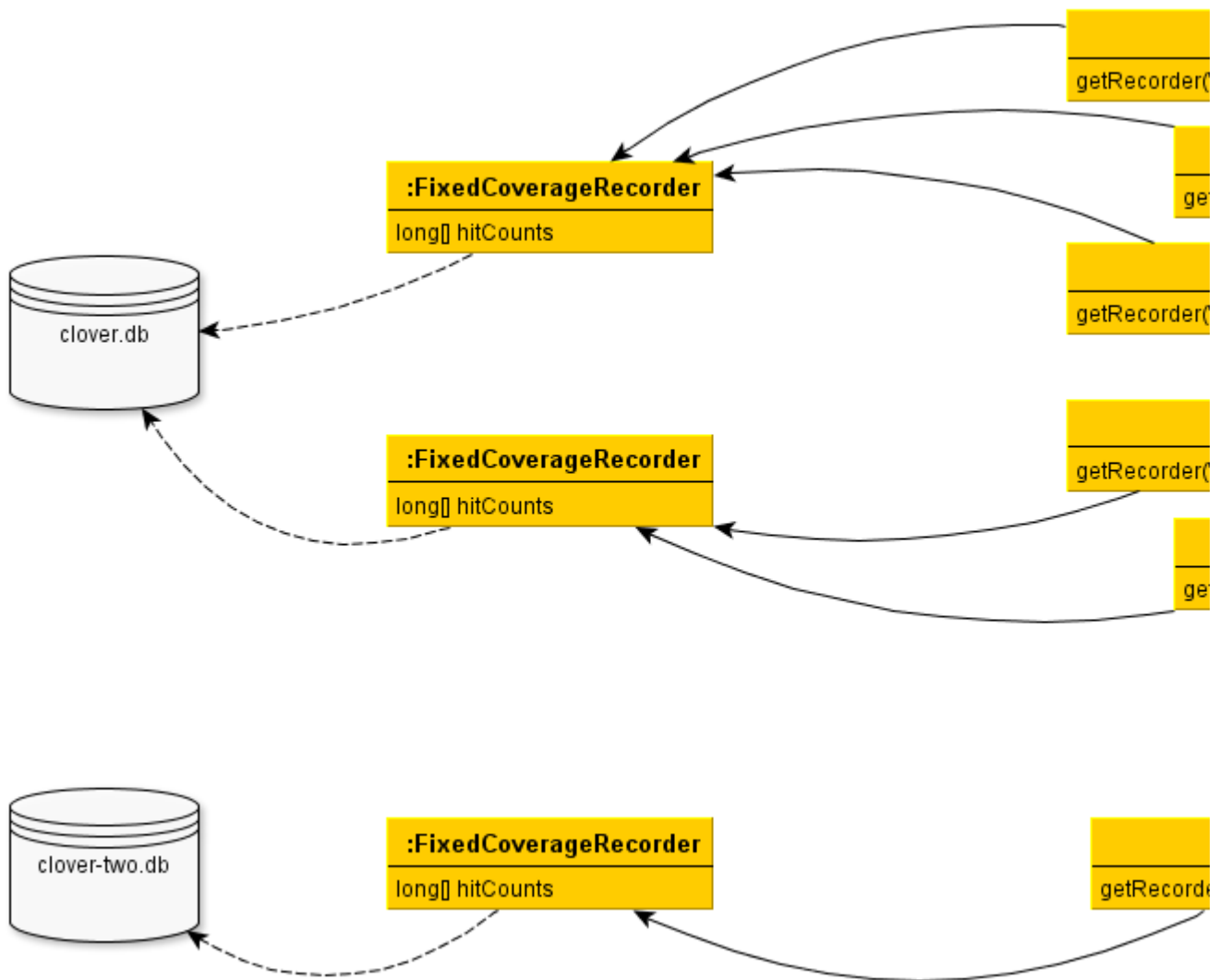
## Executing code

**Fixed Coverage Recorder**

 This is a default coverage recorder and we strongly recommend using it.

It's using an in-memory fixed-size **long[]** array for recording hit counts for methods, statements and branches. Calculation of the long[] array size requires access to the Clover Database (clover.db) at runtime, however. All classes which were compiled in the same instrumentation session (i.e. within the same <clover-instr> or javac or groovy call) will share the same instance of the coverage recorder.

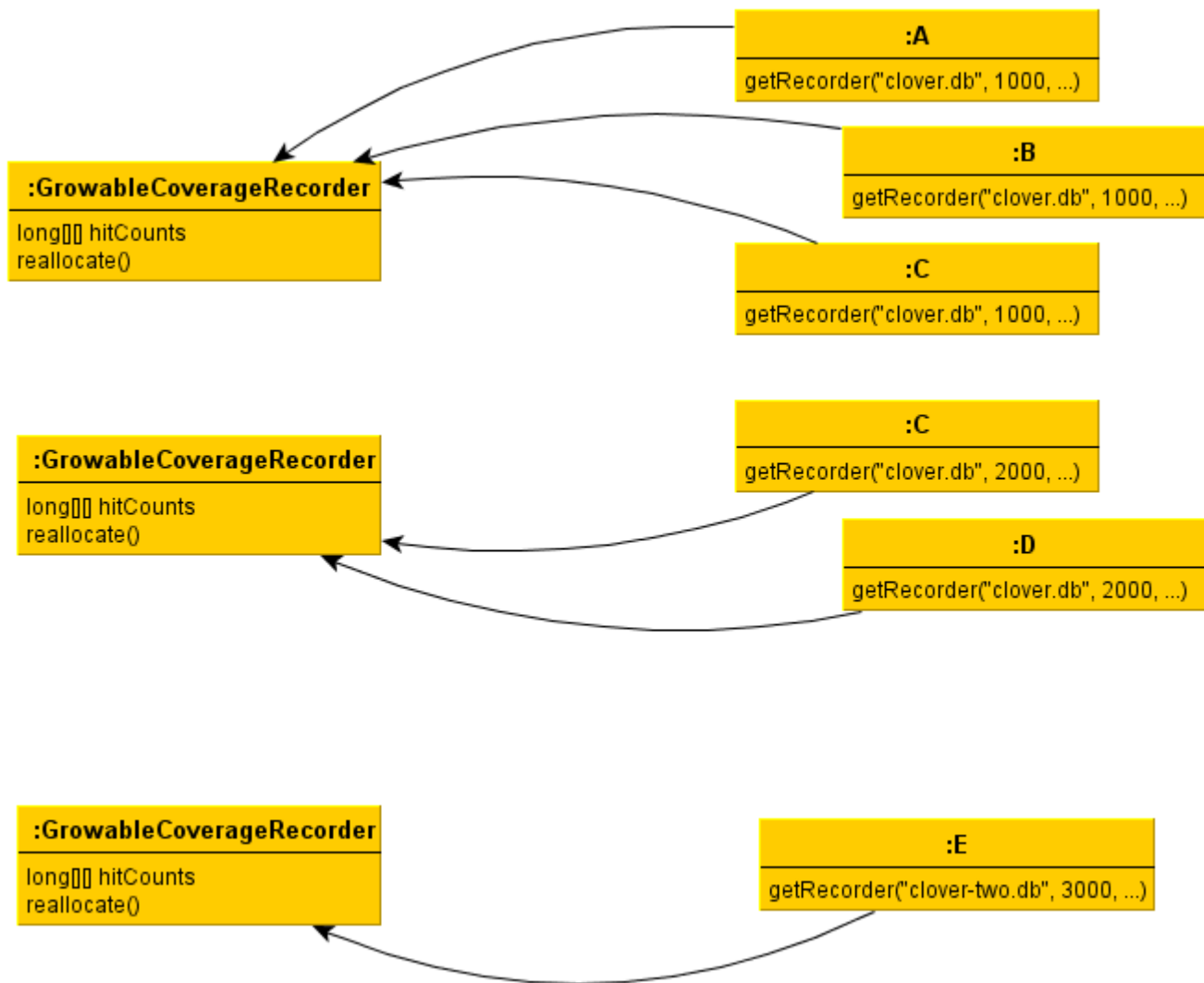




### Growable Coverage Recorder

**i** This recorder simplifies a deployment and test process as `clover.db` is not required at runtime. It's useful for cases like in-container tests, tests executed on application server, builds on remote agents or for Android applications.

It's using a dynamically resizeable two-dimensional `long[][1000000]` array for recording hit counts for methods. Thanks to this it does not require access to the Clover Database at runtime and this is its main advantage. Its performance is slightly lower than the Fixed Coverage Recorder due to memory allocation and two-level indexing. All classes which were compiled in the same instrumentation session (i.e. within the same `<clover-instr>` or `javac` or `groovyc` call) will share the same instance of the coverage recorder.



### Shared Coverage Recorder

**i** Use this coverage recorder only in case when:

- you have a large Grails project with hundreds of Domain Classes or Services **and**
- you instrument and run test classes **and**
- you have a significant performance problem related with coverage data writing and/or generating Clover reports

It's a modification of the Growable Coverage Recorder designed specially for Grails-based projects. It shares the same coverage recorder instance for every instrumented class which was compiled with the same database initstring and configuration settings (like flush policy). It means that it ignores instrumentation session timestamps.

Grails build system works in such way that it compiles every domain class and service class separately. As a consequence, Clover "sees" this as a separate instrumentation session. It means that in case of the fixed or growable coverage recorder it creates a separate instance of the recorder for every domain or service class.

It might become a performance problem if you have many such classes **and** you execute test methods, because end of **every** test method will force creation of the per-test coverage file (*clover.db\*.s*) from **all** coverage recorder instances. For example:

- 500 domain classes \* 1000 unit tests = 500 coverage recorders \* 1000 snapshots = 500'000 files on disk

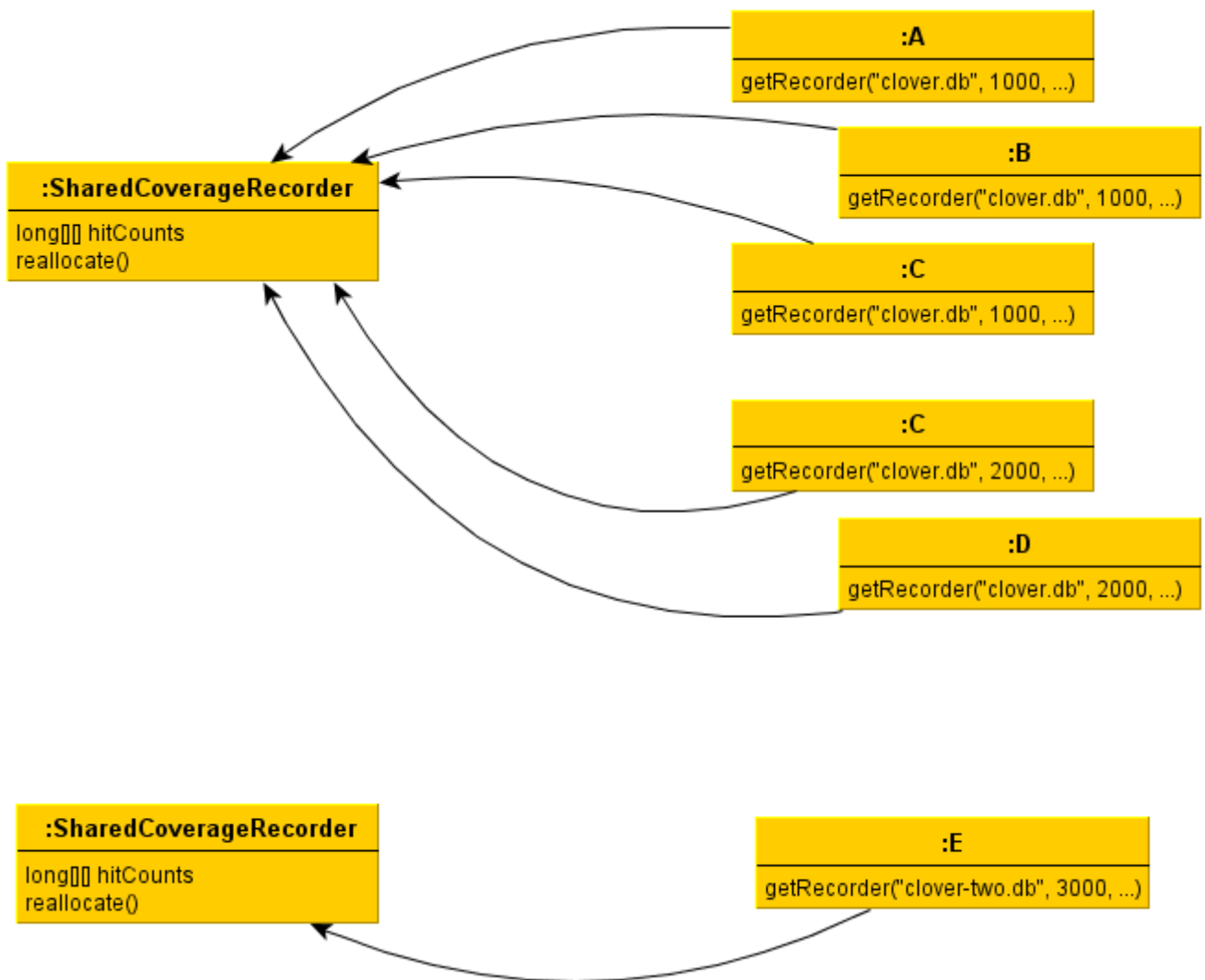


Note that the performance problem does not occur during normal application run or when test code is not instrumented. For example:

- 500 domain classes (normal app execution) = 500 coverage recorders \* 1-2 snapshots (depends on flush policy) = 500-1000 files on disk

If you decide to use a Shared Coverage Recorder, you must keep in mind that:

- you **cannot** have multiple modules which have the same initstring value, but they actually point to different files (it's a typical case if you have a multi-module Maven project and you use the same relative initstring for each module)
  - TIP: use an absolute path for initstrings or share a same database among modules
- you **cannot** deploy outdated class files, for which information in the Clover database is not longer valid (as instrumentation timestamp is ignored);
  - TIP: perform a full project build, deleting all old classes as well as Clover database and coverage recording files (see "Using shared coverage recorder" example for Grails)



## Managing the Coverage Database

### Database structure and lifecycle

The Clover database consists of several files that are constructed at various stages of the instrumentation and coverage recording process. The following files are created if Clover is initialised with an `initstring` of "clover.db"

#### Registry file

**Filename:** `clover.db`

**Description:** The Registry file contains information about all of the classes that have been instrumented by Clover. This file does not contain any actual coverage recording data.

**Lifecycle:** The Registry file is **written** during the instrumentation process. If an existing Registry file is found, the existing file is updated. If no Registry file is found, a new Registry file is created. The Registry file is **read** by Clover-instrumented code when it is executed, and also during report generation or coverage browsing (such as via an IDE plugin or the Swing Viewer).

#### ContextDef file

**Filename:** `clover.db.ctx`

**Description:** The ContextDef file contains user-defined context definitions. Note that while this file is in plain text, it is managed by Clover and should not be edited directly by the user.

**Lifecycle:** The ContextDef file is **written** prior to Clover instrumentation. The ContextDef file is **read** during instrumentation, report generation and coverage browsing.

#### CoverageRecording Files

**Filename:** `clover.dbHHHHHHHH_TTTTTTTTTT` or `clover.dbHHHHHHHH_TTTTTTTTTT.1` (where HHHHHHHH and TTTTTTTTTT are both hex strings)

**Description:** CoverageRecording files contain actual coverage data. When running instrumented code, Clover creates one or more Coverage Recorders. Each Coverage Recorder will write one CoverageRecording file. The number of Coverage Recorders created at runtime depends the nature of the application you are Clovering. In general a new Coverage Recorder will be created for each new ClassLoader instance that loads a Clovered class file. The first hex number in the filename (HHHHHHHH) is a unique number based on the recording context. The second hex number (TTTTTTTTTT) is the timestamp (ms since epoch) of the creation of the Clover Recorder. CoverageRecording files are named this way to try to minimise the chance of a name clash. While it is theoretically possible that a name clash could occur, in practice the chances are very small.

**Lifecycle:** CoverageRecording files are **written** during the execution of Clover-instrumented code. CoverageRecording files are **read** during report generation or coverage browsing.



#### Note

Clover has a failsafe mechanism for writing recording files to disk when using interval-based flush policies. The mechanism alternates between writing to a primary recording file and a secondary recording file. This prevents data loss in the event of abnormal JVM termination. The secondary recording file has the same name as a normal recording file but with `.1` appended to its name.

### Managing the Clover database

Because the Clover database can consist of many recording files, you might find it easier to create the database in its own directory. This directory can be created at the start of a Clover build, and deleted once coverage reports have been generated from the database.

Although Clover will update an existing database over successive builds, it is in general recommended that the database be deleted after it is used to generate reports, so that a fresh database is created on the next build. **Doing this improves the runtime performance of Clover.** The `<clover-clean>` Ant task is provided to allow easy deletion of a Clover database. Note that the IDE Plugins all have a feature to automatically manage the Clover database for you.

### Using a Flush Policy

How Clover writes coverage data to disk at runtime can be configured by changing Clover's *flush policy*. Clover provides three policies: `directed`, `interval` and `threaded`. The default mode is `directed`. The flush policy is set at instrumentation time, either via the `<clover-setup>` Ant task, or via the IDE plugin configuration screen.


Which flush policy you choose depends on the runtime environment in which the instrumented code is executing. In the most common unit testing scenarios, the default flush policy will suffice. In situations where

instrumented code is executing in a hosted environment (e.g. a J2EE container) and shutting down the JVM at the end of testing is not desirable, you will want to use one of the interval-based flush policies.

Policy	Description
directed	<i>default.</i> Coverage recordings are flushed only when the hosting JVM is shut down, or where the user has directed a flush using the <code>///<code>CLOVER:FLUSH</code> inline directive. Directed flushing has the lowest runtime performance overhead of all flush policies (depending on the use of the flush inline directive). <b>Note that no coverage recordings will be written if the hosting JVM is not shut down, or if the hosting JVM terminates abnormally.</b></code>
interval	The <code>interval</code> policy flushes as per the directed policy, and also at a <i>maximum</i> rate determined by the interval set at instrumentation time (see the <code>flushinterval</code> attribute on <code>&lt;clover-setup&gt;</code> ). The <code>interval</code> mode is a 'passive' mode in that flushing potentially occurs only while instrumented code is still being executed. <b>There exists the possibility that coverage data recorded just prior to the end of execution of instrumented code may not be flushed, because the flush interval has not elapsed between the last flush and the end of execution of instrumented code.</b> Any coverage not flushed in this manner will be flushed if/when the hosting JVM shuts down. The <code>interval</code> policy should be used in environments where shutdown of the hosting JVM is not practical <b>and thread creation by Clover is not desired.</b> If you don't mind Clover creating a thread, use the <code>threaded</code> policy. Runtime performance overhead is determined by the flush interval.
threaded	The <code>threaded</code> policy flushes as per the directed policy, and also at a rate determined by the interval set at instrumentation time (see the <code>flushinterval</code> attribute on <code>&lt;clover-setup&gt;</code> ). The <code>threaded</code> mode starts a separate thread to perform flushes. The <code>threaded</code> policy should be used in environments where shutdown of the hosting JVM is not practical. Runtime performance overhead is determined by the flush interval.

## Using Source Directives

Clover supports a number of inline source directives that you can use in your source to control instrumentation. Directives can be on a line by themselves or part of any valid single or multi-line Java comment.

 Clover source directives currently do not support Groovy.

### Switching Clover on and off

```
///CLOVER:ON
///CLOVER:OFF
```

This directive will switch Clover instrumentation on/off. This might be useful if you don't want Clover to instrument a section of code for some reason. Note that the scope of this directive is the current file only.

### Force Clover to flush

```
///CLOVER:FLUSH
```

Clover will insert code to flush coverage data to disk. The flush code will be inserted as soon as possible after the directive. See [Using a Flush Policy](#).

## Working with Distributed Applications

### Introduction

In some cases the application you wish to test has many components running on separate nodes in a network, or even on disconnected machines. You can use Clover to test such applications, although some additional

configuration is required. This page describes how to configure Clover in order to get a per-test code coverage for distributed business logic.

⚠ Please note that having an application deployed on multiple machines does not necessarily mean that the application logic is truly distributed. For instance, an application might run on multiple machines for the sake of load balancing, but do not have communication between nodes. We recommend to read the [Using Clover in various environment configurations](#) tutorial and especially to have a look at the "Decision matrix" which can help you to decide which approach would best fit your needs.

⚠ When deploying your application in container environments, you should also check to ensure that Clover [has sufficient permissions to function](#).

⚠ When deploying, please ensure you deploy your clovered version of the war/ear file. If you use "clover2:instrument" goal, then the clovered version of the war/ear will have a "-clover" in the name and can usually be found in the `target/clover` directory. For example, the filename could resemble this: `money-test-tutorial-1.0-SNAPSHOT-clover.jar`

#### On this page:

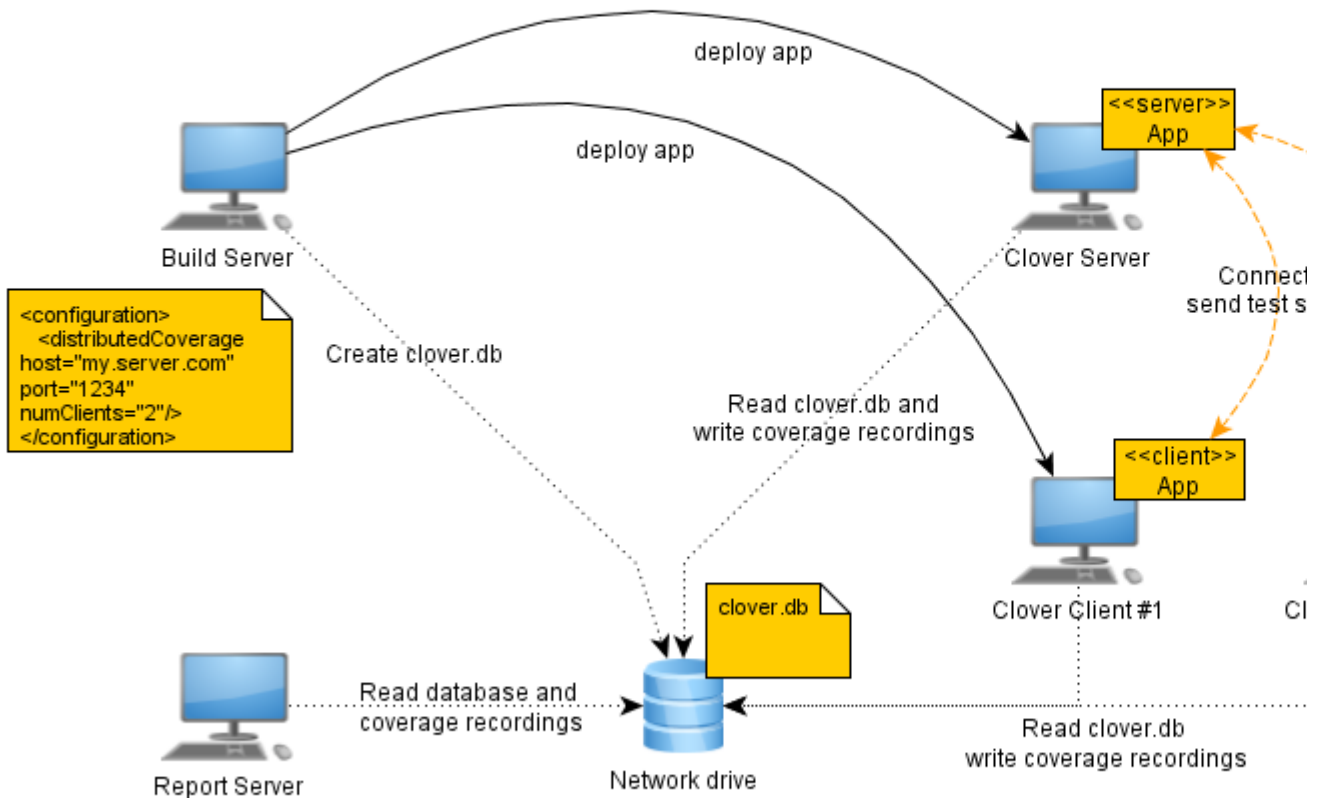
- [Introduction](#)
- [Overview](#)
- [Collecting Overall Coverage from Distributed Builds](#)
  - [Step 1: Understanding the Clover 'initstring'](#)
  - [Step 2: Choosing a Location for the Clover Registry](#)
  - [Step 3: Set Your Classpath Correctly](#)
- [Collecting Per-Test Coverage from Distributed Builds](#)
  - [Enabling or Disabling Distributed Coverage at Runtime](#)
- [Troubleshooting](#)
  - [Server does not wait for clients, despite having numClients != 0 in build configuration](#)
  - [Execution of tests hangs when numClients != 0](#)

#### Overview

## Distributed Coverage - general picture

1. Instrument code with distributed coverage

2. Deploy application.jar, clover.jar, clover.db (or put on a network drive)



### 5. Generate reports

In Clover Distributed Coverage we have two main machine roles:

**Clover Server** - is a JVM in which Clover sends "test start" and "test end" events in order to inform Clover Clients about test boundaries; Clover opens a port on which it waits for Clover Clients to connect

**Clover Client** - is a JVM in which Clover will connect to Clover Server

- 💡 Please note that Clover's Server/Client designation is actually unrelated with your application structure.
- 💡 Both Clover Server and Clover Client write coverage files to disk.

Clover Server is the place where you will typically execute unit tests (or integration tests). These tests will call application logic on Clover Client #N machines.

If unit tests (or integration tests) are executed by a build script, then the **Build Server** actually performs a role of the Clover Server.

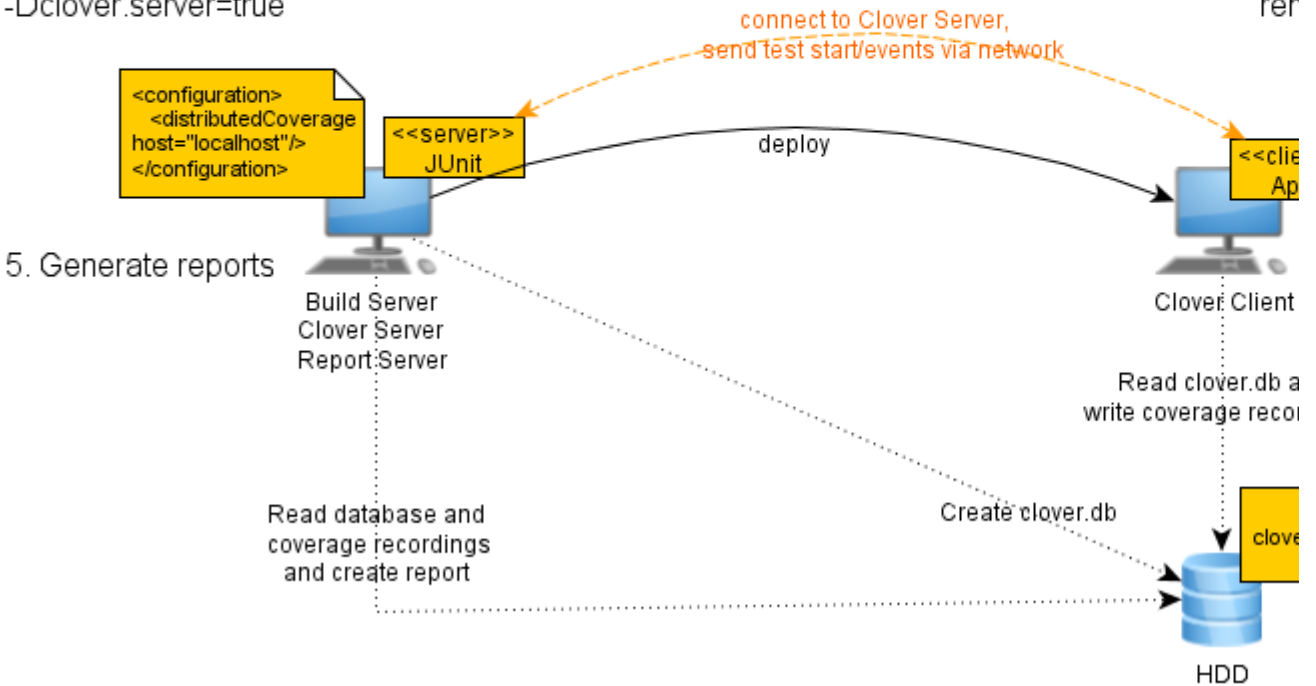
**Report Server** is the place where Clover reports are generated - it can be the same physical machine as Build Server, of course.

## Distributed Coverage - simplified setup for integration tests during build (V

1. Instrument code with distributed coverage

3. Run unit tests with `-Dclover.server=true`

2. I  
de  
4. ,  
rer



5. Generate reports

Build Server  
Clover Server  
Report:Server

Clover Client

Read clover.db and  
write coverage record

Read database and  
coverage recordings  
and create report

Create clover.db

HDD

The diagram above shows a simplified configuration used in the [WebApp example](#). There are only two JVMs used:

- **1st one** is a Maven build, which instantiates a container using Cargo Maven Plugin, deploys compiled application and clover.jar, executes unit tests with `"-Dclover.server=true"` option (Clover records coverage of test classes) and finally creates a report
- **2nd one** is a Tomcat container, where instrumented application is running, Clover listens to "test start/end" events to record per-test coverage (in addition to the global coverage)

### Collecting Overall Coverage from Distributed Builds

The first step in setting up coverage from distributed builds is to configure Clover for overall coverage reporting.

#### Step 1: Understanding the Clover 'initstring'

At build time, Clover constructs a registry of your source code, and writes it to a file at the location specified in the Clover initialisation string (`initstring`). When Clover-instrumented code is executed (e.g. by running a suite of unit tests), Clover looks in the same location for this registry file to initialise itself. Clover then records coverage data and writes coverage recording files next to the registry file during execution. See [Clover Database Structure](#) for more information.

#### Step 2: Choosing a Location for the Clover Registry

If you are deploying and running your Clover-instrumented code on different machines, you must provide a way for Clover to find the registry file, and provide a place for Clover to write coverage recording files; otherwise no coverage will be recorded.



Clover provides three different ways to achieve this:

1. **Specify an `initstring` that is a globally accessible file path**

The compile-time `initstring` should be an absolute path to the *same* filesystem location, and be accessible and writable from the build machine and all execution machines. This could be a path on shared drive or filesystem.

OR:

2. **Specify an `initstring` that is a relative path, resolved at runtime**

The compile-time `initstring` represents a relative path (relative to the CWD of each execution context). To do this you need to specify `relative="yes"` on the `<clover-setup>` task.

OR:

3. **Specify an `initstring` at runtime via system properties**

You can override the Clover `initstring` at runtime via system properties. Two (three?) system properties are supported:

<code>clover.initstring</code>	If not null, the value of this property is treated as an absolute file path to the Clover registry file
<code>clover.initstring.basedir</code>	If not null (and the <code>clover.initstring</code> system property is not set), the value of this property is used as the base directory for the file specified at compile-time in the <code>initstring</code> to resolve the full path to the Clover registry.
<code>clover.initstring.prefix</code>	If not null (and the <code>clover.initstring</code> or <code>clover.initstring.basedir</code> system properties are not set), the value of this property is prepended to the string value of compile-time specified <code>initstring</code> to resolve the full path to the Clover registry.

To set one of these properties, you need to pass it on the command line when Java is launched, using the `-D` parameter:

```
java -Dclover.initstring=... myapplication.Server
```

For application servers, this may involve adding the property to a startup script or batch file.



For methods two and three in the sequence above, you will need to copy the Clover registry file from the location on the build machine to the appropriate directory on each of the execution machines (as part of the test deployment process).

This needs to occur:

- a. after the Clover build is complete, and
- b. before you run your tests.

Once test execution is complete, you will need to copy the coverage recording files from each remote machine to the `initstring` path on the build machine in order to generate coverage reports.

### Step 3: Set Your Classpath Correctly

You must put `clover.jar` (or the appropriate Clover plugin jar) in the classpath for any JVM that will load classes that have been instrumented by Clover. How you go about this depends on the nature of the application you are testing and the environment you are deploying to.

In some cases, the `clover.jar` must be on the classpath of the actual webserver, not just on the classpath of the webapp that is instrumented. This is to ensure Clover can properly flush its coverage data when the JVM of the webserver is shutdown.

### Collecting Per-Test Coverage from Distributed Builds

**i** The steps below require you to have carried out the previous steps on this page (related to 'Collecting Overall Coverage from Distributed Builds').

### Enabling or Disabling Distributed Coverage at Runtime

Clover's Distributed Coverage feature is enabled at runtime by making use of command-line options.

This can be done without the need for re-instrumentation or compilation of source files.

### Enabling Distributed Coverage

Distributed coverage can be enabled via setting this System property:

```
-Dclover.distributed.coverage=ON
```

This will enable distributed coverage with default settings (host=localhost, port=1198, timeout=5000ms, numClients=0, retryPeriod=1000ms, name=clover.tcp.server).

In case when you cannot use default settings, you can pass specific value for any of attributes using the "key=value" syntax passed as *clover.distributed.coverage* value:

- host - host name of the "Clover Server"
- port - port on which the Clover will listen
- numClients - number of "Clover Clients" to connect until server starts test execution
- timeout - connection timeout in milliseconds
- retryPeriod - interval between connection retries in milliseconds
- name - name of the Clover server service (URL is host:port/name)

Example:

```
-Dclover.distributed.coverage=host=myhost;port=7777;numClients=2
```

Clover also needs to know which JVM is hosting your unit tests ("Clover Server"), by providing the following system property:

```
-Dclover.server=true
```

### Disabling Distributed Coverage

Distributed coverage can be disabled by setting this System property on either the Test or the Application JVM:

```
-Dclover.distributed.coverage=OFF
```

This will turn off distributed coverage for the JVM in which this is set, regardless of what was instrumented.

For more configuration options and how to do this in Ant and Maven, see the [Clover-for-Ant](#) and [Clover-for-Maven 2](#) documentation.

### Configuration Complete

Distributed Per-Test Coverage in Clover will now operate when running distributed builds. Detailed reports will now be available.

### Troubleshooting

**Server does not wait for clients, despite having numClients != 0 in build configuration**

❗ Do not use `-Dclover.distributed.coverage=ON` runtime option if `numClients!=0` was set in instrumentation. The `clover.distributed.coverage` provided at runtime will override `numClients` setting from instrumentation, setting it to 0.

As a consequence your tests on server will start immediately, without waiting for clients to connect. It can result in lower or zero coverage.

Instead of this:

- enable `clover.distributed.coverage` option in build file or
- use `-Dclover.distributed.coverage=numClients=N` (where N is a number  $\geq 0$ ) at runtime

**Execution of tests hangs when numClients != 0****❗ Server-client dependency loop**

It can happen that your "Server" will wait for "Clients" to connect, while clients will wait until server starts unit test execution - it depends on how tests are written.

This is a typical case for web applications running in container (like Tomcat, JBoss), when your unit test calls a servlet class (e.g. via HTTP request). The issue is as follows:

- unit tests on `<<server>>` are waiting until all clients are connected (`numClients != 0`) but
- none of the clients will connect until servlet class is loaded in the container, which happens only when first request comes (and it will not come, due to the point above)

In order to avoid this circular dependency you have to:

- create a servlet context listener (and instrument it by Clover)
  - the class can do virtually nothing
  - the listener class will be automatically loaded by container at application deployment (without waiting for any web request)
  - as soon as class is loaded by classloader, it will automatically connect Clover recorder instance to the "Clover server"; Clover server will start its execution

**Sample code**

```
package com.my.webapp;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

public class MyServletContextListener implements ServletContextListener {
    @Override
    public void contextInitialized(ServletContextEvent servletContextEvent) {
        System.out.println("Web App Initialized");
    }
    @Override
    public void contextDestroyed(ServletContextEvent servletContextEvent) {
        System.out.println("Web App Destroyed");
    }
}
```

```
<web-app>
  <!-- ... -->
  <listener>
    <listener-class>com.my.webapp.MyServletContextListener</listener-class>
  </listener>
  <!-- ... -->
</web-app>
```

Full code example is available on Bitbucket: <https://bitbucket.org/atlassian/maven-clover2-plugin> (src/it/webapp).

## Using Distributed Per-test Coverage with Clover-for-Ant

This page contains instructions on how to collect per-test coverage from a set of functional tests, which run in multiple JVMs (Java Virtual Machines). This may be necessary when starting a web server with the **Jetty Runner** or **Tomcat Tasks**, for example.

On this page:

- **General Overview**
  - **Option 1. Enabling Distributed Coverage at Runtime**
    - **Setting the System Properties in the Ant JUnit Task**
    - **Setting a System Property in the Java task that starts the WebServer**
  - **Option 2. Configuring Distributed Coverage at Instrumentation Time**
    - **Step 1: Activate the Distributed Per-Test Coverage Feature**
    - **Step 2: Specify the 'clover.server' property on JVM running the Tests**
  - **Related Links**

### General Overview

Clover collects per-test coverage for tests running in a separate JVM by sending messages using the tcp protocol. The JVM hosting the tests is the 'Clover Server' and the JVM(s) hosting the application are the 'Clover Clients'. The 'Clover Server' ( ie. the JVM running your tests) needs to be marked as such via a System Property: 'clover.server=true'. If this property is not set, or is set to 'false', the JVM will be in 'Clover Client' mode.

If you are testing multiple projects on the same machine at the same time (such as in a Continuous Integration environment), you will need to ensure a unique port for each build is reserved and configured. By default, Clover starts a socket server on port 1198.

Distributed per-test coverage will give you insight as to what functional tests covered what application code. It also allows you to drastically reduce test execution time by using [Clover's Test Optimization](#) to only run the tests for code that was modified since the previous build.


To configure Clover for collection of per-test coverage from distributed builds, you have two options:

1. *Recommended:* By setting a System Property on the JVM running your tests and the JVM hosting your application (e.g. the webserver)
2. By configuring Clover before instrumenting (clover-setup, clover-instr) your source files - and setting a System property on the test JVM (Only Recommended if setting a System Property on your webserver poses a problem.)

### Option 1. Enabling Distributed Coverage at Runtime

Once your Clover-instrumented application has been deployed, and your tests have been instrumented and compiled with Clover, distributed per-test coverage can be enabled and configured at runtime using just two System Properties.

Both JVMs require the 'clover.distributed.coverage' property set to ON, and the JVM running the tests require the 'clover.server' property set to 'true'.

 TIP: the `clover.distributed.coverage=ON` takes default settings (host=localhost, port=1198, timeout=5000ms, numClients=0, retryPeriod=1000ms, name=clover.tcp.server). In case when you cannot use default settings, you can pass specific value for any of attributes using the "key=value" syntax passed as `clover.distributed.coverage` value:

- host - host name of the "Clover Server"
- port - port on which the Clover will listen
- numClients - number of "Clover Clients" to connect until server starts test execution
- timeout - connection timeout in milliseconds
- retryPeriod - interval between connection retries in milliseconds
- name - name of the Clover server service (URL is `host:port/name`)

Example:

```
-Dclover.distributed.coverage=host=myhost;port=7777;numclients=2
```

For the following examples, we are using the [Jetty Runner](#) to start the Jetty Webserver, and the Ant JUnit Task to run the tests.

#### Setting the System Properties in the Ant JUnit Task

```
<junit fork="true" forkmode="once" showoutput="true" printsummary="true">
  <sysproperty key="clover.server" value="true"/>
  <sysproperty key="clover.distributed.coverage" value="ON"/>
  ...
</junit>
```

#### Setting a System Property in the Java task that starts the WebServer

The JVM running your webserver also requires the `clover.distributed.coverage` property set to ON.

```
<java jar="${jetty.jar}" fork="true">
  ...
  <jvmarg value="-Dclover.distributed.coverage=ON"/>
</java>
```

If you are unable to set a System Property on the JVM running your webserver, use the second approach described below.

#### Option 2. Configuring Distributed Coverage at Instrumentation Time

It is sometimes more convenient to enable distributed Coverage when you enable Clover - before instrumentation of your source code.

The following approach removes the need to set any system properties at all on the JVM running the WebServer.

##### Step 1: Activate the Distributed Per-Test Coverage Feature

Both the `<clover-setup>` and `<clover-instr>` tasks can be configured with this nested element:

```
<distributedCoverage/>
```

If this element is present, then Clover will run in 'distributed mode' at test time. If you wish to modify any configuration options such as the port to listen on, or the number of clients expected to attach to the testing session, you can specify these as attributes on the `<distributedCoverage>` element like so:

```
<clover-setup>
  <distributedCoverage port="1234" numClients="1"/>
</clover-setup>
```

This will enable distributed per-test coverage to be collected. Please see the [documentation](#) for the `<distributedCoverage/>` element for more options.

**Step 2: Specify the 'clover.server' property on JVM running the Tests**

Add the `clover.server` system property to the JUnit or TestNG Ant task configuration, and ensure the `forkMode` parameter is set to 'once':

e.g.

```
<junit fork="true" forkmode="once" showoutput="true" printsummary="true">
  <sysproperty key="clover.server" value="true"/>
  ...
</junit>
```

If you have specified the `numClients` option to something greater than 0, your tests can be started prior to starting the webserver. Clover will wait until `numClients` have connected to the testing session before allowing the tests to start running.

**Related Links**

[About Distributed Per-Test Coverage](#)

[About Test Optimization](#)

## Working with Restricted Security Environments

In some Java environments, such as J2EE containers, applet environments, or applications deployed via Java [Webstart](#), security restrictions are applied to hosted Java code that restrict access to various system resources.

To use Clover in these environments, Clover needs to be granted various security permissions for it to function. This requires the addition of a `grant` entry to the security policy file for the Clover jar. For background on the syntax of the policy file, see [Default Policy Implementation and Policy File Syntax](#). For background on setting Java security policies in general, see [Permissions in the Java SDK](#).

**Recommended Permissions**

Clover requires access to the Java system properties for runtime configurations, as well as read write access to areas of the file system to read the Clover coverage database and to write coverage information. Clover also uses a shutdown hook to ensure that it flushes any as yet unflushed coverage information to disk when Java exits. To support these requirements, the following security permissions are recommended:

```
grant codeBase "file:/path/to/clover.jar" {
  permission java.util.PropertyPermission "*", "read";
  permission java.io.FilePermission "<<ALL FILES>>", "read, write";
  permission java.lang.RuntimePermission "shutdownHooks";
}
```

## Working in OJVM

OJVM - Oracle Java Virtual Machine

### Step by step

1. Download and install Oracle 11 database [Standard Edition One](#) or higher (note that the Express Edition does not support Java)
2. Download and unpack [OJVMTutorial](#)
3. Download and unpack [Clover-for-Ant](#) into <clover\_home>
4. Install <clover\_home>/lib/clover.jar into Oracle database using the loadjava tool:
  
5. Compile OJVMTutorial e.g. by using javac or loadjava tool
  - a. define location where coverage snapshots will be written
  
6. Configure security manager as follows:

Provide Run application,

### Example


Down

- OJVMTutorial
- [http://docs.oracle.com/cd/B19306\\_01/java.102/b14187/chnine.htm#BABJBJGE](http://docs.oracle.com/cd/B19306_01/java.102/b14187/chnine.htm#BABJBJGE) - Security for Oracle Database Java Applications

## 9. Clover Target Reference

Clover provides a set of high level, preconfigured Ant targets. A target is a high-level, pre-configured set of functionality that you can use to quickly integrate Clover. They can be launched by adding them as values to Ant on the command line.

Each target contains a number of logically grouped lower-level pieces of functionality made up of Clover-specific [Tasks](#) and [Types](#). These allow you to harness Clover's feature set, applying the concept of *convention over configuration*. This should help you avoid laboriously coding your own targets from scratch — allowing you to rapidly begin using Clover, no matter how complex your environment.

 To enable Ant targets in Clover, you need to firstly follow the [Clover Quickstart Guide](#).

On this page:

- [Using a Target](#)
  - [with.clover \(target\)](#)
  - [clover.all \(target\)](#)
  - [clover.clean \(target\)](#)
  - [clover.current \(target\)](#)
  - [clover.report \(target\)](#)
  - [clover.save-history \(target\)](#)
  - [clover.snapshot \(target\)](#)
  - [clover.snapshot.file \(target\)](#)
- [Custom Targets](#)

### Using a Target

Each Clover target typically has a name (and optional Ant properties) that can be used on the command line, as follows:

```
ant clover.all
```

In the above example, we are running the `clover.all` target. This runs a 'clean' process, creates a build with Clover applied and then generates a Clover report.

#### `with.clover` (target)

Enables Clover on the current build. There are no properties for this target.

#### Example:

```
ant with.clover
```

#### `clover.all` (target)

Runs `clover.clean`, `with.clover`, `<test.target>`, `clover.report`, `clover.log` (in that order) from a single target.

#### `clover.all` Properties:

Property name	Description	Options	Dependency
test.target	Defines the name of a custom target to run the tests.	inheritrefs (TRUE/FALSE)	clover.clean, with.clover

#### Example:

```
ant clover.all -Dtest.target=run.tests
```

In the example above, we are specifying the `test.target` property. This is fed another value, ( shown as `run.tests`). Note that `run.tests` in this example could be replaced by the name of any custom target you may have created yourself in `build.xml`.

#### `clover.clean` (target)

Deletes the clover database and the `clover.dest` directory. There are no properties for this target.

#### Example:

```
ant clover.clean
```

#### `clover.current` (target)

Generates HTML and XML reports to `clover.dest` using `project.title`.

#### `clover.current` Properties:

Property name	Description	Options	Dependency
project.title	A string value, the user-specified name for this project	None	None
clover.dest	A system path, the destination directory for clover reports for this project	None	None
clover.span	A time <a href="#">interval</a> defining the <a href="#">time span</a> to use when creating historical reports	None	None



**Example:**

```
ant clover.current -Dproject.title="MyProject"
```

**clover.report (target)**

This is the same as `clover.current`, i.e. it generates HTML and XML reports to `clover.dest` using `project.title`. Additionally, a history report will also be created using the `historypoints` in `clover.project.historydir`.

**clover.report Properties:**

Property name	Description	Options	Dependency
<code>clover.historypoint.projectdir</code>	A path location, where history points are stored.	None	None
<code>clover.dest</code>	A system path, the destination directory for clover reports for this project	None	None
<code>clover.span</code>	A time <a href="#">interval</a> defining the <a href="#">time span</a> to use when creating historical reports	None	None

**Example:**

```
ant clover.report -Dclover.historypoint.projectdir="\myprojects\project3\history\ "
```

**clover.save-history (target)**

Saves a history point to `clover.project.historydir`.

**clover.save-history Properties:**

Property name	Description	Options	Dependency
<code>clover.project.historydir</code>	A path location, where history points are stored.	None	None
<code>test.target</code>	Defines the name of a custom target to run the tests.	<code>inheritrefs</code> (TRUE/FALSE)	<code>clover.clean</code> , <code>with.clover</code>
<code>clover.span</code>	A time <a href="#">interval</a> defining the <a href="#">time span</a> to use when creating historical reports	None	None

**Example:**

```
ant clover.save-history
```

**clover.snapshot (target)**

Saves a snapshot file to assist with Clover's [Test Optimization](#) feature. There are no properties for this target.

**Example:**

```
ant clover.snapshot
```

**clover.snapshot Properties:**

Property name	Description	Options	Dependency
clover.snapshot.file	Defines the location of the snapshot file to use when saving optimized data. For use with Clover's <a href="#">Test Optimization</a> feature.	None	None

#### clover.snapshot.file (target)

Defines the location of the snapshot file to use when saving optimized data. For use with Clover's [Test Optimization](#) feature.

#### Example:

```
ant clover.snapshot.file
```

## Custom Targets

Clover targets can be modified or overwritten. You can also create your own targets by specifying targets with the same names in `build.xml`.

### A. Integrating Clover-for-Ant with other tools

This page contains tutorials showing how Clover-for-Ant can be integrated with different tools or frameworks:

- [Using Clover-for-Ant with GWT](#)
- [Instrumenting JSP files](#)
- [Using Clover with non-standard test framework \(i.e. not handled by Clover default test detection\)](#)
- [Integrating Clover with JUnit4 Parameterized Tests](#)

Clover can be configured in many ways, depending on how the development environment is configured or application is structured. Useful manuals:

- [Using Clover for web applications](#)
- [Using Clover in various environment configurations](#)

## Integrating Clover with JUnit4 Parameterized Tests

### Introduction

JUnit4 framework version **4.10** has introduced a feature which allows to run the same test multiple times, using different data as input.

In order to use this, you have to:

- annotate test class with `@RunWith(Parameterized.class)`
- declare a `data()` method returning collection of input values and annotate this method with `@Parameters` annotation.
- declare a test method annotated with `@Test`

Furthermore, the JUnit version 4.11 has added a 'name' attribute to the `@Parameters` annotation - thanks to this, you can define a custom name for a test. You can use variables such as "{index}" for an iteration number and "{0}, {1}, ..." for N-th input argument in a test name.

For example:

```

@RunWith(Parameterized.class)
public class PersonTest {
    @Parameterized.Parameters(name = "{0} is a {1} [{index}]")
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][]{
            {"Alice", "woman"}, {"Bob", "man"}, {"Rex", "unknown"}
        });
    }
    protected String input;
    protected String expected;
    public PersonTest(String input, String expected) {
        this.input = input;
        this.expected = expected;
    }
    @Test
    public void test() {
        assertEquals(expected, new Person(input).getSex());
    }
}

```


See more details on [JUnit wiki page](#).

## Integrating Clover





As such parametrized tests are being executed by JUnit's test runner, Clover has no problem with recording test results for them. However, till Clover 3.3.0 there's was no information about which test iteration has failed - all test iterations had the same same:

### Class SquareTest

Source file Test results

Class	Tests	Fail	Error	Time (secs)	% Tests Succeeded
SquareTest	6	4	0	0,009	33,3% 


  

Tests	Started	Status	Time (secs)	Message
<a href="#">SquareTest.test</a>	7 sie 12:17:29	FAIL	0,003	 expected:<1> but was:<2>
<a href="#">SquareTest.test</a>	7 sie 12:17:29	FAIL	0	 expected:<9> but was:<6>
<a href="#">SquareTest.test</a>	7 sie 12:17:29	FAIL	0	 expected:<16> but was:<8>
<a href="#">SquareTest.test</a>	7 sie 12:17:29	FAIL	0,001	 expected:<25> but was:<10>
<a href="#">SquareTest.test</a>	7 sie 12:17:29	PASS	0	
<a href="#">SquareTest.test</a>	7 sie 12:17:29	PASS	0,005	

**Clover 3.3.0** introduced a `JUnit4TestRunnerInterceptor`, which can be attached to JUnit's runner. It "listens" which test is being executed and what runtime name it has (evaluated by JUnit). Thanks to this, you can see an iteration number:

## Class SquareTest

Source file Test results

Class	Tests	Fail	Error	Time (secs)	% Tests Success
SquareTest	6	4	0	0,007	33,3% 


  

Tests	Started	Status	Time (secs)	Message
SquareTest.test[0](SquareTest)	7 sie 12:13:08	PASS	0,005	
SquareTest.test[1](SquareTest)	7 sie 12:13:08	FAIL	0,001	expected:<1> but was:<2>
SquareTest.test[2](SquareTest)	7 sie 12:13:08	PASS	0	
SquareTest.test[3](SquareTest)	7 sie 12:13:08	FAIL	0	expected:<9> but was:<6>
SquareTest.test[4](SquareTest)	7 sie 12:13:08	FAIL	0,001	expected:<16> but was:<8>
SquareTest.test[5](SquareTest)	7 sie 12:13:08	FAIL	0	expected:<25> but was:<10>

as well as full test names ( `@Parameters(name=...)` ) in the reports:

## Class PersonTest

Source file Test results

Class	Tests	Fail	Error	Time (secs)	% Tests Success
PersonTest	3	1	0	0,007	66,7% 

Tests	Started	Status	Time (secs)	Message
PersonTest.test[Rex is a unknown [2]](PersonTest)	7 sie 12:25:04	FAIL	0,001	expected:<[unknown]> but was:<[ma]
PersonTest.test[Alice is a woman [0]](PersonTest)	7 sie 12:25:04	PASS	0,006	
PersonTest.test[Bob is a man [1]](PersonTest)	7 sie 12:25:04	PASS	0	

Unfortunately, neither Ant's `<junit>` task nor JUnit itself (via command line argument) has a way to attach test listeners. It must be done programmatically. You have to instantiate a `JUnitCore`, add Clover's `JUnitTestRunnerInterceptor` to it and call `core.run()` method passing test class(es) as an argument.

Example:

```
import org.junit.runner.JUnitCore;
import com.atlassian.clover.recorder.junit.JUnitTestRunnerInterceptor;

public class RunJUnit4WithClover {
    public static void main(String[] args) {
        JUnitCore core= new JUnitCore();
        core.addListener(new JUnitTestRunnerInterceptor());
        core.run(SquareTest.class);
    }
}
```

As soon as test execution is finished you can generate a Clover report.

**i** **LIMITATION**

Clover's JUnit4TestRunnerInterceptor can correctly handle parameterized test names when test methods from a single test case class are executed sequentially. It means that you shall not use a test runner which will run all iterations in parallel.

On the other hand, running entire test cases or test suites in parallel is allowed.

**References**

- <https://github.com/junit-team/junit/wiki/Parameterized-tests>
- <https://bitbucket.org/atlassian/clover-examples> parameterized-junit4-example

**Using Clover-for-Ant with GWT**

**i** You can instrument server-side code only. This is due to a nature of the Google Web Toolkit which translates client and shared parts into a JavaScript. If you try to instrument client code, GWT will search for sources of all referenced classes, including the Clover instrumentation, which would cause a build failure.

**Clover with manual GWT integration**

*The following example is based on GWT SDK 2.5.*

## 1) Download and install

- GWT SDK 2.5 - <http://code.google.com/p/google-web-toolkit/downloads/list> (referred as <gwt-sdk>)

2) Open the sample DynaTable application (located in <gwt-sdk>/samples/DynaTable; referred as <project\_dir>) and enhance *build.xml* file by adding:

- <clover-setup> with includes="" for server-side code
- <clover-report> or <clover-html-report>
- clover.jar to runtime classpath

Example:

**build.xml**

```

<project name="DynaTable" default="build" basedir=". ">
  <!-- Add following properties and targets -->

  <property name="clover.jar" location="${user.home}/clover.jar"/>
  <property name="clover.license" location="${user.home}/clover.license"/>
  <property name="clover.db" location="clover/db/clover.db"/>
  <property name="clover.report" location="clover/report"/>
  <taskdef resource="cloverlib.xml" classpath="${clover.jar}"/>

  <target name="init" if="with.clover">
    <clover-setup initstring="${clover.db}">
      <fileset dir="src"
includes="com/google/gwt/sample/dynatable/server/**"/>
    </clover-setup>
  </target>

  <target name="report">
    <clover-html-report initstring="${clover.db}" outdir="${clover.report}"/>
  </target>

  <!-- ... -->

  <!-- Add the "init" target to depends="..." -->
  <target name="javac" depends="libs, init" description="Compile java source to
bytecode">
    <!-- ... -->
  </target>

  <!-- Add the clover.jar to classpath -->

  <target name="devmode" depends="javac" description="Run development mode">
    <java failonerror="true" fork="true"
classname="com.google.gwt.dev.DevMode">
      <classpath>
        <pathelement location="src"/>
        <path refid="project.class.path"/>
        <pathelement location="../../validation-api-1.0.0.GA.jar" />
        <pathelement location="../../validation-api-1.0.0.GA-sources.jar"
/>
        <pathelement location="${clover.jar}"/> <!-- ADD THIS -->
      </classpath>
      <!-- ... -->
    </java>
  </target>

  <!-- ... -->

</project>

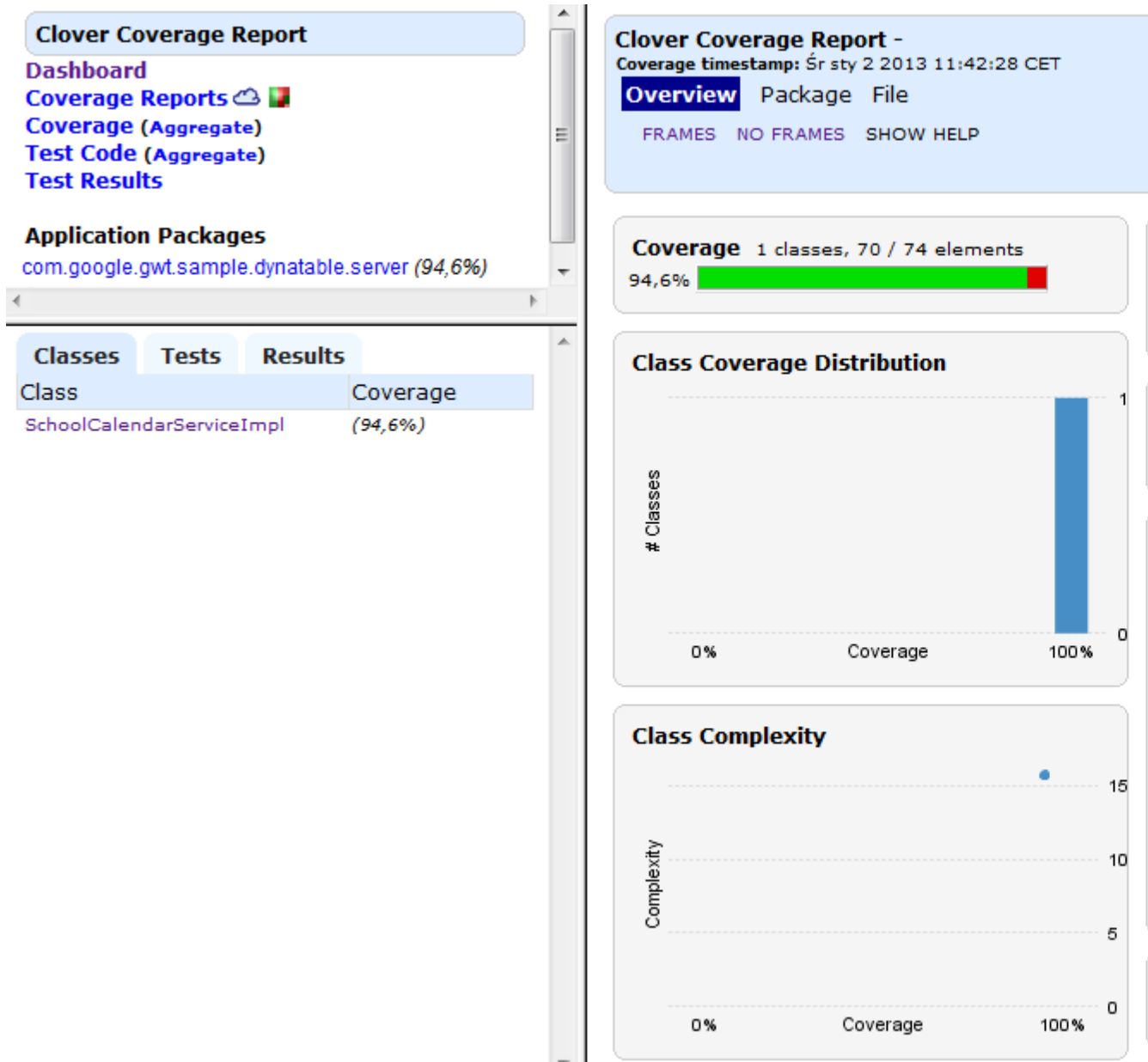
```

**3) Build application, run tests and generate Clover report.**

 GWT requires Java 1.6 or above

```
ant devmode -Dwith.clover=true
... open web browser, close GWT console ...
ant report
```

Open `<project_dir>/clover/report/index.html` file. You shall see a report like this:



#### References

See also

- [Using Clover with the GWT-maven plugin.](#)

## Clover-for-Ant Installation Guide

For the quickest and easiest installation options, see the [QuickStart Guide](#) instead of this page.

### 1. Download Clover

Download Clover from <http://www.atlassian.com/software/clover/download>.

Unzip the Clover distribution into a directory. This directory will be referred to as `CLOVER_HOME` in this guide.

## 2. Install your Clover license

To install your Clover license file, you need to do one of the following:

- Place the license file next to the Clover jar file (i.e. in `CLOVER_HOME/lib`);  
or:
- Place the license file on the Java Classpath that will be used to run Clover;  
or:
- Place the license file on the file system somewhere, and then set the Java System Property `clover.license.path` to the absolute path of the license file.

Note that the name of the license file must be `clover.license`

## 3. Install clover.jar

[add to Ant's build.xml](#)

**(recommended)**

or

[add to Ant's Classpath](#)

## Related topics

- [Supported Platforms](#)
- [Clover-for-Ant Upgrade Guide](#)
- [Clover Release Notes](#)

## Adding to Ant's build.xml

This is the recommended method of installing Clover.

Adding Clover to your build is done by adding the following to your buildfile (e.g. build.xml):

```
<taskdef resource="cloverlib.xml" classpath="/path/to/clover.jar"/>
```

**i** '/path/to' is the path to the 'clover.jar' file. Hence, a typical example of the 'classpath' parameter might be `classpath=" ../lib/clover.jar"`.

### Checking if Clover is available for the build (optional)

In some cases you may want to check if Clover is available before executing Clover-related targets. For example, you may need to ship the build file to others who may not have Clover installed. To check Clover's availability you can make use of the standard Ant `<available>` task:



```
<target name="-check.clover">
<available property="clover.installed"
classname="com.atlassian.clover.CloverInstr" />
</target>

<target name="guard.noclover" depends="-check.clover" unless="clover.installed">
<fail message="The target you are attempting to run requires Clover, which doesn't
appear to be installed"/>
</target>

<target name="with.clover" depends="guard.noclover">
...

```

## Troubleshooting

- To enable logging of the Clover installation, set the environment variable `ANT_OPTS` to `'-Dclover.debug=true'`
- Run ant with the `-debug` and `-verbose` options
- Certain environments may require the `clover.jar` to be placed directly on Ant's Classpath. Details are outlined [here](#).
- To enable logging of Clover at runtime set the environment variable `-Dclover.logging.level=debug` on the JVM that is running your Clover instrumented code. e.g. the JUnit JVM, the Tomcat JVM.

## NEXT STEP

See [Clover for Ant Best Practices](#)

## Adding to Ant's Classpath



### Note

This is an alternative method of installing Clover, and only applies to certain environments. The normal and recommended method of installing Clover is outlined on [Adding to Ant's build.xml](#).

Below are three options for adding the `clover.jar` to your Ant classpath directly.

### Installing Clover locally for a single user

1. Create a directory `${user.home}/.ant/lib`
2. Copy `clover.jar` to `${user.home}/.ant/lib`



### Note

The location of `${user.home}` depends on your JVM and platform. On Unix systems, `${user.home}` usually maps to the user's home directory. On Windows systems, `${user.home}` will map to something like `C:\Documents and Settings\username\`. Check your JVM documentation for more details.

### Installing Clover at an arbitrary location

You can install and use Clover at an arbitrary location and then refer to it using the `-lib` command line option with Ant:

```
ant -lib CLOVER_HOME/lib buildWithClover
```

(where `CLOVER_HOME` is the directory where Clover was installed.)

### Installing Clover globally into Ant

Copy `clover.jar` into `ANT_HOME/lib` (since all jars in this directory are automatically added to Ant's classpath by the scripts that start Ant).

Alternatively, you can add `CLOVER_HOME/clover.jar` to the `CLASSPATH` system environment variable before running Ant. For information about setting this variable, please consult your Operating System documentation.

#### NEXT STEP

See [Clover for Ant Best Practices](#)

## Clover-for-Ant Upgrade Guide

### General instructions

We've taken care to make upgrading Clover straightforward. Follow these simple steps to upgrade Clover-for-Ant:

#### 1. Replace your existing `clover.jar` with the new `clover.jar`

You can do this by simply replacing the the old `.jar` file with the new `.jar` file.

#### 2. Obtain and install a Clover license (optional)

Installing new license is necessary when you're installing a Clover version released after end of support date of your current license.

#### 3. Delete any existing Coverage database (optional)

Clover's database format may change in newer versions. In such case you'll get a build error with a message informing about database incompatibility. In such case you have to delete old database files. The Clover database is created at the location specified in the `initstring` attribute of `<clover-setup>`.



It's quite common to have Clover-for-Ant configured in such way that database files are removed on every 'ant clean' call; thanks to this incompatible databases can be removed automatically.

### Upgrading from specific releases

Please see the [Clover Release Notes](#) and the [Clover-for-Ant Changelog](#) for version-specific upgrade instructions.

### Clover-for-Ant Changelog

Please also refer to the [Clover-for-Eclipse Changelog](#) and [Clover-for-IDEA Changelog](#).

### Clover-for-Ant Changelog

The changes for the latest version are as follows:

#### Changes in Clover-for-Ant 4.0.0

July 11, 2014

This is a feature release with new HTML report with the ADG (Atlassian Design Guidelines) look. See the [Clover 4.0 Release Notes](#) for more details.

#### Implemented features and fixes

Key	Summary	T	P
CLOV-1164	Drop support for Ant 1.6.x (internal)		
CLOV-1345	Apply ADG in the HTML report		
CLOV-1481	Invalid instrumentation code for test methods inside anonymous inline classes		

CLOV-1512      The "Show methods" link does not open the modal dialog



4 issues

See also change log for Clover-for-Maven2&3, Clover-for-Eclipse, Clover-for-IDEA, Clover-for-Grails.

#### Known major bugs

Key	Summary	T	P	Fix Version/s	Resolution
CLOV-1490	Clover safeEval method is incompatible with Groovy's @CompileStatic annotation			4.0.1	Unresolved

1 issue

## Changes in Clover-for-Ant 3.3.0

**March 31, 2014**

This is a feature release with dedicated support for the Spock framework and JUnit4 Parameterized Tests.

See [Clover 3.3 Release Notes](#) for more details.

#### Implemented features and fixes

Key	Summary	T	P
CLOV-1382	Add lambda toggle to report wizards in Eclipse and IDEA		
CLOV-1256	as a developer I'd like to instrument tests written in the Spock framework		
CLOV-1462	ClassNotFoundException when running tests in IDEA 13.1 RC with Clover enabled		
CLOV-1458	Grails Clover Plugin Causing ConcurrentModification Error		
CLOV-1455	///Clover:OFF does not work with lambdas		
CLOV-1451	Fix unchecked warnings for code instrumented by Clover		
CLOV-1441	Clover plugin doesn't load on IDEA 13 Startup		

7 issues

See also change log for Clover-for-Maven2&3, Clover-for-Eclipse, Clover-for-IDEA, Clover-for-Grails.

#### Known major bugs

Key	Summary	T	P	Fix Version/s	Resolution
CLOV-1490	Clover safeEval method is incompatible with Groovy's @CompileStatic annotation			4.0.1	Unresolved

1 issue

#### Older versions

Looking for older versions? See [Clover-for-Ant Changelog](#) for Clover 3.2.

## Changes in Clover-for-Ant 4.0.0

### Changes in Clover-for-Ant 4.0.0

**July 11, 2014**

This is a feature release with new HTML report with the ADG (Atlassian Design Guidelines) look. See the [Clover 4.0 Release Notes](#) for more details.

**Implemented features and fixes**

Key	Summary	T	P
CLOV-1164	Drop support for Ant 1.6.x (internal)		
CLOV-1345	Apply ADG in the HTML report		
CLOV-1481	Invalid instrumentation code for test methods inside anonymous inline classes		
CLOV-1512	The "Show methods" link does not open the modal dialog		

4 issues

See also change log for Clover-for-Maven2&3, Clover-for-Eclipse, Clover-for-IDEA, Clover-for-Grails.

**Known major bugs**

Key	Summary	T	P	Fix Version/s	Resolution
CLOV-1490	Clover safeEval method is incompatible with Groovy's @CompileStatic annotation			4.0.1	Unresolved

1 issue

## Changes in Clover-for-Ant 3.3.0

### Changes in Clover-for-Ant 3.3.0

**March 31, 2014**

This is a feature release with dedicated support for the Spock framework and JUnit4 Parameterized Tests.

See [Clover 3.3 Release Notes](#) for more details.

**Implemented features and fixes**

Key	Summary	T	P
CLOV-1382	Add lambda toggle to report wizards in Eclipse and IDEA		
CLOV-1256	as a developer I'd like to instrument tests written in the Spock framework		
CLOV-1462	ClassNotFoundException when running tests in IDEA 13.1 RC with Clover enabled		
CLOV-1458	Grails Clover Plugin Causing ConcurrentModification Error		
CLOV-1455	///Clover:OFF does not work with lambdas		
CLOV-1451	Fix unchecked warnings for code instrumented by Clover		
CLOV-1441	Clover plugin doesn't load on IDEA 13 Startup		

7 issues

See also change log for Clover-for-Maven2&3, Clover-for-Eclipse, Clover-for-IDEA, Clover-for-Grails.

**Known major bugs**

Key	Summary	T	P	Fix Version/s	Resolution
CLOV-1490	Clover safeEval method is incompatible with Groovy's @CompileStatic annotation			4.0.1	Unresolved

1 issue

## Clover-for-Maven 2 and 3

## Clover-for-Maven 2 and 3 Documentation

### What is Clover-for-Maven 2 and 3?

Clover-for-Maven 2 and 3 integrates the industry-leading code coverage tool, [Atlassian Clover](#) with the Apache Maven build automation tool. Clover-for-Maven 2 and 3 allows you to easily measure the coverage of your unit tests, enabling targeted work in unit testing — resulting in stability and enhanced quality code with maximal efficiency of effort.

### Getting Started with Clover for Maven 2 and 3

[Download Clover for Maven 2 and 3](#)  
[Quick Start Guide](#)  
[Installation Guide](#)  
[Changelog for Clover-for-Maven 2 and 3](#)

### Using Clover for Maven 2 and 3

[User's Guide](#)

### Maven Site Documentation








### Resources and Support









[Atlassian Answers](#)  
[Clover Knowledge Base](#)  
[FAQ page](#)  
[Technical Support](#)

### Offline Documentation

You can download the Clover documentation in PDF, HTML or XML format.

## Recently Updated

-  [Clover Road Map](#)  
Aug 12, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)
-  [Upgrading third party libraries](#)  
Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)
-  [Updating optimization snapshot file](#)  
Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)
-  [Hacking Clover](#)  
Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)
-  [Part 4 - Test Optimization Tutorial](#)  
Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)
-  [Part 3 - Automating Coverage Checks](#)  
Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)
-  [Part 2 - Historical Reporting](#)  
Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)

-  [Part 1 - Measuring Coverage](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Clover 4.0 Release Notes](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [A side-by-side comparison of the Classic and the ADG HTML report](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Clover Release Notes](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Sonar Clover Plugin](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Clover Command Line Tools](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Clover-for-Grails Changelog](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Configuring method context filters](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)

## About Clover-for-Maven 2 and 3

### Overview

The Clover-for-Maven 2 and 3 plugin allows you to produce Clover code coverage reports from the Maven 2 and 3 build tools. It provides detailed information to highlight areas of low coverage in your project, helping to guide your unit-testing activities.

### Open Source status

This plugin is open source, under the Apache license. See [Developer Guide](#).

### License

The plugin includes a built in evaluation license.

### Support

Please report any issues with the Clover-for-Maven 2 and 3 plugin in the [Clover JIRA project](#) under the "Maven Plugin" component.

### Acknowledgements

The Maven 2 and 3 Clover plugin is derived from the original Maven 2 plugin for Clover 1 written by Vincent Massol. The `groupId` has been changed from `org.apache.maven.plugins` to `com.atlassian.maven.plugins`.

### Maven Site Docs

For documentation presented in the standard Maven format, see the [Maven Site Docs](#).

## Clover-for-Maven 2 and 3 Quick Start Guide

To get started with Clover-for-Maven 2 and 3, carry out the following steps.

### Add `com.atlassian.maven.plugins` in `.m2/settings.xml`

Before you get started, add this to your `.m2/settings.xml` file so you can reference Clover by its short name `clover2`.

```
<pluginGroups>
  <pluginGroup>com.atlassian.maven.plugins</pluginGroup>
</pluginGroups>
```

### Run Clover goals from a command line

The quickest and easiest way to try Clover is from the command line:

```
mvn clean clover2:setup test clover2:aggregate clover2:clover
```

#### Goals:

- *clean* - to ensure that all sources will be recompiled (Clover uses [source-code instrumentation](#))
- *clover2:setup* - to initialize Clover and instrument sources
- *test* - to compile code, run tests and record code coverage
- *clover2:aggregate* - to merge coverage data from a multi-module project
- *clover2:clover* - to generate HTML report for a project

You can find a report in `target/site/clover` directory.

#### Further reading

For more instructions, see the [Clover-for-Maven 2 and 3 User's Guide](#).

For documentation presented in the standard Maven format, see the [Maven Site Docs](#).

## Clover-for-Maven 2 and 3 User's Guide

The Maven 2 and 3 Clover plugin produces Clover reports from Maven 2 and 3 projects.



#### Maven Site Documentation

For documentation presented in the standard Maven format, see the [Maven Site Docs](#).

#### On this page:

- [Basic Usage](#)
- [Configuring instrumentation](#)
- [Configuring reports](#)
- [Configuring a coverage goal](#)
- [Using Test Optimization](#)
- [Working with distributed systems](#)
- [Working with multi-module projects](#)
- [Best practices](#)
- [Using Clover with other Maven plug-ins](#)

## Basic Usage

*How to quickly set up basic Clover configuration in `settings.xml` and `pom.xml` and run Clover's goals from a command line.*

## Configuring instrumentation

*How to set which source files shall be instrumented, control level of instrumentation, set JDK level and location of coverage data.*

## Configuring reports

*How to choose report formats, generate historical reports and customize report content.*

## Configuring a coverage goal

*How to set a coverage level as a quality gate and fail a build if it drops below certain threshold or drops compared to a previous build.*

## Using Test Optimization

Test Optimization saves valuable time in the build and test cycle, by only running tests that cover code which has changed since the last build.

## Working with distributed systems

- [Using Distributed Per-test Coverage](#)
- [Using Clover in various environment configurations](#)
- [Using Clover for web applications](#)

## Working with multi-module projects

You can use the `clover2:aggregate` goal to combine the Clover databases of child projects into a single database at the parent project level.

You can also create a single database for all modules with `singleCloverDatabase` parameter set to `true` in `clover2:setup` goal.

Because of this [Maven bug](#), aggregation of databases occurs before the child databases have been generated, when you use the `site` target.

You can create Clover reports for a multi-module project with the command line `mvn clover2:setup test clover2:aggregate clover2:clover`.

## Best practices

- [Best Practices for Maven](#)
- [Clover Performance Tuning](#)

## Using Clover with other Maven plug-ins

- [Compiling Groovy with GMaven plugin](#)
- [Compiling Groovy with Groovy Eclipse Plugin](#)
  
- [Using Clover via the maven-antrun-plugin](#)
- [Using with Surefire and Failsafe Plugins](#)
  - [Using Clover with Maven + surefire-test + inner test classes](#)
- [Using Clover with the GWT-maven plugin](#)
- [Using Clover with JAXB plugin](#)
- [Using Clover with the maven-bundle-plugin](#)
- [Using Clover with Maven Tycho Plugin](#)

## Basic usage

### Configuring Clover's short name in `.m2/settings.xml`

Before you get started, add this to your `.m2/settings.xml` file so you can reference Clover by its short name `clover2`.

```
<pluginGroups>
  <pluginGroup>com.atlassian.maven.plugins</pluginGroup>
</pluginGroups>
```

### Running Clover goals from the command line



The quickest and easiest way to try Clover is from the command line, for example:


```
mvn clean clover2:setup test clover2:aggregate clover2:clover
```


## Installing Clover in pom.xml

Install Clover-for-Maven 2 and 3 by adding it to your Maven build file (**pom.xml**):

1. Set up your **pom.xml** by adding:

```
pom.xml
<build>
  <plugins>
    ...
    <plugin>
      <groupId>com.atlassian.maven.plugins</groupId>
      <artifactId>maven-clover2-plugin</artifactId>
      <version>${clover.version}</version>
      <configuration>
        <licenseLocation>/path/to/clover.license</licenseLocation>
      </configuration>
    </plugin>
    ...
  </plugins>
</build>
```

 Either change `${clover.version}` to the current Clover version, or define a property in your `pom.xml` that sets this value.

 Clover ships with a 30 day evaluation license. After 30 days you need a valid Clover license file to run Clover. You can obtain a free 30 day evaluation license or purchase a commercial license at <http://my.atlassian.com>. You will need to [set up your licence](#), as a `<licenseLocation>` element in your `pom.xml` configuration file.

2. Now, simply **invoke Clover with Maven** on the command line.

```
mvn clean clover2:setup test clover2:aggregate clover2:clover
```

This will instrument your sources, build your project, run your tests and create a Clover coverage report in the **target/site/clover** directory.

You can also have Clover run as part of your build by [adding Clover's goals in pom.xml](#).

There are four basic parts executed when recording code coverage with Clover.

1. The **clover2:setup** goal will instrument your Java source files.
2. The **test** phase is Maven 2 and 3's standard command for running a unit test phase.
3. The **clover2:aggregate** goal is used for merging coverage data generated by multi-module projects.
4. The **clover2:clover** goal generates an HTML, XML, PDF or JSON report.

## Running Goals via pom.xml

The goals described above can be executed by specifying them in your `pom.xml`.

To generate a Clover report when you run the `site` goal:

```
<project>
  ...
  <reporting>
    <plugins>
      ...
      <plugin>
        <groupId>com.atlassian.maven.plugins</groupId>
        <artifactId>maven-clover2-plugin</artifactId>
        <configuration>
          ...
        </configuration>
      </plugin>
    </plugins>
  </reporting>
  ...
```

To instrument your sources whenever you build:

```
<project>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>com.atlassian.maven.plugins</groupId>
        <artifactId>maven-clover2-plugin</artifactId>
        <configuration>
          ...
        </configuration>
        <executions>
          <execution>
            <phase>generate-sources</phase>
            <goals>
              <goal>instrument</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
```

To include aggregation of child modules:

```

<project>
...
  <build>
    <plugins>
      <plugin>
        ...
        <executions>
          <execution>
            <id>main</id>
            <phase>verify</phase>
            <goals>
              <goal>instrument</goal>
              <goal>aggregate</goal>
            </goals>
          </execution>
          <execution>
            <id>site</id>
            <phase>pre-site</phase>
            <goals>
              <goal>instrument</goal>
              <goal>aggregate</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
...

```

### Using clover2:setup and clover:instrument

**i** The Clover 'instrument' goal (**clover2:instrument**) can be used if you need to actually deploy a project's artifact to production and have Clover run at the same time. This will fork the lifecycle and cause each Clover artifact to contain the `-clover` classifier. It means that the build is performed twice.

The Clover 'setup' goal (**clover2:setup**) performs instrumentation in the main build life cycle, therefore it's not recommended to use it together with 'install' or 'deploy' goals. The benefit of this approach is that build is made only once. Furthermore, `clover2:setup` supports instrumentation of Groovy code (while `clover2:instrument` does not).

## Configuring instrumentation

### Controlling which source files are instrumented

Use configuration elements to exclude and include source files from being instrumented:

```

<configuration>
...
  <includes>
    <include>...ant style glob...</include>
    <include>**/specialpackage/*.java</include>
  </includes>
  <excludes>
    <exclude>**/*Dull.java</exclude>
  </excludes>
</configuration>

```

## Excluding tests from instrumentation

If you don't want to instrument your test classes, add the following to your pom.xml (note that this disables the reporting of per-test coverage as well as reporting of test results in Clover's HTML report):

```
<configuration>
  ...
  <includesTestSourceRoots>>false</includesTestSourceRoots>
</configuration>
```

## Controlling the level of instrumentation

You can define the level that Clover will instrument to (and the respective performance overhead). Valid values are **'method'** level (low overhead) or **'statement'** level (high overhead). The default setting is **'statement'**.

Setting this to **'method'** greatly reduces the performance overhead of running Clover, however limited or no reporting is available as a result. The typical use of the 'method' setting is:

- for [Test Optimization](#) or
- for projects with a large code base (as a rough estimate of coverage).

**To set this value in your pom.xml:**

```
<configuration>
  <instrumentation>method</instrumentation>
</configuration>
```

**To set this value on the Maven command line:**

```
-Dmaven.clover.instrumentation=method
```

The setting above will result in method level only instrumentation; no statement level coverage will be available.

## Configuring code contexts

Clover allows you to exclude [coverage contexts](#) from the coverage report.

To exclude `try` bodies and static initialiser blocks:

```
<configuration>
  ...
  <contextFilters>try,static</contextFilters>
</configuration>
```

To exclude arbitrary statements or methods you can specify one or more custom contexts like so:

```
<configuration>
  <methodContexts>
    <main>(.* )?public static void main\(String\[\] argv\).*</main>
  </methodContexts>
  <statementContexts>
    <log>System.out.println\(.*\);</log>
    <iflog>if.*\(\LOG\.is.*\).*</iflog> <!-- NB: must match entire statement,
including any semicolons. -->
  </statementContexts>
</configuration>
```

\*NB: A method context regexp must match the entire method signature. A statement context regexp must match the full statement, including the ';'.  
 Each one still needs to be 'enabled' via the `<contextFilters/>` element:

```
<configuration>
  ...
  <contextFilters>main,log,iflog</contextFilters>
</configuration>
```

If you are filtering code from your coverage reports, you can keep track of what is filtered using the custom `filteredElements` column. See [Creating custom reports](#) for more information.

### Setting JDK Level

In most cases Clover will autodetect the JDK you are using. If you are building with a 1.5 JDK but have set the `maven-compiler-plugin's` `source` and `target` parameters to use a JDK version of 1.4 you will need to set the Clover JDK level to 1.4:

```
<configuration>
  ...
  <jdk>1.4</jdk>
</configuration>
```

### Setting a Clover flush policy

You can set the Clover [Flush Policy](#) and interval:

```
<configuration>
  ...
  <flushPolicy>threaded</flushPolicy>
  <flushInterval>5000</flushInterval>
</configuration>
```

### Setting the Clover DB location

To specify a particular location for your Clover database:

```
<configuration>
  ...
  <cloverDatabase>/foo/bar</cloverDatabase>
</configuration>
```

and to set a location for the merged database:

```
<configuration>
  ...
  <cloverMergeDatabase>/foo/bar</cloverMergeDatabase>
</configuration>
```

**Do not set these locations explicitly (using absolute path) if you have a multi-module project.**

## Configuring reports

### Choosing Report Formats

The `clover2:clover` goal generates an HTML and an XML report by default. You can use the `generateHtml`, `generatePdf`, `generateXml` and `generateJson` configuration elements to choose which report formats should be produced:

```
<configuration>
  ...
  <generatePdf>true</generatePdf>
  <generateXml>true</generateXml>
  <generateHtml>false</generateHtml>
  <generateJson>false</generateJson>
</configuration>
```

 You may need to run `clover2:aggregate` or `clover2:merge` goals before running `clover2:clover`.

### Getting Information about your Clover Database

The `clover2:log` goal will summarize your Clover database.

### Generating Historical Reports

To include the generation of [historical reports](#) in your Clover reports, add the `generateHistorical` element to your Clover plugin configuration:

```
<configuration>
  ...
  <generateHistorical>true</generateHistorical>
</configuration>
```

That will include your historical savepoints, if any, in the generated report.

To generate a savepoint, run the `clover2:save-history` goal.

To avoid having `mvn clean` remove your savepoints you should set the location of the history directory, which defaults to `$project.build.directory/clover/history`:

```
<configuration>
  ...
  <historyDir>${user.home}/history/${project.artifact}</historyDir>
</configuration>
```

## Creating Custom Reports

It is possible to define an external clover report descriptor file, the same way one can define a site.xml descriptor file. The descriptor file is basically a stripped down Ant file which will be run to produce the reports. All options available in [clover-report](#) can be specified. The default report descriptor used by the maven-clover2-plugin is:

```
<project name="Clover Report" default="current">

  <clover-format id="clover.format" type="{type}" orderBy="{orderBy}"
  filter="{filter}"/>
  <clover-setup initString="{cloverdb}"/>

  <clover-columns id="clover.columns">
    <totalChildren/>
    <avgMethodComplexity/>
    <uncoveredElements format="raw"/>
    <totalPercentageCovered format="longbar"/>
  </clover-columns>

  <target name="historical">
    <clover-report>
      <current outfile="{output}" summary="{summary}">
        <format refid="clover.format"/>
        <testsources dir="{tests}"/>
        <columns refid="clover.columns"/>
      </current>
      <historical outfile="{historyout}" historydir="{history}">
        <format refid="clover.format"/>
        <columns refid="clover.columns"/>
      </historical>
    </clover-report>
  </target>

  <target name="current">
    <clover-report>
      <current outfile="{output}" title="{title}" summary="{summary}">
        <format refid="clover.format"/>
        <testsources dir="{tests}"/>
        <columns refid="clover.columns"/>
      </current>
    </clover-report>
  </target>

</project>
```

This is a very simple Ant file, which defines two known targets: "historical" and "current". If there are no history points saved (via: [clover2:save-history](#)) then only the "current" target will be called. Otherwise, the "historical" target gets called which generates both a historical and current report which are linked together.

To change Clover's default reporting behavior, it is easiest to copy this file and add the changes you require. For a full list of clover-report elements and attributes, please consult the [clover-report](#) documentation.

This file can be stored either on your local file system, or in your maven repository as a classified artifact.

If stored on the file system, set this system property:

```
-Dmaven.clover.reportDescriptor=/path/to/clover-report.xml
```

or specify this element:

```
<reportDescriptor>/path/to/clover-report.xml</reportDescriptor>
```

in the `<configuration>` element for the `maven-clover2-plugin` in your `pom.xml`.

If you wish to keep this descriptor in your maven repository you must specify this system property:

```
-Dmaven.clover.resolveReportDescriptor=true
```

or set this element:

```
<resolveReportDescriptor>true</resolveReportDescriptor>
```

in the `<configuration>` element for the `maven-clover2-plugin` in your `pom.xml`.

The descriptor should be deployed using the "clover-report" classifier. For example:

```
mvn deploy:deploy-file -DgroupId=my.group.id -DartifactId=my-artifact-id
-Dversion=X.X -Dclassifier=clover-report \
-Dpackaging=xml -Dfile=/path/to/file -Durl=[url] -DrepositoryId=[id]
```

## Properties for Custom Reports

### Standard Maven properties

A custom clover report descriptor can access the standard Maven 2 and 3 properties. The following properties are available:

- `project.url`
- `project.version`
- `project.name`
- `project.description`
- `project.id`
- `project.groupId`
- `project.inceptionYear`

In addition to these properties, any additional properties defined in `pom.xml` will also be available.

### Clover configuration properties

Configuration options defined for **clover2:clover** MOJO are available in report descriptor under following names:

- `${cloverdb}` = full path to Clover database, depending on settings:
  - default database location (`<build directory>/clover/clover.db`) or
  - `maven.clover.cloverDatabase` or
  - `maven.clover.cloverMergeDatabase`
- `${output}` = full path to report directory or file:
  - absolute path of `maven.clover.outputDirectory` (HTML/JSON reports) or
  - `maven.clover.outputDirectory + "/clover.pdf"` (PDF report) or
  - `maven.clover.outputDirectory + "/clover.xml"` (XML report)
- `${history}` = `maven.clover.historyDir`
- `${title}` = report title, one of:
  - `maven.clover.title` or
  - project's artifactId + version
  - in case when `maven.clover.cloverMergeDatabase` is set then the "(Aggregated)" word is appended
- `${titleAnchor}` = `maven.clover.titleAnchor`
- `${projectDir}` = project base dir
- `${testPattern}` = `"/src/test/"`



- `${filter}` = `maven.clover.contextFilters`
- `${orderBy}` = `maven.clover.orderBy`
- `${charset}` = `maven.clover.charset`
- `${type}` = `html / pdf / xml / json`
  - depends on `maven.clover.generateHtml / maven.clover.generatePdf / maven.clover.generateXml / maven.clover.generateJson`
  - custom report will be run for each of types enabled
- `${reportStyle}` = style of the report ("adg" / "classic") - since Clover 4.0
- `${span}` = `maven.clover.span`
- `${alwaysReport}` = `maven.clover.alwaysReport`
- `${summary}` = whether to generate a summary, true for PDF report
- `${historyout}` = location of history report
  - `${output}` for HTML
  - `${output}/historical.pdf` for PDF

## Creating custom reports

It is possible to define an external clover report descriptor file, the same way one can define a site.xml descriptor file. The descriptor file is basically a stripped down Ant file which will be run to produce the reports. All options available in [clover-report](#) can be specified. The default report descriptor used by the maven-clover2-plugin is:

```

<project name="Clover Report" default="current">

  <clover-format id="clover.format" type="${type}" orderBy="${orderBy}"
  filter="${filter}"/>
  <clover-setup initString="${cloverdb}"/>

  <clover-columns id="clover.columns">
    <totalChildren/>
    <avgMethodComplexity/>
    <uncoveredElements format="raw"/>
    <totalPercentageCovered format="longbar"/>
  </clover-columns>

  <target name="historical">
    <clover-report>
      <current outfile="${output}" summary="${summary}">
        <format refid="clover.format"/>
        <testsources dir="${tests}"/>
        <columns refid="clover.columns"/>
      </current>
      <historical outfile="${historyout}" historydir="${history}">
        <format refid="clover.format"/>
        <columns refid="clover.columns"/>
      </historical>
    </clover-report>
  </target>

  <target name="current">
    <clover-report>
      <current outfile="${output}" title="${title}" summary="${summary}">
        <format refid="clover.format"/>
        <testsources dir="${tests}"/>
        <columns refid="clover.columns"/>
      </current>
    </clover-report>
  </target>

</project>

```

This is a very simple Ant file, which defines two known targets: "historical" and "current".

If there are no history points saved (via: [clover2:save-history](#)) then only the "current" target will be called. Otherwise, the "historical" target gets called which generates both a historical and current report which are linked together.

To change Clover's default reporting behavior, it is easiest to copy this file and add the changes you require. For a full list of clover-report elements and attributes, please consult the [clover-report](#) documentation.

This file can be stored either on your local file system, or in your maven repository as a classified artifact.

If stored on the file system, set this system property:

```
-Dmaven.clover.reportDescriptor=/path/to/clover-report.xml
```

or specify this element:

```
<reportDescriptor>/path/to/clover-report.xml</reportDescriptor>
```

in the `<configuration>` element for the `maven-clover2-plugin` in your **pom.xml**.

If you wish to keep this descriptor in your maven repository you must specify this system property:

```
-Dmaven.clover.resolveReportDescriptor=true
```

or set this element:

```
<resolveReportDescriptor>true</resolveReportDescriptor>
```

in the `<configuration>` element for the `maven-clover2-plugin` in your **pom.xml**.

The descriptor should be deployed using the "clover-report" classifier. For example:

```
mvn deploy:deploy-file -DgroupId=my.group.id -DartifactId=my-artifact-id
-Dversion=X.X -Dclassifier=clover-report \
-Dpackaging=xml -Dfile=/path/to/file -Durl=[url] -DrepositoryId=[id]
```

### Properties for Custom Reports

#### Standard Maven properties

A custom clover report descriptor can access the standard Maven 2 and 3 properties. The following properties are available:

- `project.url`
- `project.version`
- `project.name`
- `project.description`
- `project.id`
- `project.groupId`
- `project.inceptionYear`

In addition to these properties, any additional properties defined in `pom.xml` will also be available.

#### Clover configuration properties

Configuration options defined for **clover2:clover** MOJO are available in report descriptor under following names:

- `${cloverdb}` = full path to Clover database, depending on settings:
  - default database location (`<build directory>/clover/clover.db`) or
  - `maven.clover.cloverDatabase` or
  - `maven.clover.cloverMergeDatabase`
- `${output}` = full path to report directory or file:
  - absolute path of `maven.clover.outputDirectory` (HTML/JSON reports) or
  - `maven.clover.outputDirectory + "/clover.pdf"` (PDF report) or
  - `maven.clover.outputDirectory + "/clover.xml"` (XML report)
- `${history}` = `maven.clover.historyDir`
- `${title}` = report title, one of:
  - `maven.clover.title` or
  - project's artifactId + version
  - in case when `maven.clover.cloverMergeDatabase` is set then the "(Aggregated)" word is appended
- `${titleAnchor}` = `maven.clover.titleAnchor`
- `${projectDir}` = project base dir
- `${testPattern}` = `**/src/test/**`
- `${filter}` = `maven.clover.contextFilters`
- `${orderBy}` = `maven.clover.orderBy`
- `${charset}` = `maven.clover.charset`
- `${type}` = `html / pdf / xml / json`
  - depends on `maven.clover.generateHtml / maven.clover.generatePdf / maven.clover.generateXml /`

*maven.clover.generateJson*

- custom report will be run for each of types enabled
- `${reportStyle}` = style of the report ("adg" / "classic") - since Clover 4.0
- `${span}` = *maven.clover.span*
- `${alwaysReport}` = *maven.clover.alwaysReport*
- `${summary}` = whether to generate a summary, true for PDF report
- `${historyout}` = location of history report
  - `${output}` for HTML
  - `${output}/historical.pdf` for PDF

## Configuring a coverage goal

### Setting coverage threshold as a quality gate

You can check that your test coverage has reached a certain threshold, and fail the build if it has not by adding a `targetPercentage` tag to your plugin configuration in `pom.xml`:

```
<configuration>
  ...
  <targetPercentage>75%</targetPercentage>
  ...
</configuration>
```

You can then use the `clover2:check` target to examine the Clover database and check that you have reached the coverage threshold.

If you want the build to succeed anyway (printing a warning to your log), use the command line option `-DfailOnViolation=false`.

### Ratcheting Up Coverage

Clover can be configured to fail the build or warn you when the code coverage for a project drops relative to the previous build.

The steps to configure the `maven-clover2-plugin` to do this are as follows:


- specify the `clover2:historyDir` configuration for `clover2:check`
- save a history point using the `clover2:save-history` goal at report time. This will be used by the subsequent build.

You can also optionally specify a `historyThreshold` parameter which is the leeway used by `clover2:check` when comparing the coverage with the previous build.

## Using Test Optimization in Maven

Follow the steps in this document to set up Clover's Test Optimization, which allows targeted testing of only the code which has changed since the last build.

This page contains the basic steps for adding Clover's Test Optimization to an existing Maven configuration.

 **WARNING:** Clover's build optimization in Maven runs in the default build lifecycle - not the forked Clover lifecycle. Please pay attention not to deploy clovered artifacts to your repository. Test Optimization is only recommended for saving time during development - not production deployment.

### Command-line Quick Start

The quickest possible way to start using Test Optimization in Clover-for-Maven 2 and 3 is to run the following command:

```
mvn clover2:setup clover2:optimize test clover2:snapshot
```

By default, the snapshot file gets saved to `${basedir}/.clover/clover.snapshot`. It can be deleted by running the `clover2:clean` goal. You can also specify an alternative location using `-Dmaven.clover.snapshot=/path/to/clover.snapshot`. This file is needed to optimize subsequent builds, so please ensure that it won't be removed between builds (don't keep it in a `/target` directory, for instance).

For further documentation on these goals, see the Maven site docs:

- [clover2:clean](#)
- [clover2:setup](#)
- [clover2:optimize](#)
- [clover2:snapshot](#)

Alternatively, add the following profile to the profiles element in your **pom.xml**.

### Editing pom.xml for Test Optimization

To enable Clover's test optimization functionality, add the following profile to the profiles element in your **pom.xml**:

```
<profiles>
  <profile>
    <id>clover.optimize</id>
    <build>
      <plugins>
        <plugin>
          <groupId>com.atlassian.maven.plugins</groupId>
          <artifactId>maven-clover2-plugin</artifactId>
          <version>4.0.0</version>
          <executions>
            <execution>
              <goals>
                <goal>setup</goal>
                <goal>optimize</goal>
                <goal>snapshot</goal>
              </goals>
            </execution>
          </executions>
        </plugin>
      </plugins>
    </build>
  </profile>
</profiles>
```

This will then make it possible to run the following command:

```
mvn integration-test -Pclover.optimize
```

### Test Optimization In-Action

The first time Clover Test Optimization is used a full test run will be done. You should see the following log message appear in the maven stdout:

```

c:\Work\testcases\tutorial>mvn integration-test -Pclover.optimize
[INFO] Scanning for projects...
[INFO] -----
[INFO] Building Money Test
    ...
[INFO] [clover2:optimize {execution: clover}]
[INFO] Adding fileset: directory=c:\Work\testcases\tutorial\target\clover\src-test-instrumented
, **/*Test.java **/*TestCase.java!, excludes=null
[INFO] Clover was unable to save any time on this optimized test run.
[INFO] Clover included 2 test classes in this run (total # test classes : 2)
[INFO] [surefire:test {execution: default-test}]
[INFO] Surefire report directory: c:\Work\testcases\tutorial\target\surefire-reports

-----
T E S T S
-----
Running com.cenqua.samples.money.MoneyBagTest
Tests run: 22, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.164 sec
Running com.cenqua.samples.money.MoneyTest
Tests taking too long? Try Clover's test optimization.
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.013 sec

Results :

Tests run: 23, Failures: 0, Errors: 0, Skipped: 0

[INFO] [clover2:snapshot {execution: clover}]
[INFO] No span specified, using span of: 6s
[INFO] Saving snapshot to: c:\Work\testcases\tutorial\.clover\clover.snapshot
[INFO] Clover Version 3.1.9, built on January 04 2013 (build-dev)
[INFO] Loaded from: d:\Data\.m2\repository\com\cenqua\clover\clover\3.1.9-SNAPSHOT\clover-3.1.9
[INFO] Clover Evaluation License registered to Clover Maven Plugin.
[INFO] You have 27 day(s) before your license expires.
[INFO] Snapshot file not found, creating new file at c:\Work\testcases\tutorial\.clover\clover.
[INFO] [jar.jar {execution: default-jar}]
[INFO] Building jar: c:\Work\testcases\tutorial\target\money-test-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESSFUL

```

If you then rerun the build, without modifying any source files (and ensuring the snapshot file is not deleted) you should see the following:

```

c:\Work\testcases\tutorial>mvn integration-test -Pclover.optimize
[INFO] Scanning for projects...
[INFO] -----
[INFO] Building Money Test
...
[INFO] [clover2:optimize {execution: clover}]
[INFO] Adding fileset: directory=c:\Work\testcases\tutorial\target\clover\src-test-instrumented
, **/*Test.java, **/*TestCase.java], excludes=null
[INFO] Clover estimates having saved around 3 seconds on this optimized test run. The full test
nds
[INFO] Clover included 0 test classes in this run (total # test classes : 2)
[INFO] [surefire:test {execution: default-test}]
[INFO] Surefire report directory: c:\Work\testcases\tutorial\target\surefire-reports

-----
T E S T S
-----
There are no tests to run.

Results :

Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

[INFO] [clover2:snapshot {execution: clover}]
[INFO] No span specified, using span of: 2s
[INFO] Saving snapshot to: c:\Work\testcases\tutorial\.clover\clover.snapshot
[INFO] Clover Version 3.1.9, built on January 04 2013 (build-dev)
[INFO] Loaded from: d:\Data\m2\repository\com\cenqua\clover\clover\3.1.9-SNAPSHOT\clover-3.1.9
[INFO] Clover Evaluation License registered to Clover Maven Plugin.
[INFO] You have 27 day(s) before your license expires.
[INFO] Updating snapshot 'c:\Work\testcases\tutorial\.clover\clover.snapshot' against Clover da
ses\tutorial\target\clover\clover.db
[INFO] [jar:jar {execution: default-jar}]
[INFO] Building jar: c:\Work\testcases\tutorial\target\money-test-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESSFUL

```

If a source file is modified in any way (including whitespace changes), and you re-run the build, only TestCases that cover the modified file will be run, for instance:

```

c:\Work\testcases\tutorial>mvn integration-test -Pclover.optimize
[INFO] Scanning for projects...
[INFO] -----
[INFO] Building Money Test
...
[INFO] [clover2:optimize {execution: clover}]
[INFO] Adding fileset: directory=c:\Work\testcases\tutorial\target\clover\src-test-instrumented
, **/*Test.java, **/*TestCase.java], excludes=null
[INFO] Clover estimates having saved around 3 seconds on this optimized test run. The full test
nds
[INFO] Clover included 1 test class in this run (total # test classes : 2)
[INFO] [surefire:test {execution: default-test}]
[INFO] Surefire report directory: c:\Work\testcases\tutorial\target\surefire-reports

-----
T E S T S
-----
Running com.cenqua.samples.money.MoneyBagTest
Tests run: 22, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.186 sec

Results :

Tests run: 22, Failures: 0, Errors: 0, Skipped: 0

[INFO] [clover2:snapshot {execution: clover}]
[INFO] No span specified, using span of: 2s
[INFO] Saving snapshot to: c:\Work\testcases\tutorial\.clover\clover.snapshot
[INFO] Clover Version 3.1.9, built on January 04 2013 (build-dev)
[INFO] Loaded from: d:\Data\m2\repository\com\cenqua\clover\clover\3.1.9-SNAPSHOT\clover-3.1.9
[INFO] Clover Evaluation License registered to Clover Maven Plugin.
[INFO] You have 27 day(s) before your license expires.
[INFO] Updating snapshot 'c:\Work\testcases\tutorial\.clover\clover.snapshot' against Clover da
ses\tutorial\target\clover\clover.db
[INFO] [jar:jar {execution: default-jar}]
[INFO] Building jar: c:\Work\testcases\tutorial\target\money-test-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESSFUL

```

By default, the same snapshot file is updated for 10 consecutive builds. On the 10th build, the snapshot file is deleted and recreated. You can adjust this setting via the `fullRunEvery` option on the `clover2:optimize` goal.

TIP: If your terminal supports ANSI escape sequences (OS X, Linux by default), supply the `-Dansi.color=true` property on the command line to have your build optimization in full color.

```
[INFO] [clover2:optimize {execution: clover}]
[INFO] Adding fileset: directory=/Work/testcases/testCloverWithMaven/target/clover/src-test-include
*TestCase.java], excludes=null
[INFO] Clover estimates having saved around 1 minute on this optimized test run. The full test
[INFO] Clover included 0 test classes in this run (total # test classes : 2)
[INFO] [surefire:test {execution: default-test}]
```

#### Related Links

- [Overview of Test Optimization](#)
- [Test Optimization Technical Details](#)
- [Test Optimization Quick Start for Ant](#)
- [Clover for Maven 2 and 3 - Test Optimization Best Practices](#)
- [Using Distributed Per-test Coverage](#)

## Using Distributed Per-test Coverage

This page contains instructions on how to collect per-test coverage from a set of functional tests, which run in multiple JVMs (Java Virtual Machines). This may be necessary when starting a web server with the [Maven Cargo plugin](#) or the [Tomcat plugin](#), for example.

### Why measure per-test coverage?

Clover's per-test coverage data gives you unique insight into how each of your tests exercises the codebase. Clover's per-test reporting gives statement level detail about the behaviour of each test. Furthermore, once you've measured per-test coverage, you can use Clover's powerful new **test optimization** feature, where Clover can choose a smart subset of tests to run for a given code change, saving you the time and hassle of running a full test suite for every change.

On this page:

- [Option 1. Enabling Distributed Coverage at Runtime](#)
  - [Setting a System Property in the Maven Cargo plugin](#)
  - [Setting a System Property in the Maven Surefire plugin](#)
- [Option 2. Activate Distributed Coverage during Instrumentation.](#)
  - [Step 1: Activate the Distributed Coverage Feature](#)
  - [Step 2: Specify the JVM running the Tests as the Server](#)
- [Execution](#)
- [Related Links](#)

To set up collection of per-test coverage from distributed builds, carry out the following steps.

### Option 1. Enabling Distributed Coverage at Runtime

It is recommended, but not necessary, to enable distributed coverage collection at runtime. This can be done by defining the `clover.distributed.coverage` system property in all JVMs running Clovered code, including your Surefire JVM, and the JVM running your web server.

For the following examples, we are using the [Maven Cargo plugin](#) to start the webserver and the [Maven Surefire](#)




plugin to run the tests.

### Setting a System Property in the Maven Cargo plugin

The "clover.distributed.coverage" System Property must be set to "ON" in the Maven Cargo Plugin configuration.

```
<systemProperties>
  <clover.distributed.coverage>ON</clover.distributed.coverage>
</systemProperties>
```

 **TIP:** the *clover.distributed.coverage=ON* takes default settings (host=localhost, port=1198, timeout=5000ms, numClients=0, retryPeriod=1000ms, name=clover.tcp.server). In case when you cannot use default settings, you can pass specific value for any of attributes using the "key=value" syntax passed as *clover.distributed.coverage* value:

- host - host name of the "Clover Server"
- port - port on which the Clover will listen
- numClients - number of "Clover Clients" to connect until server starts test execution
- timeout - connection timeout in milliseconds
- retryPeriod - interval between connection retries in milliseconds
- name - name of the configuration (URL is *host:port/name*)

Example:

```
<systemProperties>
  <clover.distributed.coverage>host=myhost;port=7777;numclients=2</clover.distributed
  .coverage>
</systemProperties>
```

### Setting a System Property in the Maven Surefire plugin

The following System properties must be set in the Maven Surefire Plugin configuration:

- "clover.distributed.coverage" System Property must be set to "ON",
- "clover.server" System Property must be set to "true".

```
<configuration>
  <forkMode>once</forkMode>
  <systemProperties>
    <property>
      <name>clover.server</name>
      <value>true</value>
    </property>
    <property>
      <name>clover.distributed.coverage</name>
      <value>ON</value>
    </property>
  </systemProperties>
</configuration>
```

## Option 2. Activate Distributed Coverage during Instrumentation.

### Step 1: Activate the Distributed Coverage Feature

To add the `<distributedCoverage/>` element to the `maven-clover2-plugin` configuration section, apply the following code:

```
<configuration>
  <distributedCoverage/>
</configuration>
```

This will enable distributed per-test coverage to be collected. Clover running in the JVM that hosts the tests will start a TCP server to do so. By default, it listens on `localhost:1198`.

The `<distributedCoverage>` element takes the following nested elements:

Element name	Description	Required
host	The hostname the test JVM should bind to.	No; defaults to 'localhost'
name	The name of this configuration.	No; defaults to 'tcp-config'
numClients	The number of clients that need to connect to the test server before the tests will continue.	No; defaults to '0'
port	The port the test JVM should listen on.	No; defaults to '1198'
retryPeriod	The amount of time (in milliseconds) to wait before attempting to reconnect in the event of a network failure.	No; defaults to '1000'
timeout	The amount of time (in milliseconds) to wait before a connection attempt will fail.	No; defaults to '5000'


 All attributes are optional.

### Step 2: Specify the JVM running the Tests as the Server

Add the `clover.server` system property to the `maven-surefire-plugin` configuration section, and ensure the `forkMode` parameter is set to 'once':

```
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <configuration>
    <forkMode>once</forkMode>
    <systemProperties>
      <property>
        <name>clover.server</name>
        <value>>true</value>
      </property>
    </systemProperties>
  </configuration>
</plugin>
```

## Execution

 In order to run your tests and generate reports, you might need to copy Clover database (`clover.db`) and

coverage recordings (clover.dbHHHHHHH\_TTTTTTTTTT) between machines. You might also need to provide clover.jar at runtime. It depends on how your environment is configured and especially whether Clover database is accessible via shared network drive. Read the [Using Clover in various environment configurations](#).

### Related Links

- [Working with Distributed Applications](#)
- [Using Test Optimization in Maven](#)
- [Using Clover in various environment configurations](#)

## Using Clover in various environment configurations

- [Introduction](#)
- [Scenario 1 - single-module application executed on several servers independently](#)
- [Scenario 2 - multi-module application executed on several servers independently](#)
- [Scenario 2b - multi-module application executed on several servers independently](#)
- [Scenario 3 - multiple applications executed on several servers in isolation](#)
- [Scenario 3b - multiple applications executed on several servers in partial isolation](#)
- [Scenario 4 - multi-module application with distributed execution on several servers](#)
- [Scenario 5 - multi-module application with history points](#)
- [Decision matrix](#)
- [References](#)

### Introduction


This is a "multi-multi" tutorial showing how to use Clover:

- with projects containing multiple modules,
- running on multiple application servers,
- in multiple test phases (e.g. unit tests, integration, manual testing),
- in multiple test runs (snapshots and history points)
- in distributed environment

If you have questions like:

- should I use cloverDatabase, singleCloverDatabase or cloverMergeDatabase?
- should I declare Clover in master pom.xml or in child modules' pom.xmls as well?
- should I deploy instrumented code on all application servers and run tests at once or sequentially?
- what should I copy (or not copy) to test server?
- should I use distributed.coverage=ON?
- should I use merge clover databases?

then this tutorial is for you 😊

 Confused which scenario you shall use? Have a quick look at the [Decision Matrix](#).

### **Assumptions for all scenarios:**

For the simplicity of the tutorial it's assumed that:

- we have separated machines for build, tests and reporting
- we have a shared network drive accessible from all machines at the same absolute path

In case when your environment is different and:

- build, application or reporting server is the same machine => skip points talking about copying files

- shared network drive is accessible from all machines but at different absolute paths => instrument sources using relative paths, use `clover.initstring.basedir` / `clover.initstring.prefix` at runtime
- shared network drive is not available => copy `clover.db` / recording snapshots / history points between machines

## Scenario 1 - single-module application executed on several servers independently

### Assumptions:

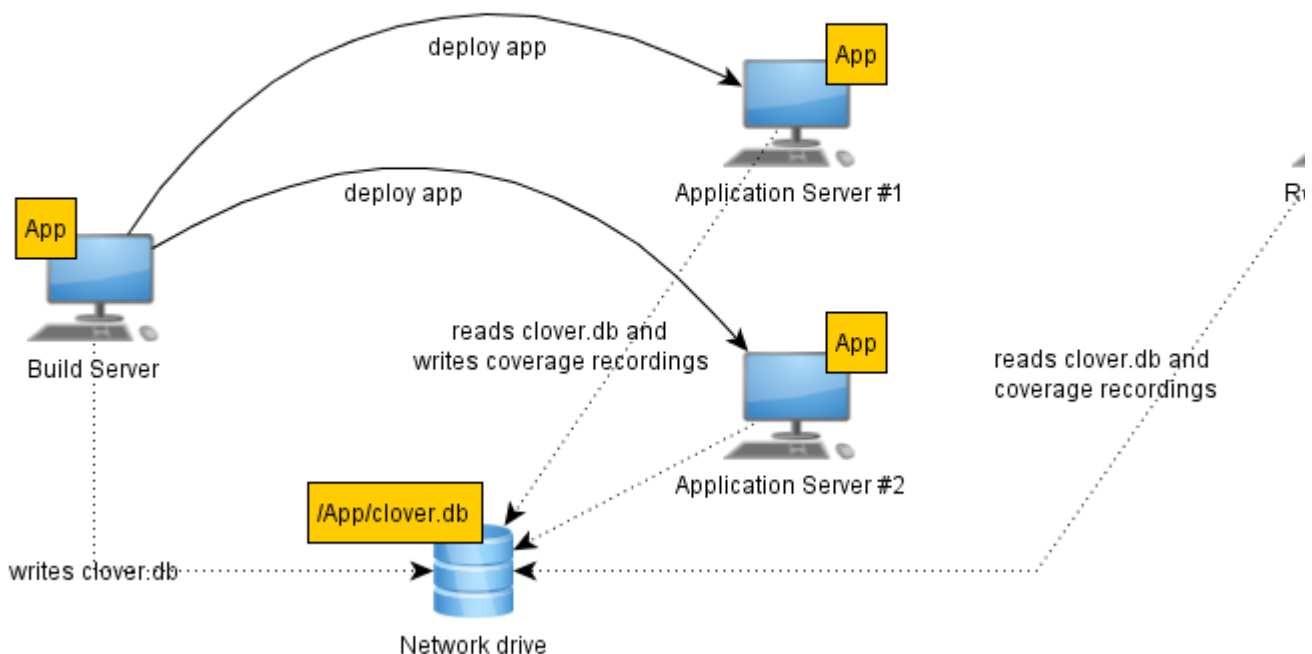
- we have a single-module application (like maven project with one `pom.xml` or ant project with one `build.xml`)
- we deploy instrumented application (the same code) to several application servers
- tests run independently on each application server
- we're not interested in per-test coverage recording
- we want to have a single report showing combined coverage from all application servers

 TIP: this scenario applies also to a case when

- you deploy the application on one server or
- you deploy the application on one server and run multiple instances in separate JVMs or
- you execute your application multiple times (for example unit tests + integration tests + manual testing)

### Overview chart:

Scenario 1 - single-module application executed on several servers independently



### Solution approach:

1) because of fact that

- we're not interested in per-test coverage AND

- we run tests independently

there is **no need** to configure a distributed coverage feature.

2) because of fact that

- we deploy the same code to all application servers AND
- we want to have a single report showing combined coverage from all application servers

we have the same code base, so the same clover.db should be used on each application server and merging clover databases has no sense

we should keep coverage recordings from all servers in the same location as well - note that they will not overwrite each other, because every snapshot file has unique file name

3) because of fact that

- we have a shared network drive accessible from all machines at the same absolute path

it will be very convenient to use it to store clover.db as well as recordings

we will not need to use `-Dclover.initstring` at runtime, because the absolute path will be hardcoded in the instrumented code

#### Steps:

### 1) Build application with Clover

#### a) using Ant

Define `initstring` attribute for `<clover-setup>` or `<clover-instr>` tag, e.g.:

```
<clover-setup initstring="/path/to/network/drive/clover.db">
```

#### b) using Maven

Define `<cloverDatabase>` property for Clover plugin in pom.xml, e.g.:

```
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <configuration>
    <cloverDatabase>/path/to/network/drive/clover.db</cloverDatabase>
  </configuration>
</plugin>
```

```
mvn clean clover2:setup install
```

### 2) Deploy instrumented application to Application Servers

Copy `clover.jar` and your application jar/war. There's no need to copy `clover.db` as it's on a network drive.

### 3) Run tests on Application Servers

Execute your application. There is no need to provide `clover.initstring` parameter as path to `clover.db` database is already hardcoded in instrumented sources.

#### 4) Generate coverage report

##### a) using Ant

Execute `<clover-report>` task; use `initstring` pointing to `clover.db` on a network drive, e.g.:

```
<clover-report initstring="/path/to/network/drive/clover.db">
```

##### b) using Maven

Execute `clover2:clover` goal. Note that you don't need to call `clover2:aggregate` as the project is not multi-module. The `<cloverDatabase>` defined in `pom.xml` will be used for reporting.

```
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <configuration>
    <cloverDatabase>/path/to/network/drive/clover.db</cloverDatabase>
  </configuration>
</plugin>
```

```
mvn clover2:clover
```

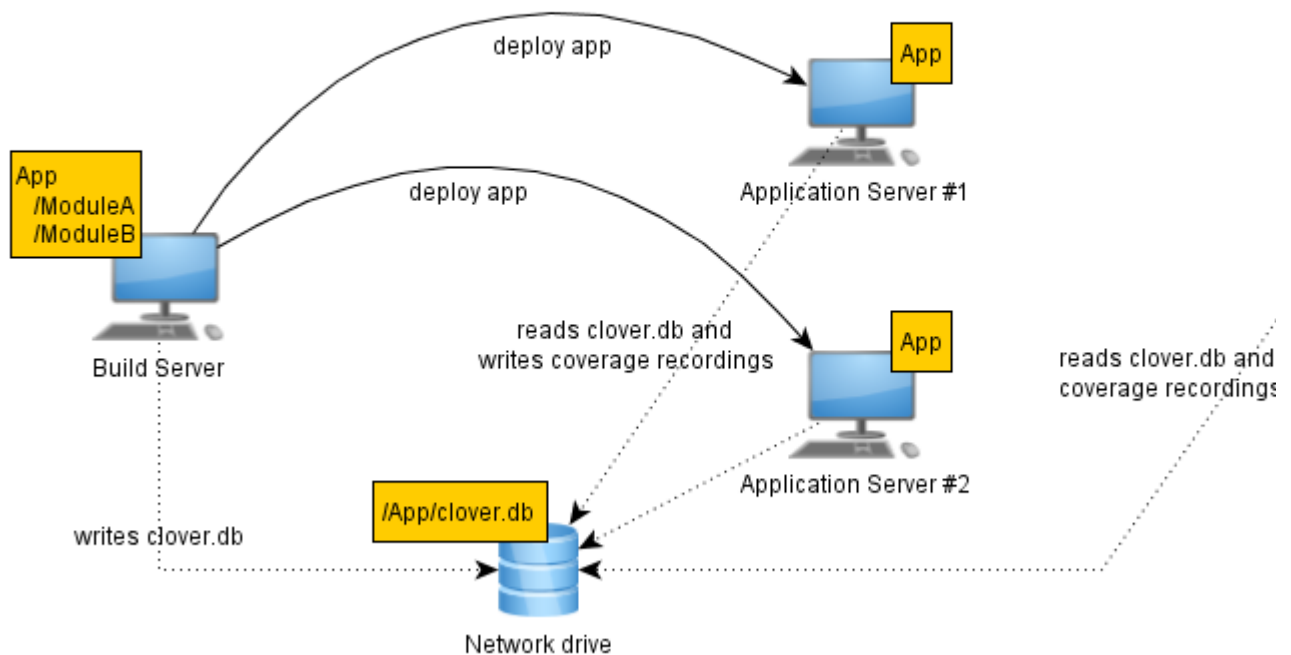
### Scenario 2 - multi-module application executed on several servers independently

#### Assumptions:

- we have a multi-module project
  - like maven project with modules, or
  - ant project with several build.xml files in which `<ant inheritrefs="true" inheritprops="true">` is used (so that all properties are passed)
- we deploy instrumented application (the same code) to several application servers
- tests run independently on each application server
- we're not interested in per-test coverage recording
- we want to have a single report showing combined coverage for all modules from all application servers

#### Overview chart:

## Scenario 2 - multi-module application executed on several servers independently - Approach #



In this scenario there are two approaches possible:

**Approach #1:** use one database for all modules

**Approach #2:** use separate database for every module but under common root

We will focus on **Approach #1** as it's easier to manage. For Approach #2 - see chapter below.

**Solution approach:**

1) because of fact that

- we're not interested in per-test coverage AND
- we run tests independently

there is no need to configure a distributed coverage feature.

2) because of fact that

- we deploy the same code to all application servers AND
- we want to have a single report showing combined coverage from all application servers

we have the same code base, so one clover.db should be used on each application server and merging of clover databases has no sense

we should keep coverage recordings from all servers in the same location as well - note that they will not overwrite each other, because every snapshot file has unique file name

3) because of fact that

- we have a shared network drive accessible from all machines at the same absolute path
- we use single clover database for all modules

we will not need to use `-Dclover.initstring` at runtime, because the absolute path will be hardcoded in the instrumented code

**Steps:****1) Build application with Clover****a) using Ant**

```
<target name="all">
  <!-- Enable clover for top level module -->
  <clover-setup initstring="/path/to/network/drive/clover.db">
  <!-- Build sub-modules ensuring that properties are passed -->
  <ant inheritrefs="true" inheritprops="true" file="sub-module-a/build.xml"
target="all"/>
  <ant inheritrefs="true" inheritprops="true" file="sub-module-b/build.xml"
target="all"/>
</target>
```

**b) using Maven**

- define `<singleCloverDatabase>true</singleCloverDatabase>` in top-level pom.xml; sub-modules will inherit this setting
- define `<cloverDatabase>/path/to/network/drive/clover.db</cloverDatabase>` in top-level pom.xml;
- do **not** define `<cloverDatabase>` attribute in sub-modules, because it would override the `singleCloverDatabase` parameter

```
<!-- TOP LEVEL POM.XML -->
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <configuration>
    <cloverDatabase>/path/to/network/drive/clover.db</cloverDatabase>
    <singleCloverDatabase>true</singleCloverDatabase>
  </configuration>
</plugin>

<!-- CHILD MODULES POM.XML -->
<!-- No need to define anything for Clover, unless you wish to have
some module-specific settings -->
```

**2) Deploy instrumented application to Application Servers**

Copy clover.jar and your application jar/war. There's no need to copy *clover.db* as it's on a network drive.

**3) Run tests on Application Servers**

Execute your application. As we have used `singleCloverDatabase` and `cloverDatabase` pointing to absolute path on a network drive, we don't need to provide *clover.initstring* parameter.

**4) Generate coverage report****a) using Ant**

```
<clover-report initstring="/path/to/network/drive/clover.db">
```



**b) using Maven**

Execute `clover2:clover` goal. Note that you don't need to call `clover2:aggregate` as the project is not multi-module. The `<cloverDatabase>` defined in top-level pom.xml will be used for reporting.

```
mvn clover2:clover
```

**Scenario 2b - multi-module application executed on several servers independently**

This scenario is the same as Scenario 2, but we use separate clover.db database for every module **and** all of them are stored under common root.

**Steps:****1) Build application with Clover****a) using Ant**

This approach is generally not applicable for ant scripts.

**b) using Maven**

Don't define `<cloverDatabase>` and `<singleCloverDatabase>` so that default values will be used (relative path `target/clover/clover.db` and `false`).

**2) Deploy instrumented application to Application Servers**

No differences.

**3) Run tests on Application Servers**

Provide `clover.initstring.basedir=/path/to/top-level-module/dir` at runtime. Alternatively: use `clover.initstring.prefix`.

**4) Generate coverage report**

Before generating report you have to merge all databases. For example:

**a) using Ant**

```
<clover-merge initstring="/path/to/mergedClover.db">
  <cloverDb initstring="/path/to/network/drive/moduleA/clover.db"/>
  <cloverDb initstring="/path/to/network/drive/moduleB/clover.db"/>
</clover-merge>
<clover-report initstring="/path/to/mergedClover.db"/>
```

**b) using Maven**

```
<!-- Top-level pom.xml -->
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <configuration>

  <cloverMergeDatabase>/path/to/network/drive/cloverMerged.db</cloverMergeDatabase>
  </configuration>
</plugin>
```

```
mvn clover2:aggregate clover2:clover
```

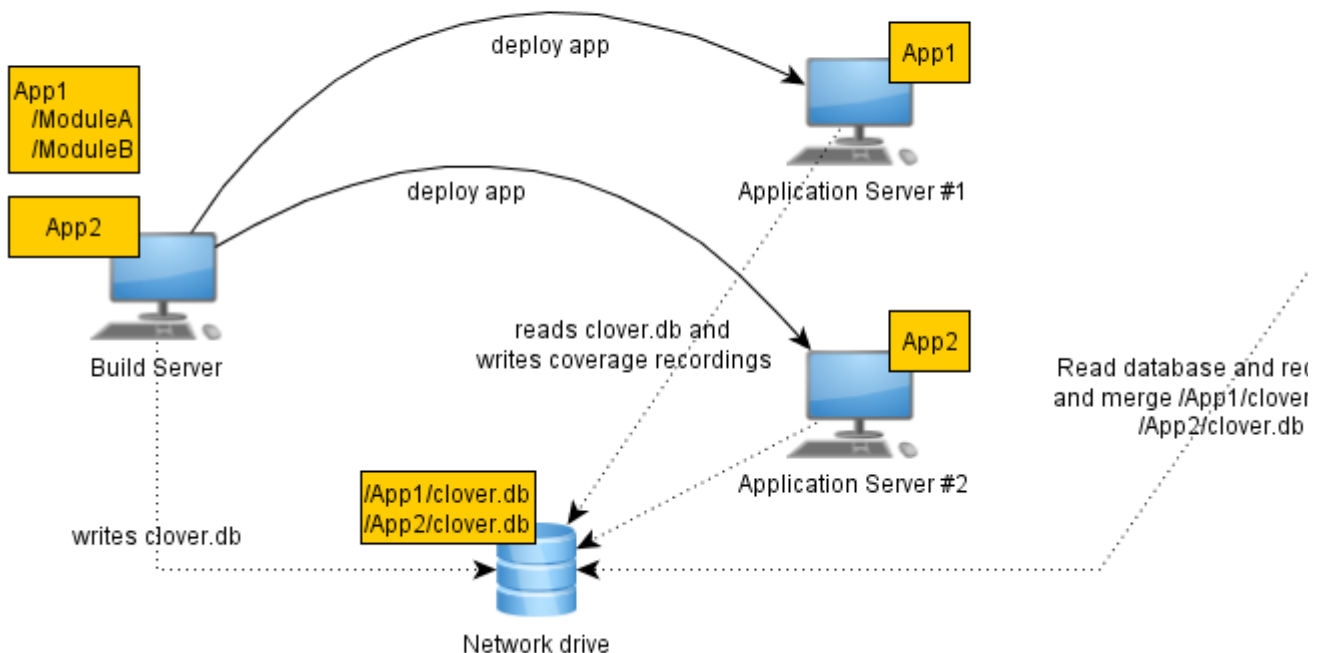
### Scenario 3 - multiple applications executed on several servers in isolation

#### Assumptions:

- we have multiple applications (each of them can have one or more modules)
  - every application has separate code base (i.e. no shared source files)
  - every application is built and instrumented separately (i.e. separate clover.db for every app)
- we deploy instrumented applications to several application servers
  - every application runs on it's own server (i.e. no case when two apps runs in the same JVM)
- tests run independently on each application server and for each application
- we're not interested in per-test coverage recording
- we want to have a single report showing combined coverage for all modules of all applications from all application servers

#### Overview chart:

Scenario 3 - multiple applications executed on several servers in isolation



#### Solution approach:

1) because of fact that

- we're not interested in per-test coverage AND
- we run tests independently

there is no need to configure a distributed coverage feature.

2) because of fact that

- every application has separate code base AND
- every application is built and instrumented separately AND
- we want to have a single report showing combined coverage from all application servers

it implies that

- different clover.db files should be used (one for each application on every application server where application runs)
- merging of clover databases is necessary after tests
- we should keep coverage recordings from all servers in separate locations - one location for every clover.db file

3) because of fact that

- we have a shared network drive accessible from all machines at the same absolute path
- every application runs on it's own server

we will not need to use `-Dclover.initstring` at runtime, because the absolute path will be hardcoded in the instrumented code

### Steps:

#### 1) Build application with Clover

##### a) using Ant

Define *initstring* attribute for `<clover-setup>` or `<clover-instr>` tag, which will point to different directory for every application ,e.g.:

```
<!-- App1 build.xml -->
<clover-setup initstring="/path/to/network/drive/app1/clover.db">

<!-- App2 build.xml -->
<clover-setup initstring="/path/to/network/drive/app2/clover.db">
```

##### b) using Maven

Define `<cloverDatabase>` property for Clover plugin in pom.xml. You can use `singleCloverDatabase` in case your application is multi-module. Databases for all applications **must** be stored under common root (it's a limitation of `clover2:merge` goal; Ant `clover-merge` task is more flexible regarding paths). Example:

```

<!-- App1 pom.xml -->
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <configuration>

<cloverDatabase>/path/to/network/drive/common-root/app1/clover.db</cloverDatabase>
  </configuration>
</plugin>

<!-- App2 pom.xml -->
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <configuration>

<cloverDatabase>/path/to/network/drive/common-root/app2/clover.db</cloverDatabase>
  <singleCloverDatabase>true</singleCloverDatabase> <!-- assuming that app2
is multi-module -->
  </configuration>
</plugin>

```

```
mvn clean clover2:setup install
```

## 2) Deploy instrumented application to Application Servers

Copy `clover.jar` and your application jar/war to proper machines. There's no need to copy `clover.db` as it's on a network drive.

## 3) Run tests on Application Servers

Execute your applications. As every application runs in their own JVM and due to fact that we have used `cloverDatabase` (and optionally `singleCloverDatabase`) pointing to absolute path on a network drive, we don't need to provide `clover.initstring` parameter at runtime, because correct path is hardcoded in instrumented classes.

## 4) Generate coverage report

Before generating report you have to merge all databases. for example:

### a) using Ant

```

<clover-merge initstring="/path/to/network/drive/cloverMerged.db">
  <cloverDb initstring="/path/to/network/drive/app1/clover.db"/>
  <cloverDb initstring="/path/to/network/drive/app2/clover.db"/>
</clover-merge>
<clover-report initstring="/path/to/cloverMerged.db"/>

```

### b) using Maven

```

<!-- Top-level pom.xml -->
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <configuration>

<cloverMergeDatabase>/path/to/network/drive/cloverMerged.db</cloverMergeDatabase>
<!-- output database -->
  <baseDir>/path/to/network/drive/common-root</baseDir> <!-- common root -->
  <includes>*.db</includes> <!-- filename pattern, separated by comma or
space -->
  </configuration>
</plugin>

```

```

mvn clover2:aggregate          # run it for multi-module applications
mvn clover2:merge clover2:clover # run it for final report

```

### Scenario 3b - multiple applications executed on several servers in partial isolation

This scenario is a variation of Scenario 3 in such way, that:

- location of clover.db(s) on test server is different than on build server (so you have to provide/change *initstring* at runtime)
- some applications are executed in the same JVM (as a consequence, you cannot pass *clover.initstring* as JVM argument, because you need a different value for each application).

In such case, you have to:

- instrument these applications using relative paths in `<cloverDatabase>` parameter (Maven) or `<clover-setup initstring="">` (Ant), like below:

```

app1/clover.db
app2/clover.db
app2/moduleA/clover.db
app2/moduleB/clover.db

```

- copy generated clover.db(s) to test server, keeping their relative paths (under a common root), for instance:

```

/path/to/common-root/app1/clover.db
/path/to/common-root/app2/clover.db
/path/to/common-root/app2/moduleA/clover.db
/path/to/common-root/app2/moduleB/clover.db

```

- provide *clover.iniststring.basedir=/path/to/common-root* at runtime

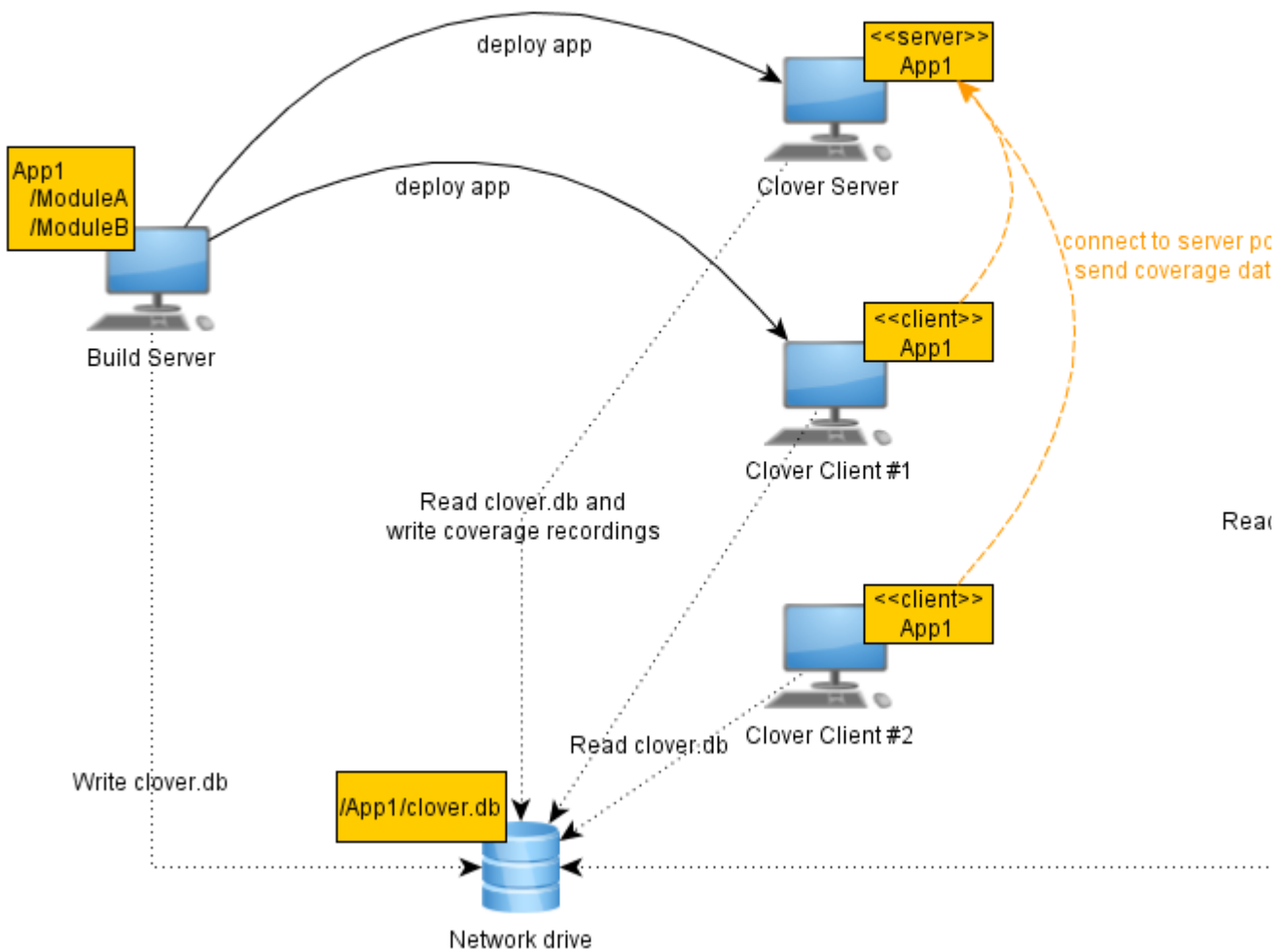
### Scenario 4 - multi-module application with distributed execution on several servers

**Assumptions:**

- we have a multi-module project (like Maven project with modules, or Ant project with several build.xml files with inheriting properties across ant calls)
- we deploy instrumented application (the same code) to several application servers
- unit tests are executed on one <<server>> machine, but these unit tests call business logic on one or more <<client>> machines
- we are interested in per-test coverage recording, showing combined coverage from distributed execution
- we want to have a single report showing combined coverage for all modules from all application servers

**Overview chart:**

Scenario 4 - multi-module application with distributed execution on several servers



**Solution approach:**

1) because of fact that

- we are interested in per-test coverage AND
- we run tests on several machines in parallel (so that a single test case is executed on several nodes)

it implies that

- we must configure a distributed coverage feature
- we must designate a single machine which will be our host of unit tests - see <<server>> on a diagram
- we must open network ports so that <<client>> machines will be able to connect to <<server>> on a designated port number
- we must decide how to launch <<server>> and <<clients>>, for example whether server should hold with test execution until clients are ready

2) because of fact that

- we deploy the same code to all application servers

we don't have to merge clover databases as the same database should be used on each application server

3) because of fact that

- we have a shared network drive accessible from all machines at the same absolute path

we will not need to use *clover.initstring* at runtime, because the absolute path will be hardcoded in the instrumented code

#### Steps:

##### 1) Build application with Clover

- define how many clients will connect to JVM running unit tests;
  - ⚠ a number greater than 0 means that server will hold until all clients are connected before it continues execution; number equal 0 means that tests will start immediately
  - ⚠ you might have a dependency loop so that server waits for clients and clients wait for server - see below
- define server host name (default is localhost) and listening port (default is 1198)
- optionally define connection timeout (in milliseconds), retry period

##### a) using Ant

```
<clover-setup initstring="/path/to/network/drive/clover.db">
  <distributedCoverage host="my.server.com" port="1234" numClients="2"
  timeout="10000"/>
</clover-setup>
```

##### b) using Maven

```

<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <configuration>
    <cloverDatabase>/path/to/network/drive/clover.db</cloverDatabase>
    <singleCloverDatabase>true</singleCloverDatabase> <!-- In case of
multi-module application (optional) -->
    <distributedCoverage>
      <host>my.server.com</host>
      <port>1234</port>
      <numClients>2</numClients>
      <timeout>10000</timeout>
    </distributedCoverage>
  </configuration>
</plugin>

```

## 2) Deploy instrumented application to Application Servers

Copy `clover.jar` and your application jar/war to proper machines. There's no need to copy `clover.db` as it's on a network drive.

## 3) Run tests on Application Servers

### On <<server>> machine

```
java ... -Dclover.server=true
```

### On <<client>> machines

You don't have to provide any runtime options for JVM. They're already compiled in the code.

## Potential problems

**! Server does not wait for clients, despite having `numClients != 0` in build configuration**  
Do not use `-Dclover.distributed.coverage=ON` runtime option if `numClients!=0` was set in instrumentation. The `clover.distributed.coverage` provided at runtime will override `numClients` setting from instrumentation, setting it to 0. As a consequence your tests on server will start immediately, without waiting for clients to connect. It can result in lower or zero coverage.

Instead of this:

- enable distributed coverage option in build file or
- use `-Dclover.distributed.coverage=numClients=N` (where N is a number  $\geq 0$ ) at runtime

**! Execution of tests hangs when with `numClients != 0`**  
It can happen that your server will wait for clients to connect, while clients will wait until server starts unit test execution. This is a typical case for web applications running in container (like Tomcat, JBoss), when your unit test calls a servlet class (e.g. via HTTP request). The issue is as follows:

- unit tests on <<server>> are waiting until all clients are connected (`numClients != 0`) but
- none of the clients will connect until servlet class is loaded in the container, which happens only when first request comes (and it will not come, due to point above)

See [Working with Distributed Applications](#) how to fix this circular dependency.



## 4) Generate coverage report

### a) using Ant

```
<clover-report initstring="/path/to/network/drive/clover.db">
  <current showUniqueCoverage="true" outfile="/path/to/clover/report">
    <format type="html"/>
    <fileset dir="src"/>
  </current>
</clover-report>
```

### b) using Maven

In order to show per-test coverage in the HTML report (*showUniqueCoverage*), you have to use the custom *<reportDescriptor>* in pom.xml and in the report descriptor set the *showUniqueCoverage=true*. For example:

#### report-descriptor.xml

```
<project name="Clover Report" default="current">
  <clover-setup initString="${cloverdb}"/>
  <target name="historical"/>
  <target name="current">
    <clover-report>
      <current showUniqueCoverage="true" outfile="${output}"> <!-- Show
per-test coverage in report -->
        <format type="html"/>
      </current>
    </clover-report>
  </target>
</project>
```

#### pom.xml

```
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <configuration>
    <cloverDatabase>/path/to/network/drive/clover.db</cloverDatabase>
    <outputDirectory>/path/to/clover/report</outputDirectory>
    <reportDescriptor>report-descriptor.xml</reportDescriptor> <!-- Use
custom report -->
  </configuration>
</plugin>
```

```
mvn clover2:clover
```

**i** More information about format of report descriptor can be found here:

- <http://docs.atlassian.com/maven-clover2-plugin/latest/clover-mojo.html#reportDescriptor>
- <https://confluence.atlassian.com/display/CLOVER/clover-report>
- <https://bitbucket.org/atlassian/maven-clover2-plugin> in src/main/resources/default-clover-report.xml

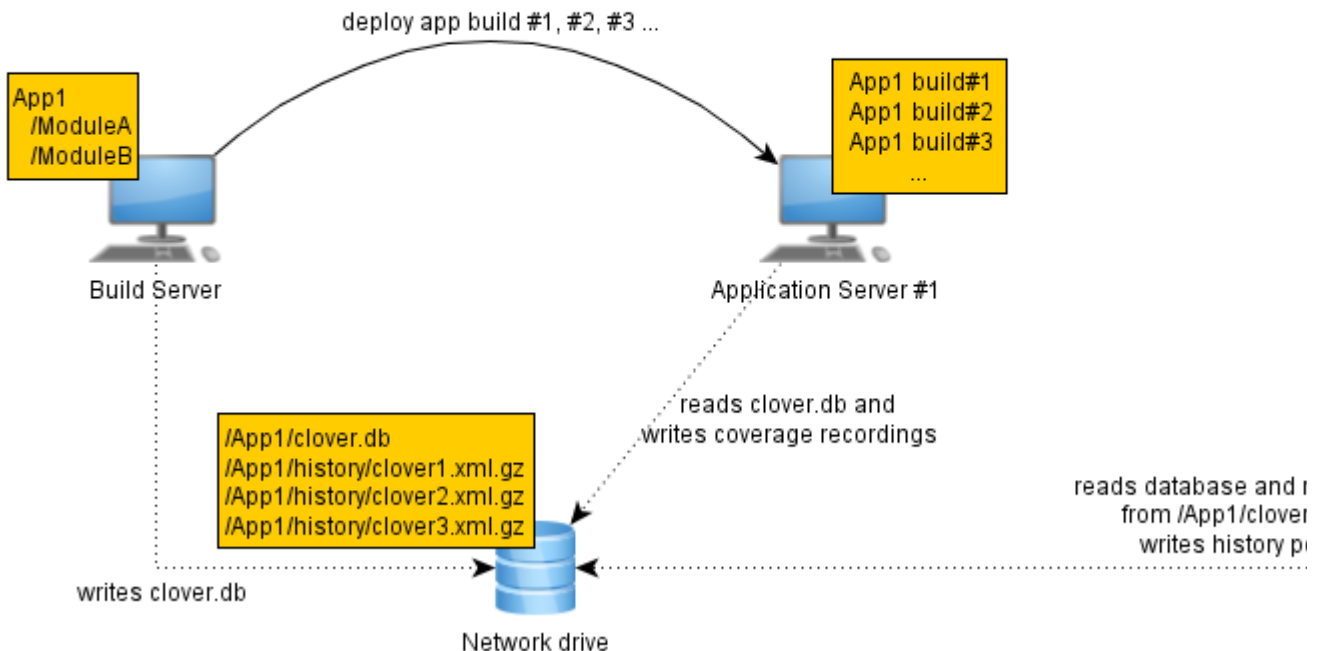
## Scenario 5 - multi-module application with history points

### Assumptions:

- we have a multi-module project (like Maven project with modules, or Ant project with several build.xml files with inheriting properties across ant calls)
- we deploy instrumented application to one application server
- we are not interested in per-test coverage or test optimization
- we want to have a single report showing combined coverage for all modules

### Overview chart:

Scenario 5 - multi-module application with history points



### Solution approach:

- 1) because of fact that we're not interested in per-test coverage or test optimization, there is no need to configure a distributed coverage feature
- 2) because of fact that we have one multi-module application and we want to have a single report showing combined coverage from all modules, one clover.db should be used and there's no need to merge clover databases
- 3) because of fact that we will generate reports with history,
  - we must keep all \*.xml.gz history snapshots between builds and
  - we must delete clover.db and coverage files between builds
- 4) because of fact that
  - we have a shared network drive accessible from all machines at the same absolute path
  - we use single clover database for all modules

we will not need to use *clover.initstring* at runtime, because the absolute path will be hard-coded in the

instrumented code.

#### Steps:

### 1) Build application with Clover

Configuration is the same as for Scenario 2.

#### a) using Ant

```
<target name="all">
  <!-- Enable clover for top level module -->
  <clover-setup initstring="/path/to/network/drive/clover.db">
  <!-- Build sub-modules ensuring that properties are passed -->
  <ant inheritrefs="true" inheritprops="true" file="sub-module-a/build.xml"
target="all"/>
  <ant inheritrefs="true" inheritprops="true" file="sub-module-b/build.xml"
target="all"/>
</target>
```

#### b) using Maven

```
<!-- TOP LEVEL POM.XML -->
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <configuration>
    <cloverDatabase>/path/to/network/drive/clover.db</cloverDatabase>
    <singleCloverDatabase>true</singleCloverDatabase>
  </configuration>
</plugin>

<!-- CHILD MODULES POM.XML -->
<!-- No need to define anything for Clover, unless you wish to have some
module-specific settings -->
```

```
mvn clover2:setup test
```

### 2) Deploy instrumented application to Application Servers

Remove previous version of application and copy clover.jar and your application jar/war. There's no need to copy *clover.db* as it's on a network drive.

### 3) Run tests on Application Servers

Execute your application. As we have used *singleCloverDatabase* and *cloverDatabase* pointing to absolute path on a network drive, we don't need to provide *clover.initstring* parameter at runtime.

### 4) Generate coverage report

#### a) using Ant

```

<clover-report initstring="/path/to/network/drive/clover.db">
  <current outfile="/path/to/clover/report/current" title="Coverage Report">
    <format type="html"/>
    <fileset dir="src"/>
  </current>
  <historical outfile="/path/to/clover/report/historical" title="Historical
Report" historyDir="/path/to/clover/historypoints">
    <format type="html"/>
  </historical>
</clover-report>

<clover-historypoint historyDir="/path/to/clover/historypoints">
  <fileset dir="src"/>
</clover-historypoint>

```

## b) using Maven

```

<!-- Top-level pom.xml -->
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <configuration>
    <cloverDatabase>/path/to/network/drive/clover.db</cloverDatabase>
    <singleCloverDatabase>true</singleCloverDatabase>
    <generateHistorical>true</generateHistorical>
    <generateHtml>true</generateHtml>
    <historyDir>/path/to/clover/historypoints</historyDir>
    <outputDirectory>/path/to/clover/report</outputDirectory>
  </configuration>
</plugin>

```

```
mvn clover2:clover clover2:save-history
```

## Decision matrix

### Instrumentation of source code

<b>Q1</b>	<b>How many applications do you build? (term 'application' means a separate source code and independent build)</b>			
	one application		many applications	
<b>Q2</b>	<b>How many modules application(s) has(have)? (term 'module' means a part of source code, built in the same session as other parts of code)</b>			
	one module	many modules	one module	many modules
<b>Solution</b>				

Recommended	define <i>cloverDatabase</i> in pom.xml  no need to merge	define <i>cloverDatabase</i> in master pom.xml  <i>set singleCloverDatabase=true</i> in master pom.xml  no need to merge	define <i>cloverDatabase</i> in pom.xml  all applications' databases must be stored under a common root (for sake of reporting via clover2:clover)  merge databases from all applications after tests (using clover2:merge)	define <i>cloverDatabase</i> in master pom.xml  <i>set singleCloverDatabase=true</i> in master pom.xml  all applications' databases must be stored under a common root (for sake of reporting via clover2:clover)  merge databases from all applications after tests (using clover2:merge)
Alternative #1	n/a	<i>set singleCloverDatabase=false</i> (or don't define)  merge after tests via clover:aggregate	n/a	<i>set singleCloverDatabase=false</i> (or don't define)  all applications' databases must be stored under a common root (for sake of reporting via clover2:clover) merge after tests via clover:aggregate + clover2:merge

**Application execution**

<b>Q3</b>	<b>Is directory with clover database(s) accessible at the same absolute same path on build</b>				
	yes		no		
<b>Q4</b>	<b>Do you have many applications (i.e. you have many clover databases generated)?</b>				
	one application	many applications		one application	man
<b>Q5</b>	<b>Do you run each application in a separate JVM?</b>				
	n/a	separately	together	n/a	separately
<b>Solution</b>					

Recommended	nothing to do; path to clover.db is already hardcoded in instrumented source code	instrument source code with different <i>initstring</i> for every app; no need to provide <i>clover.initstring</i> at runtime as it's hardcoded in instrumented source code	instrument code with relative path in <i>initstring</i> for all applications; provide common root in <i>clover.initstring.basedir</i> at runtime	provide <i>clover.initstring=/path/to/clover.db</i> at runtime	instrument source code with different <i>initstring</i> for every app; provide different <i>clover.initstring</i> for every application at runtime
-------------	---	---	--	--	--

### Environment configuration

<b>Q6</b>	<b>In case you use different machines for build/test/reporting - do you have a shared network drive?</b>			
	yes		no	
<b>Q7</b>	<b>Do you execute the same application (i.e. binaries produced in one build and using the same clover.db) on several machines?</b>			
	no	yes	no	yes
<b>Solution</b>				
	nothing to do (clover.db created during build is available on test machine too thanks to a network drive; coverage recording files are written to the same directory)	nothing to do (clover.db created during build is accessible on all test machines too; coverage recordings from all test machines are written to the same directory; coverage files generated on different machines will not clash because they're using unique file names)	copy clover.db from build to test server execute tests copy clover.db and coverage files from test to report server	copy clover.db from build to test servers execute tests copy clover.db from build to report server copy coverage files from all test servers to report server into the same directory

### Per-test coverage and test optimization vs distributed coverage

<b>Q8</b>	<b>Are you interested in per-test coverage report or in test optimization?</b>			
	no		yes	
<b>Q9</b>	<b>Do you have distributed application, so that single test case executes application logic on several machines?</b>			
	no	yes	no	yes
<b>Solution</b>				

nothing to do	don't set up distributed coverage feature	just run your application and gather coverage recording files from all machines in one common directory	don't set up distributed coverage feature use <code>showUniqueCoverage=true</code> for reporting	instrument code with <code>distributedCoverage</code> option (define host/port for server hosting unit tests)  at runtime, designate one server where unit tests are executed and run with <code>clover.server=true</code> runtime property  see <a href="#">Using Distributed Per-test Coverage</a>  use <code>showUniqueCoverage=true</code> for reporting
---------------	---	---	---	--

## References

- [About Distributed Per-Test Coverage](#)
- [Working with Distributed Applications](#)
  - [Using Distributed Per-test Coverage with Clover-for-Ant](#)
  - [Using Distributed Per-test Coverage](#)
- [Using Clover for web applications](#)
- [FAQ / Atlassian Answers](#)
  - [When using Clover, why do I get a java.lang.NoClassDefFoundError when I run my code?](#)
  - [No coverage being generated from WebLogic hosted instrumented EAR](#)
  - [Clover on J2EE: Manual Testing](#)

## Using Clover for web applications

- [Introduction](#)
  - [Flushing coverage data in application container](#)
  - [Providing necessary permissions in restricted security environments](#)
  - [Configuring distributed code coverage](#)
  - [Application based on the Google Web Toolkit](#)
  - [Running in the 'test' or 'integration-test' phase](#)
- [Examples](#)
  - [WebApp](#)

## Introduction

This manual presents how to configure Clover in order to get code coverage for web applications - and this issue can actually split into several problem areas:

### *Flushing coverage data in application container*

By default, Clover dumps coverage data at JVM shutdown. In case when your application sever is never shut down, you have to change the flush policy during instrumentation and use the "interval" or "threaded" option. See:

- `clover-instr` or `clover-setup` task and theirs `flushpolicy`, `flushinterval` attributes (for Ant)
- `clover2:instrument` or `clover2:setup` and their `flushPolicy`, `flushInterval` parameters (for Maven)

### *Providing necessary permissions in restricted security environments*

Clover requires permissions to read and write files on disk, read properties, register to JVM shutdown hook. It can happen that your application container has security restrictions applied and you have to grant Clover necessary permissions. See:

- [Working with Restricted Security Environments](#)

### **Configuring distributed code coverage**

In case when:

- your tests are being executed in a different JVM than application code called by them (for example JUnits executed during build calls servlets deployed on Tomcat, or EJBs on JBoss) **and**
- you're interested in [the per-test coverage feature](#) (i.e. which test methods have covered which business methods) or you're interested in [the test optimization feature](#)

then your project must be configured with a **distributed coverage** option. See:

- [Working with Distributed Applications](#)
  - [Using Distributed Per-test Coverage with Clover-for-Ant](#)
  - [Using Distributed Per-test Coverage](#)
- [Using Clover in various environment configurations](#)

### **Application based on the Google Web Toolkit**

GWT performs translation from Java to JavaScript of a client-side code. In this case Clover shall either instrument server-side code only or GWT application shall be executed in a sandbox. See:

- [Using Clover with the GWT-maven plugin](#)

### **Running in the 'test' or 'integration-test' phase**

Depends on whether your build is running tests in 'test' and/or 'integration-test' phase as well as is using maven-surefire-plugin and/or maven-failsafe-plugin, Clover configuration may vary a little. See:

- [Using with Surefire and Failsafe Plugins](#)

## **Examples**

### **WebApp**

Get sources from Bitbucket: <https://bitbucket.org/atlassian/maven-clover2-plugin>, next go to `src/it/webapp`.

It's a simple web application with two servlets named MyServlet and RemoteServlet and a one JSP page. Maven during integration test phase runs unit test which calls servlets locally (by instantiating them) or remotely (in container).

A project build has been configured with a distributed coverage feature in such way that the "Server" (i.e. the JVM where JUnits are launched) will wait until at least one "Client" (i.e. the JVM where business code is executed; in our case inside a Tomcat container) connects to it. Because of fact that our servlet classes will not be loaded by Tomcat class loader until first HTTP request comes, an extra MyServletContextListener class was created. This class does virtually nothing, but due to fact that it's being loaded by class loader during application deployment, the Clover initialization code bundled into this class will run and connect to "Server" on a specified port. As soon as connection is established, "Server" will start execution of unit tests.



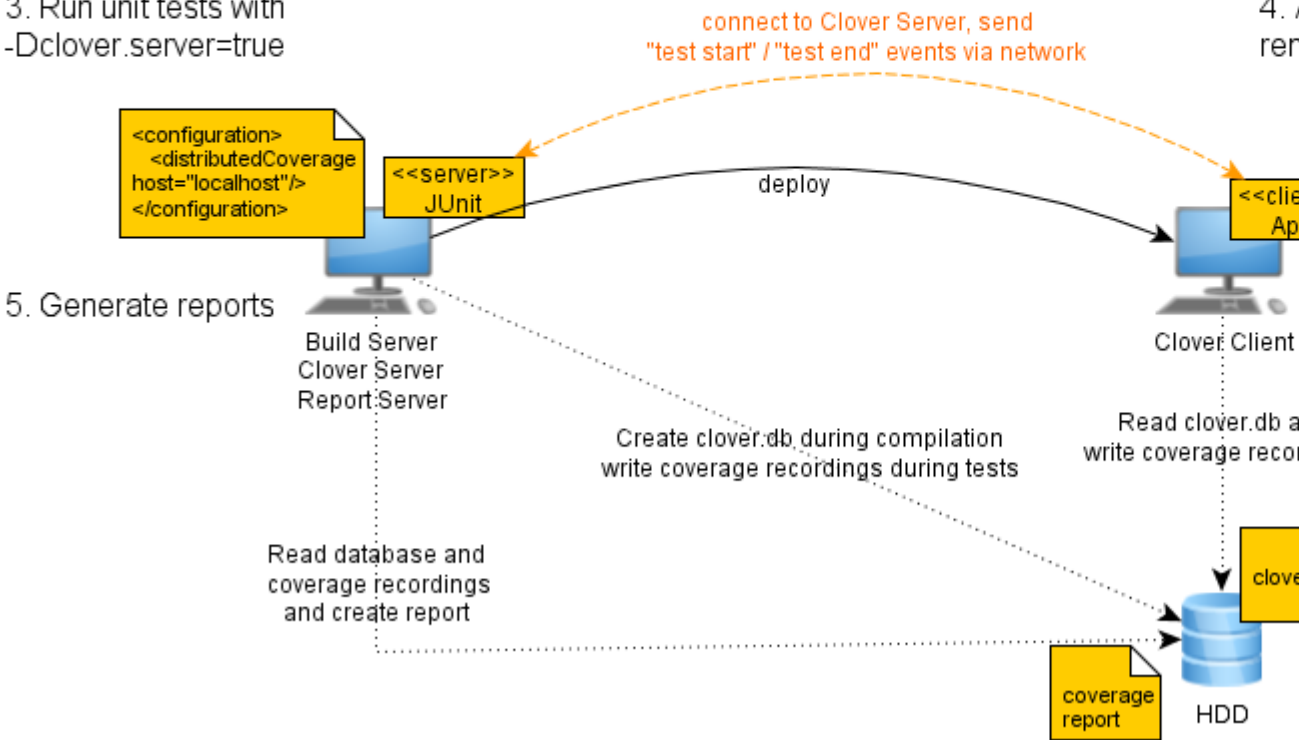
## Distributed Coverage - simplified setup for integration tests during build (V

1. Instrument code with distributed coverage

2. I  
de

3. Run unit tests with -Dclover.server=true

4. ,  
rer



5. Generate reports

Build Server  
Clover Server  
Report:Server

Clover Client

Create clover.db during compilation  
write coverage recordings during tests

Read clover.db and  
write coverage recordings

Read database and  
coverage recordings  
and create report

coverage  
report

HDD

## Best Practices for Maven

This page contains best practices for using Clover for Maven 2 and 3.

On this page:

- Test Optimization in a CI environment
  - Setting up a CI profile
  - Running the Clover goals directly
- Test optimization on the desktop
- Combining build optimization with site coverage reporting
- Test optimization across a multi-module project
- Using clover2:setup for non-forked life cycle
- Cross compilation using Groovy
- Colouring test optimization
- Build Profiles
  - clover.report Profile
  - Clover Optimize Profile
  - Clover Optimize, Report, Log and Check Profile
    - Related Links

### Test Optimization in a CI environment

There are two recommended ways to utilize Clover's test optimization in a CI (Continuous Integration) environment, either using a Profile, or to run the goals directly.

NB. Clover Test Optimization will not work if you have added the maven-clover2-plugin to the default build

section of the pom with an execution binding the 'instrument' goal.

### Setting up a CI profile

1. Add a 'clover.optimize' profile to the project's pom.xml.
2. Create a new '**Gateway**' build plan in your CI server. A 'Gateway' build plan is one that gets run before any others and if successful, triggers any subsequent builds.
3. The gateway plan should execute the **verify** phase, with the 'clover.optimize' profile activated. Example:


```
mvn verify -Pclover.optimize
```

4. If your build plan is configured to do a full clean checkout before each build — you will need to ensure the Clover snapshot file is stored in a location that will not be removed between builds. The following configuration added to the pom.xml is one option:

```
<configuration>
<snapshot>${user.home}/.clover/${groupId}-${artifactId}/clover.snapshot</snapshot>
</configuration>
```

5. Beware however, that this set up will instrument your source and test files and compile them to the usual Maven output location. If you run this command:

```
mvn deploy -Pclover.optimize
```

then you will be deploying class files that have been instrumented by Clover .

### Running the Clover goals directly

Add a new build plan with the following command line:

```
mvn clover2:setup verify clover2:snapshot
```

### Test optimization on the desktop

Running Clover's test optimization locally is very advantageous. This is achieved using the 'clover.optimize' profile that can be activated like so:

```
mvn verify -Pclover.optimize
```

### Combining build optimization with site coverage reporting

Maven2 will merge any executions defined in the default build section of the pom, with those defined in a profile. It is therefore recommended practice to always use two profiles — one for test optimization and one for generating a Clover report when you generate a site. The `clover2:instrument` goal forks the build lifecycle ensuring that Clover instrumented sources are kept completely separate from production sources. This also means that your tests get run twice — which is obviously not desirable in an optimized build.

The '`clover.report`' profile is an example of a build profile to activate when running this command:

```
mvn site -Pclover.report
```

Using separate profiles for site generation and Test Optimization is currently the recommended way to have both a site and a Test Optimization Clover build configured in the same `pom.xml`.

## Test optimization across a multi-module project

By default, Clover will generate a new `clover.db` and `clover.snapshot` file for each module. This means, that if you have tests in module A that cover code in module B, and you modify code in module B, the tests in module A will not be run. You can achieve the desired behaviour however, by configuring Clover to use a single `clover.db` and `clover.snapshot` for the entire project:

```
<configuration>
  <snapshot>${user.home}/.clover/atlassian-plugins-clover.snapshot</snapshot>
  <singleCloverDatabase>true</singleCloverDatabase>
</configuration>
```

If you have many modules, you may need to set `fullBuildEvery` to a value higher than the default of 10. See also `singleCloverDatabase`.

## Using `clover2:setup` for non-forked life cycle

The `clover2:setup` is designed to make integration with integration and functional tests a lot simpler than using the forked lifecycle that comes with `clover2:instrument`. It also has the added advantage of not having to run your tests twice.

Executing `clover2:setup` does the following:

- Instruments all source and test files found in `src/main/java`, `src/test/java`.
- Copies the instrumented source files to `target/clover/src-instrumented`, `target/clover/src-test-instrumented` respectively.
- Redirects the Maven project's source and test directories to `target/clover/src-instrumented`, `target/clover/src-test-instrumented`. Subsequent plug-ins in the build life cycle then use these locations as the source directories.

Therefore, executing the following line will instrument all source and test files, compile the instrumented source files, run all tests and then install the compiled and **instrumented** classes.

```
mvn clover2:setup install clover2:clover
```

**WARNING:** It is not recommended to deploy your Clover instrumented classes to an external Maven repository.

Note: `clover2:setup` will automatically bind itself to the 'process-sources' phase if defined in the goals list of the plugin's executions.

## Cross compilation using Groovy

If you are using cross-compilation with Groovy code, you should ensure that the `maven-clover2-plugin:setup` goal runs before the GMaven Plugin's `gmaven:generateStubs` goal in your `pom.xml`. Otherwise, you may end up with errors when running the Clover-for-Maven 2 plugin.

Alternatively, if you run `clover2:setup` directly from the `mvn` command line, then this Clover goal will run before the `gmaven:generateStubs` goal and you will avoid these errors when cross-compiling Groovy code.

## Colouring test optimization

If your terminal supports ANSI escape codes, run your Maven build with the `-Dansi.color` flag. Currently a few important log messages dealing with Clover's Test Optimization will be logged in colour:

```
[INFO] [clover2:optimize {execution: clover}]
[INFO] Adding fileset: directory=/Work/testcases/testCloverWithMaven/target/clover/src-test-ins
*TestCase.java], excludes=null
[INFO] Clover estimates having saved around 1 minute on this optimized test run. The full test
[INFO] Clover included 0 test classes in this run (total # test classes : 2)
[INFO] [surefire:test {execution: default-test}]
```

## Build Profiles

The following profiles can be used directly in the pom.xml. This avoids the need to modify the ~/.m2/settings.xml file.

### clover.report Profile

```
<profile>
  <id>clover.report</id>
  <build>
    <plugins>
      <plugin>
        <groupId>com.atlassian.maven.plugins</groupId>
        <artifactId>maven-clover2-plugin</artifactId>
        <version>${cloverVersion}</version>
        <executions>
          <execution>
            <id>clover</id>
            <phase>verify</phase>
            <goals>
              <goal>instrument</goal>
              <goal>check</goal>
              <goal>clover</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  <reporting>
    <plugins>
      <plugin>
        <groupId>com.atlassian.maven.plugins</groupId>
        <artifactId>maven-clover2-plugin</artifactId>
        <version>${cloverVersion}</version>
      </plugin>
    </plugins>
  </reporting>
</profile>
```

### Clover Optimize Profile

```
<profile>
  <id>clover.optimize</id>
  <build>
    <plugins>
      <plugin>
        <groupId>com.atlassian.maven.plugins</groupId>
        <artifactId>maven-clover2-plugin</artifactId>
        <version>${cloverVersion}</version>
        <configuration>
<snapshot>${user.home}/.clover/${groupId}-${artifactId}/clover.snapshot</snapshot>
        </configuration>
        <executions>
          <execution>
            <id>clover</id>
            <goals>
              <goal>setup</goal>
              <goal>optimize</goal>
              <goal>snapshot</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</profile>
</profiles>
```

## Clover Optimize, Report, Log and Check Profile

```
<profile>
  <id>clover.all</id>
  <build>
    <plugins>
      <plugin>
        <groupId>com.atlassian.maven.plugins</groupId>
        <artifactId>maven-clover2-plugin</artifactId>
        <configuration>
          <targetPercentage>93%</targetPercentage>
        </configuration>
      </plugin>
    </plugins>
  </build>
</profile>
```

```
<snapshot>${user.home}/.clover/${groupId}-${artifactId}/clover.snapshot</snapshot>
</configuration>
<executions>
  <execution>
    <id>clover</id>
    <goals>
      <goal>setup</goal>
      <goal>optimize</goal>
      <goal>snapshot</goal>
    </goals>
  </execution>
  <execution>
    <phase>verify</phase>
    <goals>
      <goal>clover</goal>
      <goal>log</goal>
      <goal>check</goal>
    </goals>
  </execution>
</executions>
</plugin>
</plugins>
</build>
</profile>
```

#### Related Links

- [Overview of Test Optimization](#)
- [Test Optimization Technical Details](#)
- [Test Optimization Quick Start for Ant](#)
- [Test Optimization Quick Start for Maven 2 and 3](#)

## Compiling Groovy with GMaven plugin

### Configuring the GMaven Plugin for Groovy Support in Maven 2 and 3

If you are using Clover-for-Maven 2 and 3 on Groovy code, you would typically need to define a `plugin` element for the [GMaven Plugin](#) in your `pom.xml` file.

As shown in the example definition below, the GMaven Plugin definition requires the Groovy dependency (`groovy-all`). However, within this dependency, you *must* define a [version of Groovy that Clover supports](#) inside a `version` sub-element. If you omit this `version` element, the GMaven Plugin will default to using Groovy version 1.6.0, which is not compatible with Clover.

```
<properties>
  <groovy.version>1.8.8</groovy.version>
  <gmaven.version>1.5</gmaven.version>
</properties>
...
<plugins>
  <plugin>
    <groupId>org.codehaus.gmaven</groupId>
    <artifactId>gmaven-plugin</artifactId>
    <version>${gmaven.version}</version>
    <configuration>
      <providerSelection>1.8</providerSelection>
    </configuration>
    <dependencies>
      <dependency>
        <groupId>org.codehaus.gmaven.runtime</groupId>
        <artifactId>gmaven-runtime-1.8</artifactId>
        <version>${gmaven.version}</version>
      </dependency>
      <dependency>
        <groupId>org.codehaus.groovy</groupId>
        <artifactId>groovy-all</artifactId>
        <version>${groovy.version}</version>
      </dependency>
    </dependencies>
    <executions>
      <execution>
        <goals>
          <goal>generateStubs</goal>
          <goal>compile</goal>
          <goal>generateTestStubs</goal>
          <goal>testCompile</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
</plugins>
```

## Tips

### **Cross-compilation**

If you use cross-compilation with Groovy code, please refer to the [Cross Compilation using Groovy on the Best Practices for Maven](#) page.

### **Stub generation**

Use the **clover2:setup** goal for Clover instrumentation in case you have generateStubs or generateTestStubs goal declared in GMaven plugin configuration.

In case you use clover2:instrument a build will fail with an error message like:

```
org.apache.maven.BuildFailureException: Compilation failure
... error: duplicate class: com.acme.MyClass
```

A reason is that GMaven will generate stubs twice and will add */generated-sources/groovy-stubs* source root for both the default build life cycle (*/target*) and the Clover's forked build life cycle (*/target/clover*) resulting in

duplicated source files passed to the Maven compiler.

### Setting the providerSelection

Remember to configure a providerSelection parameter. Otherwise build might fail with the following error: "*org.apache.maven.lifecycle.LifecycleExecutionException: Unexpected node: Node[7:1,64,ANNOTATIONS]*" (see [stac koverflow](#)).

### Code example

See <https://bitbucket.org/atlassian/maven-clover2-plugin> repository, **src/it/groovy** directory.

## Compiling Groovy with Groovy Eclipse Plugin

### Compiling Groovy with Groovy Eclipse Plugin

There are several possible ways to configure Groovy Eclipse Plugin - see official <http://groovy.codehaus.org/Groovy-Eclipse+compiler+plugin+for+Maven> page.

Our recommendation is to use configuration similar to the following:

#### Source layout

- keep Java in src/main/java and src/test/java
- keep Groovy in src/main/groovy and src/test/groovy
- do **not** define Groovy source locations for maven-compiler-plugin directly, e.g.:

```
<sourceDirectory>src/main/groovy</sourceDirectory>
<testSourceDirectory>src/test/groovy</testSourceDirectory>
```

- instead of this use:
  - `extensions=true` for groovy-eclipse-compiler (Maven 3) or
  - build-helper-maven-plugin to define additional source roots (Maven 2)

#### Maven 3 POM

##### Maven 3 pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.atlassian.samples</groupId>
  <artifactId>groovy-eclipse-plugin-maven3-sample</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>Groovy Eclipse Plug-in Sample for Maven 3</name>

  <!-- Dependencies for test execution and runtime -->
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.10</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.codehaus.groovy</groupId>
      <artifactId>groovy-all</artifactId>
      <version>1.8.6</version>
    </dependency>
  </dependencies>
```



```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.5.1</version>
      <configuration>
        <!-- Bind Groovy Eclipse Compiler -->
        <compilerId>groovy-eclipse-compiler</compilerId>
        <source>1.6</source>
        <target>1.6</target>
      </configuration>
      <dependencies>
        <!-- Define which Groovy version will be used for build
(default is 2.0) -->
        <dependency>
          <groupId>org.codehaus.groovy</groupId>
          <artifactId>groovy-eclipse-batch</artifactId>
          <version>1.8.6-01</version>
        </dependency>
        <!-- Define dependency to Groovy Eclipse Compiler (as it's
referred in compilerId) -->
        <dependency>
          <groupId>org.codehaus.groovy</groupId>
          <artifactId>groovy-eclipse-compiler</artifactId>
          <version>2.7.0-01</version>
        </dependency>
      </dependencies>
    </plugin>
    <!-- Define Groovy Eclipse Compiler again and set extensions=true.
Thanks to this, plugin will -->
    <!-- enhance default build life cycle with an extra phase which adds
additional Groovy source folders -->
    <!-- It works fine under Maven 3.x, but we've encountered problems with
Maven 2.x -->
    <plugin>
      <groupId>org.codehaus.groovy</groupId>
      <artifactId>groovy-eclipse-compiler</artifactId>
      <version>2.7.0-01</version>
      <extensions>true</extensions>
    </plugin>
    <!-- Configure Clover for Maven plug-in. Please note that it's not
bound to any execution phase, -->
    <!-- so you'll have to call Clover goals from command line. -->
    <plugin>
      <groupId>com.atlassian.maven.plugins</groupId>
      <artifactId>maven-clover2-plugin</artifactId>
      <version>3.1.7</version>
      <configuration>
        <generateHtml>true</generateHtml>
        <historyDir>.cloverhistory</historyDir>
      </configuration>
    </plugin>
  </plugins>

```

```

    </build>

</project>

```

In the build log you'll find messages like:

```

[INFO] --- groovy-eclipse-compiler:2.7.0-01:add-groovy-build-paths
(default-add-groovy-build-paths)
@ groovy-eclipse-plugin-maven3-sample ---
[INFO] Adding /src/main/groovy to the list of source folders
[INFO] Adding /src/test/groovy to the list of test source folders

```

### Maven 2 POM

Build life cycle extension (used by groovy-eclipse-compiler) is not supported in Maven 2.x. Therefore, you can add source locations via build-helper-maven-plugin.

#### Maven 2 pom.xml

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.atlassian.samples</groupId>
  <artifactId>groovy-eclipse-plugin-maven2-sample</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>Groovy Eclipse Plug-in Sample for Maven 2</name>

  <!-- Dependencies for test execution and runtime -->
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.10</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.codehaus.groovy</groupId>
      <artifactId>groovy-all</artifactId>
      <version>1.8.6</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.5.1</version>
        <configuration>
          <!-- Bind Groovy Eclipse Compiler -->
          <compilerId>groovy-eclipse-compiler</compilerId>

```

```

        <source>1.6</source>
        <target>1.6</target>
    </configuration>
    <dependencies>
        <!-- Define which Groovy version will be used for build
(default is 2.0) -->
        <dependency>
            <groupId>org.codehaus.groovy</groupId>
            <artifactId>groovy-eclipse-batch</artifactId>
            <version>1.8.6-01</version>
        </dependency>
        <!-- Define dependency to Groovy Eclipse Compiler (as it's
referred in compilerId) -->
        <dependency>
            <groupId>org.codehaus.groovy</groupId>
            <artifactId>groovy-eclipse-compiler</artifactId>
            <version>2.7.0-01</version>
        </dependency>
    </dependencies>
</plugin>
<!-- Use Build Helper plugin which adds new source folders for Groovy,
without modifying build cycle -->
<!-- (as groovy-eclipse-compiler extensions="true" does -->
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>build-helper-maven-plugin</artifactId>
    <version>1.5</version>
    <executions>
        <execution>
            <id>add-source</id>
            <phase>generate-sources</phase>
            <goals>
                <goal>add-source</goal>
            </goals>
            <configuration>
                <sources>
                    <source>src/main/groovy</source>
                </sources>
            </configuration>
        </execution>
        <execution>
            <id>add-test-source</id>
            <phase>generate-test-sources</phase>
            <goals>
                <goal>add-test-source</goal>
            </goals>
            <configuration>
                <sources>
                    <source>src/test/groovy</source>
                </sources>
            </configuration>
        </execution>
    </executions>
</plugin>
<!-- Configure Clover for Maven plug-in. Please note that it's not
bound to any execution phase, -->
<!-- so you'll have to call Clover goals from command line. -->
<plugin>
    <groupId>com.atlassian.maven.plugins</groupId>
    <artifactId>maven-clover2-plugin</artifactId>
    <version>3.1.7</version>
    <configuration>
        <generateHtml>true</generateHtml>

```

```
        <historyDir>.cloverhistory</historyDir>
    </configuration>
</plugin>
</plugins>
```

```
    </build>

</project>
```

In the build log you'll find messages like:

```
[INFO] [build-helper:add-source {execution: add-source}]
[INFO] Source directory: c:\MyProject\src\main\groovy added.
...
[INFO] [compiler:compile {execution: default-compile}]
[INFO] Using Groovy-Eclipse compiler to compile both Java and Groovy
files

[INFO] [build-helper:add-test-source {execution: add-test-source}]
[INFO] Test Source directory: c:\MyProject\src\test\groovy added.
...
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Using Groovy-Eclipse compiler to compile both Java and Groovy
files
```

### Build Command

Run your build with Clover using a following command (Maven 2 & 3):

```
mvn clean clover2:setup install clover2:aggregate clover2:clover
```

### Tips

- Note that it's possible to bind Clover goals to build phases using the `<executions>` tag in `pom.xml`. See [Clover-for-Maven 2 and 3 User's Guide](#), "Running goals via `pom.xml`" chapter. Just ensure that `clover2:setup` goal is called in the `process-sources` phase the latest.

### Troubleshooting



#### Bug Warning for Clover 3.1.11 and older

Because of bug [CLOV-1144](#) (fixed in 3.1.12) you have to keep your `*.groovy` files **not** in the location for Java code. If you put your Groovy sources into `src/main/java` or `src/test/java` (and this is unfortunately suggested solution on [Groovy-Eclipse+compiler+plugin+for+Maven](#) official page), you will end up with an error message like below:

```
[INFO] BUILD FAILURE
...
[ERROR] Failed to execute goal
com.atlassian.maven.plugins:maven-clover2-plugin:3.1.7:setup (default-cli)
...
Clover has failed to instrument the source files in the
[C:\MyProject\target\clover\src-instrumented] directory
at
org.apache.maven.lifecycle.internal.MojoExecutor.execute(MojoExecutor.java:2
17)
...
[ERROR] For more information about the errors and possible solutions, please
read the following articles:
[ERROR] [Help 1]
http://cwiki.apache.org/confluence/display/MAVEN/MojoExecutionException
*** ERROR: No source files specified

  USAGE: com.cenqua.clover.CloverInstr [OPTIONS] PARAMS [FILES...]
...
```

## References

- [CLOV-1150](#) - Clover does not instrument groovy source files, (groovy-eclipse compiler) when located in src/main/groovy **CLOSED**
- [CLOV-1144](#) - Support \*.groovy files in src/main/java folder for groovy-eclipse-plugin **CLOSED**
- [Atlassian Answers: Does Clover work with the groovy-eclipse-plugin](#)
- [Compiling Groovy with GMaven plugin](#)

## Using with Surefire and Failsafe Plugins

### Introduction

Clover can be used to generate code coverage statistics from practically any kind of test - unit, integration, functional, regression ... - both automatic and manual. The only thing that has to be done is to instrument source code and run it with proper options.

The most frequent Clover usage is to run unit test with code coverage - typically the maven-surefire-plugin is used for this purpose - and thus Clover-for-Maven was designed to cooperate with Surefire plugin "out of the box".

In this short tutorial you will learn how to configure Clover with the Maven Failsafe Plugin, which is used for integration tests.

### Comparison of maven-surefire-plugin and maven-failsafe-plugin

	maven-surefire-plugin	maven-failsafe-plugin
Main purpose	unit tests	integration tests
Bound to build phase	test	pre-integration-test integration-test post-integration-test verify
Build fails in phase	test	verify

Default wildcard pattern	**/Test*.java	**/IT*.java
	**/*Test.java	**/*IT.java
	**/*TestCase.java	**/*ITCase.java
Default output directory	\${basedir}/target/surefire-reports	\${basedir}/target/failsafe-reports

### Setting up Clover with maven-failsafe-plugin (only)

In order to have code coverage statistics from integration tests **and excluding unit tests**, you have to do the following:

1. Disable Surefire plugin, e.g. by setting `<skip>true</skip>` option.
2. Enable Failsafe plugin in your build
  - a. Failsafe plugin requires a test framework provider, e.g. JUnit or TestNG - declare it.
3. Tell Clover to use `target/failsafe-reports` as report directory - use the `<reportDescriptor>` for this.
4. Tell Clover to use test case wildcard pattern for both plugins - use the `<reportDescriptor>` for this.
5. Instrument sources, execute tests and generate reports
  - a. we recommend calling clover goals from command line (as typically projects are multi-module and we have to call `clover2:aggregate`)
  - b. we recommend calling "verify" target instead of "integration-test" (because when you call "integration-test", the Failsafe plugin will not perform post-integration-test cleanup)

### Content of pom.xml

```
<dependencies>
  <!-- Test framework which will be used by Failsafe plugin. Version number is
mandatory -->
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.7</version>
    <scope>test</scope>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>com.atlassian.maven.plugins</groupId>
      <artifactId>maven-clover2-plugin</artifactId>
      <configuration>
        <!-- Use custom report descriptor -->
        <reportDescriptor>clover-report.xml</reportDescriptor>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-failsafe-plugin</artifactId>
      <executions>
        <execution>
          <goals>
            <goal>integration-test</goal>
            <goal>verify</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <configuration>
        <!-- Disable unit tests -->
        <skip>>true</skip>
      </configuration>
    </plugin>
  </plugins>
</build>
```

### **Content of clover-report.xml**



```

<project name="Clover Report" default="current">
  <!-- Initialize Clover -->
  <clover-setup initString="\${cloverdb}"/>
  <target name="historical">
    <!-- Empty as we're not interested in historical reports right now -->
  </target>
  <target name="current">
    <clover-report>
      <current outfile="\${output}" title="\${title}">
        <format type="html"/>
        <!-- Declare naming convention in order to have test classes listed
on the "Tests" tab in HTML report -->
        <testsources dir="src/test">
          <!-- Use Maven-Failsafe-Plugin naming convention -->
          <include name="**/IT*.java"/>
          <include name="**/*IT.java"/>
          <include name="**/*ITCase.java"/>
          <!-- Use Maven-Surefire-Pugin naming convention.
NOTE: Although we don't run unit tests, we still want to have
them on "Tests" tab instead of "Classes" -->
          <include name="**/Test*.java"/>
          <include name="**/*Test.java"/>
          <include name="**/*TestCase.java"/>
        </testsources>
        <!-- Tell Clover to get test results from failsafe. They will be
listed on "Results" tab -->
        <testresults dir="target/failsafe-reports" includes="TEST-*.xml"/>
      </current>
    </clover-report>
  </target>
</project>

```

### Maven command

```
mvn clean clover2:setup verify clover2:aggregate clover2:clover
```

### Setting up Clover with maven-surefire-plugin and maven-failsafe-plugin (combined report)

In order to have combined coverage statistics from unit **and** integration tests, you have to do the following:

1. Set <reportsDirectory> option for both Surefire and Failsafe plugin pointing to the same location.
2. Enable both Surefire and Failsafe plugin in your build.
  - a. Failsafe plugin requires a test framework provider, e.g. JUnit or TestNG - declare it.
3. Tell Clover to use location from point 1 as report directory - use the <reportDescriptor> for this.
4. Tell Clover to use test case wildcard pattern for both plugins - use the <reportDescriptor> for this.
5. Instrument sources, execute tests and generate reports
  - a. we recommend calling clover goals from command line (as typically projects are multi-module and we have to call clover2:aggregate)
  - b. we recommend calling "verify" target instead of "integration-test" (because when you call "integration-test", the Failsafe plugin will not perform post-integration-test cleanup)

### Content of pom.xml

```

<properties>
  <!-- A common location in which a surefire report from 'test' and failsafe
report from
  'integration-test' phase will be stored. See also the clover-report.xml file
which refers
  to this location -->

<surefire.and.failsafe.report.dir>target/test-report</surefire.and.failsafe.report.
dir>
</properties>
<dependencies>
  <!-- Test framework which will be used by Failsafe plugin. Version number is
mandatory -->
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.7</version>
    <scope>test</scope>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>com.atlassian.maven.plugins</groupId>
      <artifactId>maven-clover2-plugin</artifactId>
      <configuration>
        <!-- Use custom report descriptor -->
        <reportDescriptor>clover-report.xml</reportDescriptor>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-failsafe-plugin</artifactId>
      <executions>
        <execution>
          <goals>
            <goal>integration-test</goal>
            <goal>verify</goal>
          </goals>
        </execution>
      </executions>
      <configuration>

<reportsDirectory>${surefire.and.failsafe.report.dir}</reportsDirectory>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <configuration>

<reportsDirectory>${surefire.and.failsafe.report.dir}</reportsDirectory>
      </configuration>
    </plugin>
  </plugins>
</build>

```

### Content of clover-report.xml

```

<project name="Clover Report" default="current">
  <!-- Initialize Clover -->
  <clover-setup initString="${cloverdb}"/>
  <target name="historical">
    <!-- Empty as we're not interested in historical reports right now -->
  </target>
  <target name="current">
    <clover-report>
      <current outfile="${output}" title="${title}">
        <format type="html"/>
        <!-- Declare naming convention in order to have test classes listed
on the "Test" tab in HTML report -->
        <testsources dir="src/test">
          <!-- Use Maven-Failsafe-Plugin naming convention -->
          <include name="**/IT*.java"/>
          <include name="**/*IT.java"/>
          <include name="**/*ITCase.java"/>
          <!-- Use Maven-Surefire-Pugin naming convention -->
          <include name="**/Test*.java"/>
          <include name="**/*Test.java"/>
          <include name="**/*TestCase.java"/>
        </testsources>
        <!-- Tell Clover to get test results directory as defined in
pom.xml. They will be listed on "Results" tab -->
        <testresults dir="target/test-report" includes="TEST-*.xml"/>
      </current>
    </clover-report>
  </target>
</project>

```

### Maven command

```
mvn clean clover2:setup verify clover2:aggregate clover2:clover
```

### Test optimization

Test Optimization feature is available for Surefire plugin. You you have to use:

- `clover2:optimize` goal for maven-surefire-plugin used in 'test' phase
- `clover2:optimizeIntegration` goal for maven-surefire-plugin used in 'integration-test' phase

**⚠ Test Optimization feature for maven-failsafe-plugin is not available yet. See [Hacking Clover / Updating optimization snapshot file](#) if you need a workaround.**

### Sample project

A sample project shows usage of Surefire and Failsafe plugins together.

Checkout code from Bitbucket: <https://bitbucket.org/atlassian/maven-clover2-plugin>

Go to: **[src/it/surefire-and-failsafe-plugins](#)**

Run mvn command with goals as specified in `goals.txt` file in this project. Use Maven 2.x or higher and Java5 or higher.

### References

See also:

- [maven-surefire-plugin](#)
- [maven-failsafe-plugin](#)
- [clover-mojo#reportDescriptor](#)
- [clover-report](#) - *Ant task reference*
- [default-clover-report.xml](#) - *default report configuration used by clover2:clover*
- [Unit Test Results and Per-Test Coverage](#)

## Using Clover with the GWT-maven plugin

For developers working with the Google Web Toolkit (GWT) software development kit and Clover for Maven 2, the maven-clover2-plugin works best with the [gwt-maven-plugin](#).

The maven-googlewebtoolkit2-plugin has known issues that can cause the build to fail if you are building with Clover. As such, the [gwt-maven-plugin](#) is recommended.

For further background reading on the gwt-maven-plugin and interoperability with the maven-clover2-plugin, please also read this [Google Groups discussion](#).

### Instrumentation of source code

Because of the nature of Google Web Toolkit, which translates Java source code (client and shared parts) into a JavaScript, which is later being executed in a web browser, instrumentation of Java sources by Clover requires few technical tricks.

#### Instrumentation of server-side code only

This is a simpler case, as server-side Java sources are being compiled to classes and executed directly in JVM. Therefore the only thing which has to be set up is to enable Clover and limit instrumentation to server-side code packages.

#### Test frameworks

The gwt-maven-plugin provides a JUnit-compatible GWTTestCase which allows to run unit tests using a web browser or htmlunit.

#### How to configure Maven project

1. Add GWT Maven Plugin to **pom.xml** and set desired test mode in <configuration> tag - for example htmlunit allows headless run. Example:

```

<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>gwt-maven-plugin</artifactId>
  <version>2.4.0</version>
  <executions>
    <execution>
      <goals>
        <goal>compile</goal>
        <goal>test</goal>
        <goal>i18n</goal>
        <goal>generateAsync</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <runTarget>GwtExample.html</runTarget>
    <hostedWebapp>${webappDirectory}</hostedWebapp>

    <i18nMessagesBundle>com.atlassian.client.Messages</i18nMessagesBundle>
    <mode>htmlunit</mode>
    <htmlunit>IE7</htmlunit>
  </configuration>
</plugin>


```

2. Add Clover Plugin definition to **pom.xml** and configure which sources should be instrumented with Clover - instrument only server-side code. Example:

```

<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <version>${clover.version}</version>
  <configuration>
    <!-- Instrument only server part -->
    <includes>
      <include>com/atlassian/server/**</include>
    </includes>
  </configuration>
</plugin>

```

 Please note that includes/excludes are supported by the `clover2:setup` goal (i.e. `clover2:instrument` will instrument all sources).

3. By default, the `gwt:test` goal is bound to integration-test phase (and not test), so run maven with `integration-test` or `install` goal. Example:

```
mvn clean clover2:setup install clover2:aggregate clover2:clover
```

#### Instrumentation of server, client and shared code

In this case we cannot use **`gwt:compile`** and **`gwt:test`** goals. The reason is that it would start translation of Java client-side and shared source code to JavaScript, searching for sources of all referenced classes, including the Clover instrumentation, which would cause a build failure.

### Test frameworks

The gwt-test-utils framework provides means to simulate GWT inside JVM, it can intercept all GWT.xyz() method calls, prepare mocks using Mockito or EasyMock etc. The JUnit-compatible GwtTest allows to run unit tests without a web browser.

### How to configure Maven project

1. Add gwt-test-utils dependency to **pom.xml**. Disable **compile** and **test** goals in gwt-maven-plugin. Increase memory for maven-surefire-plugin, if necessary. Example:

```
<dependency>
  <groupId>com.googlecode.gwt-test-utils</groupId>
  <artifactId>gwt-test-utils</artifactId>
  <version>0.38</version>
  <scope>test</scope>
</dependency>

<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>gwt-maven-plugin</artifactId>
  <version>2.4.0</version>
  <executions>
    <execution>
      <goals>
        <!-- <goal>compile</goal> DISABLED -->
        <!-- <goal>test</goal> DISABLED -->
        <goal>i18n</goal>
        <goal>generateAsync</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <runTarget>GwtExample.html</runTarget>
    <hostedWebapp>${webappDirectory}</hostedWebapp>

    <i18nMessagesBundle>com.atlassian.client.Messages</i18nMessagesBundle>
    <mode>htmlunit</mode>
    <htmlunit>IE7</htmlunit>
  </configuration>
</plugin>

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <configuration>
    <argLine>-Xmx512M -XX:MaxPermSize=128M</argLine>
  </configuration>
</plugin>
```

2. Add Clover Plugin definition to **pom.xml**. Use **setup** goal in the initialize phase in order to make sure that source files generated by GWT will be instrumented as well. Example:

```
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <executions>
    <execution>
      <id>clover-initialization</id>
      <phase>initialize</phase>
      <goals>
        <goal>setup</goal>
      </goals>
    </execution>
    <execution>
      <id>clover-reporting</id>
      <phase>install</phase>
      <goals>
        <goal>aggregate</goal>
        <goal>clover</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

3. Run build, for example:

```
mvn clean install
```

### Example project

1. Checkout GwtCloverExample sources from Bitbucket: <https://bitbucket.org/atlassian/maven-clover2-plugin>
2. Go to `src/it/gwt` directory
3. Use at least Java6 and Maven 2.x.
4. The project demonstrates build using three profiles:
  - default** - no Clover instrumentation
  - with.clover.serveronly** - only server-side code is being instrumented by Clover, integration tests are performed with gwt-maven-plugin+htmlunit framework
  - with.clover.everything** - all code is being instrumented by Clover, unit tests are performed with gwt-test-utils and mocking of server services, no integration tests
5. Usage (see also gwt/build.bat file):

```
mvn clean install

mvn -Pwith.clover.serveronly clean install
mvn -Pwith.clover.everything clean install
```
6. See output reports in `<project_dir>/target/site/clover`

## Using Clover with JAXB plugin

### Steps

Add `maven-clover2-plugin` to `<build>` section

- set the `includesAllSourceRoots` property to true (default is false) if you're interested in code coverage for

### JAXB generated sources

- bind the `clover2:setup` goal to process-sources phase (so that it's executed after sources generation)
- bind the `clover2:clover` goal to verify or install phase (so that report is generated after test execution)

```
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <version>${clover.version}</version>
  <configuration>
    <!-- Instrument all source files, also generated by JAXB. Set to false if
    you're not interested in such details (default is false) -->
    <includesAllSourceRoots>true</includesAllSourceRoots>
  </configuration>
  <executions>
    <execution>
      <!-- Call the clover2:setup after JAXB sources are generated but before
      compilation -->
      <id>main1</id>
      <phase>process-sources</phase>
      <goals>
        <goal>setup</goal>
      </goals>
    </execution>
    <execution>
      <!-- Call the clover2:clover and generate report after tests are run
      -->
      <id>main2</id>
      <phase>verify</phase>
      <goals>
        <goal>clover</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

### Add jaxb-api plugin to <dependencies> section

- this is required to run unit tests

```
<dependencies>
  <dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>${jaxb.version}</version>
  </dependency>
</dependencies>
```

### Add maven-jaxb2-plugin to <build> section

- you might need to add `jaxb-xjc` and `jaxb-impl` as plugin's dependencies (see Troubleshooting chapter)
- you can use extensions like `property-listener-injector`



```

<plugin>
  <groupId>org.jvnet.jaxb2.maven2</groupId>
  <artifactId>maven-jaxb2-plugin</artifactId>
  <version>0.8.1</version>
  <executions>
    <execution>
      <goals>
        <goal>generate</goal>
      </goals>
      <configuration>
        <extension>>true</extension>
        <schemaLanguage>DTD</schemaLanguage>
        <schemaIncludes>
          <schemaInclude>*.dtd</schemaInclude>
        </schemaIncludes>
        <bindingIncludes>
          <bindingInclude>*.jaxb</bindingInclude>
        </bindingIncludes>
        <args>
          <arg>-Xinject-listener-code</arg>
        </args>
      </configuration>
    </execution>
  </executions>
  <dependencies>
    <dependency>
      <groupId>com.sun.xml.bind</groupId>
      <artifactId>jaxb-xjc</artifactId>
      <version>${jaxb.version}</version>
    </dependency>
    <dependency>
      <groupId>com.sun.xml.bind</groupId>
      <artifactId>jaxb-impl</artifactId>
      <version>${jaxb.version}</version>
    </dependency>
    <dependency>
      <groupId>org.jvnet.jaxb2-commons</groupId>
      <artifactId>property-listener-injector</artifactId>
      <version>1.0</version>
    </dependency>
  </dependencies>
</plugin>

```

## Sample project

- Checkout code from Bitbucket: <https://bitbucket.org/atlassian/maven-clover2-plugin>
- Go to **src/it/jaxb**
- Run **mvn clean install**
- Generated report will be available in target/site/clover directory

## Troubleshooting

**Problem:** Class not found - javax.activation.DataHandler with an error message like below:

```
[INFO] [jxb2:generate {execution: default}]
[FATAL ERROR] org.jvnet.mjiip.v_2.XJC2Mojo#execute() caused a linkage
error (java.lang.NoClassDefFoundError) and may be
out-of-date. Check the realms:
[...]
[ERROR] FATAL ERROR
[INFO]
-----
[INFO] javax/activation/DataHandler
[INFO]
-----
[INFO] Trace
java.lang.NoClassDefFoundError: javax/activation/DataHandler
    at
com.sun.tools.xjc.model.CBuiltinLeafInfo.<clinit>(CBuiltinLeafInfo.java:
303)
    at
com.sun.tools.xjc.reader.dtd.TDTDReader.<clinit>(TDTDReader.java:442)
[...]
```

**Solution:** Define `com.sun.xml.bind::jaxb-xjc` and `com.sun.xml.bind::jaxb-impl` as <dependencies> for `maven-jaxb2-plugin` as in example chapter above.

**Problem:** When using the Clover Plugin together with the the JAXB2 Plugin, the build fails during instrumentation due to an unresolved clover.jar with an error message like below:

```
[ERROR] BUILD ERROR
[INFO]
-----
[INFO] Error configuring:
org.jvnet.jaxb2.maven2:maven-jaxb2-plugin. Reason: Error evaluating
plugin parameter expression: project.compileClasspathElements
[INFO]
-----
[INFO] Trace
  org.apache.maven.lifecycle.LifecycleExecutionException: Error
configuring: org.jvnet.jaxb2.maven2:maven-jaxb2-plugin.
Reason: Error evaluating plugin parameter expression:
project.compileClasspathElements
  at
org.apache.maven.lifecycle.DefaultLifecycleExecutor.executeGoals(Default
LifecycleExecutor.java:707)
  [...]
Caused by:
org.apache.maven.artifact.DependencyResolutionRequiredException:
Attempted to access the artifact
>>>>com.atlassian.clover:clover:jar:X.Y.Z:compile<<<<;
which has not yet been resolved
  at
org.apache.maven.project.MavenProject.addArtifactPath(MavenProject.java:
1906)
  [...]
```

**Solution:** Define a dependency to *com.atlassian.clover::clover* in your POM, for example:

```
<dependency>
  <groupId>com.atlassian.clover</groupId> <!-- note: com.cenqua.clover for 3.x
versions -->
  <artifactId>clover</artifactId>
  <version>4.0.0</version>
</dependency>
```

## Using Clover with Maven + surefire-test + inner test classes

If you use Maven with the surefire-test plugin, its default filter setting for searching test classes is to skip inner classes:

```
<excludes>
  <exclude>**/*$*</exclude>
</excludes>
```

In case when you have inner classes defined in your JUnit TestCases and you have configured a Surefire plugin to run your inner classes as well, you might get an error like this:

```

-----
Test set: TestUtils$__CLR2_6_34a4agh7gevmc
-----
Tests run: 2, Failures: 0, Errors: 2, Skipped: 0, Time elapsed: 0.094 sec <<<
FAILURE!
  initializationError(TestUtils$__CLR2_6_34a4agh7gevmc) Time elapsed: 0.016 sec <<<
ERROR!
  java.lang.Exception: Test class should have exactly one public constructor
  at
  org.junit.runners.BlockJUnit4ClassRunner.validateOnlyOneConstructor(BlockJUnit4Cla
  ssRunner.java:143)

```

This is because Clover generates inner class for each class (test or application code). In order to fix a problem, you have to change your **pom.xml** and filter out from your test scope any inner classes beginning with **\_\_CLR**.

For example:

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <properties>
        <excludes>
          <exclude>**/*$__CLR*</exclude>
        </excludes>
      </properties>
    </plugin>
  </plugins>
</build>

```

## Using Clover with the maven-bundle-plugin

This page contains instructions on how to use Clover with the maven-bundle-plugin.

The following configuration is required to ensure that the `clover.jar` and any instrumented source files are ignored by the maven-bundle-plugin.

### Procedure

Carry out the following steps.

1. Make the bundle plugin process instrument the class files correctly
2. Ensure the Clover artifact is not embedded in the bundle.

### Example

Here, we are configuring `pom.xml` for the maven-bundle-plugin:

```

<Import-Package> <!-- Make the bundle plugin process instrumented class files
correctly -->
  com_*;resolution:=optional
</Import-Package>
<Embed-Dependency>artifactId=!clover</Embed-Dependency> <!-- Ensure the clover
artifact is not embedded in the bundle -->

```

## Using Clover via the maven-antrun-plugin

Any of Clover's Ant Tasks may be used directly from within Maven by using the [maven-antrun-plugin](#).

Specifically, if you wanted to use the [clover-check](#) task to ensure that a particular package maintains a given coverage percentage, you could use the following configuration in Maven:

```
<profile>
  <id>clover.check</id>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-antrun-plugin</artifactId>
        <dependencies>
          <dependency>
            <groupId>com.atlassian.clover</groupId>
            <artifactId>clover</artifactId>
            <version>4.0.0</version> <!-- Ensure you
use the same version as the maven-clover2-plugin -->
          </dependency>
        </dependencies>
        <executions>
          <execution>
            <phase>verify</phase>
            <configuration>
              <tasks>
                <property
name="clover.license.path" location="\${user.home}/clover.license"/>
                <!-- Change this to point to your license -->
                <taskdef
resource="cloverlib.xml" classpathref="maven.plugin.classpath"/>
                <clover-setup
initString="\${project.build.directory}/clover/clover.db"/>
                <clover-check
filter="\${clover.filter}" haltOnFailure="true">
                  <package name="com.mypkg"
target="100%"/> <!-- Check that com.mypkg always has 100% code coverage
-->
                </clover-check>
              </tasks>
            </configuration>
            <goals>
              <goal>run</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</profile>
```

## Using Clover with Maven Tycho Plugin

### Introduction

Unfortunately, we cannot use direct integration of Tycho plugin with Clover using standard 'clover2:setup' or 'clover2:instrument' goals. The reason is Maven Tycho Plugin ignores Maven's source folders definitions.

Instead of this, Tycho Plugin reads source folders locations from Eclipse configuration files (like build.properties). As a consequence it does not see files instrumented by Clover, which are stored in the *target/src-instrumented* directory.

## Workaround

However, we can use a following trick, which is very similar to one described in [Instrumenting RCP Application / Approach#1](#) : Instrument source code manually:

### 1) Instrument all source files manually

The idea is to replace original sources with the instrumented version, still preserving the original project structure.

Use `CloverInstr` command line tool or `clover-instr` Ant task or `clover2:instrument` to instrument sources manually - see script in **Appendix 1**.

💡 Remember to put instrumented sources in another location, i.e. not in your original workspace. 😊

Clover Database (clover.db) will be created during this process.

### 2) Build instrumented project

Build will use Maven Tycho Plugin to package everything. Because of fact that instrumented sources contain calls of Clover classes, you must have the Clover JAR available on classpath during compilation. The easiest way is to add `com.atlassian.clover:clover` dependency in the `extraClasspathElements` parameter of the `tycho-compiler-plugin` in a top-level module (see Appendix).

### 3) Run tests

Run any kind of tests - JUnit, manual ... - just to get coverage data. Coverage recordings will be stored in the same directory where Clover Database is located.

Note that you must have the Clover runtime available during execution. The best is to add it to Java Xbootclasspath in order to ensure that Clover JAR is loaded before any OSGI bundle. If you are running:

- unit tests via tycho-surefire-plugin then add `<argLine>-Xbootclasspath/a:/path/to/clover-X.Y.Z.jar</argLine>` in the Tycho Surefire Plugin configuration section (see Appendix)
- manual tests in standalone product then add `-Xbootclasspath/a:/your/path/to/clover.jar` as JVM argument in Eclipse configuration file (eclipse.ini / config.ini)

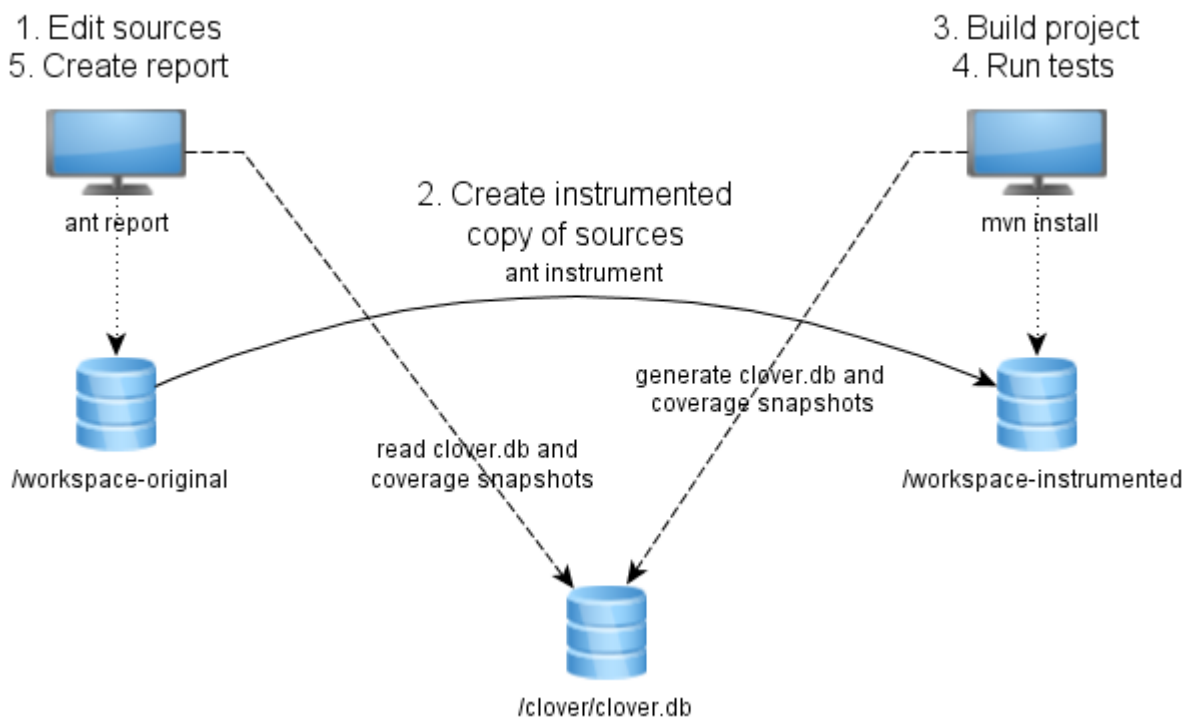
### 4) Generate report

Run `HtmlReporter` command line tool or `clover-report` Ant task or the `clover2:clover` Maven goal in order to generate report.

## Appendix 1

### *Workbench configuration*

The diagram below shows how work with manually instrumented sources (Approach #1) can be organized. A location of Clover database is configured in Ant script.



### Code sample

You can clone examples provided by Tycho team: `git clone http://git.eclipse.org/gitroot/tycho/org.eclipse.tycho-demo.git`

Go to "itp01" sample, rename it to "original\_project", use Ant script from below and save it in directory above "original\_project". Follow instructions below.

### Ant script

Sample Ant script which instruments all \*.java files from *project.original.dir* and puts them into *project.instrumented.dir*, preserving original directory structure. It copies all non-java files as well.

```
<project default="instrument">
  <property name="clover.jar" location="${user.home}/clover.jar"/>
  <property name="ant-contrib.jar"
location="${user.home}/ant-contrib-1.0b3.jar"/>
  <property name="project.original.dir" location="original_project"/>
  <property name="project.instrumented.dir" location="instr_project"/>
  <property name="project.clover.db"
location="${project.instrumented.dir}/.clover/clover.db"/>

  <taskdef resource="cloverlib.xml" classpath="${clover.jar}"/>
  <taskdef resource="net/sf/antcontrib/antlib.xml"
classpath="${ant-contrib.jar}"/>

  <target name="_instrument-dir">
    <!-- Use double-slash for windows paths -->
    <propertyregex property="original.dir.quoted"
input="${project.original.dir}" regexp="\" replace="\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\" global="true"/>
    <propertyregex property="relative.dir" input="${source.dir}"
regexp="${original.dir.quoted}(.*)" select="\1"/>
    <echo message="Instrumenting ${source.dir} into
${project.instrumented.dir}${relative.dir}"/>
  </target>
</project>
```

```

    <echo message="Clover database is ${project.clover.db}"/>
    <clover-instr destdir="${project.instrumented.dir}${relative.dir}"
initstring="${project.clover.db}">
      <fileset dir="${project.original.dir}${relative.dir}">
        <include name="**/*.java"/>
      </fileset>
    </clover-instr>
  </target>

  <target name="instrument">
    <!-- Cleanup from previous run -->
    <delete dir="${project.instrumented.dir}"/>
    <!-- Find all source directories, for each of them call clover-instr.
Please note that we cannot use sth like:
      <clover-instr srcdir="${project.original.dir}"
destdir="${project.instrumented.dir}" initstring="${project.clover.db}">
        directly, because clover-instr does not recreate original directory
structure, but puts everything
        under one destdir root.
    -->
    <foreach target="_instrument-dir" param="source.dir" inheritall="true"
inheritrefs="true">
      <path>
        <!-- Define all package roots here -->
        <dirset dir="${project.original.dir}">
          <include name="**/src"/>
          <include name="**/test"/>
        </dirset>
      </path>
    </foreach>

    <!-- Copy all other non-java files as well -->
    <echo message="Copying other files from ${project.original.dir} to
${project.instrumented.dir}"/>
    <copy todir="${project.instrumented.dir}">
      <fileset dir="${project.original.dir}">
        <exclude name="**/*.java"/>
      </fileset>
    </copy>

    <!-- Now we can build it under Tycho. Don't even try to read instrumented
sources ;-) -->
    <echo message="INSTRUMENTATION DONE. Run Maven build in
${project.instrumented.dir}"/>
  </target>

  <target name="report">
    <clover-report initstring="${project.clover.db}">
      <current outfile="current.html">
        <format type="html"/>
      </current>
      <current outfile="current.xml">
        <format type="xml"/>
      </current>
    </clover-report>
  </target>

```



```

    </clover-report>
  </target>
</project>

```

### Modified top-level pom.xml

You must have the Clover artefact available during compilation by Tycho. Modify the *tycho-compiler-plugin* configuration and add *com.atlassian.clover:clover* to *<extraClasspathElements>* option.

If necessary, add also Clover dependency to *tycho-surefire-plugin* as JVM argument.

```

<build>
  <plugins>
    <plugin>
      <groupId>org.eclipse.tycho</groupId>
      <artifactId>tycho-compiler-plugin</artifactId>
      <version>0.15.0</version>
      <configuration>
        <extraClasspathElements>
          <!-- Use the same Clover version as for source instrumentation -->
          <dependency>
            <groupId>com.atlassian.clover</groupId>
            <artifactId>clover</artifactId>
            <version>4.0.0</version>
          </dependency>
        </extraClasspathElements>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.eclipse.tycho</groupId>
      <artifactId>tycho-surefire-plugin</artifactId>
      <version>0.15.0</version>
      <configuration>
        <!-- Use the same Clover version as for source instrumentation -->
        <argLine>-Xbootclasspath/a:${user.home}/.m2/repository/com/atlassian/clover/clover/
4.0.0/clover-4.0.0.jar</argLine>
      </configuration>
    </plugin>
    ...
  </plugins>
</build>

```

### Usage

Directory layout:

```
/original_project - Tycho project
  /pom.xml        - must contain 'com.atlassian.clover:clover'
dependency for build and runtime/test
  /module1
  /src           - typical location of source folders in eclipse
plug-ins
  /test
  /module2
/instr_project   - copy of 'original_project' with instrumented files
created by build script
  /pom.xml
  /module1
  /src
  /test
  /module2
/build.xml      - Ant build script from above
```

Commands:

```
ant instrument
cd instr_project
mvn install
cd ..
ant report
```

## Clover-for-Maven 2 and 3 Installation Guide

This page contains the installation instructions for Clover-for-Maven 2 and 3.

See:

- [Clover-for-Maven 2 and 3 Quick Start Guide](#)
- [Basic usage](#)
- [Clover-for-Maven 2 and 3 User's Guide](#)

## Clover-for-Maven 2 and 3 Upgrade Guide

### General instructions

#### 1. Update version number of maven-clover2-plugin in your pom.xml

```
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-clover2-plugin</artifactId>
  <version><!-- PUT CLOVER VERSION --></version>
</plugin>
```

#### 2. Update version number of clover dependency in your pom.xml (optional)

Your pom.xml may contain a dependency to Clover Core (com.atlassian.clover:clover). Update it's version number the same version as the maven-clover2-plugin:


```
<dependency>
  <groupId>com.atlassian.clover</groupId>
  <artifactId>clover</artifactId>
  <version><!-- PUT CLOVER VERSION --></version>
</dependency>
```

### 3. Update license key (optional)

Installing new license is necessary if you're installing a Clover version released after end of support date of your current license.

### 4. Delete existing coverage databases (optional)

Clover's database format may change in newer versions. In such case you may get a build error with a message informing about database incompatibility. In such case you have to delete old database files.

 By default, databases are stored in module's /target directory; thanks to this incompatible databases are removed automatically upon 'mvn clean'.

## Upgrading from specific releases

Please see the [Clover Release Notes](#) and the [Clover-for-Maven 2 and 3 Changelog](#) for version-specific upgrade instructions.

## Clover-for-Maven 2 and 3 Changelog

Please also refer to the [Clover-for-Ant Changelog](#).

## Clover-for-Maven changelog




The changes for the latest version are as follows:

### Changes in Clover-for-Maven 4.0.0

July 14, 2014

This is a feature release with new HTML report with the ADG (Atlassian Design Guidelines) look.

#### Implemented features and fixes

T	Key	Summary	P
	CLOV-1345	Apply ADG in the HTML report	↑
	CLOV-1471	Maven clover2:setup triggers duplicate class exception	↑
	CLOV-1467	Rename com.cenqua to com.atlassian	↓

3 issues

Please see also the [Clover-for-Ant Changelog](#) for all changes in the Clover product.

### Changes in Clover-for-Maven 3.3.0

April 1, 2014

This is a major release with a dedicated support for the Spock framework and JUnit4 Parameterized Tests.

#### Implemented features and fixes

T	Key	Summary	P
	CLOV-1256	as a developer I'd like to instrument tests written in the Spock framework	

1 issue

Please see also the [Clover-for-Ant Changelog](#) for all changes in the Clover product.

#### Older versions

Looking for older versions? See [Clover-for-Maven 2 and 3 Changelog](#) for Clover 3.2.







## Changes in Clover-for-Maven 4.0.0

### Changes in Clover-for-Maven 4.0.0

**July 14, 2014**

This is a feature release with new HTML report with the ADG (Atlassian Design Guidelines) look.

#### Implemented features and fixes

T	Key	Summary	P
	CLOV-1345	Apply ADG in the HTML report	
	CLOV-1471	Maven clover2:setup triggers duplicate class exception	
	CLOV-1467	Rename com.cenqua to com.atlassian	

3 issues

Please see also the [Clover-for-Ant Changelog](#) for all changes in the Clover product.

## Changes in Clover-for-Maven 3.3.0

### Changes in Clover-for-Maven 3.3.0

**April 1, 2014**

This is a major release with a dedicated support for the Spock framework and JUnit4 Parameterized Tests.

#### Implemented features and fixes

T	Key	Summary	P
	CLOV-1256	as a developer I'd like to instrument tests written in the Spock framework	

1 issue

Please see also the [Clover-for-Ant Changelog](#) for all changes in the Clover product.

## Clover-for-Maven 2 and 3 FAQ

### Clover Maven 2 and 3 Plugin FAQ

- [Deploying Instrumented Jars](#)
- [How to keep Clover reports between builds?](#)
- [How to remove -clover suffix from artifact name?](#)
- [Is there an alternative to using the Maven Central repository?](#)
- [Preparing multi-module projects for remote deployment with Clover-for-Maven 2](#)
- [Troubleshooting License problems](#)
- [Troubleshooting problems with displaying characters](#)

## Clover-for-Eclipse

### Clover-for-Eclipse Documentation

#### What is Clover-for-Eclipse?

Clover-for-Eclipse brings the industry-leading code coverage tool, [Atlassian Clover](#) to the Eclipse integrated development environment. Clover-for-Eclipse allows you to easily measure the coverage of your unit tests, enabling targeted work in unit testing — resulting in stability and enhanced quality code with maximal efficiency of effort.

#### Getting Started with Clover for Eclipse

[Download Clover for Eclipse](#)

[Installation Guide](#)

[Clover for Eclipse in 10 minutes](#)

[Changelog for latest version of Clover-for-Eclipse](#)

#### Using Clover for Eclipse

[User's Guide](#)

[Installation & Configuration Guide](#)

#### Resources and Support

[Atlassian Answers](#)

[FAQ](#)

[Technical Support](#)










#### Offline Documentation

You can download the Clover documentation in PDF, HTML or XML format.

### Recently Updated

 [Clover Road Map](#)

Aug 12, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)

-  [Upgrading third party libraries](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Updating optimization snapshot file](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Hacking Clover](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Part 4 - Test Optimization Tutorial](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Part 3 - Automating Coverage Checks](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Part 2 - Historical Reporting](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Part 1 - Measuring Coverage](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Clover 4.0 Release Notes](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [A side-by-side comparison of the Classic and the ADG HTML report](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)

## Clover-for-Eclipse User's Guide

- [Overview](#)
- [Using the plugin](#)
- [FAQ](#)

### Overview

The Clover Eclipse Plugin allows you to instrument your Java code easily from within the [Eclipse Java IDE](#), and to view your coverage results inside Eclipse.

### Using the plugin

We recommend starting your adventure with Clover for Eclipse with a following lecture:

- [1. Clover for Eclipse in 10 minutes](#)

If you need more details how given features work, or how to efficiently work with Clover, you can read about:

- [2. Exploration of coverage in Eclipse](#)
- [3. Exploration of test results in Eclipse](#)
- [4. Scope of instrumentation in Eclipse](#)
- [5. Eclipse configuration options](#)
- [6. Generating reports in Eclipse](#)
- [7. Test Optimization for Eclipse](#)

If you work with Ant-based projects under Eclipse IDE, read the:

- [8. Launching an Ant build from Eclipse](#)

For more advanced topics, like performance tuning or instrumentation of RCP application, see:

- [9. Eclipse advanced topics](#)

### FAQ

See the [Eclipse Plugin FAQ](#). You can also search posts with the clover tag on [Atlassian Answers](#).

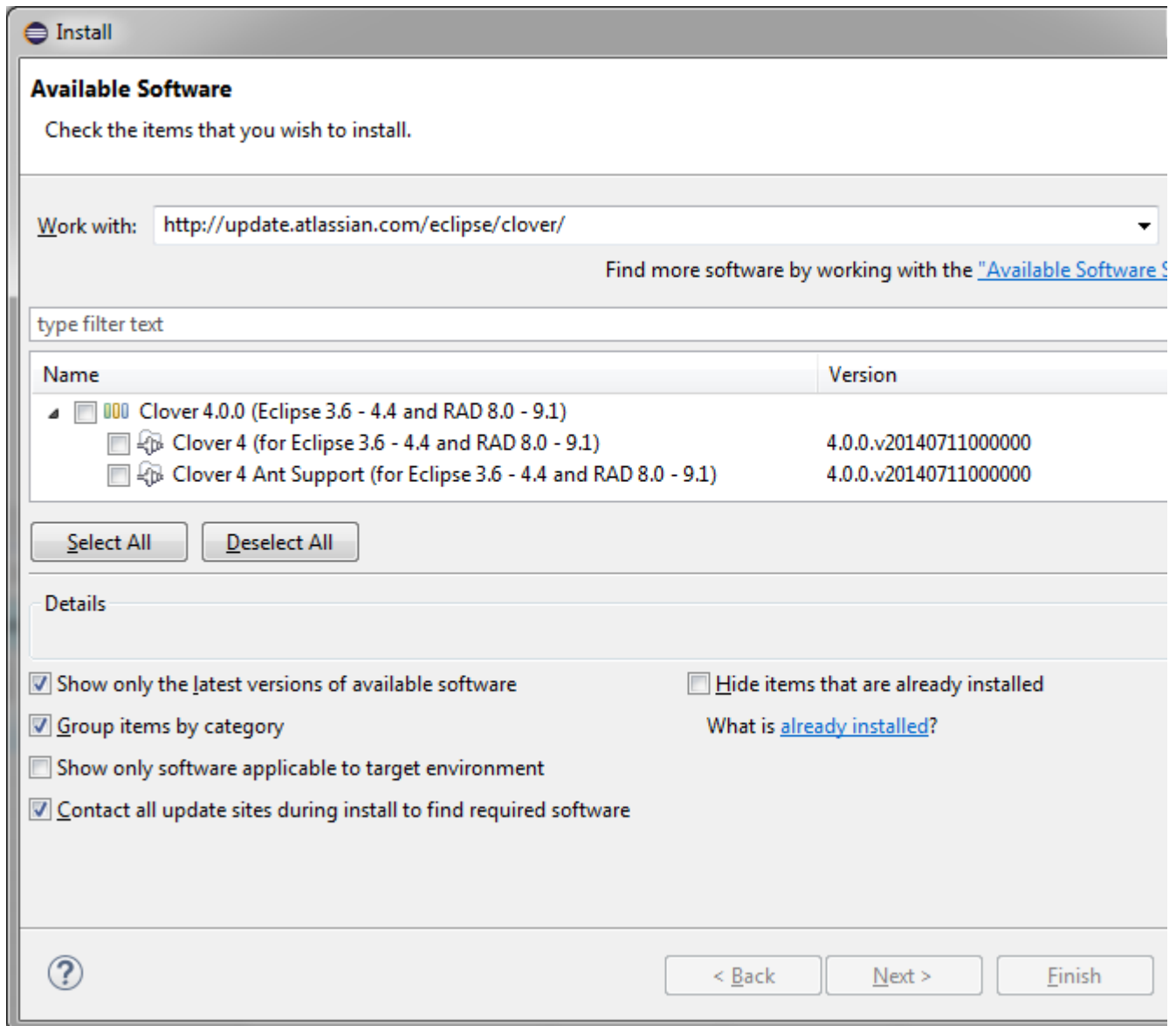
## 1. Clover for Eclipse in 10 minutes

This short guide will learn you how to install and use Clover in 10 minutes.

- Installing the plugin
- Entering license key
- Enabling Clover for Java project
- Building and running application
- Reviewing coverage results
  - Coverage Explorer
  - Clover Dashboard
  - Java Editor
  - Test Run Explorer
  - Test Contributions
  - Coverage Cloud Report
  - Coverage Treemap Report

### Installing the plugin

1. Select from the menu "Help | Install new software".
2. Click "Add" button and enter <http://update.atlassian.com/eclipse/clover> then click OK.
3. Select "Clover 4" and "Clover 4 Ant Support" features. Disable the "Contact all update sites..." checkbox (for faster installation). Click "Next" button twice.
4. Accept license agreement, click "Finish", click "OK" for warning about unsigned content, click "Restart now".



### Entering license key

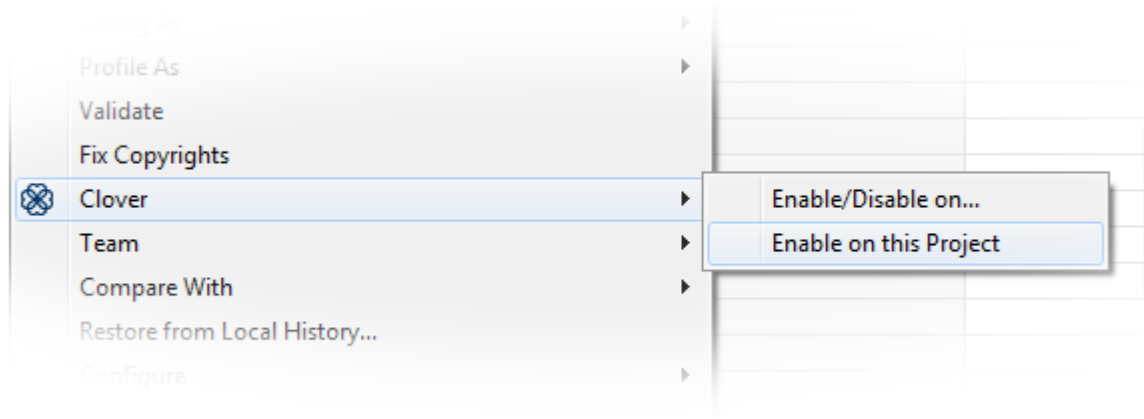
Your Clover-for-Eclipse just downloaded should be shipped with a 30-day evaluation key. You can also obtain new license keys on <http://my.atlassian.com> site.

1. Click Window > Preferences > Clover > License.
2. Paste the license key (note that the key contains newline characters). Click OK.

### Enabling Clover for Java project

Right click on a project in "Package Explorer" view, select "Clover > Enable on this Project". If you wish to enable Clover for multiple projects at once, choose "Enable/Disable on..."



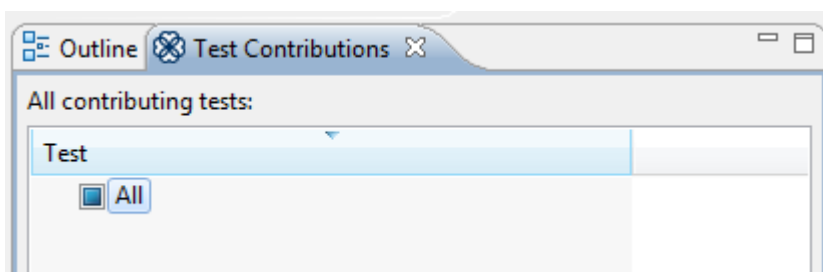


Four Clover views will be opened automatically:

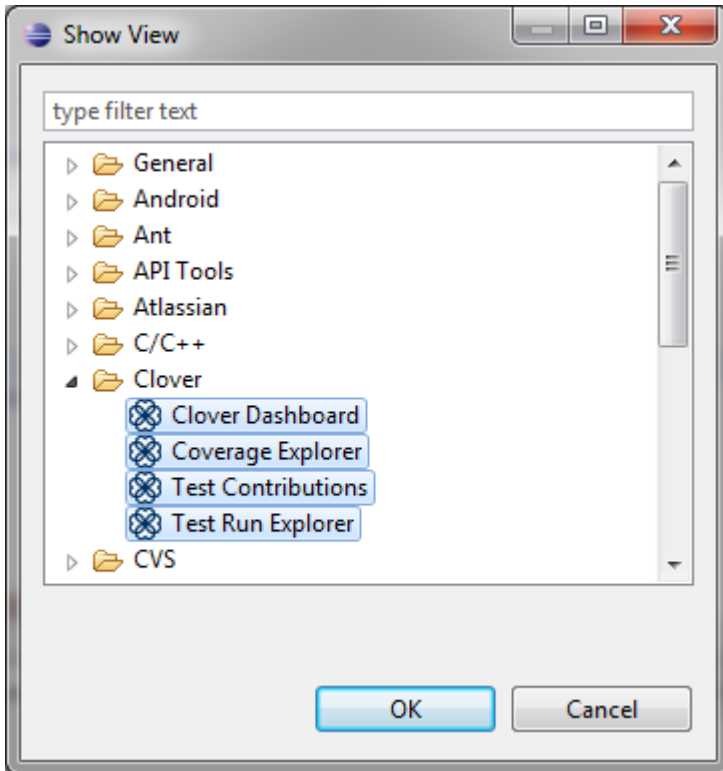
- Coverage Explorer
- Test Run Explorer
- Clover Dashboard
- Test Contributions

 A screenshot of the Eclipse IDE's Coverage Explorer view. The view shows a table with columns for 'Elem', 'Cov%', 'Av Me Cpx', and 'C'. The 'Show:' dropdown is set to 'All classes'. The table contains the following data:
 

Elem	Cov%	Av Me Cpx	C
Moneybags	0,0%	1,4	8
com.cenqua.samples.money	0,0%	1,4	8
IMoney.java	0,0%	-	
Money.java	0,0%	1,3	1
MoneyBag.java	0,0%	2,1	3
MoneyBagTest.java	0,0%	1,0	2
MoneyTest.java	0,0%	1,0	

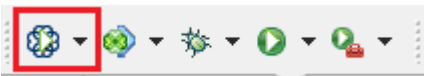


You can always open them from "Window > Show View > Other ... > Clover":



### Building and running application

Build your project as usual. In order to run it, choose "Run with Clover" button from tool bar:



### Reviewing coverage results

#### **Coverage Explorer**

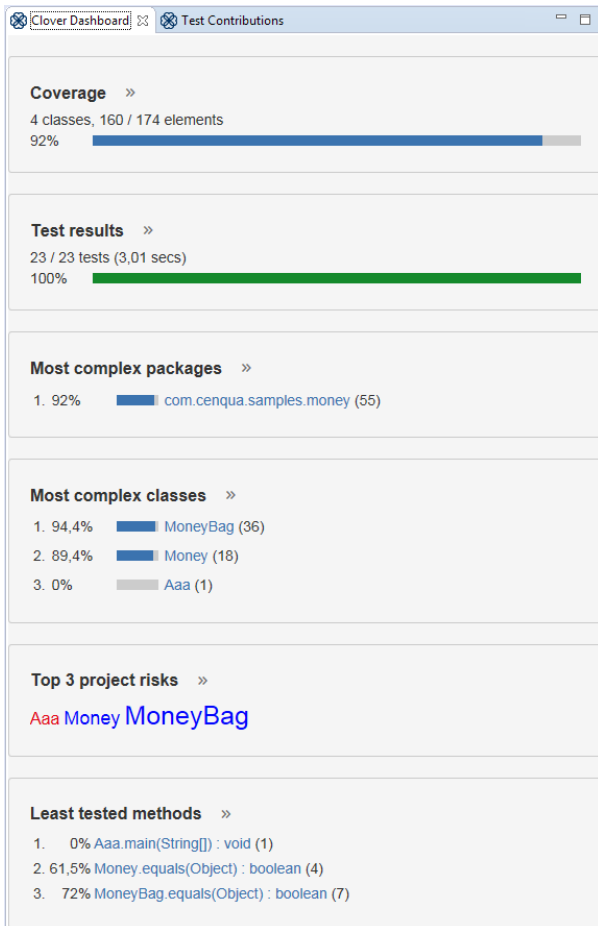
It's the best place to start. You will see coverage and metrics for all instrumented projects.

Elem	Cov%	Av Me Cpx	C
Moneybags	94,8%	1,4	8
com.cenqua.samples.money	94,8%	1,4	8
IMoney.java	0,0%	-	
Money.java	89,4%	1,3	1
Money	89,4%	1,3	1
MoneyBag.java	94,4%	2,1	3
MoneyBagTest.java	97,8%	1,0	2
MoneyTest.java	100,0%	1,0	
MoneyTest	100,0%	1,0	
testAdd()	100,0%	-	

### Clover Dashboard

It's a place where you can have a quick overview of your project and find hints about code areas you should focus on:

- **Coverage** - shows percentage coverage of the project
- **Test Results** - shows percentage of passed tests as well as duration
- **Most Complex Packages** - lists several packages with the highest complexity -> refactor
- **Most Complex Classes** - as above, but on a class level -> refactor
- **Top Project Risks** - the most complex and the least tested classes -> refactor and/or write more tests
- **Least Tested Methods** - methods with the lowest test coverage -> write more tests



### Java Editor

To view coverage information on a line-by-line basis, Clover adds coloured annotations to your project's Java source code editors.

```

public class Foo {
101 public int passed(boolean ok) {
101     System.out.println("Executed and test case passed");
101     int i = 10;
101     if (ok) {
100         return ++i;
101     } else {
1     return --i;
101     }
101 }

3 public void failed() {
3     System.out.println("Executed, but test case failed");
3 }

0 public void neverExecuted() {
0     System.out.println("Never executed");
0 }

1 public void coveredButNotByTest() {
1     System.out.println("Executed, by not by test case");
1 }

1 public void partiallyCovered(boolean what) {
0     if (what) { System.out.println("partiallyCovered true"); } else { System.out.p
1 }

1 public void partiallyCoveredWithFailed(boolean what) {
0     if (what) { System.out.println("partiallyCoveredWithFailed true"); } else { Sy
1 }

```

Colours used for background highlighting are as follows:

Colour	Meaning
Green	Coverage from passing tests or due to execution outside tests (e.g. main() method).
Squiggly Red Lines	Partial branch coverage (caused when only one part of a branch has been covered).
Yellow	Failed test coverage (where coverage has only been caused by one or more failing tests and no passing tests).
Grey	Filtered out code.
Red	Code with no coverage.

A left margin ruler shows three colour markers:

- left half - green if at least part of line is covered, yellow if at least part of line is covered but test has failed, red for line not covered at all (it's an "optimistic marker")
- right half - red if at least part of line is not covered, yellow if at least part of line is covered but test has failed, green for fully covered line (it's a "pessimistic marker")
- right strip - dark green for coverage related with passed tests, dark yellow for coverage related with failed tests

### Test Run Explorer

The Test Run Explorer view, lets you explore your recently run tests - showing whether they passed or failed, their duration and any error messages that they generated. Furthermore, it allows you to explore the code coverage caused by an individual test, a test class, a package or even your entire project.

After running tests in a Clover-enabled project, on the left-hand side of this view you will see a tree of tests that were run. On selecting an element on the left-hand side, the right-hand side displays all the application (i.e. non-test) classes that had coverage caused by the selection.

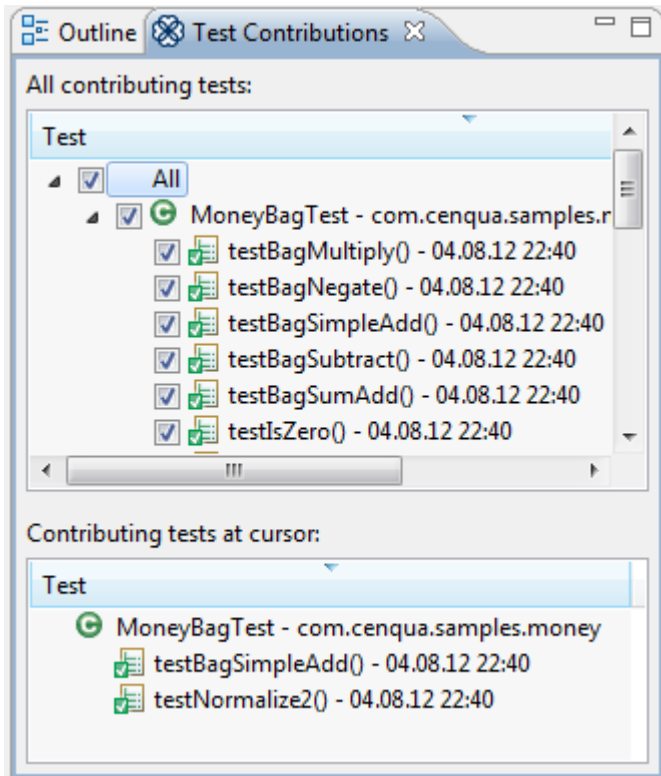
The right-hand table displays not only the names of the classes that were partially or fully covered by the test methods, but also the percentage of class' total coverage that is attributable to the test method (the test method's coverage contribution) as well as the percentage of the class' coverage that was attributable only to the selected test method (the test method's unique coverage).

Test	Start	Rslt	Time	Message
Moneybags				
MoneyBagTest.testBagMultiply()	05.09.12 11:35	PASS	0.001s	
MoneyBagTest.testBagNegate()	05.09.12 11:35	PASS	0.001s	
MoneyBagTest.testBagNotEquals()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testBagSimpleAdd()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testBagSubtract()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testBagSumAdd()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testIsZero()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMixedSimpleAdd()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMoneyBagEquals()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMoneyBagHash()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMoneyEquals()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMoneyHash()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testNormalize2()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testNormalize3()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testNormalize4()	05.09.12 11:35	PASS	0.001s	

### Test Contributions

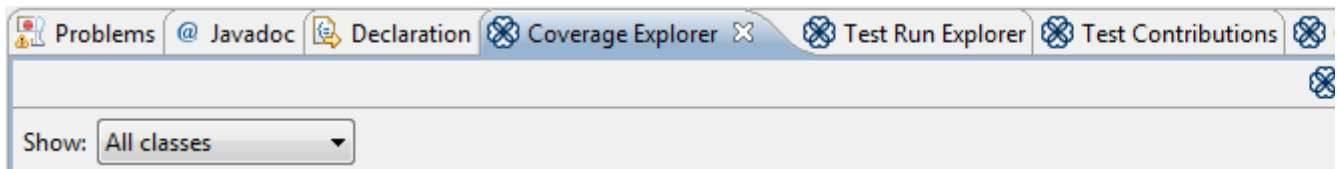
The Test Contributions view shows unit tests and methods that generated coverage for the currently opened and selected Java source file. As you switch between Java source file editors, the top most tree is updated with the test methods that contributed to coverage of this file. The bottom tree tracks the test methods that contributed to coverage of the file at the current cursor position.

The Test Contributions view allows you to better understand the relationship between your test code and your application code.



### Coverage Cloud Report

Coverage cloud reports are a great way to work out the classes that form major risks (low coverage but high complexity) to your project and its packages, and also to highlight potential quick wins for increasing the overall project or per-package coverage. The coverage cloud report can be generated by right-clicking on your Clover-enabled project in the Coverage Explorer view or Test Run Explorer view and selecting Generate Coverage Cloud.



Clover Coverage Cloud Report

[AbstractExecutorService](#) [AbstractList](#) [AbstractMap](#) [AbstractMap.SimpleEntry](#) [AbstractMap.SimpleImmutableEntry](#) [AbstractSequentialList](#) [ArrayBlockingQueue](#) [ArrayBlockingQueue](#) [ArrayDeque](#) [ArrayDeque.DeqIterator](#) [ArrayDeque.DescendingIterator](#) [Arrays](#) [AtomicBoolean](#) [AtomicInteger](#) [AtomicIntegerArray](#) [AtomicLong](#) [AtomicLongArray](#) [AtomicMarkableReference](#) [AtomicReference](#) [AtomicReferenceArray](#) [AtomicStampedReference](#) [BrokenBarrierException](#) [CancellationException](#) [Collections.AsLifoQueue](#) [Collections.CheckedCollection](#) [Collections.CheckedCollection](#) [Collections.CheckedList](#) [Collections.CheckedList.ListIterator](#) [Collections.CheckedMap.EntrySetView](#) [Collections.CheckedMap.EntrySetView](#) [Collections.CheckedSet](#) [Collections.CheckedSortedMap](#) [Collections.CheckedSortedMap](#) [Collections.ReverseComparator](#) [Collections.SetFromMap](#) [ConcurrentHashMap](#) [ConcurrentHashMap.EntrySet](#) [ConcurrentHashMap.HashIterator](#)

Project risks Quick wins

Project backport-util-concurrent  Include classes from sub-packages

Refresh

### Coverage Treemap Report

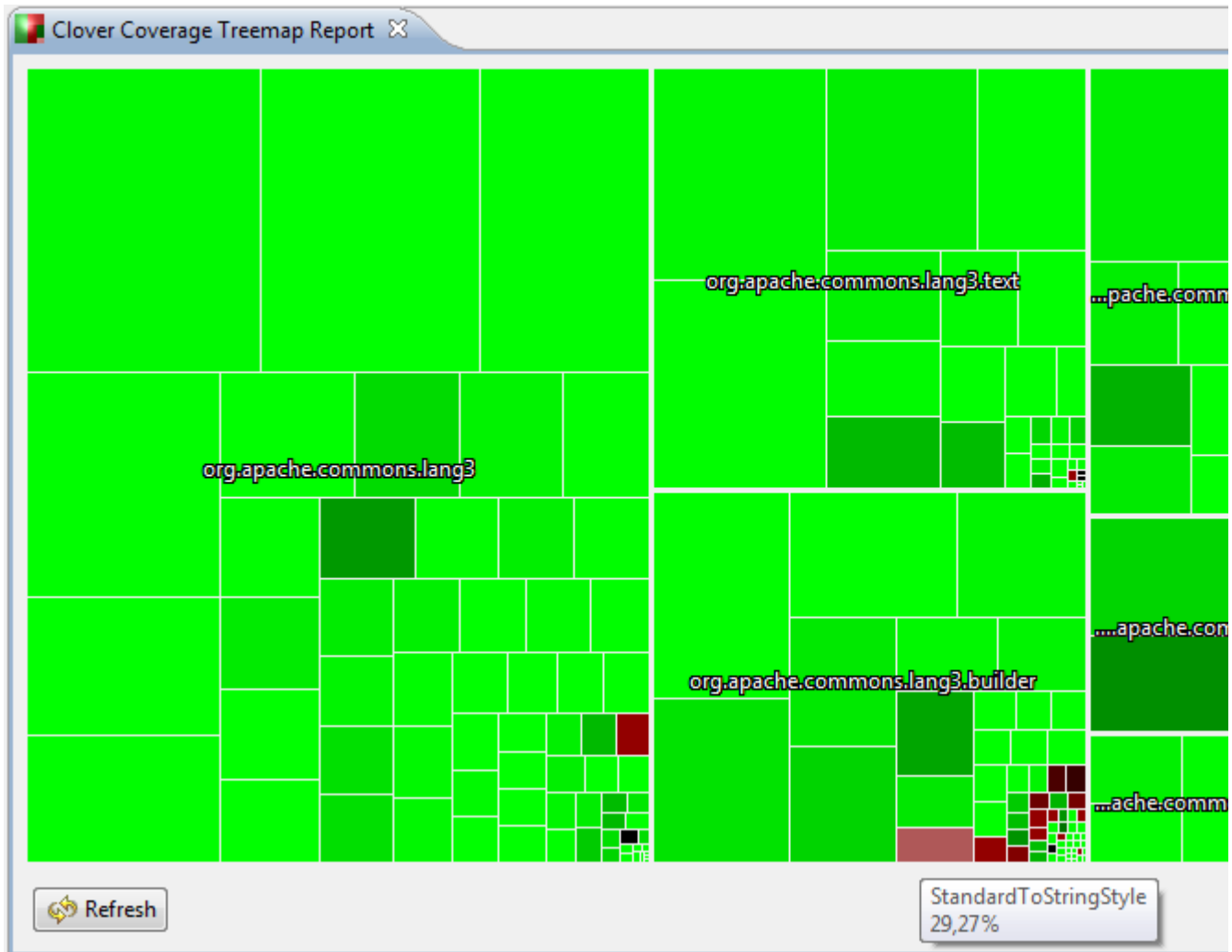
The coverage treemap report allows simultaneous comparison of classes and package by complexity and by code coverage. The treemap is divided by package (labelled) and then further divided by class (unlabelled). The size of the package or class indicates its complexity (larger squares indicate great complexity). Colours indicate the level of coverage (bright green - most covered, bright red - uncovered).

The treemap cloud report can be generated by right clicking on your Clover-enabled project in the Coverage Explorer and selecting Generate Coverage Treemap.

Problems Javadoc Declaration Coverage Explorer Test Run Explorer Test Contributions

Show: All classes





Congratulations! You now know basics of *Clover for Eclipse* plugin - enough to start your daily work with it.

If you like to learn more, read the [2. Exploration of coverage in Eclipse](#) chapter.

## 2. Exploration of coverage in Eclipse

- The Clover Coverage Explorer
  - Columns
  - Summary Panel
  - Actions and Menus
- Viewing Coverage Results
  - Source Code Annotations
  - Coverage Cloud Reports
    - Package Risks
    - Quick Wins
  - Coverage Treemap Reports

### The Clover Coverage Explorer

The Coverage Explorer allows you to view and control Clover's instrumentation of your Java projects, and shows you the coverage statistics for each project based on recent test runs or application runs. It is automatically added to the workbench when you enable Clover for your project. If the viewer is closed, you can open it again using "Window | Show View | Other..." and selecting "Clover | Coverage Explorer".

The main tree shows coverage and metrics information for packages, files, class and methods of any Clover-enabled project in your workspace. Clover will auto-detect which classes are your tests and which are your application classes - by using the drop-down box above the tree you can then restrict the coverage tree shown so that you only see coverage for application classes, test classes or both.

*Screenshot: Clover Coverage Explorer*

The screenshot shows the Clover Coverage Explorer window with the following data:

Elem	Cov%	Av Me Cpx	C
Moneybags	94,8%	1,4	8
com.cenqua.samples.money	94,8%	1,4	8
IMoney.java	0,0%	-	
Money.java	89,4%	1,3	1
Money	89,4%	1,3	1
MoneyBag.java	94,4%	2,1	3
MoneyBagTest.java	97,8%	1,0	2
MoneyTest.java	100,0%	1,0	
MoneyTest	100,0%	1,0	
testAdd()	100,0%	-	

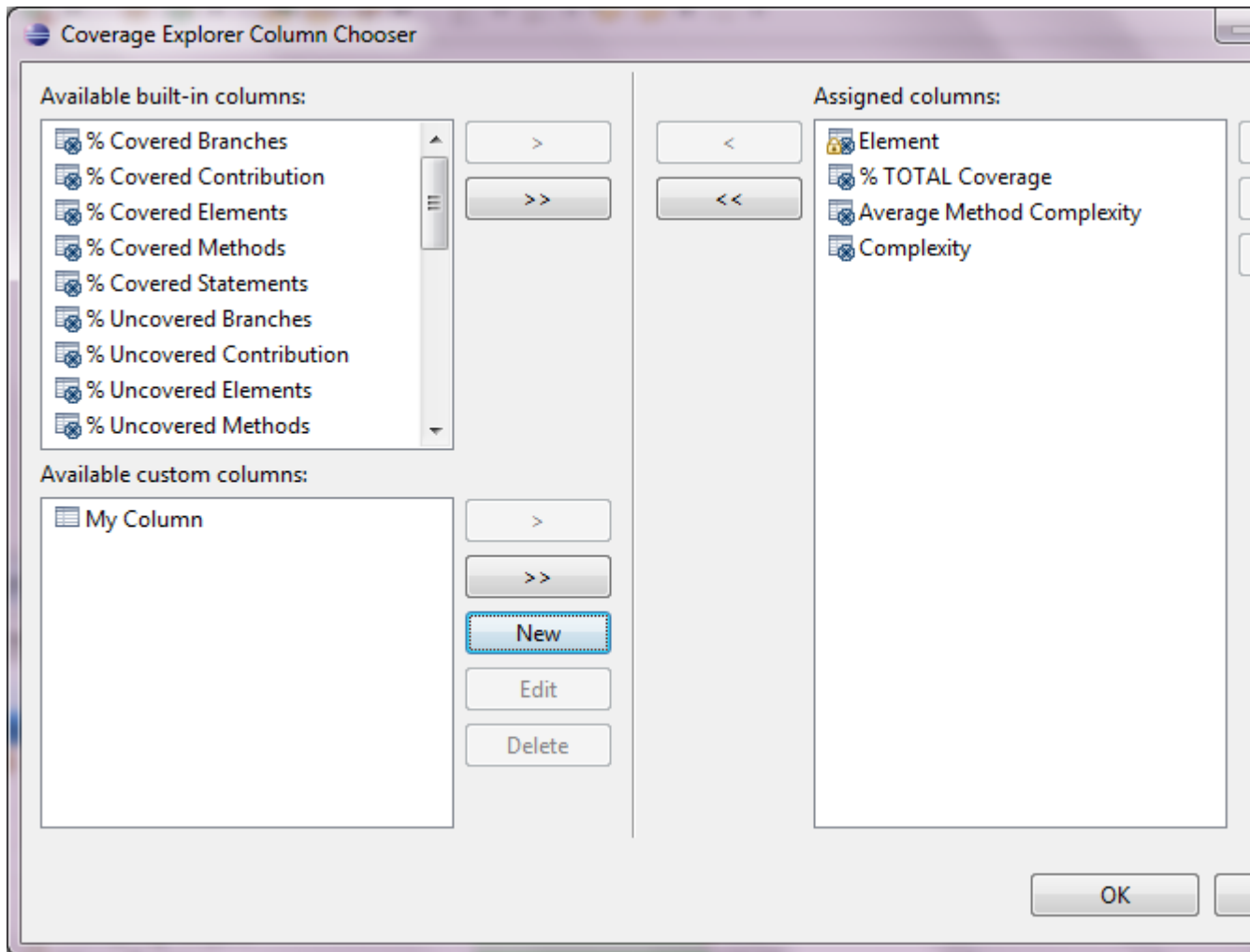
### Columns

Initially the Coverage Explorer's tree displays the following four columns:

- **Element** - The name of the package, file, class or method.
- **%TOTAL Coverage** - The total coverage of the element as a percentage.
- **Complexity** - The cyclomatic complexity of the element
- **Average Method Complexity** - The average method complexity of the element

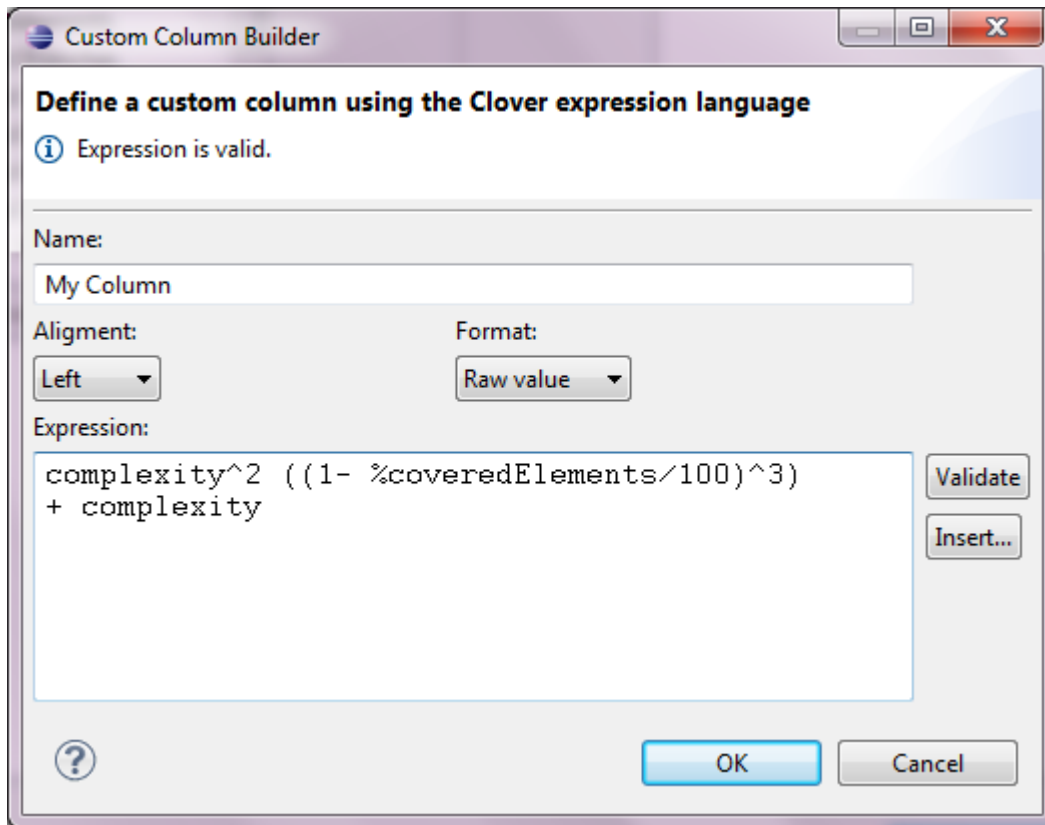
Clover supports 24 standard column types which can be chosen in the Coverage Explorer Column Chooser. The Column Chooser can be summoned by selecting "Columns..." in the Coverage Explorer view menu.

*Screenshot: Coverage Explorer Column Chooser*



Clover-for-Eclipse also lets you define your own custom columns for display within the Coverage Explorer tree. While choosing columns in the Column Chooser dialog, click the "New" button to summon the Custom Column Builder dialog. Custom columns must have a name and a valid [Clover expression](#). Custom columns may be left, center or right justified and the value may be displayed as a decimal value or as a percentage value. Clover expressions can refer to any of the 24 standard columns; these references can be easily inserted into the expression by selecting "Insert..." and choosing the appropriate column name.

*Screenshot: Coverage Explorer Custom Column Builder*



### Summary Panel

Summary metrics are displayed alongside the tree for the selected project, package, file, class or method in the tree. The following metrics are provided:

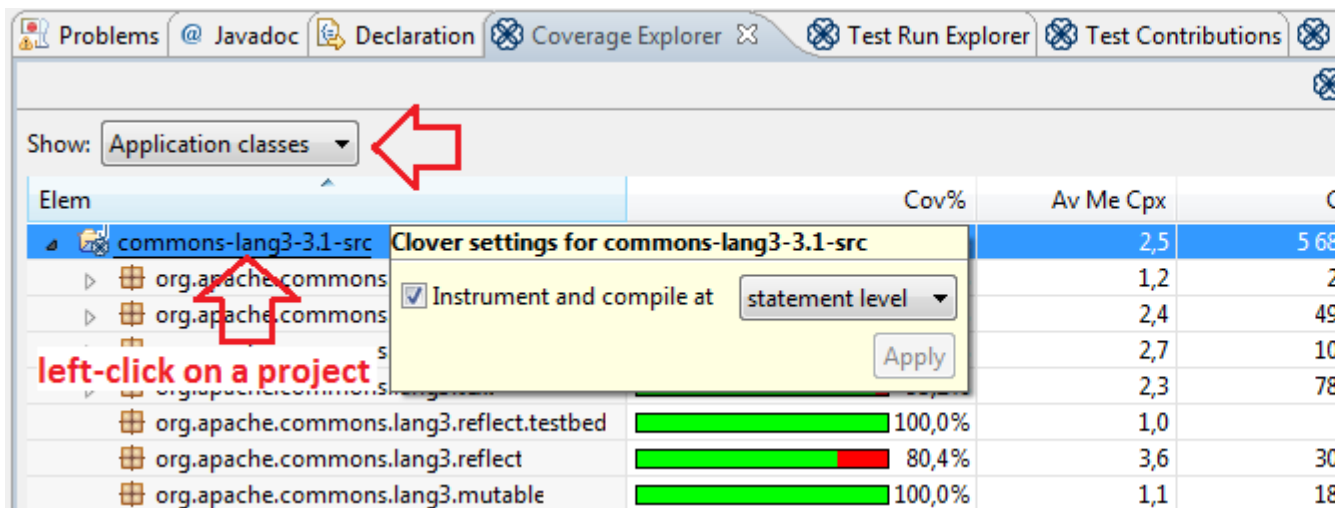
- **Structure**
  - **Packages:** The number of packages the project or package root contains
  - **Files:** The number of files the package, package root or project contains
  - **Classes:** The number of classes the file, package, package root or project contains
  - **Methods:** The number of methods the class, file, package, package root or project contains
  - **Statements:** The number of statements the method, class, file, package, package root or project contains
  - **Branches:** The number of branches the method, class, file, package, package root or project contains
- **Tests:**
  - **Tests:** The total number of tests in the class, file, package, package root or project
  - **Passes:** The total number of passing tests in the class, file, package, package root or project
  - **Fails:** The total number of failing tests in the class, file, package, package root or project
  - **Errors:** The total number of tests with errors in the class, file, package, package root or project. Currently Clover does not differentiate between fails and errors and counts all errors as failures.
- **Source:**
  - **LOC:** The total number of lines of code in the class, file, package, package root or project
  - **NC LOC:** The total number of non-commented lines of code in the class, file, package, package root or project
  - **Total Cmp:** The total cyclomatic complexity of code in the method, class, file, package, package root or project
  - **Avg Method Cmp:** The average method complexity of the class, file, package, package root or project
  - **Cmp Density:** The complexity density of code in the class, file, package, package root or project

### Actions and Menus

The Coverage Explorer view allows the following actions (from the view toolbar, menu or both):

- **Enable/Disable Clover On...** Allows you to select multiple project to enable or disable Clover for. This allows you, for example, to quickly turn Clover on for all projects in your workspace in one action.
- **Refresh Coverage Data.** Re-loads from disk the Clover coverage data for the selected project.
- **Delete Coverage Recordings.** Deletes the coverage recording data recorded from test runs or applications runs for the selected project.
- **Clear Snapshot** Deletes the [Test Optimization](#) snapshot file
- **Compiled with Clover.** Toggles the use of Clover instrumentation when Eclipse compiles the selected Java project.
- **Edit Context Filter...** Allows you to edit the block and custom coverage 'contexts' used to help you filter out unwanted coverage data. An explanation of what contexts are can be found [here](#).
- **Generate Coverage Treemap.** Generates a mini report page in treemap format for the selected project. This report arranges package and classes enabling easy comparison of their complexity while also indicating their level of code coverage.
- **Generate Coverage Cloud.** Generates a report page in cloud tag format for the selected project. This lists classes considered project risks or quick wins.
- **Coverage Reports > Run new report...** Launches the report generation wizard that will take you through the steps required to generate a PDF, HTML or XML. These reports can be generated for single or multiple projects.
- **Coverage Reports > View report** Browse recently generated reports.
- **Coverage in Explorer > Show All Classes.** Shows code coverage for all application and test classes in the Coverage Explorer.
- **Coverage in Explorer > Show Only Application Classes.** Shows code coverage only for application (non-test) classes in the Coverage Explorer.
- **Coverage in Explorer > Show Only Test Classes.** Shows code coverage only for test classes in the Coverage Explorer.
- **Enable Clover Working Set.** Enables or disables usage of the Clover working set. This filters the files, directories and projects that Clover will report on and are especially handy for large projects.
- **Edit Clover Working Set...** Allows you to edit the files, directories and projects in the Clover working set.
- **Clear Clover Working Set.** Removes all files, directories and projects from the Clover working set.
- **Hide Unavailable.** This check button hides any elements in the Coverage Explorer's tree that don't have associated coverage information and helps reduce visual clutter. For instance, with this button checked any project that isn't Clover-enabled will not be shown in the Coverage View.
- **Hide 100% Covered.** This check button hides any elements in the Coverage Explorer's tree that have full coverage and helps you focus on only those classes that require more testing.
- **Layout > Packages.** Arranges the coverage tree by package. This is useful if you have a single source directory for your project.
- **Layout > Package Roots.** Arranges the coverage tree first by package root, then by package. This is useful if you have multiple source directories within your project.
- **Layout > Hierarchical packages** Toggle showing packages in hierarchical or flat list.
- **Columns...** Allows you to customise the columns shown shown in the Coverage Explorer.
- **Coverage in Editors > Show All.** Shows red/green coverage areas in open Java editors. This is useful for finding out exactly which parts of the code are being covered.
- **Coverage in Editors > Show Uncovered.** Shows only red (uncovered) areas in open Java editors. This is useful for finding out exactly which parts of the code are not being covered while not cluttering your editor with the overwhelming large green areas (covered code).
- **Coverage in Editors > Show None.** Hides all red/green coverage areas in open Java editors.
- **Include passed only** If enabled, code coverage from failed tests will not be taken into account.
- **Show exclusion annotations** If enabled, draws small Clover icons in the Package Explorer view for every source file, indicating whether it's included or excluded from instrumentation.
- **About** Shows information about Clover version, copyright and licences.

The Coverage Explorer view has also actions accessible from buttons and links in browser (left panel) as well as on left/right click on elements in a browser:



- **Show: All Classes** Shows code coverage for all application and test classes in the Coverage Explorer.
- **Show: Application classes** Shows code coverage only for application (non-test) classes in the Coverage Explorer.
- **Show: Test classes** Shows code coverage only for test classes in the Coverage Explorer.
- **Workspace Settings > Aggregate coverage generated since the last clean build** If enabled, coverage from all test or application runs will sum up (the "Delete Coverage Recordings" action or full project rebuild will clean them).
- **Workspace Settings > Track per-test coverage** If enabled, Clover will track individual code coverage generated by every test case. It's useful for checking what code has been exactly covered by given tests or which tests are the most relevant for given application code.
- **Workspace Settings > Keep per-test coverage data fully in memory** If enabled, per-test coverage will be stored in memory instead temporary files on disk. Attention: can consume a lot of memory.
- **Workspace Settings > Look for updated coverage every N s** Whether and how frequently Clover should refresh coverage view
- **Project Settings > Instrument and compile** The "statement level" is recommended as it gives the most accurate information. The "method level" records only whether given method was called or not; it can be used for very large projects (as produces smaller instrumentation overhead and compiles faster) or when you need just rough information about coverage.

## Viewing Coverage Results

The Clover-for-Eclipse plugin shows coverage data in number of ways. Viewing any coverage data, of course, requires that you have executed tests or you have launched your application and it also requires that Clover has detected the change in coverage. On the Clover preferences page, you can change how often Clover looks for new coverage data for your application or tests.

### Source Code Annotations

To view coverage information on a line-by-line basis, Clover adds coloured annotations to your project's Java source code editors.

*Screenshot: Clover Source Annotations in Java editor*

```

public class Foo {
101 public int passed(boolean ok) {
101     System.out.println("Executed and test case passed");
101     int i = 10;
101     if (ok) {
100         return ++i;
101     } else {
1     return --i;
101     }
101 }

3 public void failed() {
3     System.out.println("Executed, but test case failed");
3 }

0 public void neverExecuted() {
0     System.out.println("Never executed");
0 }

1 public void coveredButNotByTest() {
1     System.out.println("Executed, by not by test case");
1 }

1 public void partiallyCovered(boolean what) {
0     if (what) { System.out.println("partiallyCovered true"); } else { System.out.p
1 }

1 public void partiallyCoveredWithFailed(boolean what) {
0     if (what) { System.out.println("partiallyCoveredWithFailed true"); } else { Sy
1 }

1 public static void main(String args[]) {
1     new Foo().coveredButNotByTest();
1     new Foo().partiallyCovered(true);
1 }
}

```

The table below explains the meanings of various colours:

Annotation Colour	Meaning
Green	Coverage from passing tests.
Green	Coverage caused incidentally (caused by something other than test methods, e.g. main() methods, test setUp() and tearDown() methods). A different tooltip is presented to show it is incidental.
Squiggly Red Lines	Partial branch coverage (caused when only one part of a branch has been covered).
Yellow	Failed test coverage (where coverage has only been caused by one or more failing tests and no passing tests).
Grey	Filtered out code.
Red	Code with no coverage.

A left margin ruler shows three colour markers:

- left half - green if at least part of line is covered, yellow if at least part of line is covered but only from failed tests, red if line is not covered at all (it's an "optimistic marker")



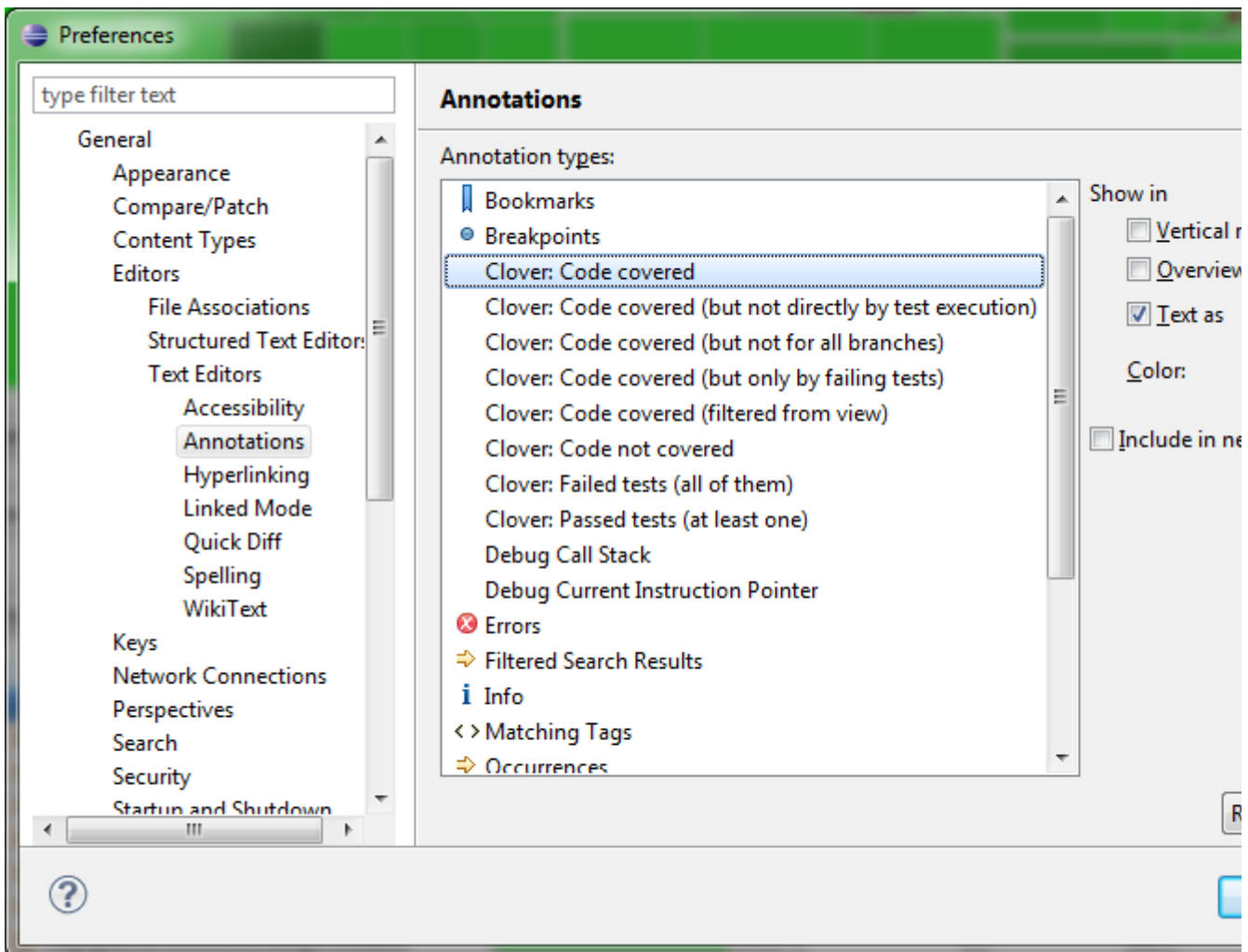
- right half - red if at least part of line is not covered, yellow if at least part of line is covered but only from failed tests, green for fully covered line (it's an "pessimistic marker")
- right strip - dark green for coverage related with passed tests, dark yellow for coverage related with failed tests

Each coloured source annotation displays a tool tip providing more information about the relevant line or lines. These coverage annotations are added automatically to any opened source files from a Clover-enabled project. For example, if a statement was executed ten times, or a conditional expression was only executed twice in the true case but never in the false case, this information will appear in the tool tip.

If you wish to switch off the source code annotations, you can easily do so by de-selecting the menu / toolbar item "Show Coverage in Editor" on the Coverage Explorer view or Test Run Explorer view. Using an adjacent menu / toolbar item you can also force Clover to only annotate code without coverage, thus only showing the red annotations.

You can also change these colours and effects in Eclipse. To do so, access '**Eclipse Preferences > General > Editors > Text Editors > Annotations**'. On this page, see the '**Annotation Types**' list and click the settings that begin with 'Clover'.

Screenshot: Clover Source Annotation Settings



## Coverage Cloud Reports

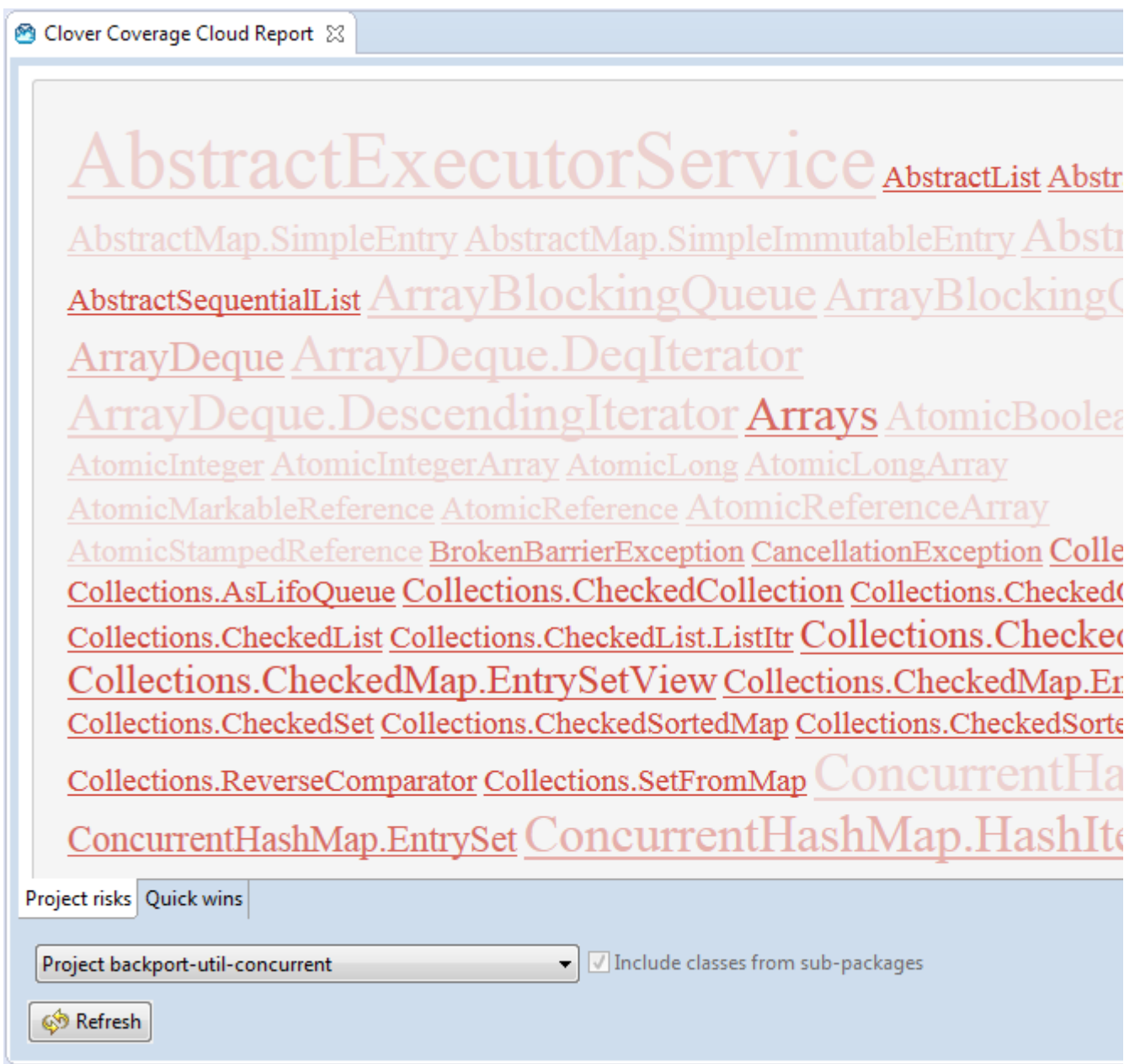
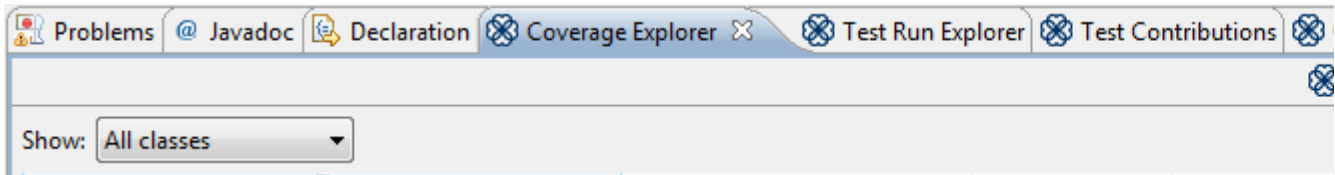
Coverage cloud reports are a great way to work out the classes that form major risks (low coverage but high complexity) to your project and its packages, and also to highlight potential quick wins for increasing the overall project or per-package coverage.

The coverage cloud report has two tabs, one tab for the project/package risks cloud and the other for quick wins



cloud. Each cloud shows the classes of the project/package enlarged or reduced depending on their significance to the report and clicking on any class name will open the corresponding source file in an editor. At the bottom of the report you can select the target for the clouds - either the entire project or one of the packages within the project - changing this will update the cloud tabs. Where a package is selected and has sub-packages, the checkbox "Include classes from sub-packages" allows you to include or exclude sub-package clouds from the clouds.

The coverage cloud report can be generated by right clicking on your Clover-enabled project in the Coverage Explorer view or Test Run Explorer view and selecting Generate Coverage Cloud.



**Package Risks**

The Package Risks Cloud highlights those classes that are the **most complex, yet are the least covered** by

your tests. The larger and redder the class, the greater the risk that class poses for your project or package. Package risk clouds can be toggled to include or exclude classes in sub-packages.

Metric	Attribute
Average Method Complexity	Font Size
% Coverage	Font Color

### Quick Wins

This Cloud highlights the "**low hanging coverage fruit**" of your project or package. You will achieve the greatest increase in overall project coverage by covering the largest, reddest classes first. Package Quick Win clouds can be toggled to include or exclude classes in sub-packages.

Metric	Attribute
Number of Elements	Font Size
Number of Elements Untested	Font Color

### Coverage Treemap Reports

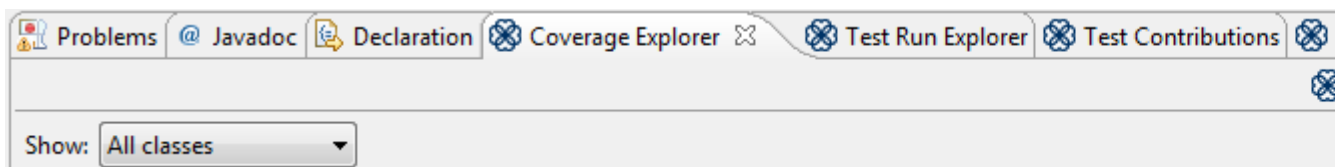
The coverage treemap report allows simultaneous comparison of classes and package by complexity and by code coverage. The treemap is divided by package (labelled) and then further divided by class (unlabelled). The size of the package or class indicates its complexity (larger squares indicate great complexity, while smaller squares indicate less complexity). Colours indicate the level of coverage, as follows:

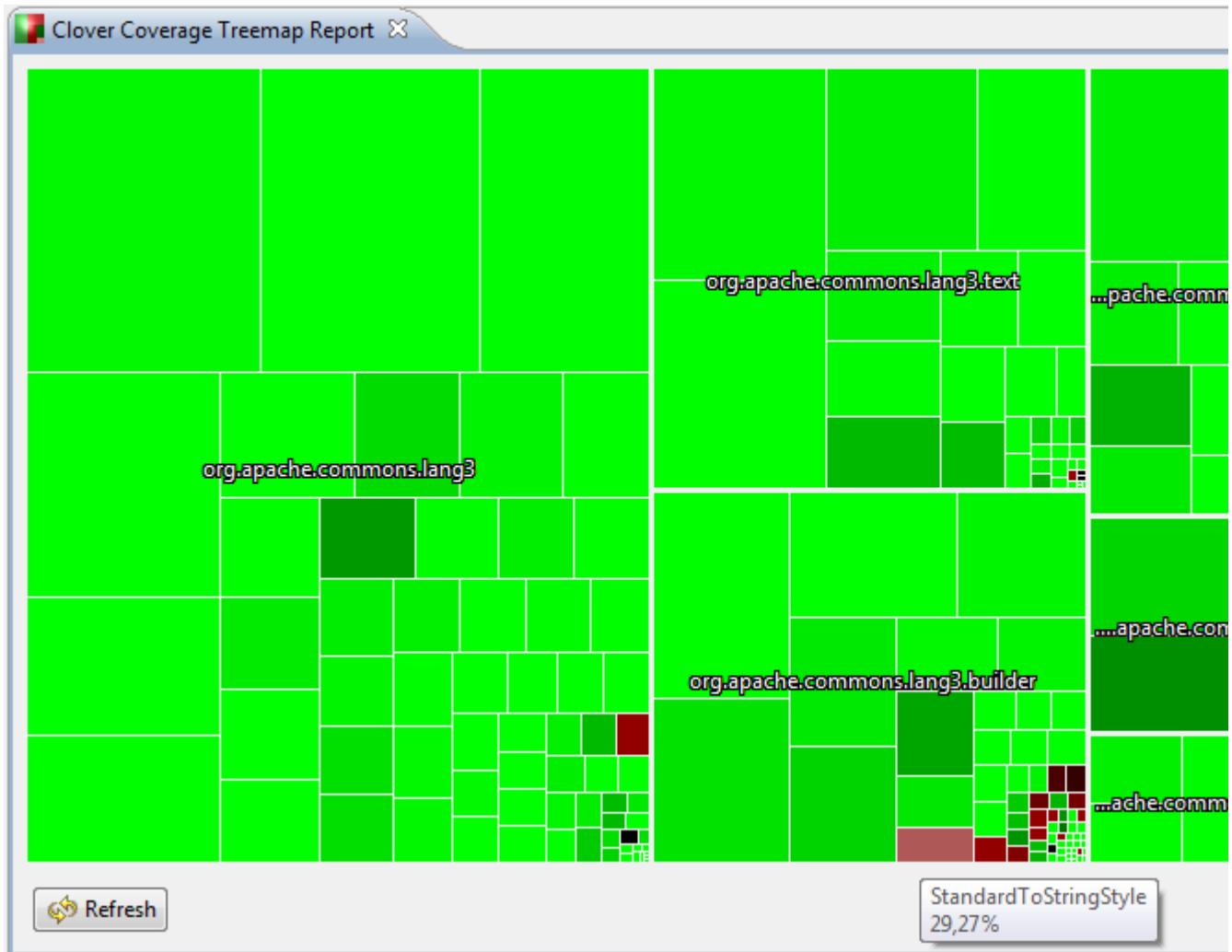
- Bright green (most covered)
- Dark green (more coverage)
- Black (around 50% coverage)
- Dark Red (little coverage)
- Bright Red (uncovered)

The percentage shown in the yellow box is the code coverage for the class currently under the mouse.

Right clicking on a package area and selecting the magnify option will then focus the treemap on the package's classes. Right clicking again and selecting the demagnify option will re-focus on the project's top level. Double clicking on a class will open the corresponding source file in an editor.

The treemap cloud report can be generated by right clicking on your Clover-enabled project in the Coverage Explorer and selecting *Generate Coverage Treemap*. You can also click on a *Generate a coverage treemap* button.





Now you perfectly know how your application is covered. But how to efficiently navigate between tests and application code? Read the [3. Exploration of test results in Eclipse](#) chapter.

### 3. Exploration of test results in Eclipse

- Test Run Explorer
  - Actions and Menus
- Test Contributions

#### Test Run Explorer

The Test Run Explorer view, like several popular plugins such as the JUnit Plugin or TestNG Plugin, lets you explore your recently run tests - showing whether they passed or failed, their duration and any error messages that they generated. Clover-for-Eclipse takes this one step further and allows you to explore the code coverage caused by an individual test, a test class, a package or even your entire project.

After running tests in a Clover-enabled project, select the Test Run Explorer view (or to open it, select *Window > Show Views > Other... > Clover > Test Run Explorer*). On the left-hand side of this view you will see a tree of tests that were run and their containers such as classes, packages, projects. On selecting an element on the left-hand side, the right-hand side displays all the application (i.e. non-test) classes that had coverage caused by the selection.

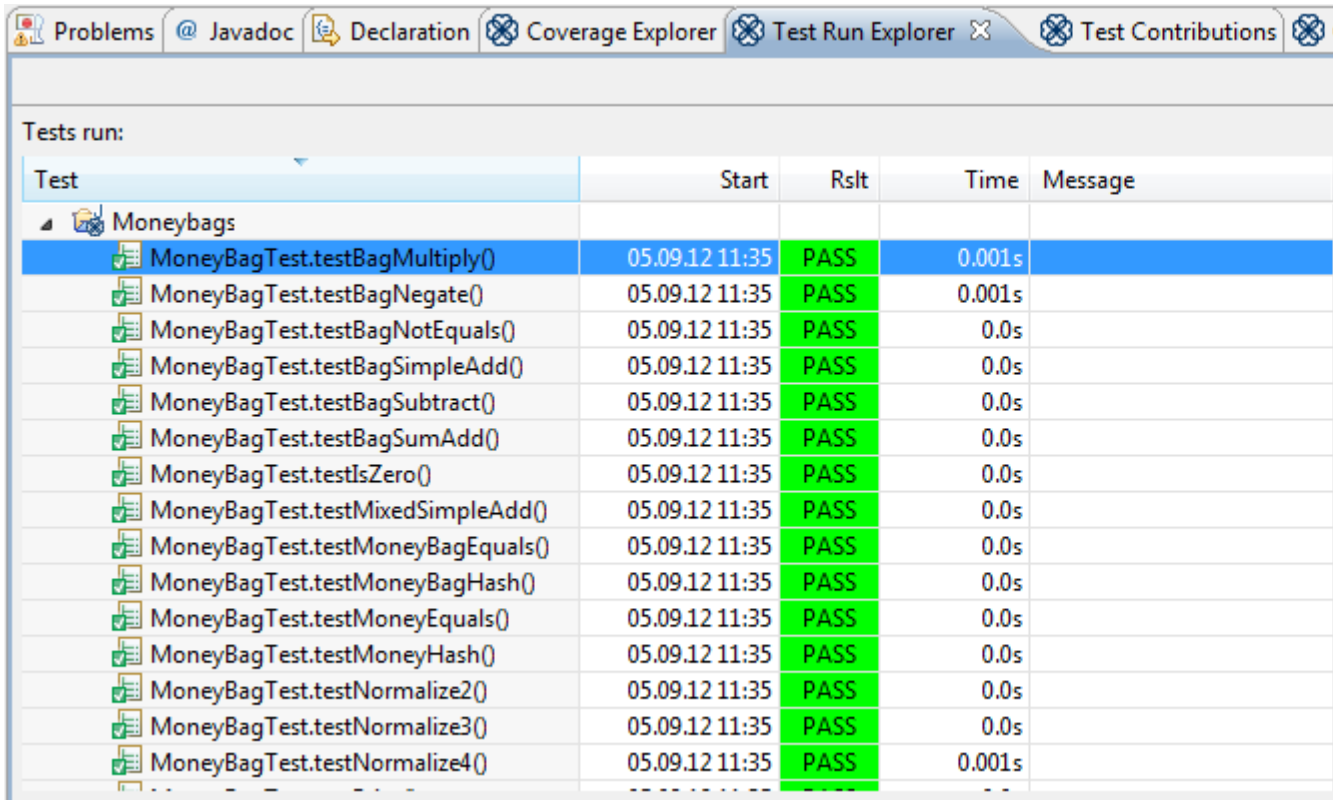
For example, selecting a test method on the left side will cause the right-hand side to show the application

classes with code executed (directly or indirectly) by that test method. Selecting a test class or test package will cause the right-hand side to show application classes with code executed by all the test methods contained by the test classes or test package.

The right-hand table displays not only the names of the classes that were partially or fully covered by the test methods, but also the percentage of class' total coverage that is attributable to the test method (the test method's coverage contribution) as well as the percentage of the class' coverage that was attributable only to the selected test method (the test method's unique coverage).

The tree of tests and their containers on the left-hand tree can be presented in three different ways for maximum convenience:

1. Test Cases: shows "Project > Test Methods" (qualified with Test Class names)
2. Packages : shows "Project > Package > Test Class > Test Methods" layout
3. Source Roots: shows "Project > Source Root > Package > Test Class > Test Methods" layout



Test	Start	Rslt	Time	Message
Moneybags				
MoneyBagTest.testBagMultiply()	05.09.12 11:35	PASS	0.001s	
MoneyBagTest.testBagNegate()	05.09.12 11:35	PASS	0.001s	
MoneyBagTest.testBagNotEquals()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testBagSimpleAdd()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testBagSubtract()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testBagSumAdd()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testIsZero()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMixedSimpleAdd()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMoneyBagEquals()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMoneyBagHash()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMoneyEquals()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMoneyHash()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testNormalize2()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testNormalize3()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testNormalize4()	05.09.12 11:35	PASS	0.001s	

### Actions and Menus

The Test Run Explorer view allows the following actions (from the view toolbar, menu or both):

- **Enable/Disable Clover On...** Allows you to select multiple project to enable or disable Clover for. This allows you, for example, to quickly turn Clover on for all projects in your workspace in one action.
- **Refresh Coverage Data.** Re-loads from disk the Clover coverage data for the selected project.
- **Delete Coverage Recordings.** Deletes the coverage recording data recorded from test runs or applications runs for the selected project.
- **Clear Snapshot** Deletes the [Test Optimization](#) snapshot file
- **Compiled with Clover.** Toggles the use of Clover instrumentation when Eclipse compiles the selected Java project.
- **Edit Context Filter...** Allows you to edit the block and custom coverage 'contexts' used to help you filter out unwanted coverage data. An explanation of what contexts are can be found [here](#).
- **Generate Coverage Treemap.** Generates a mini report page in treemap format for the selected project. This report arranges package and classes enabling easy comparison of their complexity while also indicating their level of code coverage.
- **Generate Coverage Cloud.** Generates a report page in cloud tag format for the selected project. This lists classes considered project risks or quick wins.

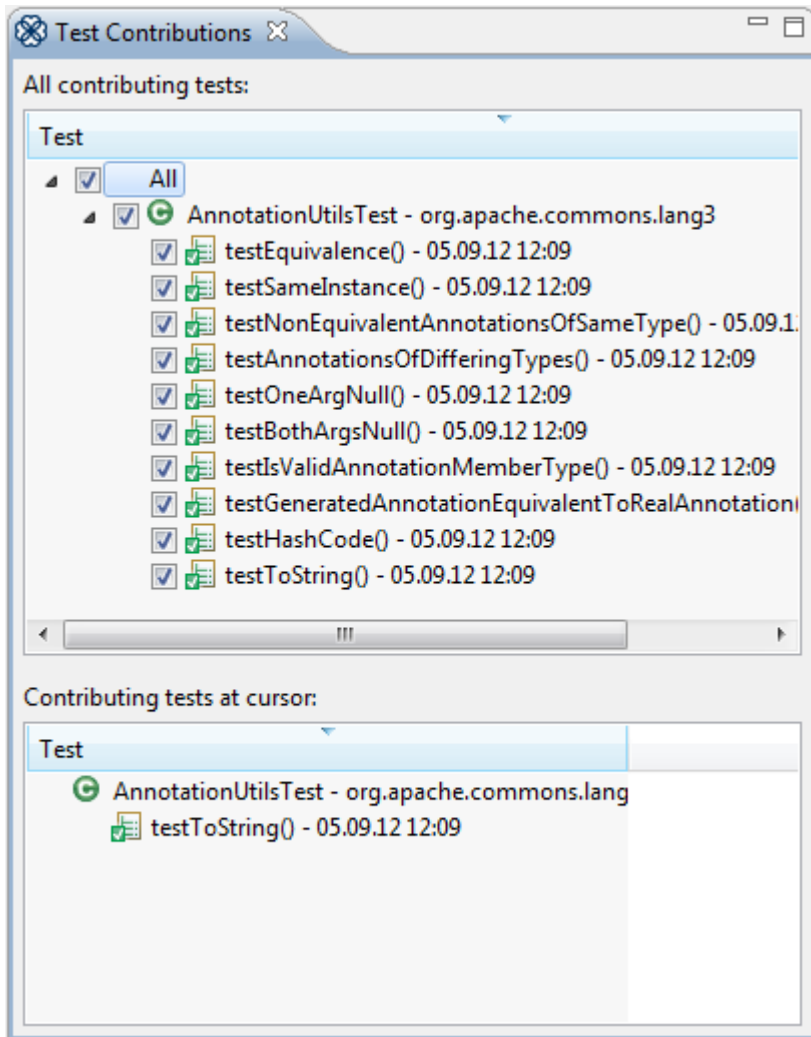
- **Coverage Reports > Run new report...** Launches the report generation wizard that will take you through the steps required to generate a PDF, HTML or XML. These reports can be generated for single or multiple projects.
- **Coverage Reports > View report** Browse recently generated reports.
- **Layout > Test cases** Shows "Project > Test Methods" layout
- **Layout > Packages** Shows "Project > Package > Test Class > Test Methods" layout
- **Layout > Source roots** Shows "Project > Source Root > Package > Test Class > Test Methods" layout
- **Coverage in Editors > Show All.** Shows red/green coverage areas in open Java editors. This is useful for finding out exactly which parts of the code are being covered.
- **Coverage in Editors > Show Uncovered.** Shows only red (uncovered) areas in open Java editors. This is useful for finding out exactly which parts of the code are not being covered while not cluttering your editor with the overwhelming large green areas (covered code).
- **Coverage in Editors > Show None.** Hides all red/green coverage areas in open Java editors.
- **Include passed only** If enabled, code coverage from failed tests will not be taken into account.
- **Show exclusion annotations** If enabled, draws small Clover icons in the Package Explorer view for every source file, indicating whether it's included or excluded from instrumentation.
- **Enable Clover Working Set.** Enables or disables usage of the Clover working set. This filters the files, directories and projects that Clover will report on and are especially handy for large projects.
- **Edit Clover Working Set...** Allows you to edit the files, directories and projects in the Clover working set.
- **Clear Clover Working Set.** Removes all files, directories and projects from the Clover working set.
- **About** Shows information about Clover version, copyright and licences.

Note that double-clicking on any element in left or right pane will navigate you directly to the source code.

## Test Contributions

The Test Contributions view shows unit tests and methods that generated coverage for the currently opened and selected Java source file. As you switch between Java source file editors, the top most tree is updated with the test methods that contributed to coverage of this file. Clicking on any of the tests and their methods will take you to their associated source. Unchecking a test or test method will remove the coverage it provides from the open source file. The bottom tree tracks the test methods that contributed to coverage of the file at the current cursor position.

The Test Contributions view allows you to better understand the relationship between your test code and your application code as you move between Java source files.



You have already learned how to navigate through code coverage and test results. But don't you have a feeling that your coverage reports could be more accurate when focused on *really* important areas of application? If you do so, don't hesitate to read [4. Scope of instrumentation in Eclipse](#) to learn how to configure instrumentation scope from whole projects down to a single line of code.

## 4. Scope of instrumentation in Eclipse

Clover provides many ways to fine-tune instrumentation scope, which gives you an ability to concentrate your work on the most important code.

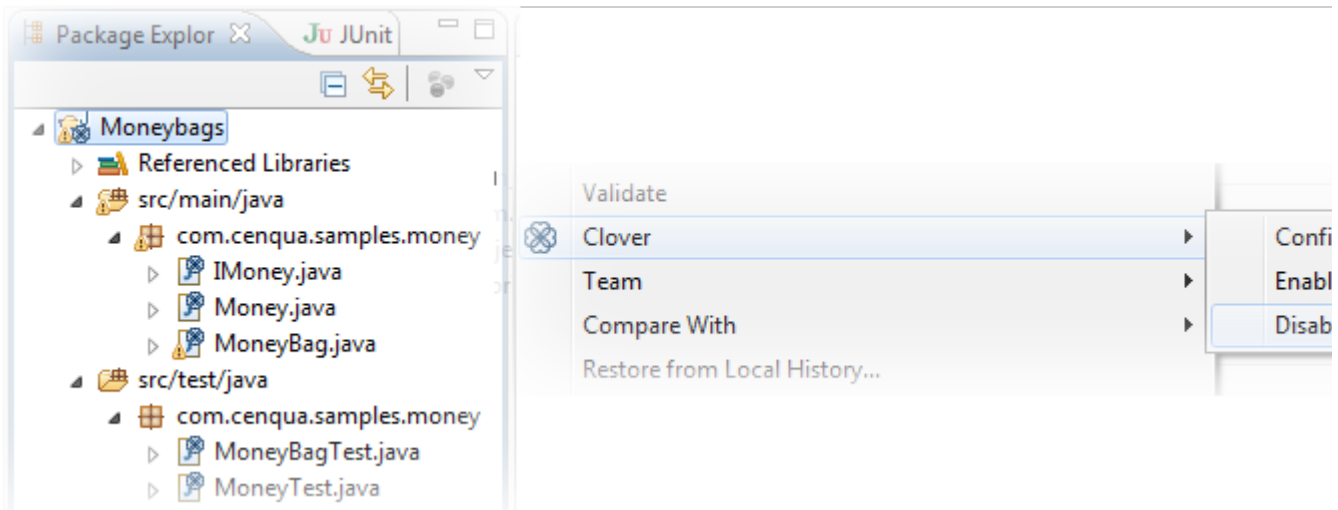
- Enabling and disabling Clover for selected projects
- Enabling and disabling instrumentation for Clover-enabled projects
- Excluding and including packages
- Excluding and including files
- Excluding certain blocks of code
- Excluding methods and statements matching regular expression
- Excluding arbitrary lines of code
- Setting instrumentation detail level
- Showing Clover coverage annotations in Java source editors

### ***Enabling and disabling Clover for selected projects***

If you want to completely disable Clover support for a project (which will remove it from the three Clover views,



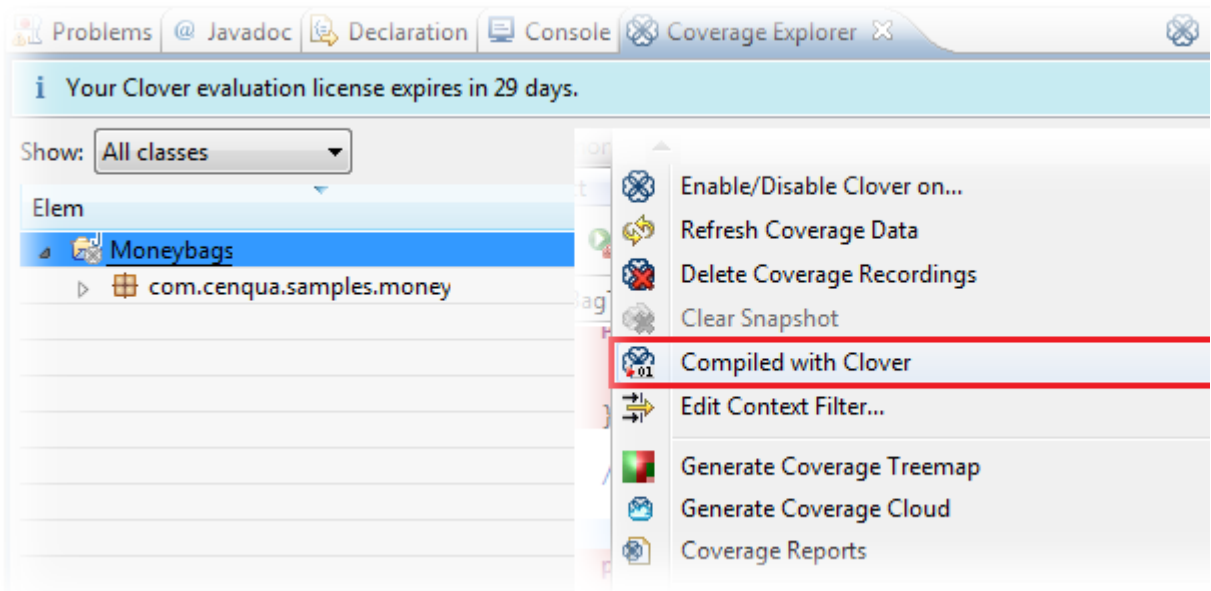
remove all Clover data etc), then right click on the project and select "*Clover > Disable on this Project*". If you wish to enable/disable Clover on multiple projects, right click on one of them and select "*Clover | Enable/Disable on...*" and select the projects you wish to have Clover enabled/disabled for.



### Enabling and disabling instrumentation for Clover-enabled projects

In order to track the code coverage of your projects, Clover must insert special code into your programs at compilation time - called instrumentation - to record this coverage. When Clover is enabled on your projects, Clover will automatically perform this task for every file you compile in the project. You can tell Clover not to instrument your projects by:

- right clicking on them in the *Coverage Explorer* or *Test Run Explorer* views and deselecting "*Compiled with Clover*" option or

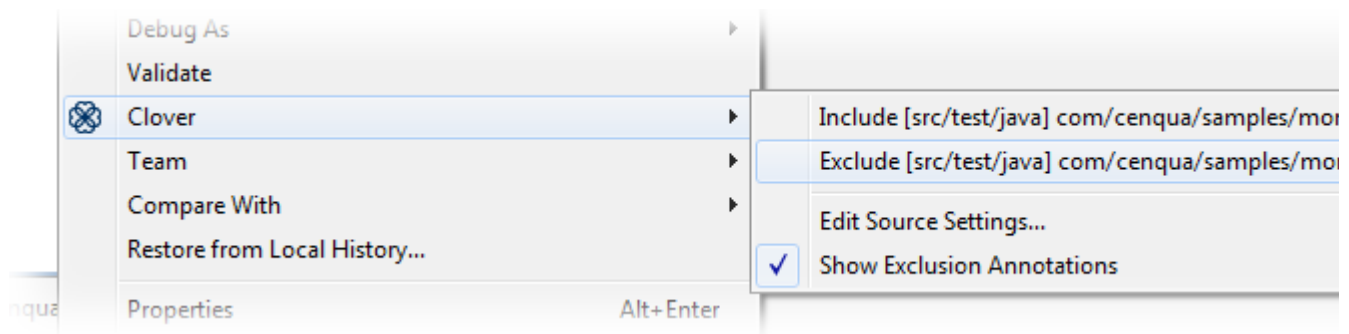


- selecting them in the *Coverage Explorer* or *Test Run Explorer* views and clicking "*Toggle whether Clover Instrumentation...*" button



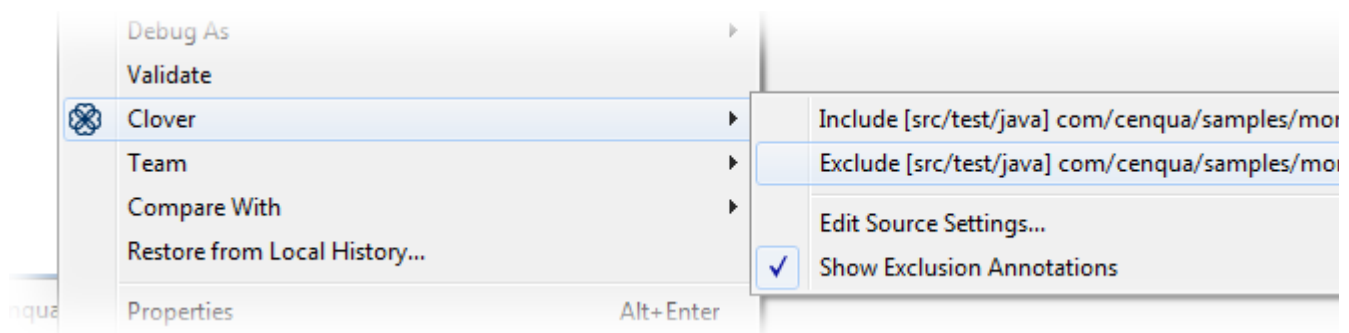
### Excluding and including packages

Right click on a package in *Package Explorer* view, choose "Clover > Include/Exclude <package name>" from context menu.



### Excluding and including files

Right click on a file in *Package Explorer* view, choose "Clover > Include/Exclude <file name>" from context menu.

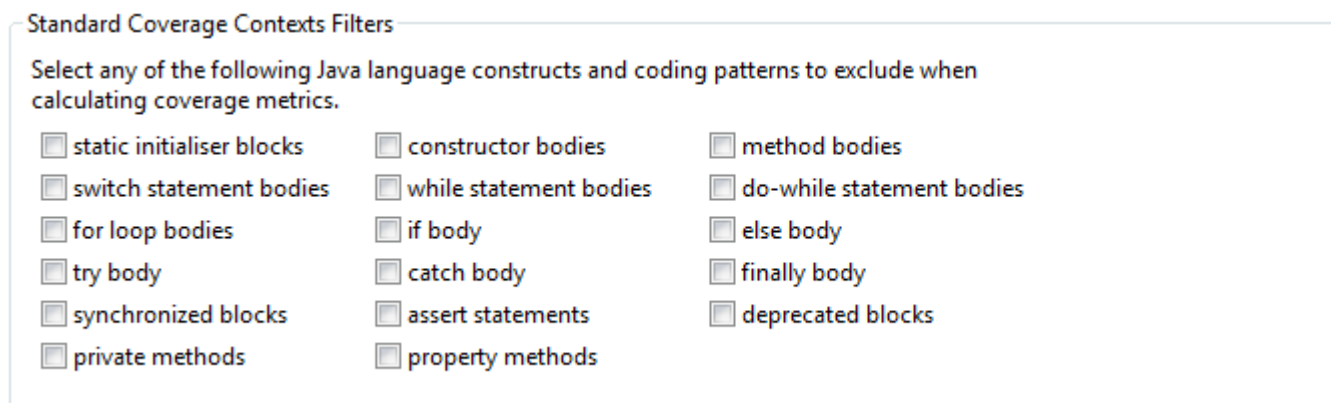


### Excluding certain blocks of code

Right click on a project in *Package Explorer* view, choose "Properties" from context menu, next "Clover > Instrumentation" tab.

In the *Standard Coverage Context Filters* you can choose Java language constructs or coding patterns to be excluded. The most interesting are:

- assert statements
- catch body
- finally body
- private methods (all methods having private keyword)
- property methods (all methods having name like getXYZ() / setXYZ() / isXYZ(), being public and having no arguments for isXYZ/getXYZ)

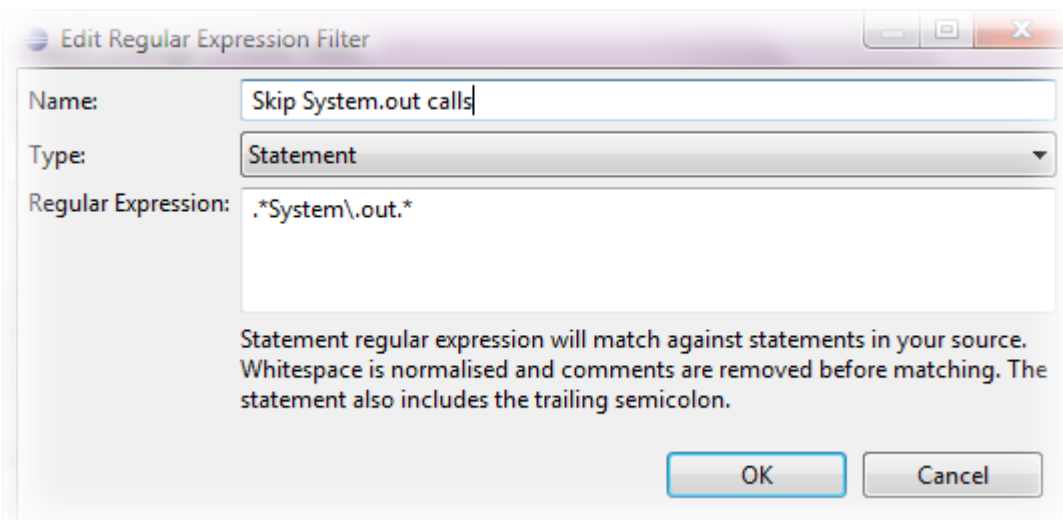
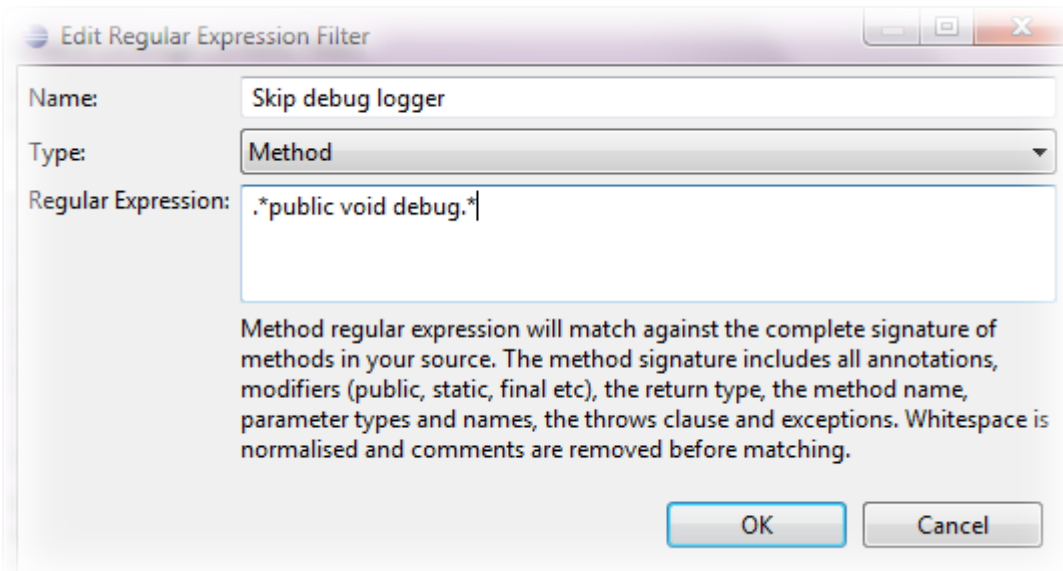




### Excluding methods and statements matching regular expression

Right click on a project in *Package Explorer* view, choose "Properties" from context menu, next "Clover > Instrumentation" tab.

In the *Custom Coverage Context Filters* you can define regular expressions for method signatures and statements.



### Excluding arbitrary lines of code

Put `///CLOVER:OFF` and `///CLOVER:ON` in source code (note that three slashes are used) to exclude given sections.

```

public Money(int amount, String currency) {
    fAmount = amount;
    fCurrency = currency;
}

/**
 * Adds a money to this money. Forwards the request to the addMoney helper.
 */
public IMoney add(IMoney m) {
    return m.addMoney(this);
}

///

```

**Setting instrumentation detail level**

Left click on a project in Coverage Explorer view. A pop-up will show with "Instrument and compile at" having two options:

- **statement level** - it is a recommended setting as it gives the most accurate information about code coverage; Clover will record information about: method calls, statement calls and whether given branch was evaluated to true/false.
- **method level** - it records only whether given method was called or not; this option can be useful for very large projects (as it produces smaller instrumentation overhead and code compilation is faster) or when you need just rough information about coverage

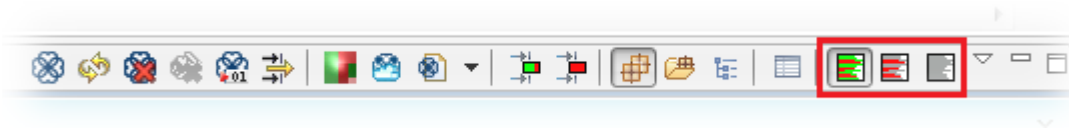
Elem	Cov%	Av Me Cpx	C
commons-lang3-3.1-src		2,5	5 68
org.apache.commons		1,2	2
org.apache.commons		2,4	49
org.apache.commons		2,7	10
org.apache.commons		2,3	78
org.apache.commons.lang3.reflect.testbed	100,0%	1,0	

**Showing Clover coverage annotations in Java source editors**

If you wish to temporarily disable the red/green code coverage annotations in your Java source editors (but wish to continue using Clover on your projects), you can simple toggle one of three toggles:

- "Show All"
- "Show Uncovered"
- "Show None"

This setting applies to all Clover-enabled projects in the workspace.



Now you have your project instrumentation tuned to your needs. Are you looking for more tweaks? Read the [5. Eclipse configuration options](#) chapter.

## 5. Eclipse configuration options

- [Project Properties](#)
  - [Clover - Instrumentation](#)
  - [Clover - Contexts](#)
  - [Clover - Source Files](#)
  - [Clover - Test Classes](#)
- [Global Preferences](#)
  - [Clover - General](#)
  - [Clover - License](#)
  - [Clover - Test Optimization](#)
  - [Text Editors - Annotations](#)

The Clover Eclipse plugin's configuration can be accessed in three places:

- From the "Clover" page of a project's properties dialog (Project > Properties > Clover).
- From the "Clover" page of the workspace preferences (Window > Preferences > Clover).
- From the "Annotations" page of the workspace preferences (Window > Preferences > General > Editors > Text Editors > Annotations)


### Project Properties

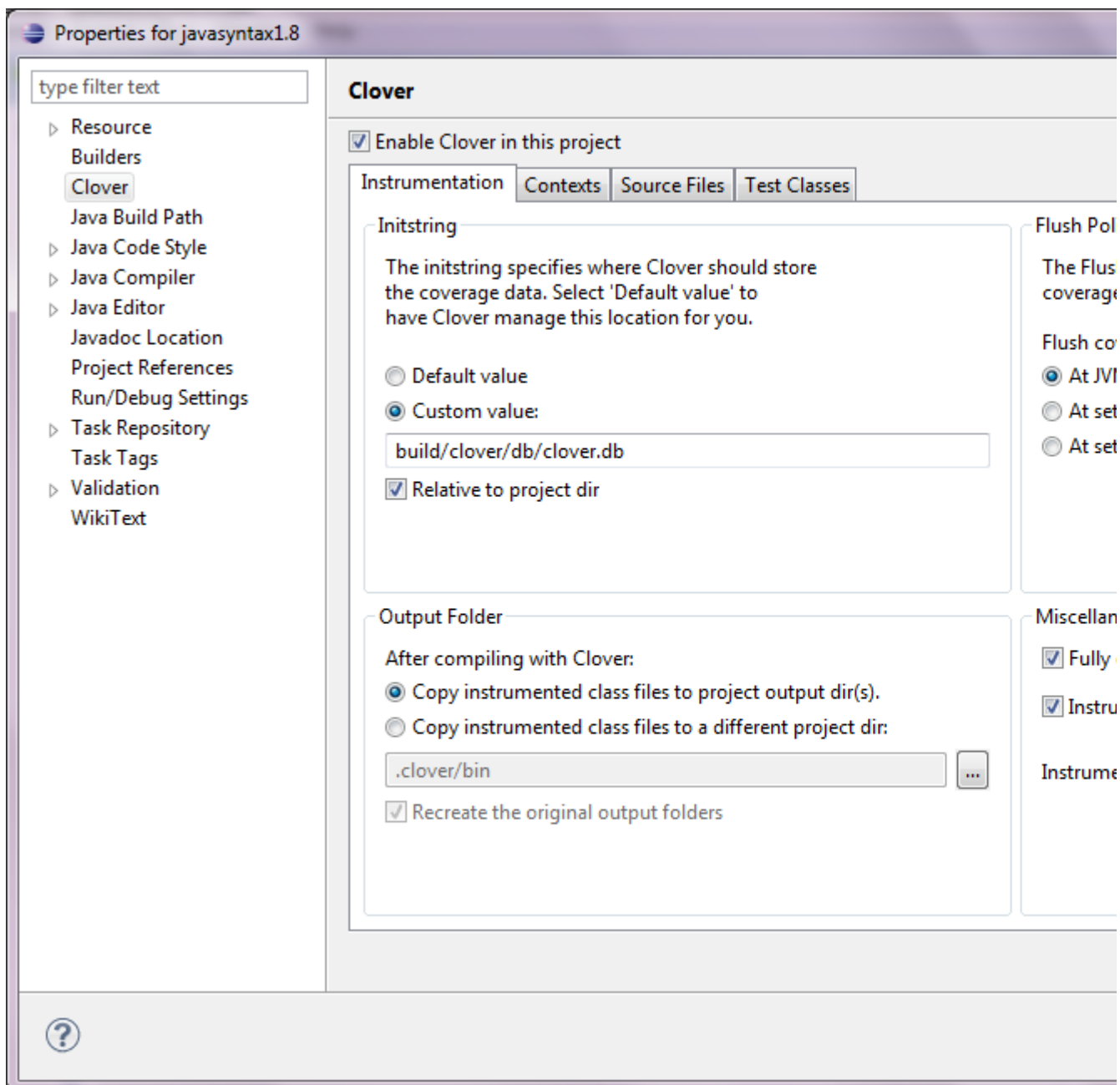
#### Clover - Instrumentation

These options control how Clover instrumentation works when "Compile with Clover" is selected on your project.

- **Initstring** - This controls where the Clover plugin stores (and looks for) the coverage database. You may want to specify a custom value if you want the Clover coverage data generated to be available to an external merge task or report generation.
- **Flush Policy** - The Flush Policy controls how Clover writes coverage data to disk when your application runs.
  - "At JVM shutdown and on special instruction" is the default and means coverage data is written to disk when the JVM shutsdown or when Clover encounters an inline comment directive instructing it to flush.
  - "At set intervals" allows you to specify that coverage data should be written out at regular intervals.
  - "At set intervals from a Clover thread" causes Clover to create a separate thread within your application which regularly flushes coverage data to disk. See [Flush Policies](#).
 These options control the compilation of the instrumented source code that Clover generate.
- **Output Folder** - This specifies where instrumented classes are placed after compilation.
  - "Copy instrumented class files to project output dir(s)" will cause instrumented classes to be placed in the same folder or folders as Eclipse would normally place class files according to your project settings. In this mode Clover will only output instrumented class files.
  - "Copy instrumented class files to a different project dir" will place instrumented class files in the

folder specified by you and continue to place un-instrumented class files in the project's configured output folder(s).

- "Recreate original output folders" if checked, instrumented packages and classes will be placed in subfolders that mimic the original output folders relative to the project root
  - **Miscellaneous** - other settings
    - "Fully qualify instrumentation references to java.lang classes" - if checked, all classes from *java.lang* package will be referenced using their fully qualified name; this option avoids name conflicts in case your project contains classes named the same as those from *java.lang* package.
    - "*Instrument and compile at: statement level / method level*" - statement level instrumentation is more accurate but has a runtime performance penalty, method level instrumentation is less accurate but will run faster. We recommend using statement level unless you have a performance issue. (*Since Clover 3.1.8*).
    - "*Instrument lambda functions*" - whether Java 8 lambda functions shall be instrumented. If instrumented, they're treated like normal methods (they can be shown in HTML report and considered in code metrics, for example). Possible values:
      - *none* - do not instrument lambda functions,
      - *expression* - instrument lambdas in expression-like form, e.g. "*(a, b) -> a + b*",
      - *block* - instrument lambdas in code blocks, e.g. "*(a, b) -> { return a + b; }*",
      - *all* - instrument all lambda functions.
-  Due to Clover's restrictions related with code instrumentation and javac compiler's type inference capabilities, you may get compilation errors when expression-like lambda functions are passed to generic methods or types. In such case disable instrumentation of expression-like form (i.e. use the *none* or *block* setting). See the [Java 8 code instrumented by Clover fails to compile](#) Knowledge Base article for more details. (*Option available since Clover 3.3.0*).



## Clover - Contexts

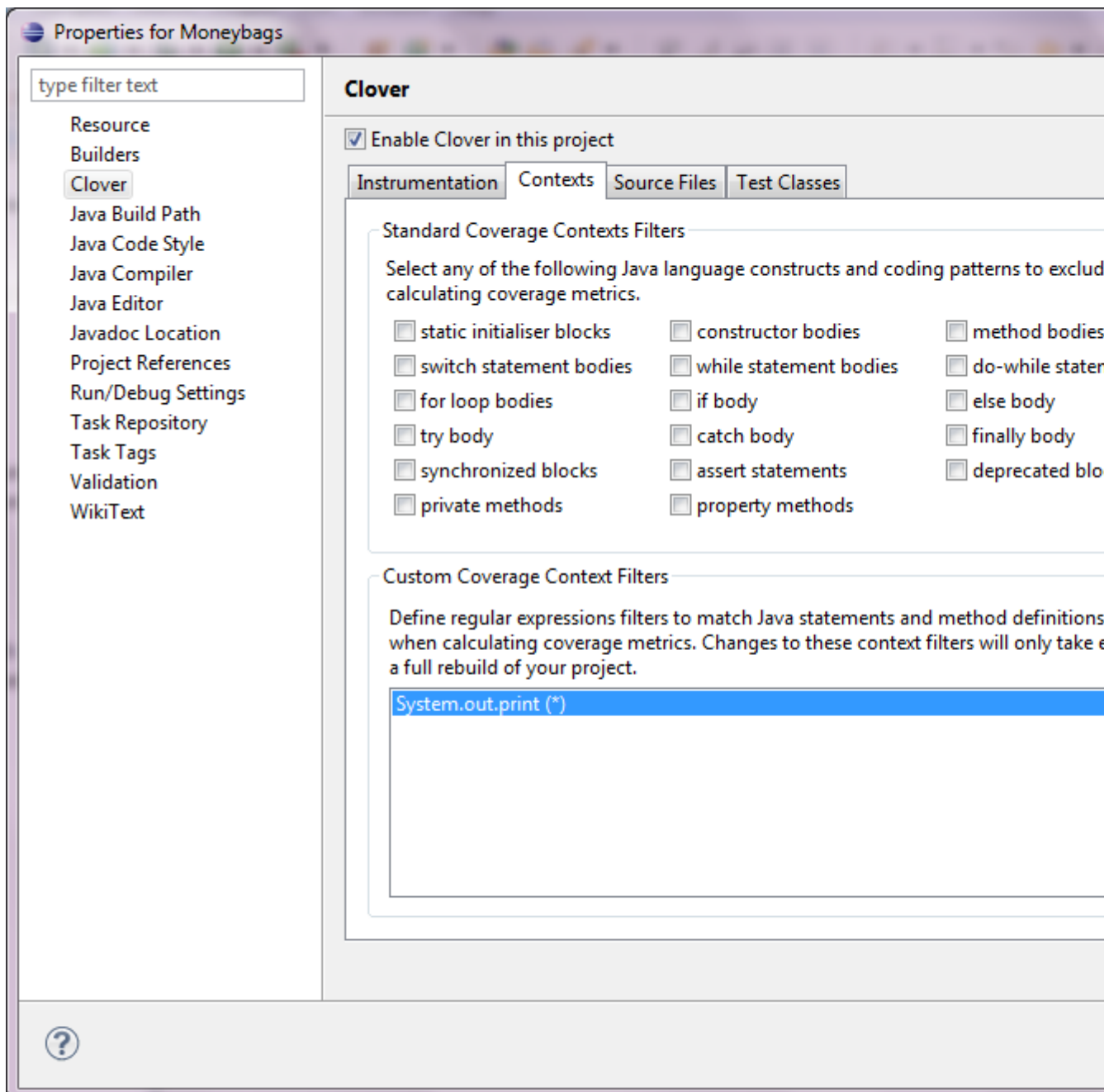
Clover's coverage contexts allow you to filter out certain parts of your project's code when considering how much of your code is covered. This is useful when you do not have 100% code coverage yet don't feel the uncovered code is of sufficient importance to invest time to test. See [Using Coverage Contexts](#) for an in-depth discussion of contexts and how they are used.

Clover recognises two main types of contexts - block contexts which are pre-defined by Clover, and custom contexts which are defined by you.

- **Block Contexts** - these are pre-defined by Clover and refer to common Java coding constructs or idioms such as the body of if statements; static initialiser blocks; or property style methods.
- **Custom Contexts** - Are defined by you and come in two flavours - method contexts and statement contexts.
  - *Methods contexts* are specified as regular expressions that match methods in your classes. For example, a method context of "(.\*)?public .(foo|bar).\*" will identify any public method named foo or bar. You could then later filter out coverage for such methods if you considered them

unimportant to your project's total coverage figures. See [Using Coverage Contexts - Method Contexts](#).

- *Statement contexts* are also specified as regular expressions and they match statements in your methods. For example, a statement context of `".LOG\debug."` would identify statements that perform debug level logging. You could then filter out coverage for these logging statements if you found their coverage (or lack of coverage) too distracting.



### Clover - Source Files

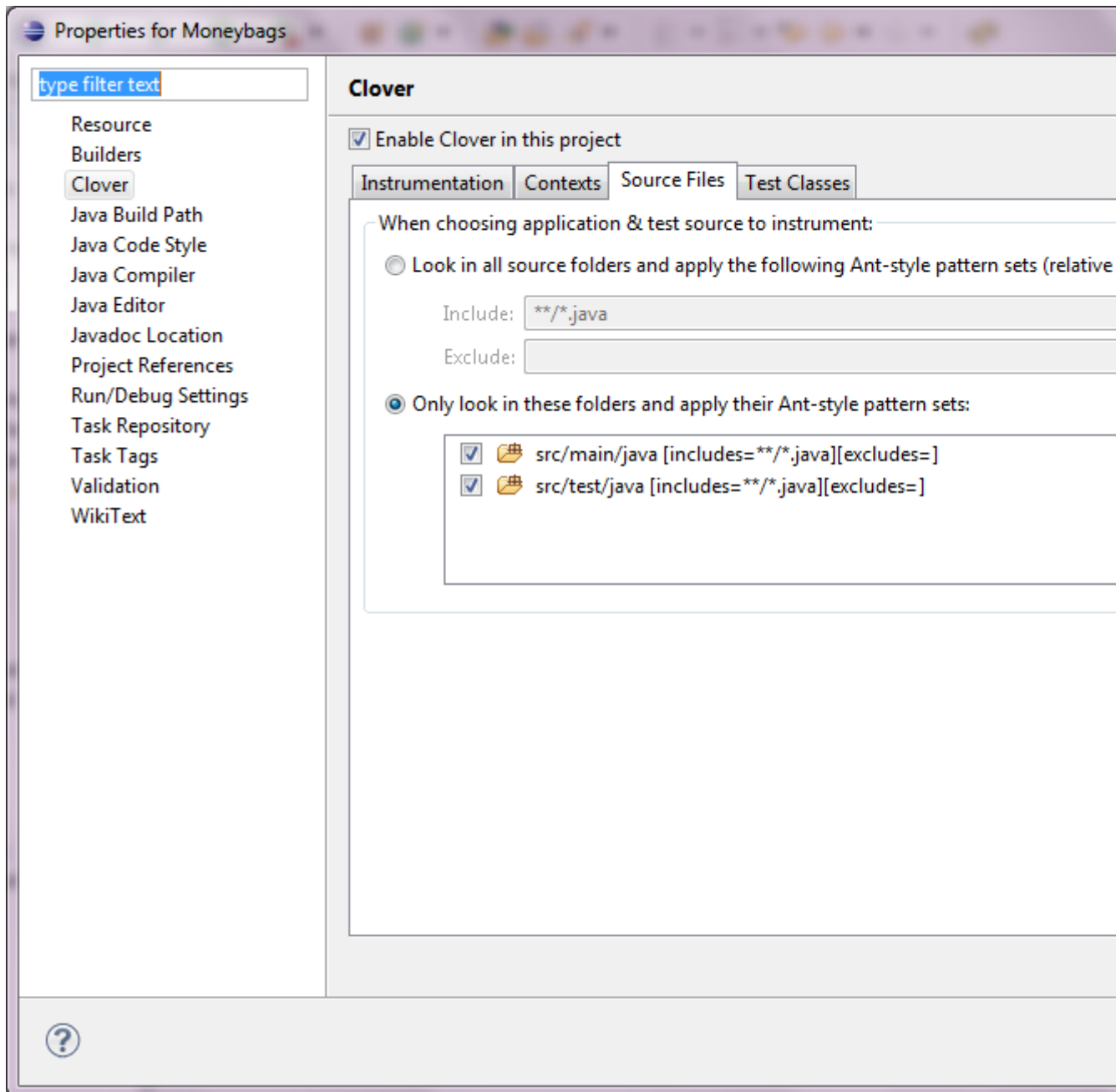
The Source Files tab let you select in detail what exactly should be instrumented. You can select one of the following:

- A global pattern for including and/or excluding source files.
- A per-source root selection (including per-source root includes/excludes).

This configuration window allows you to use Ant-style pattern sets. If there are only certain source files you want instrumented, then by using these Ant-style include and exclude pattern sets you can fine-tune how Clover determines whether files are eligible for instrumentation. For example, by using an "Excludes" value of `**/remote`

`/*.java` you will stop instrumentation of files in the "remote" folder of your project.

- **Look in all source folders and apply the following Ant-style pattern sets (relative to the project root) -**  
Applies your pattern sets to all source folders.
- **Only look in these folders and apply their Ant-style pattern sets (relative to the project root) -**  
Applies your pattern sets to all specified set of folders.

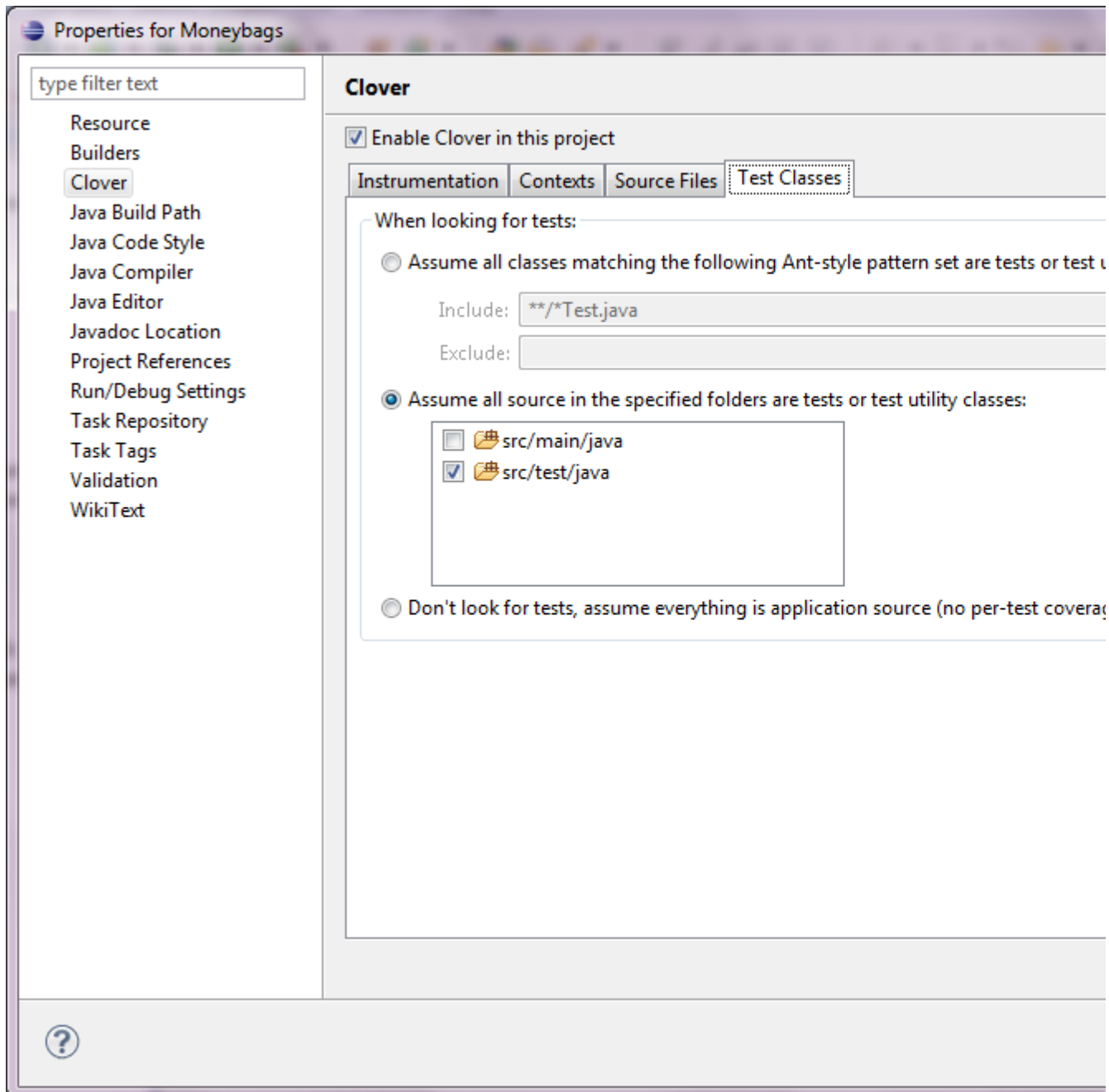


### Clover - Test Classes

The Test classes tab lets you select how Clover should look for your tests. You can select from one of the following:

- A global pattern for including and/or excluding source files.
- A per-source root selection.
- None (Clover makes no attempt to look for tests).





## Global Preferences

### Clover - General

These options control how Clover works across all projects which have "Compile with Clover" enabled.

- **When rebuilding project** - when you rebuild a project, Clover will ask you whether you want to delete the existing coverage information. This section allows you to specify what the default action should be, and whether Clover should prompt you at all.
- **Coverage visualisation should**
  - **Refresh automatically every Ns** - if enabled, the plugin will check for updated coverage data at the frequency given below and display and new coverage. If it is not enabled, then you will need to use the "Refresh Coverage Data" button to see updated coverage data.
  - **Span coverage before build** - the span attribute allows you to control which coverage recordings are merged to form a current coverage report. For more information, see [Using Spans](#).
- **When changing the coverage context filter for a project** - when you change context filter, Clover can perform full rebuild of a project or let you rebuild it manually; "Ask me each time" sets whether Clover should prompt about it

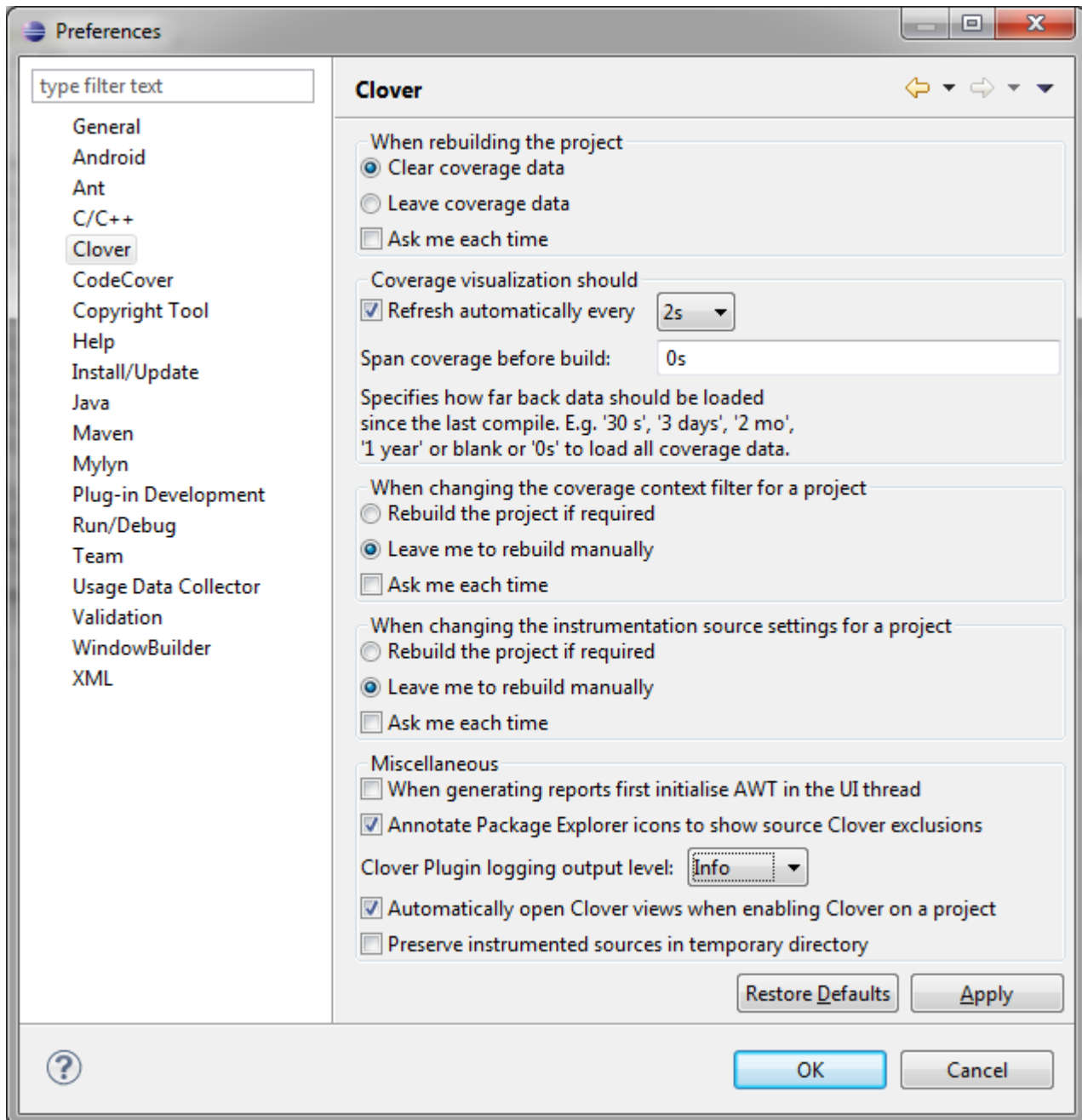


- **When changing the instrumentation source settings for a project** - similar as above
- **Miscellaneous settings**
  - **When generating reports first initialise AWT in the UI thread** - is a compatibility setting for certain systems. Please leave this in its default setting unless advised otherwise by Atlassian support personnel.
  - **Annotate Package Explorer icons to show source Clover exclusions** - If enabled, Clover will add small icons in the Package Explorer view for every source file, indicating whether it's included or excluded from instrumentation.
  - **Clover plugin logging output level** allows you to set the logging output of the plugin to one of four levels: `info`, `verbose`, `debug` or `none`. Output will be written to standard Eclipse log file (`<eclipse_workspace>\.metadata\.log`).
  - **Automatically open Clover views when enabling Clover on a project**

When selected, Clover will automatically open four standards views ("Coverage Explorer", "Test Run Explorer", "Test Contributions", "Clover Dashboard") in the current perspective every time Clover is being enabled on a project (e.g. by right click on a project and selecting "Clover -> Enable/Disable on"). De-selecting this option is useful when you don't use some of Clover views and don't want to have them re-appearing
  - **Preserve instrumented sources in temporary directory**

When enabled, Clover will keep a copy of every instrumented source file in the temporary directory (in Eclipse workspace). This allows to see the exact content of file which was passed to the compiler. Use this option for troubleshooting only, i.e. when you believe that code was incorrectly instrumented. There is no reason to have this option enabled during normal work. Files will not be automatically removed, so you'd have to clean temporary directory manually. Files are stored in a structure reflecting project layout. When you enable this option, the "Logging Level" will automatically switch to "Info". After every compilation you can find a message in Error Log view about location of instrumented files, for example:

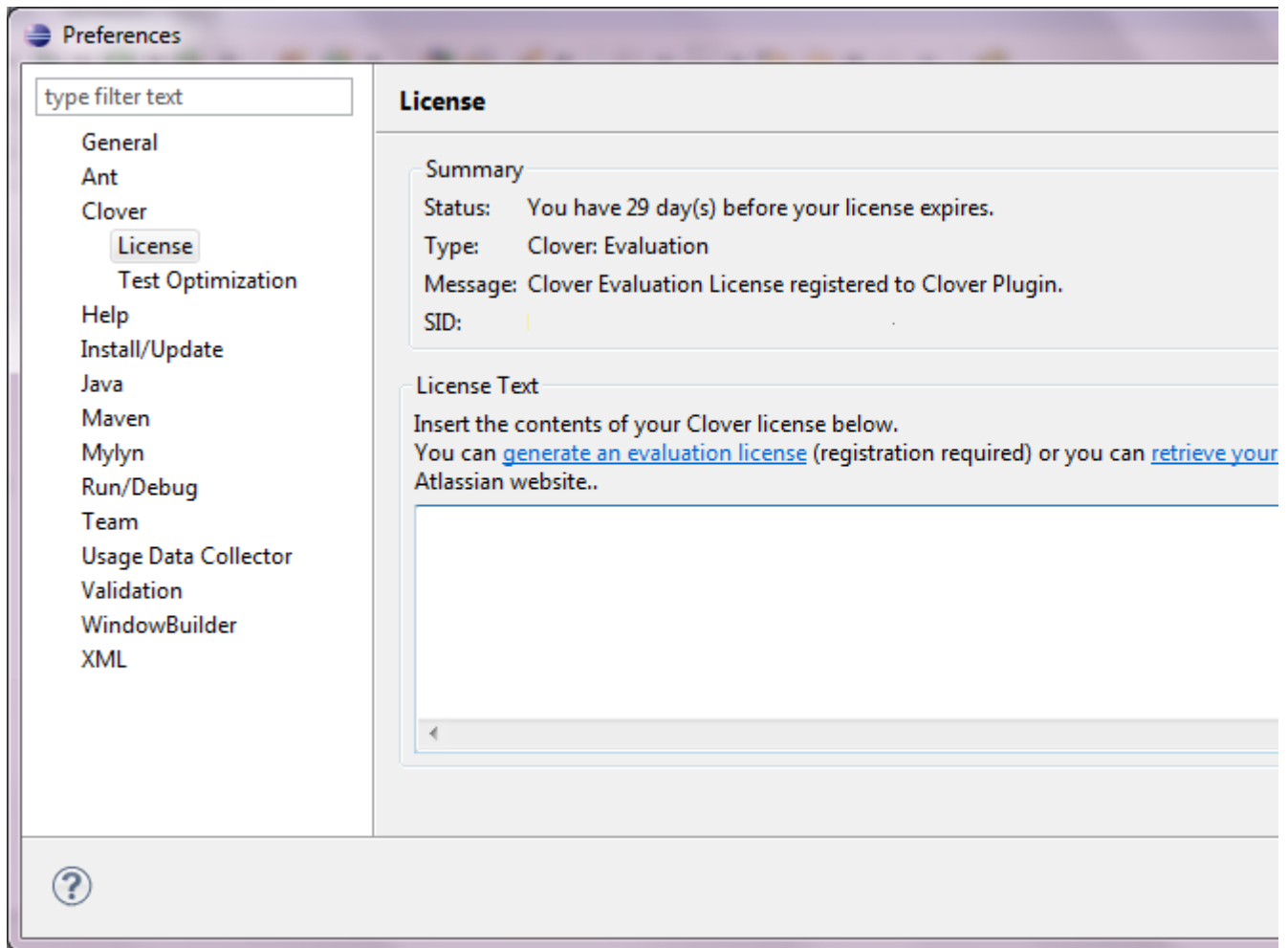
```
CLOVER: Instrumented sources have been preserved in  
<workspace>\.metadata\plugins\org.eclipse.core.resources\projects\Moneybags\com.cenqua.clover.core\CLOV_INSTR_SRC
```



## Clover - License

This option allows you to view and change the license you use with Clover.

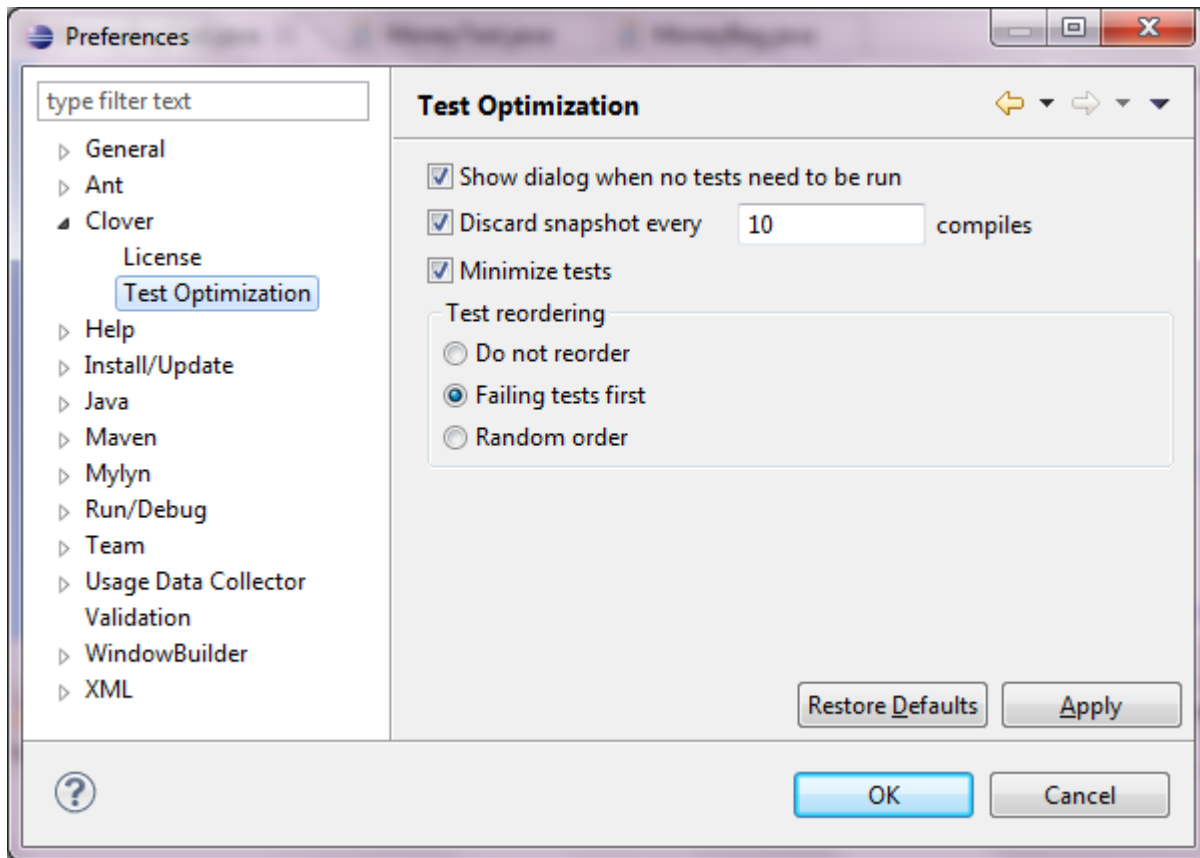
- **License text** — When you receive a Clover license from Atlassian, you should copy its contents to this text box. If the license text is on the clipboard then clicking the "Paste" button will paste it to this text box automatically. If you have saved the Clover license to a file then clicking the "Load..." button will allow you to select the file and fill the text box with its contents.
- **Summary** — The summary box summarises the nature of the currently-enabled Clover license.



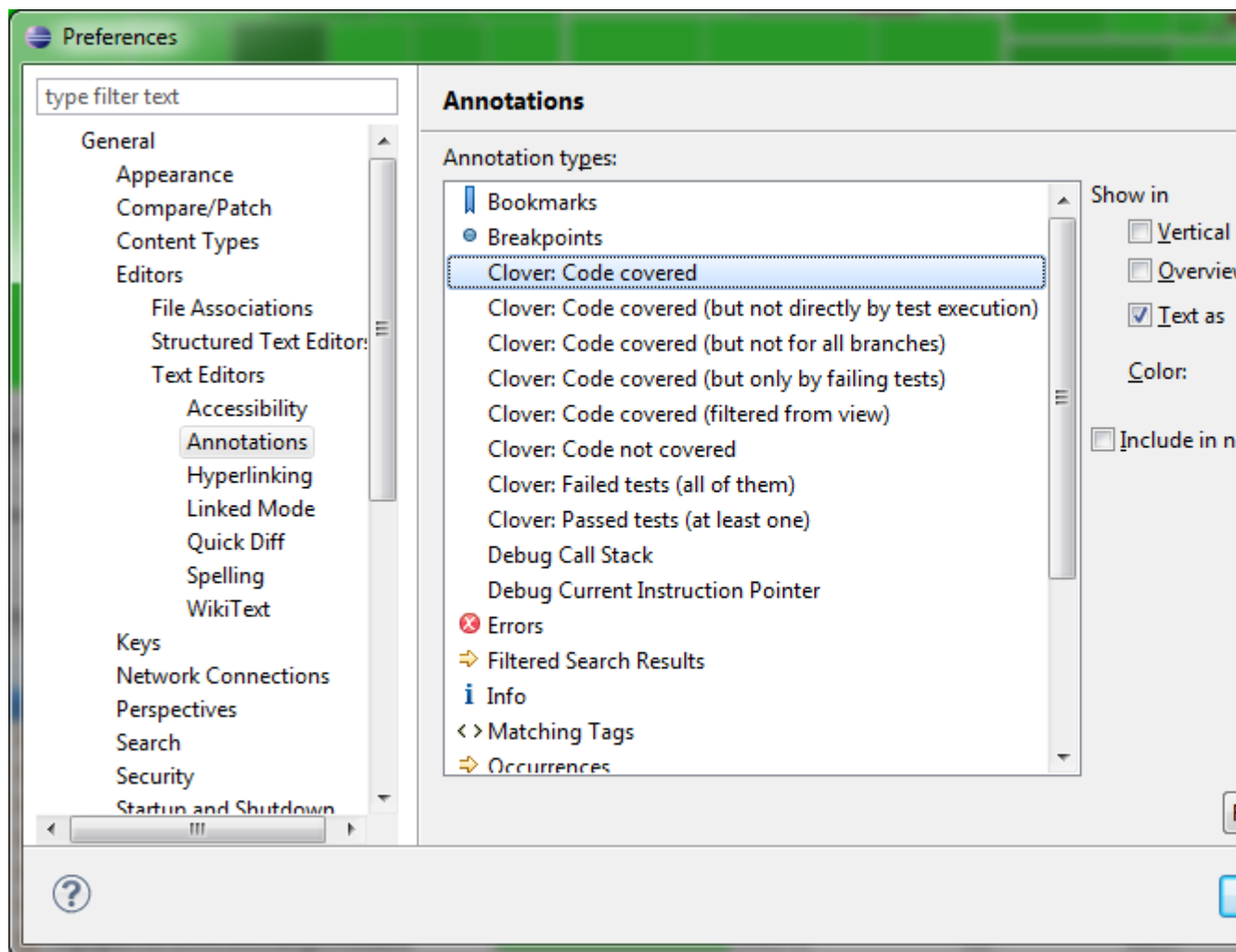
## Clover - Test Optimization

This page allows to view and change Test Optimization settings.

- **Show dialog when no tests need to be run** - if enabled, Clover will show pop-up in case when due to test optimization no test case have been executed
- **Discard snapshot every N compiles** - if enabled, Clover will delete test optimization file after N builds (thus causing that no tests will be optimized during next test run)
- **Minimize tests** - if enabled, Clover will execute only these tests which have failed or these which are affected by code changes; otherwise all tests are executed
- **Test reordering**
  - **Do not reorder** - no change in test execution order
  - **Failing tests first** - tests are first sorted by test result from previous run (failing first) and next by their execution time (shortest first)
  - **Random order** - tests are shuffled in random order; this option is good for detecting unwanted cross-test dependencies



### Text Editors - Annotations



Code markers are as follows:

- **Clover: Code covered** - code which has been covered during execution
- **Clover: Code covered (but not directly by test execution)** - code which has been covered, but there is no related test case (e.g. from a call of the *main()* method)
- **Clover: Code covered (but not for all branches)** - code with a partial branch coverage (caused when only one part of a branch has been covered)
- **Clover: Code covered (but only by failing tests)** - code which has been covered, but all related test cases are failing
- **Clover: Code covered (filtered from view)** - code which has been [filtered out](#)
- **Clover: Code not covered** - code which has been never executed
- **Clover: Passed tests (at least one)** - colour for overview ruler marker stripe indicating that at least one passing test case is related with given line
- **Clover: Failed tests (all of them)** - colour for overview ruler marker stripe indicating that all tests related with given line are failing

Now you have tweaked and hacked Clover according to your developer needs. But you would like to share information about code coverage with you colleagues or present it to management? If yes, read next chapter: [6. Generating reports in Eclipse](#).

## 6. Generating reports in Eclipse

- [Introduction](#)
- [Generating a Report](#)
  - [Select an Output Format](#)
  - [Select the Source Project](#)
  - [Configure General Settings for the Report](#)

- Configuring a Report Filter
- Configuring Your Report JVM
- Finalise the Report
- Opening the Generated Report

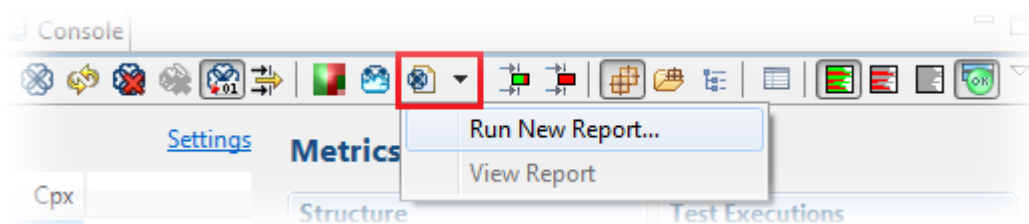
## Introduction

The Clover-for-Eclipse plugin allows you to generate HTML, PDF or XML reports from **one or more** Clover-enabled projects in the current workspace.

## Generating a Report

To create a report, select a project or source file then click the left-hand side of the Report Button in Eclipse. The **'Generate Report: Report Format'** dialog opens.

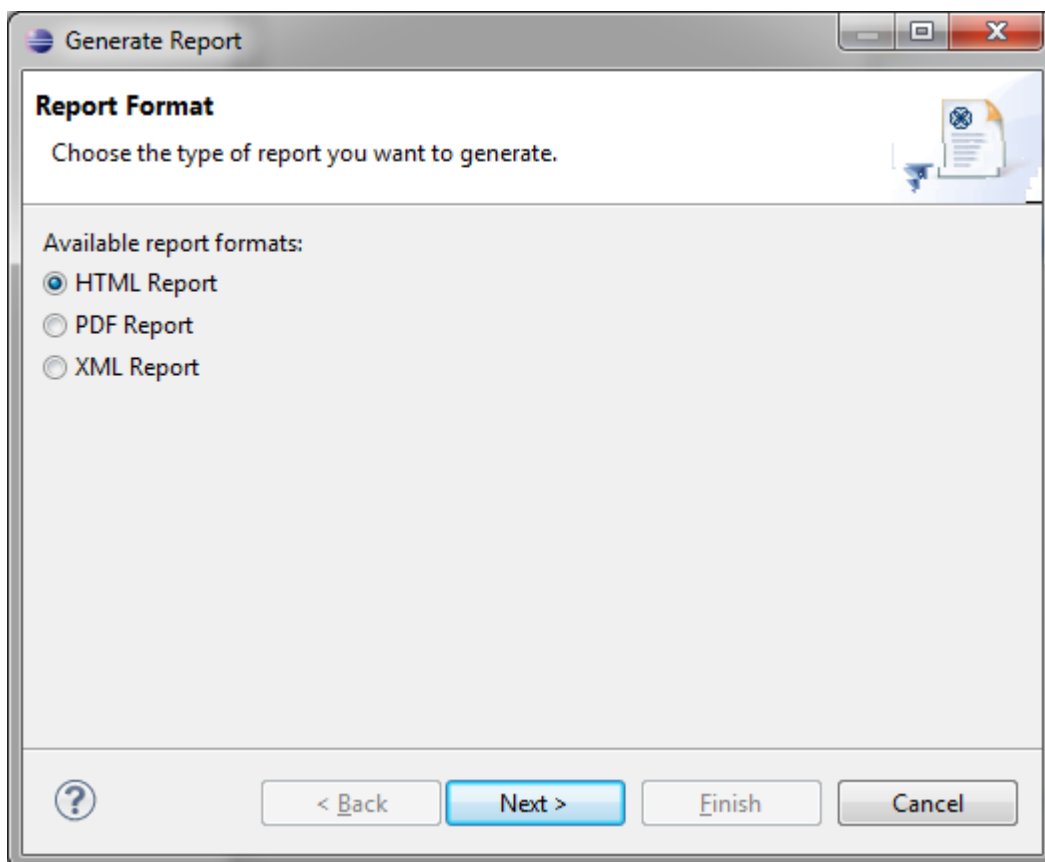
**i** It doesn't matter whether you select a project or an element of a project — the reports operate at the project level. Currently, Clover doesn't support reporting on the sub-sets of a project.



### Select an Output Format

For your report, you can select an output format of HTML, PDF or XML.

To select the desired output format, click the corresponding radio button in the **'Generate Report: Report Format'** dialog and click **'Next'**. The **'Generate Report: Project Selection'** dialog opens.

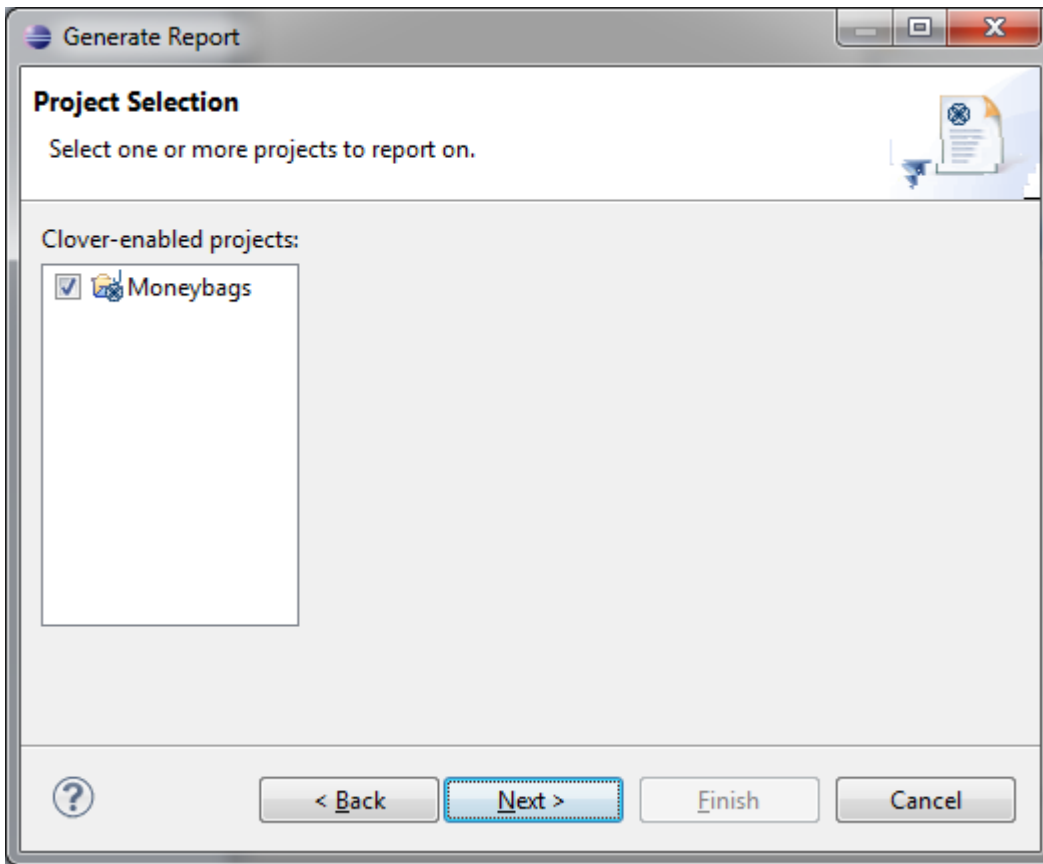


### Select the Source Project

A list of available projects will be displayed. To choose one, click the tick box shown next to the desired project

and click **Next**. A dialog box with specific settings for the chosen report type opens.

**i** More than one project can be selected (this would be used for instance if the user has multiple related projects or has application code in one project, with test code in another).

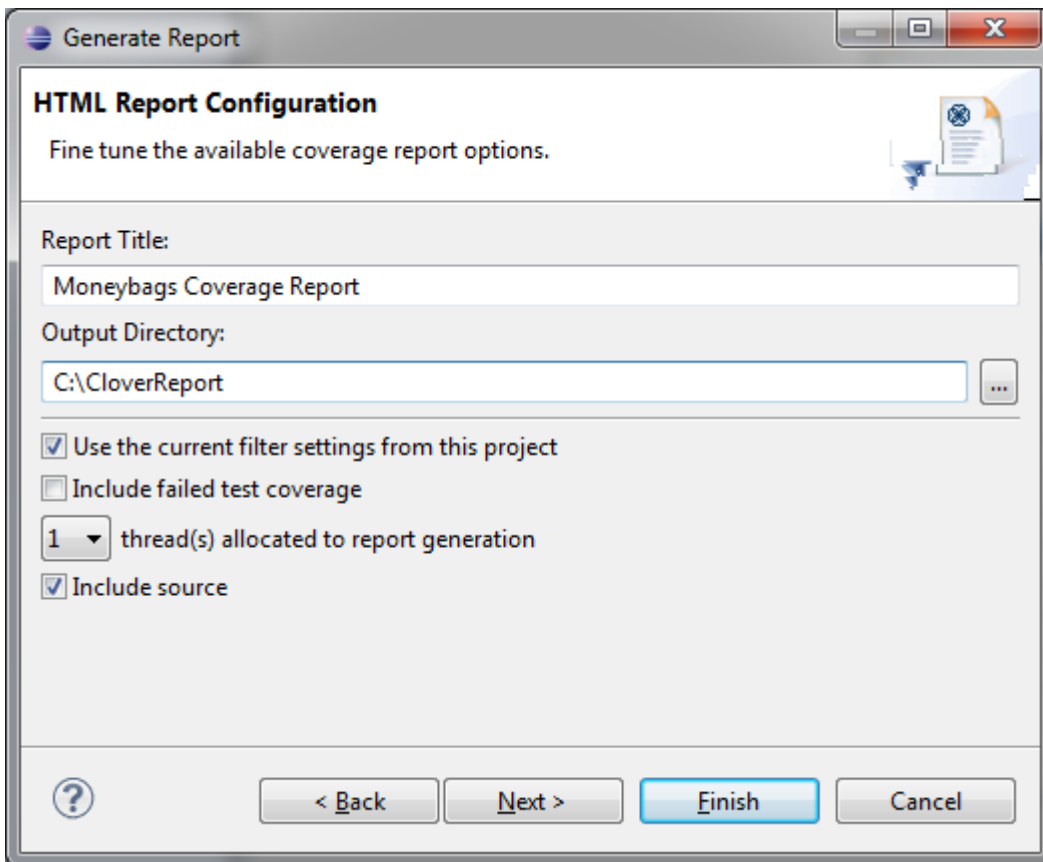


#### Configure General Settings for the Report

Your report can make use of the following settings:

Setting	Default	Description
Report Title	<project name> Coverage Report	Report title.
Output directory	<project dir>report/html	Directory where report will be written to. In case when report already exists in this location, appropriate warning will be displayed.
Use current filter settings from this project	True	If the user un-checks this they will be given the opportunity to set a custom context.
Include failed test coverage	True	Tests from failed tests are included by default but can be excluded if they wish to discount this as worthy of being reported.
Threads allocated (only for HTML reports)	1 (range: 1-4)	Using more threads may product the report faster but will use more memory. 1 is recommended for large projects.
Include source (only for HTML reports)	True	Whether to include source code in the HTML reports. Not including source will mean users can't see per-line coverage information but report generation will run faster.

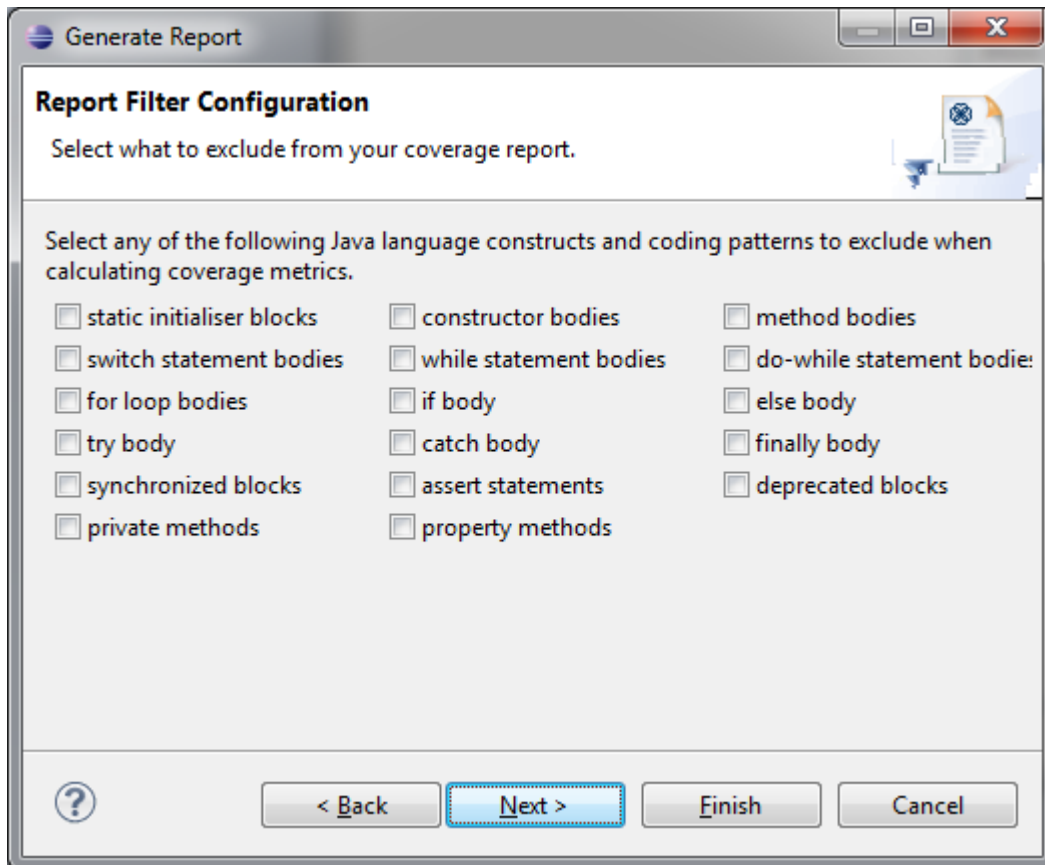
Include line info (only for XML reports)	False	Whether line by line coverage information is added to the report.
--	-------	---



### Configuring a Report Filter

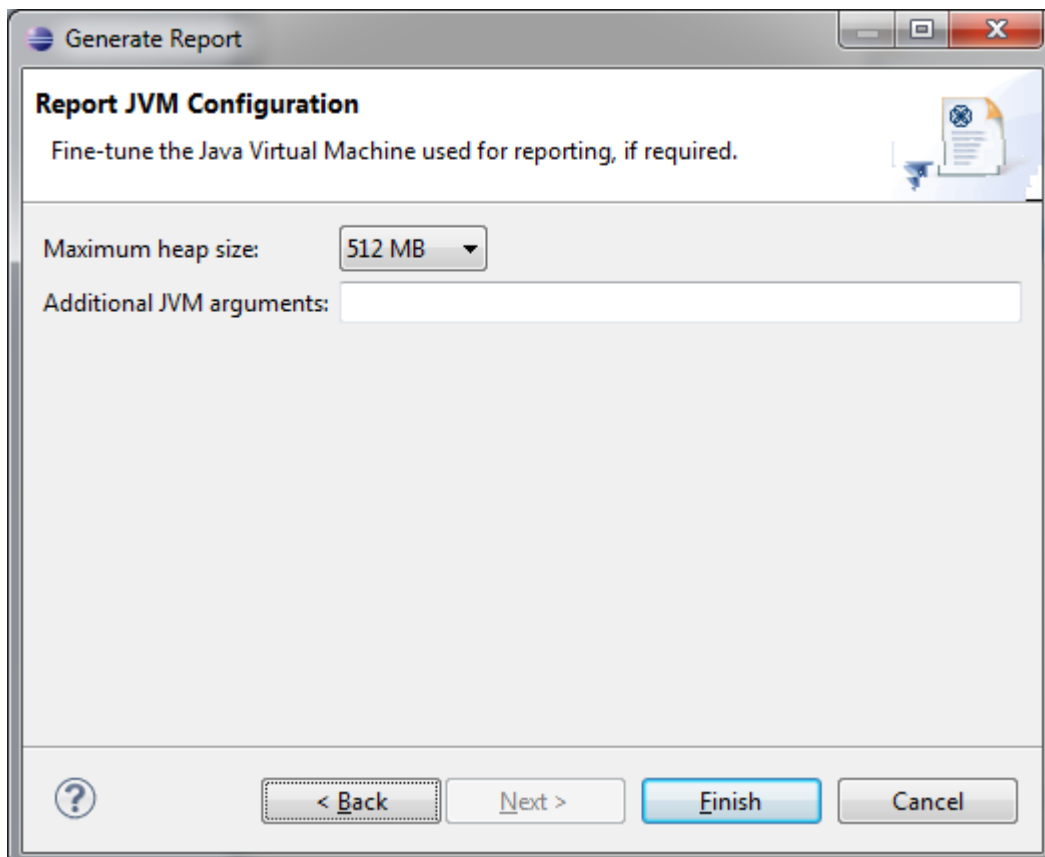
Report Filter Configuration is only shown if you choose not to accept default filter settings. This lets you select any of the predefined pre-defined filters or any of the custom method or statement filters you have previously configured.





### Configuring Your Report JVM

On this page you can set maximum heap size (default: 512 MB) and additional JVM arguments (typically don't need to supply these unless Atlassian support tells to).



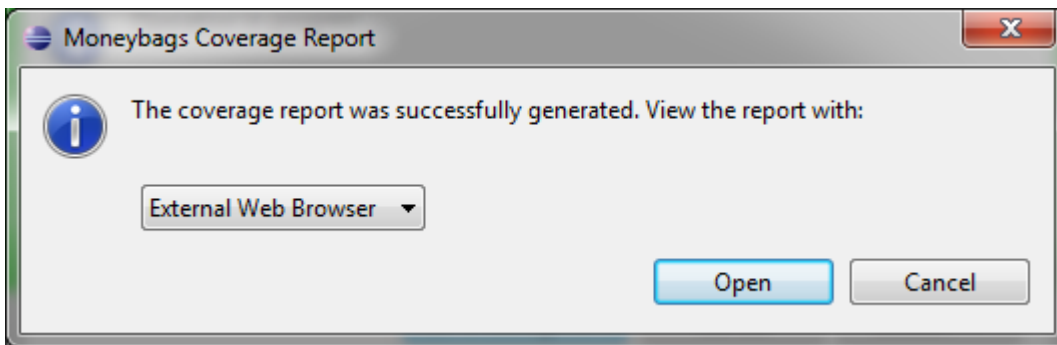
### Finalise the Report

Clicking **'Finish'** will start the report generation process. Log output will appear in the Eclipse console area.

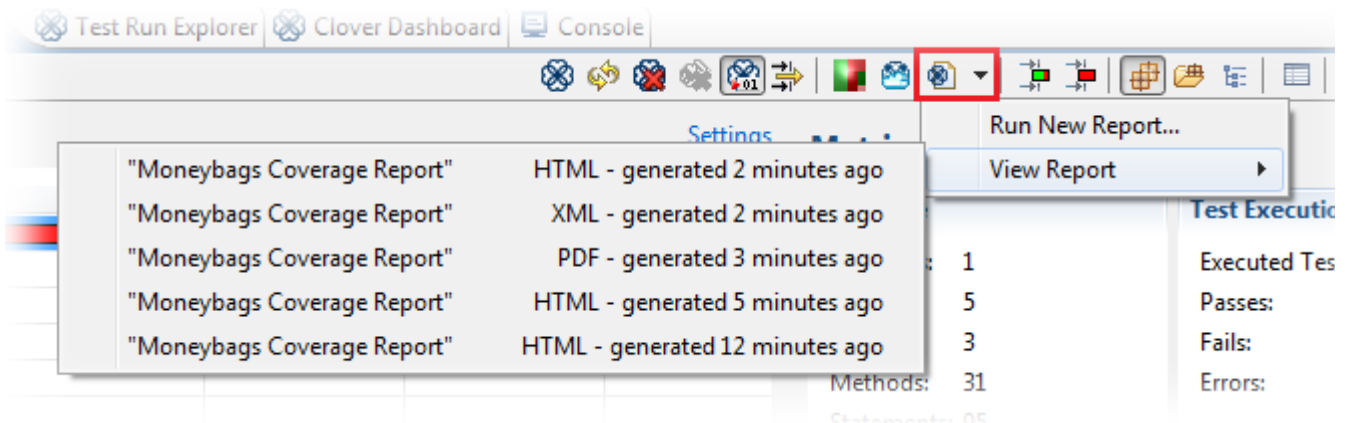
### Opening the Generated Report

When report generation is complete, a dialog box will prompt you to open one of the following options:

- **HTML report**  
HTML reports will be viewed either via the External Web Browser / Eclipse Web Browser (on supported platforms).
- **XML report**  
XML files will be viewed in either the External XML Editor / Eclipse XML Editor / Eclipse Text Editor
- **PDF report**  
PDF files will be viewed in the External PDF Viewer.



After a report is generated, an entry is added to the drop-down on the right-hand side of the report button in the Clover views which allow users to quickly re-open the report.



### Sample PDF Report

Clover Coverage Report		project stats:			
Test project for documentation Coverage Report		LOC:	283	Methods:	183
Coverage timestamp: Fri May 1 2009 15:22:48 EST		NCLOC:	183	Classes:	3
		Files:	3	Pkgs:	3

	Branch	Stmt	Method	Total
Test project for documentation	84.8%	94.7%	100%	93%

Packages	Branch	Stmt	Method	Total
com.cenqua.samples.money	84.8%	94.7%	100%	93%

Next chapter: [7. Test Optimization for Eclipse.](#)

## 7. Test Optimization for Eclipse

This page explains how to set up Clover's Test Optimization feature in the Eclipse development environment.

On this page:

- [Before You Begin](#)
- [Launching Test Optimization](#)
- [Measuring Test Optimization Results](#)
- [Test Optimization Settings](#)
  - [Setting Global Preferences](#)
  - [Setting Per-launcher Preferences](#)
- [Configurations Unsuitable For Test Optimization](#)
  - [Limitations with Multi-Project Set-Ups](#)
  - [Limitations with Test Suites](#)
  - [Limitations with Testing Frameworks](#)
- [Troubleshooting](#)


### Before You Begin

Before using Test Optimization with Clover-for-Eclipse, be aware of the following.

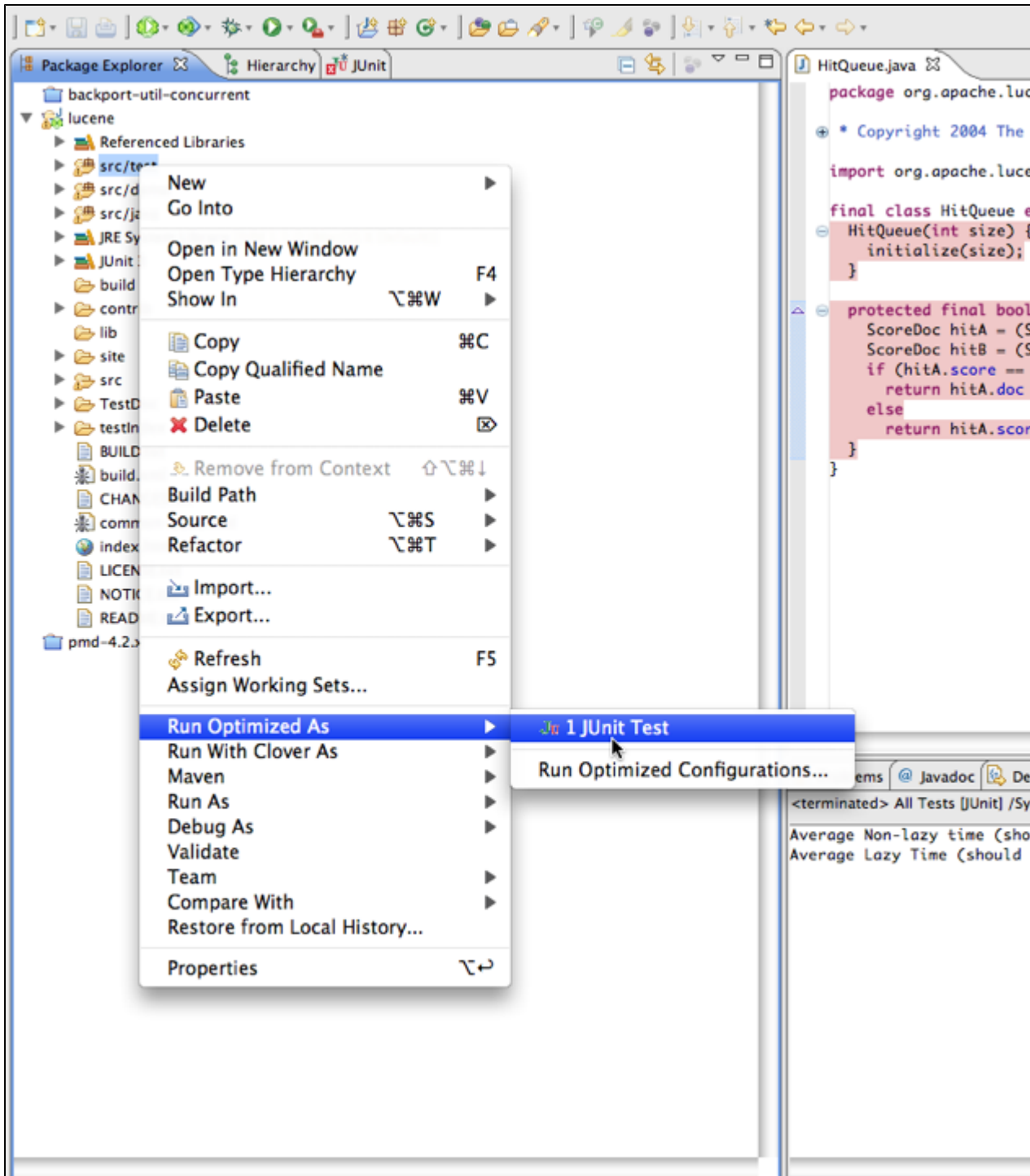
- Test Optimization is available as a launch command, similar to 'Run' or 'Debug'.
- Test Optimization supports JUnit launch configurations only.
- Ensure you have Clover enabled on the project - when there is no Clover instrumentation, there is no Test Optimization.

### Launching Test Optimization

To establish Test Optimization in Clover-for-Eclipse, carry out one of the following actions:

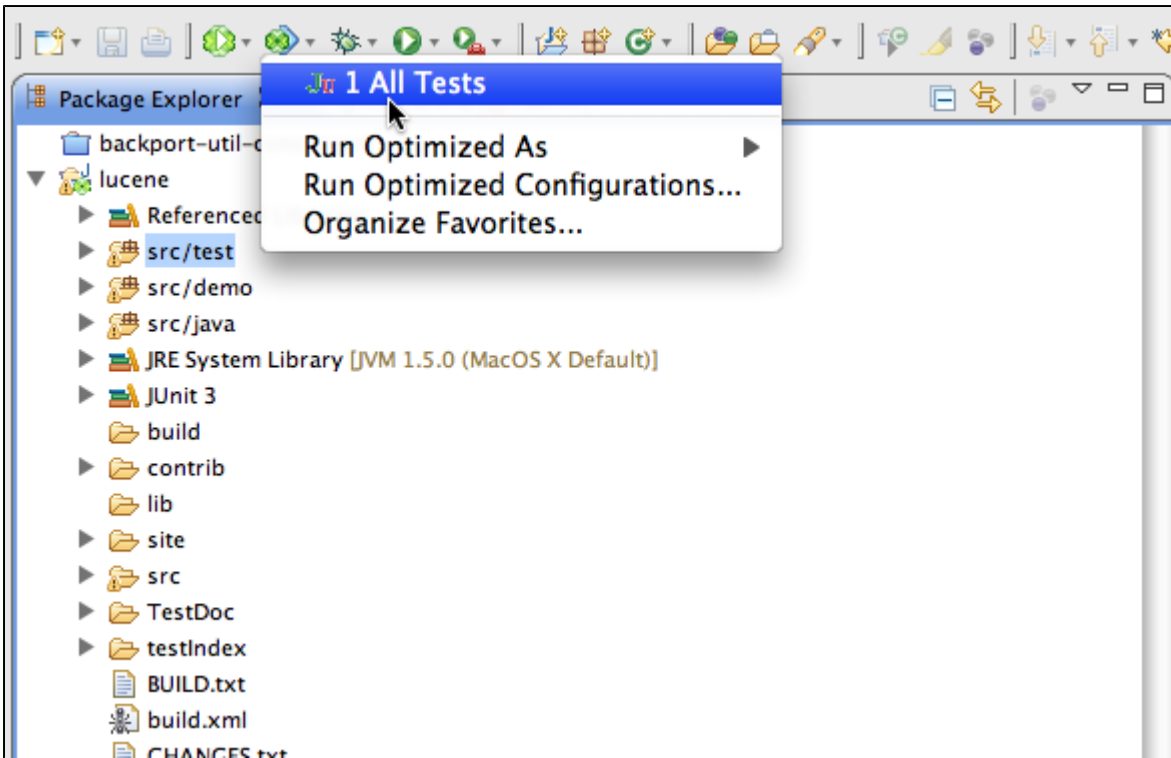
- Select a folder or package containing test classes and click the 'Run Optimized' icon , OR
- Right-click on a folder or package containing test classes and select 'Run Optimized As: JUnit Test', OR

*Screenshot: Launching a Build with Test Optimization from the Context Menu*



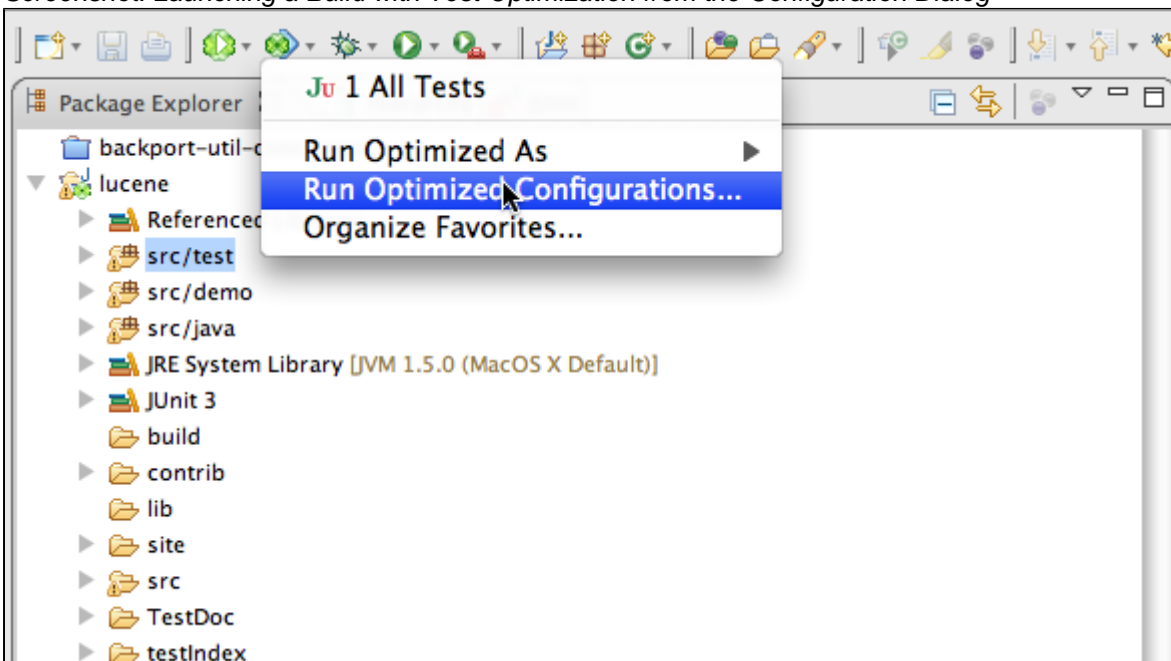
- Select an existing launch configuration from the Run Optimized dropdown, OR

*Screenshot: Launching a Build with Test Optimization from the Drop-Down Menu*



- Create a Run Optimized configuration and execute it from the configuration dialog.

Screenshot: Launching a Build with Test Optimization from the Configuration Dialog

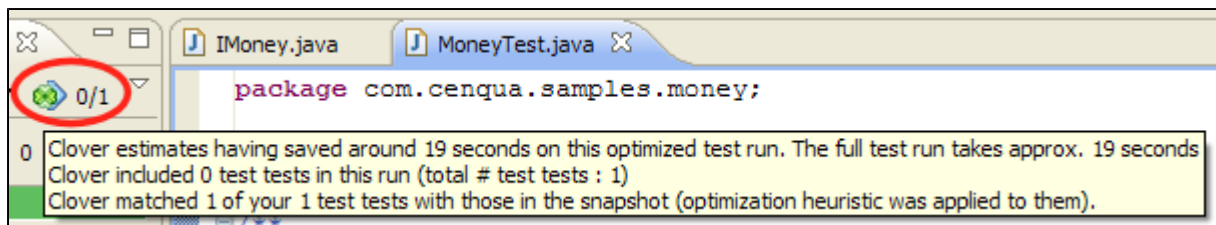


**i** When the Run Optimized button is used, Clover will run the configuration that was most recently run optimized.


### Measuring Test Optimization Results



When Optimized tests are being run, the JUnit view displays additional info about savings (as shown in the screenshot below).

Screenshot: Pop-up Notification of Time Savings With Test Optimization



After Optimized tests run, Clover saves a snapshot file with coverage information that is used to optimize the following test runs.

This file may be deleted using the Delete Snapshot icon  in the Coverage View (next to Delete Coverage Recordings button; you need to select a project first).

 The **'Delete Coverage Recordings'** button  also clears the snapshot file.

The Delete Snapshot icon is disabled when the selected project does not have the snapshot file. Test Optimization would run all tests (no optimization) when the snapshot file is deleted or absent.

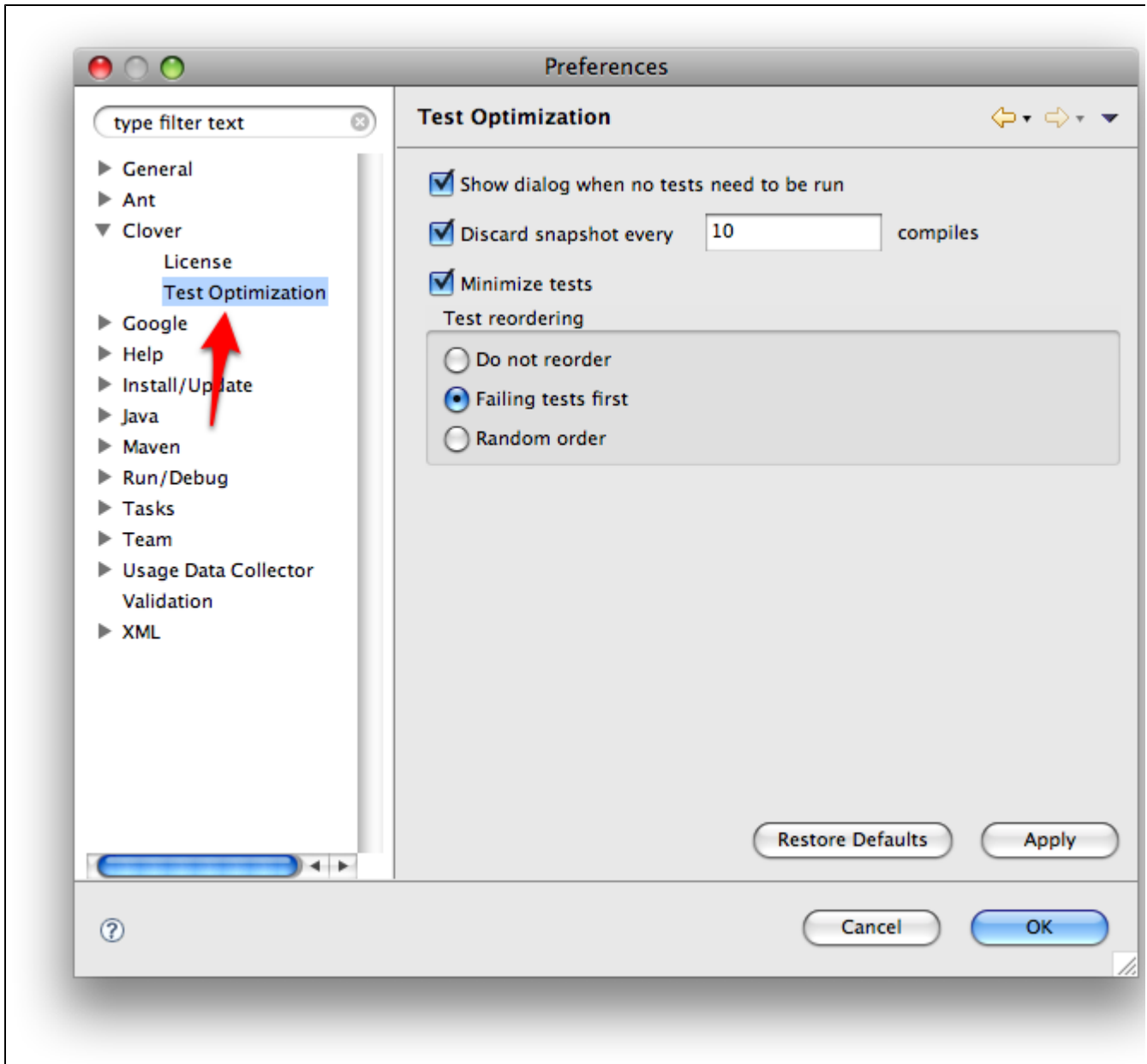
### Test Optimization Settings

The list below shows the settings available for Test Optimization.

- Show dialog when no tests need to be run: when Test Optimization reduces amount of tests to 0, display a notification dialog. Otherwise JUnit silently does nothing.
- Discard snapshot every X compiles: when enabled, snapshot is re-generated every X compiles. This is the equivalent of Ant's **'fullrunevery'** setting (See [Clover-for-Ant documentation](#)).
- Minimize tests: main functionality. When disabled Clover only reorders tests, all of them are always run.
- Test reordering:
  - Do not reorder (means NOOP if Minimize Tests is also off)
  - Failing tests first: reorder tests so that the ones which failed the last time Optimized Test was run are run first.
  - Random order.

### Setting Global Preferences

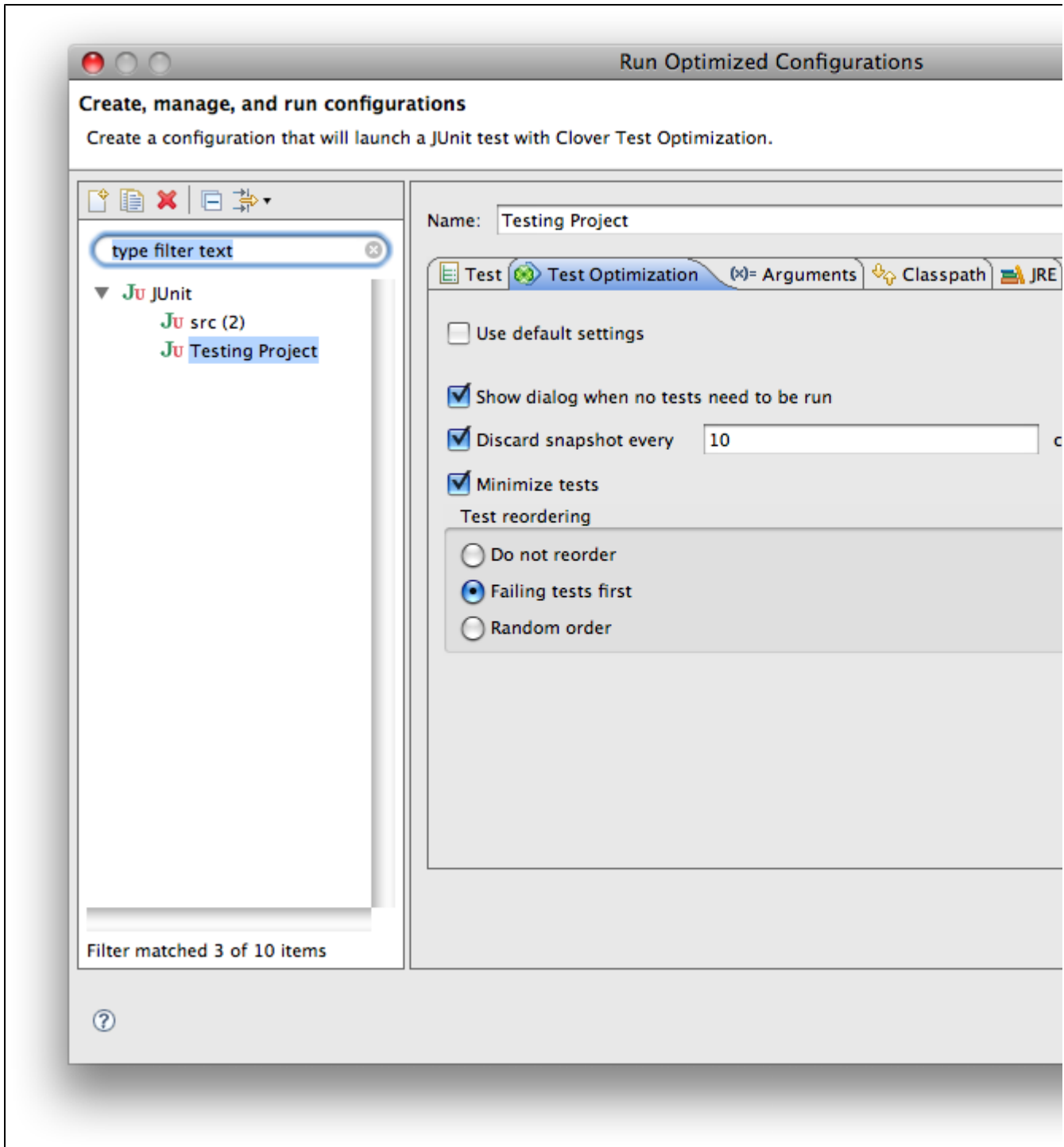
*Screenshot: Setting Global Preferences*



These are global (workspace-scope) preferences of Test Optimization; they are used as a template for per-launch configuration preferences, or used when launch configuration uses default settings.

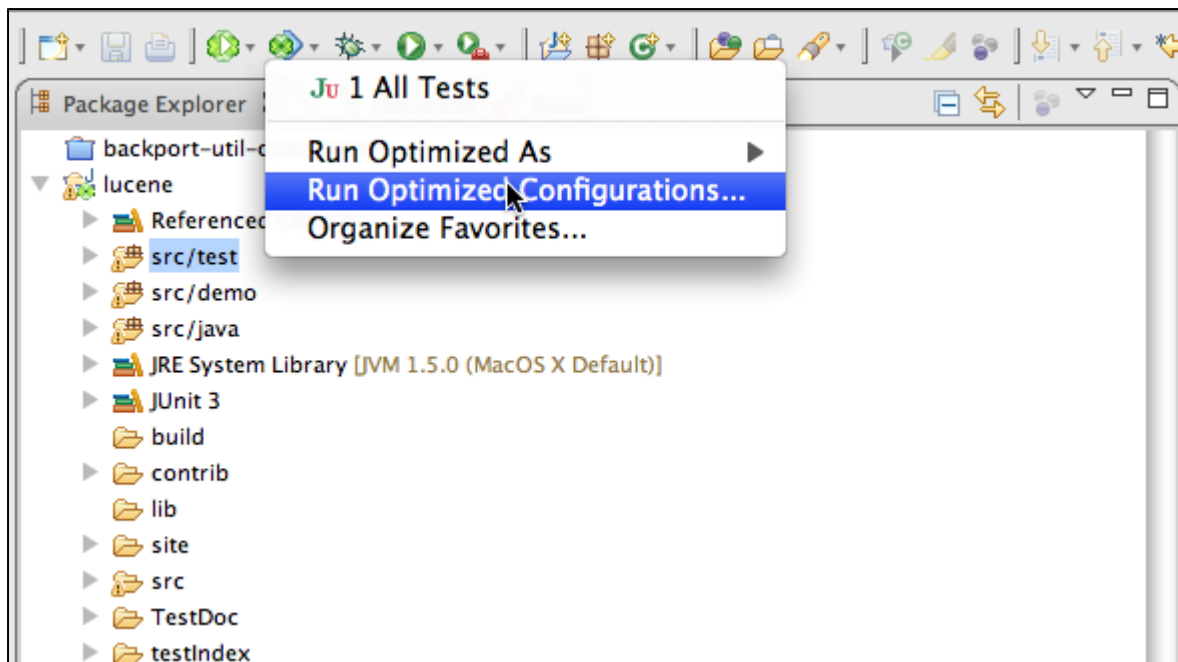
#### Setting Per-launcher Preferences

*Screenshot: Running Optimized Configurations*



Screenshot: The Run Optimized Drop-Down Menu





Allows overriding the workspace settings for single launch config.

- Use default settings: when set, use the workspace (global) settings.
- Copy defaults: copy workspace settings to current settings.

### Configurations Unsuitable For Test Optimization

Unfortunately not all configurations are suitable for Test Optimization. Please see the following points for specific details.

#### Limitations with Multi-Project Set-Ups

Clover does not aggregate data across projects, so it is not possible to detect changed sources in projects other than the one the test is in. As a result, if your project contains tests that are dependent on other projects, any change in those projects would not be detected by Test Optimization and some tests that should be run will unfortunately be 'optimized' too aggressively, resulting in their removal.

For more information, see this JIRA issue: [CEP-297](#).

#### Limitations with Test Suites

Clover does not recognise test suites as entities that should be optimized away.

As the result test suites are always run (never optimized).

If your test launch configuration includes both test suite and the test case (which is probably an incorrect configuration), then the test case would be run twice (normal behavior) or once (via test suite) when the test case is optimized away.

Resolution: Do not include test suites in launch configuration, add test cases directly.

For more information, see this JIRA issue: [CEP-299](#).

#### Limitations with Testing Frameworks

Test Optimization is only supported with JUnit tests at this time.

### Troubleshooting

To troubleshoot Test Optimization in Clover-for-Eclipse, check through the following solutions:

1. If Clover is disabled for the project or generally Clover does not work for the project;
  - Check the project icon has the nice green Clover overlay.
  - Check whether Coverage Explorer shows any coverage for the project.
2. If Clover has the test source settings wrong;
  - Check whether the Test Run Explorer shows any tests.

- Go to project Properties | Clover | Test Classes, make sure that either the Ant-style pattern or source folder list selects all your test classes properly.
3. If Test Optimization does not work when you have tests in different project than the classes being tested;
    - This is a known issue. See this JIRA issue for more details: ([CEP-297](#)).
  4. If your test case is run twice, or not optimized at all;
    - Clover does not support test suites. Make sure you don't try to run one, launch test cases directly ([CEP-299](#)).

Next chapter: [8. Launching an Ant build from Eclipse.](#)

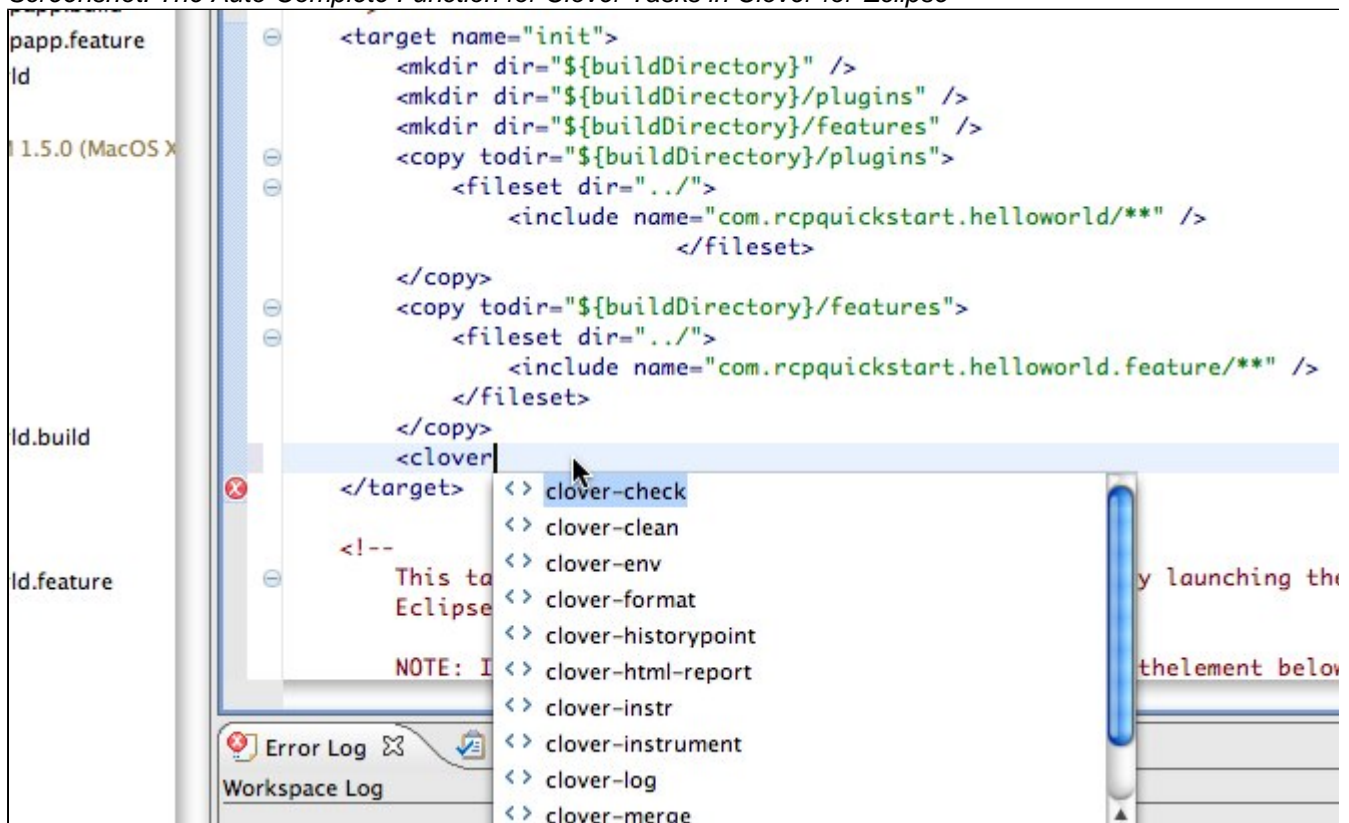
## 8. Launching an Ant build from Eclipse

Clover-for-Eclipse integrates with Eclipse's built-in Ant support and Ant execution. If you are using Ant with your builds, this makes accessing Ant tasks easier and more convenient launching them from the command line.

The Clover-Eclipse Ant support allows users to do the following:

- Use Clover Ant tasks without declaring a taskdef element.
- Use Clover Ant tasks without needing an additional license file (license information is taken from that entered in the plugin).
- Get auto-complete support for Clover tasks when writing Ant build files within Eclipse.

*Screenshot: The Auto-Complete Function for Clover Tasks in Clover-for-Eclipse*



Next chapter: [9. Eclipse advanced topics.](#)

## 9. Eclipse advanced topics

- Instrumenting RCP Application
- Performance Tuning in Clover for Eclipse

## Instrumenting RCP Application

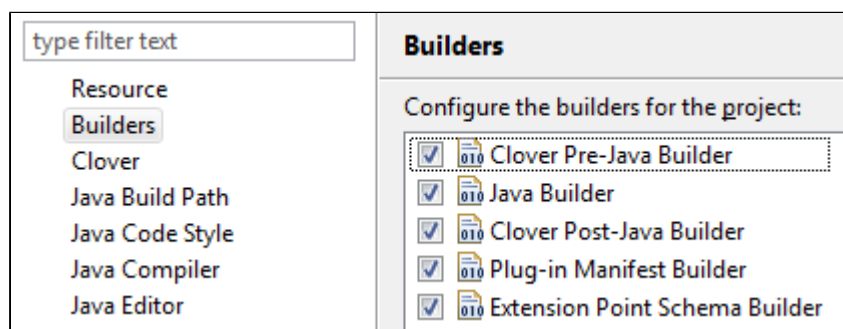
- Running application inside Eclipse IDE (with Clover)
  - Instrumenting code
  - Running product
- Running application in another Eclipse
  - Glossary
  - Instrumenting code
  - Exporting Plug-ins or Plug-in fragments
  - Exporting Features
  - Running plug-ins and features
  - Generating report
- Running product outside Eclipse
  - Exporting Product
  - Approach #1: Instrument source code manually
  - Approach #2: Overwrite product's plugins by instrumented versions
  - Running product
  - Generating report
  - Sample workbench configuration
- Appendix 1

### Running application inside Eclipse IDE (with Clover)

#### Instrumenting code

Just right click on projects you wish to instrument and choose "Clover > Enable on this project" option. Clover will:

- add "Clover Pre-Java Builder" and "Clover Post-Java Builder" on Builders tab (in project properties); typically it looks like this:



- add CLOVER\_RUNTIME variable on "Java Build Path / Libraries" tab
- rebuild project

**!** Your project must use a Java Builder in order to be instrumented by Clover. If you're using other builders for compilation (like Ant Builder) instead of Java Builder, your code won't be instrumented.

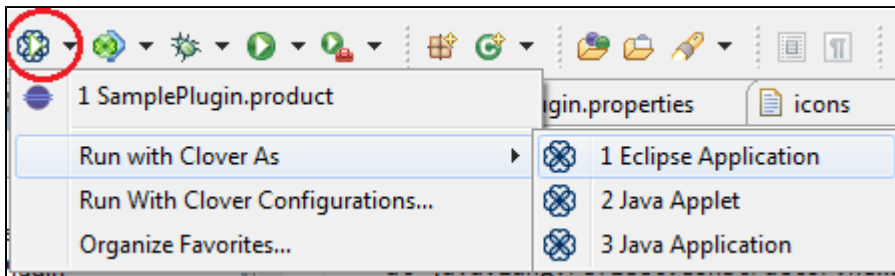
In such case, instrument sources manually before building them - for example by adding `<clover-instr/>` or `<clover-setup/>` to your Ant's build.xml. See example in [Appendix 1](#).

**i** Note that having PDE builders like Plug-in Manifest Builder / Extension Point Schema Builder in addition to Java Builder is OK, as they are just packaging compiled classes.

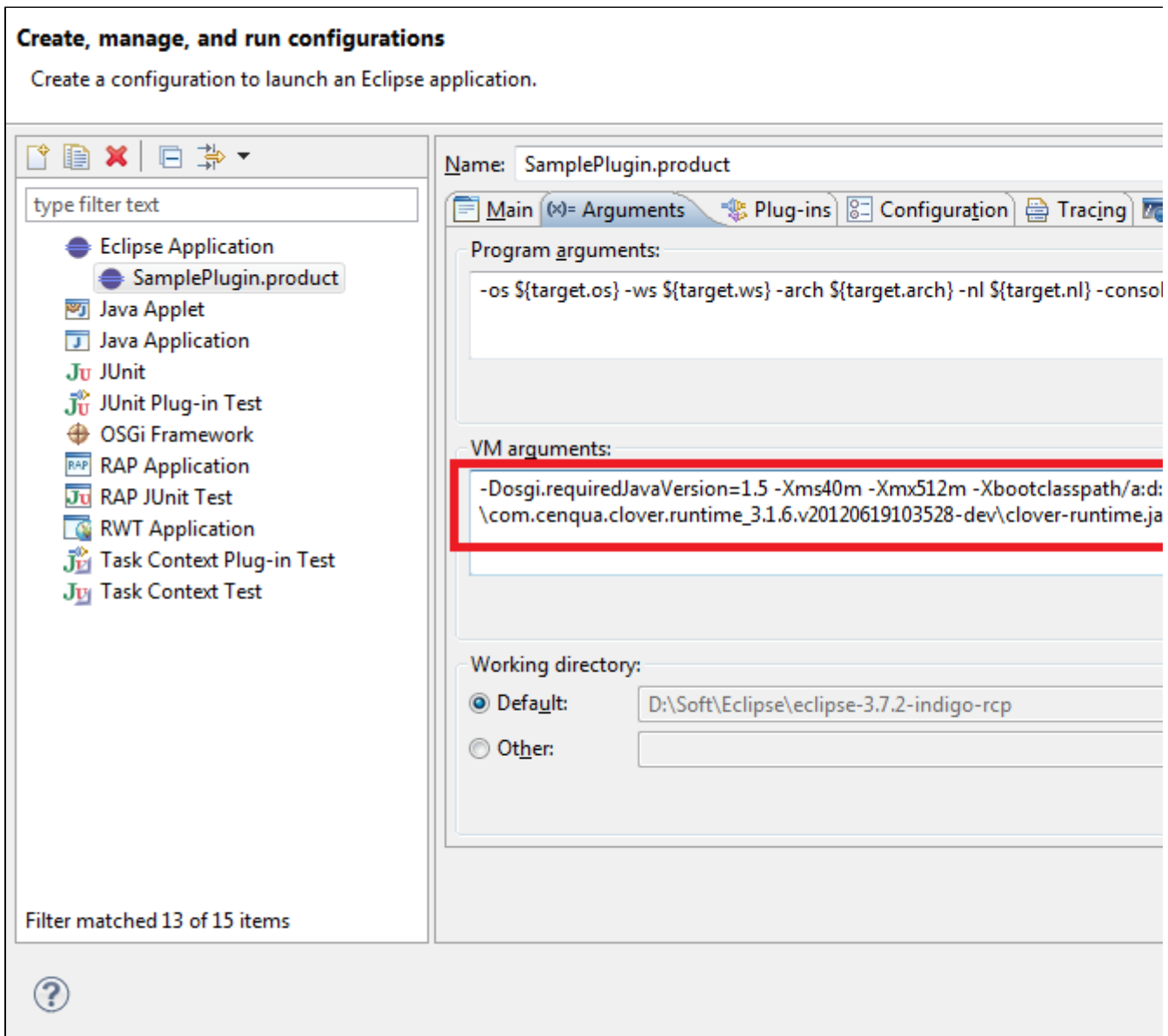
#### Running product

In order to run instrumented code you have to use "Run with Clover as..." button from tool bar. It will add Clover

jar to -Xbootclasspath.



Alternatively, if you wish to use "Run as..." (or other action like Debug, Profile) to run instrumented application, define `-Xbootclasspath/a:/your/path/to/clover-runtime.jar` in *VM Arguments* box as presented below:



**i** You can find clover-runtime.jar in `<eclipse_dir>\plugins\com.atlassian.clover.eclipse.runtime_4.X.X.vYYYYMMDD000000\clover-runtime.jar`.

*(Usage of other clover\*.jar files is discouraged, because they can have different content, build time stamp or instrumentation database version).*

## Running application in another Eclipse

This chapter describes a case, when code is instrumented in one Eclipse IDE and next features/plugins are exported, installed and tested in another Eclipse installation .

This is more tricky, because you have to ensure that instrumented classes will be packaged into jar files. You also have to change location of *clover.db*.

### Glossary

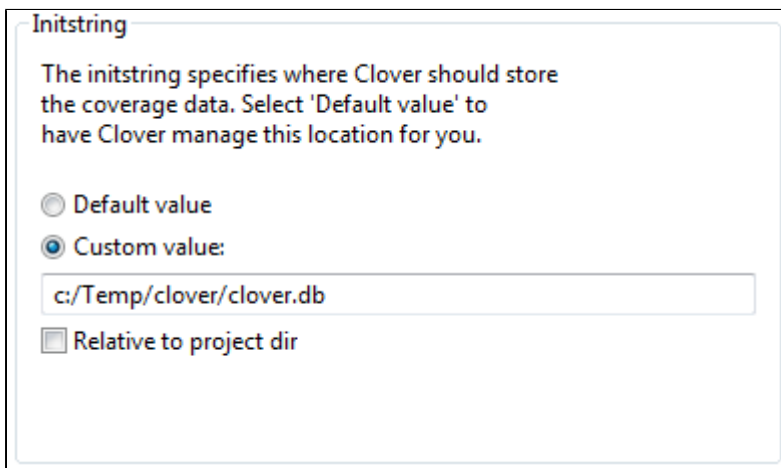
**Eclipse Test** - instance where application is executed (typically, it's a pure Eclipse installation used for testing)

**Eclipse IDE** - instance where application is compiled, it has Clover plug-in installed

### Instrumenting code

Run Eclipse IDE. Open Project Properties > Clover. Check the *"Enable Clover in this project"* box. On the *"Instrumentation"* tab in *"Initstring"* box:

- select the *"Custom value"* radio button,
- enter absolute path for Clover database
- deselect the *"Relative to project dir"* toggle.



Initstring

The initstring specifies where Clover should store the coverage data. Select 'Default value' to have Clover manage this location for you.

Default value

Custom value:

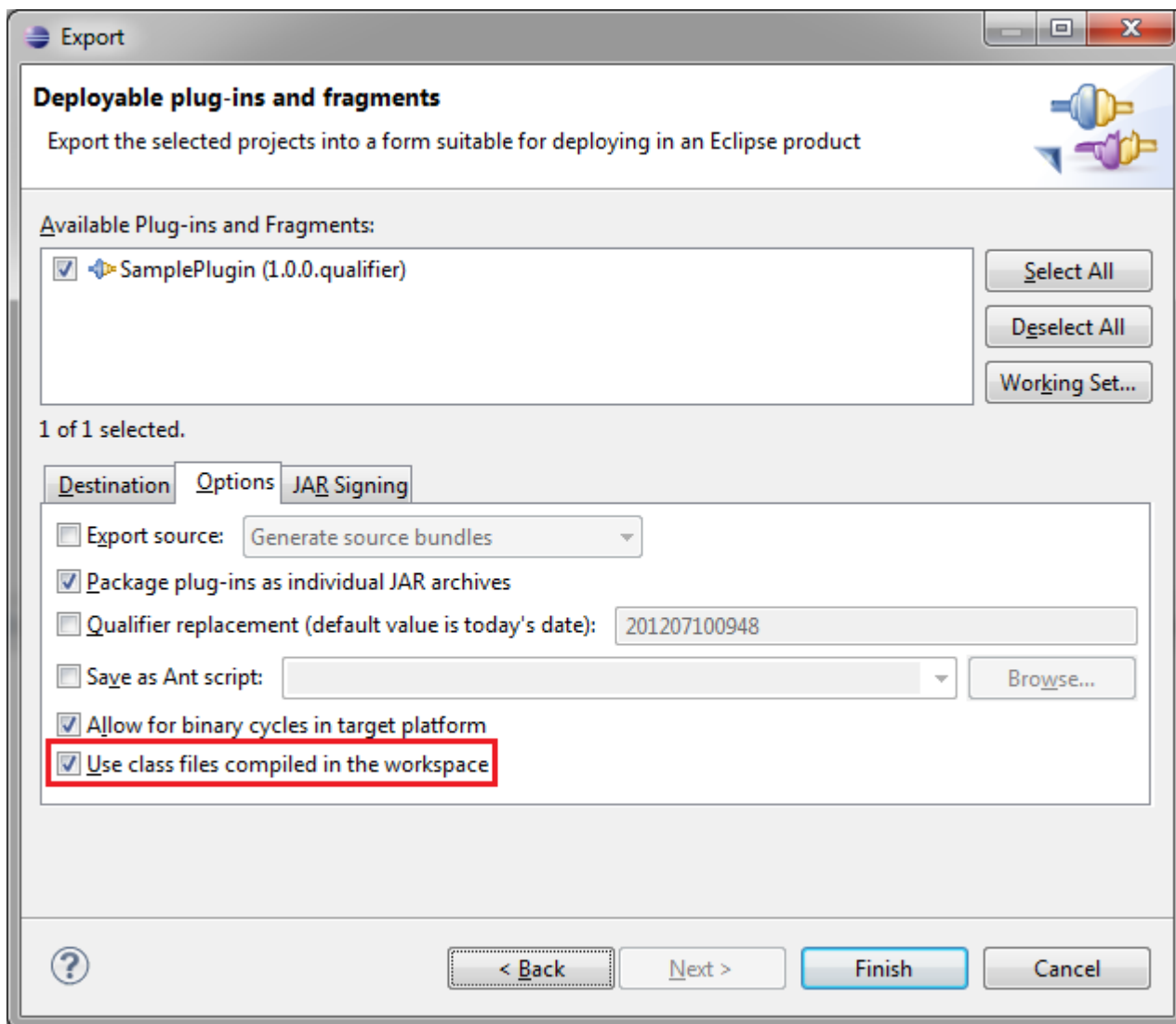
c:/Temp/clover/clover.db

Relative to project dir

Click OK and rebuild the project.

### Exporting Plug-ins or Plug-in fragments

Right click on plug-in project, choose *"Export ..."* > *"Plug-in Development"* / *"Deployable plug-ins and fragments"*. On *"Options"* tab make sure that the *"Use class files compiled in the workspace"* checkbox is selected.



### Exporting Features

Right click on feature project, choose "Export ..." > "Plug-in Development" / "Deployable features". On "Options" tab make sure that the "Use class files compiled in the workspace" checkbox is selected.

### Running plug-ins and features

Now, you can install these features or plug-ins into the Eclipse Test. Make sure that the **clover-runtime.jar** (you can find it in `<eclipse>\plugins\com.atlassian.clover.eclipse.runtime_version_number`) is available in Java **-Xbootclasspath**.

Add this to your Eclipse Test `eclipse.ini` file, for example:

```
-vmargs
-Xbootclasspath/a:/path/to/eclipse-ide/plugins/com.atlassian.clover.eclipse.runtime_4.0.0.v20140711000000/clover-runtime.jar
```

Note that **-vmargs** is necessary because all arguments listed after `-vmargs` are being passed as arguments for JVM, instead of arguments for Eclipse framework.

### Generating report

As during instrumentation the absolute path was used for Clover database, your Eclipse IDE should automatically fetch coverage files so that you can see results in Coverage Explorer.

You can also generate report manually, using `<clover-report>` Ant task for instance. See the "report" target in Appendix 1.

## Running product outside Eclipse

### Exporting Product

Unfortunately, exporting the whole product at once:

- right click on project, choose "Export ..." > "Plug-in Development / Eclipse Product" or
- open the `MyProduct.product` file and click "Use the Eclipse Product export wizard" from Overview tab.

**will not work** because the standard "Eclipse Product Export Wizard" does not use Builders defined in project properties, but calls PDE Ant build scripts. As a consequence it bypasses the "Clover Pre/Post-Java Builder".

There are two ways to solve this:

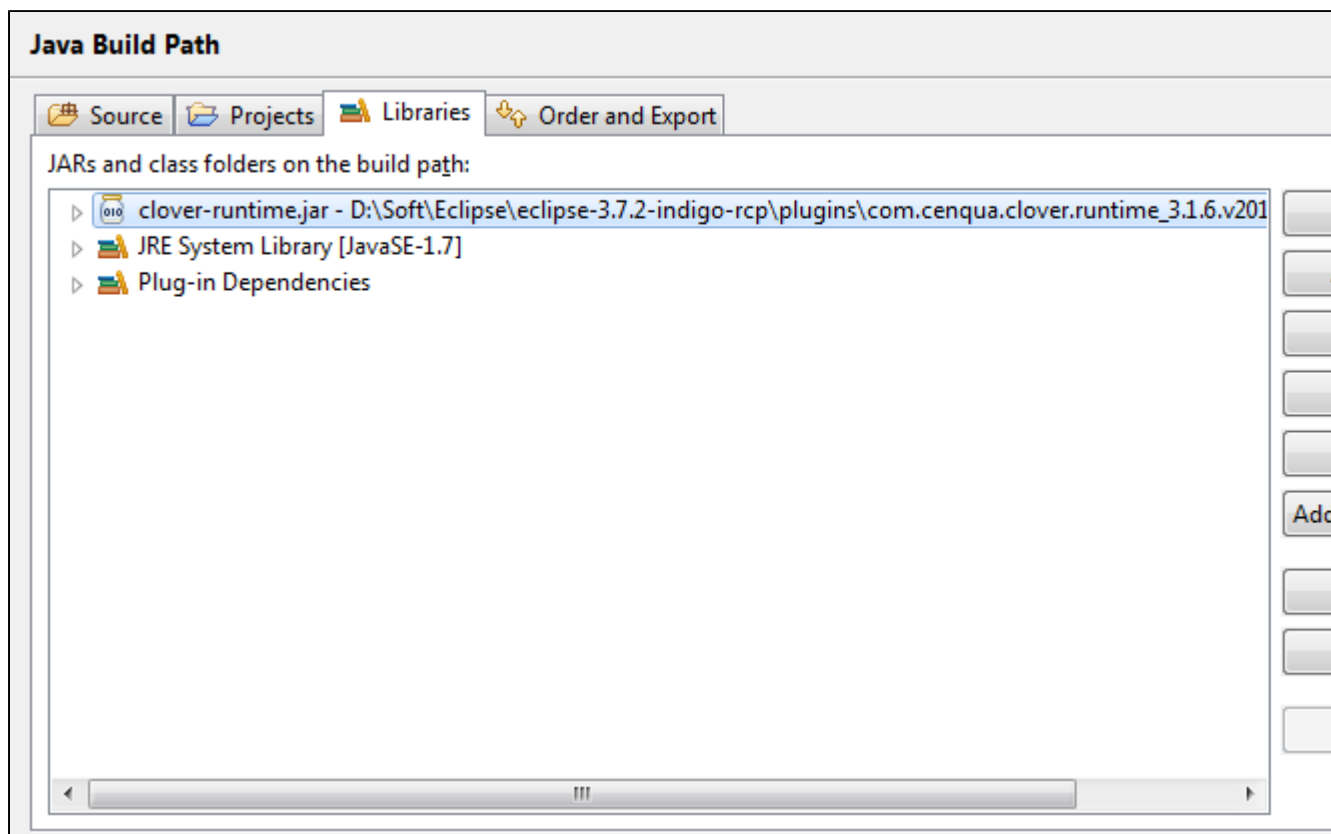
#### Approach #1: Instrument source code manually

1) Use `CloverInstr` command line tool or `clover-instr` Ant task to instrument sources manually - see script in Appendix 1

💡 remember to put instrumented sources in another location, i.e. not into your original source folder 😊

2) Open instrumented sources in Eclipse IDE.

Add the `clover-runtime.jar` from `<eclipse_home>\plugins\com.atlassian.clover.eclipse.runtime_<version_number>` as external JAR to Java Build Path.




💡 Do not add it as dependency in MANIFEST.MF as there's no reason to package this jar inside your product - it's only for compilation. At runtime, the `-Xbootclasspath` will be used.



Run Eclipse Product Export Wizard.

#### Approach #2: Overwrite product's plugins by instrumented versions

- 1) Manually export all plugins being part of your product - see [Exporting Plug-ins or Plug-in fragments](#) (they will contain instrumented classes).
- 2) Export product using "Eclipse Product Export Wizard" into some folder (it will contain un-instrumented classes).
- 3) Overwrite plugins in the exported product folder by those exported manually.


 This is less convenient compared to Approach #1 as you have to export plugins one-by-one, but might be needed if:

- you have sources generated during build (XML schema bindings, for instance) and
- you want to have them instrumented as well (which is usually not practised).

#### Running product

Make sure that the *clover-runtime.jar* (you can find it in `<eclipse>\plugins\com.atlassian.clover.eclipse.runtime_<version_number>`) is available in Java **-Xbootclasspath**. Add this to your product's *config.ini* file, for example:

```
-vmargs
-Xbootclasspath/a:/path/to/eclipse-ide/plugins/com.atlassian.clover.ecli
pse.runtime_4.0.0.v20140711000000/clover-runtime.jar
```

 Note that **-vmargs** is necessary because all arguments listed after *-vmargs* are being passed as arguments for JVM, instead of arguments for Eclipse framework.

#### Generating report

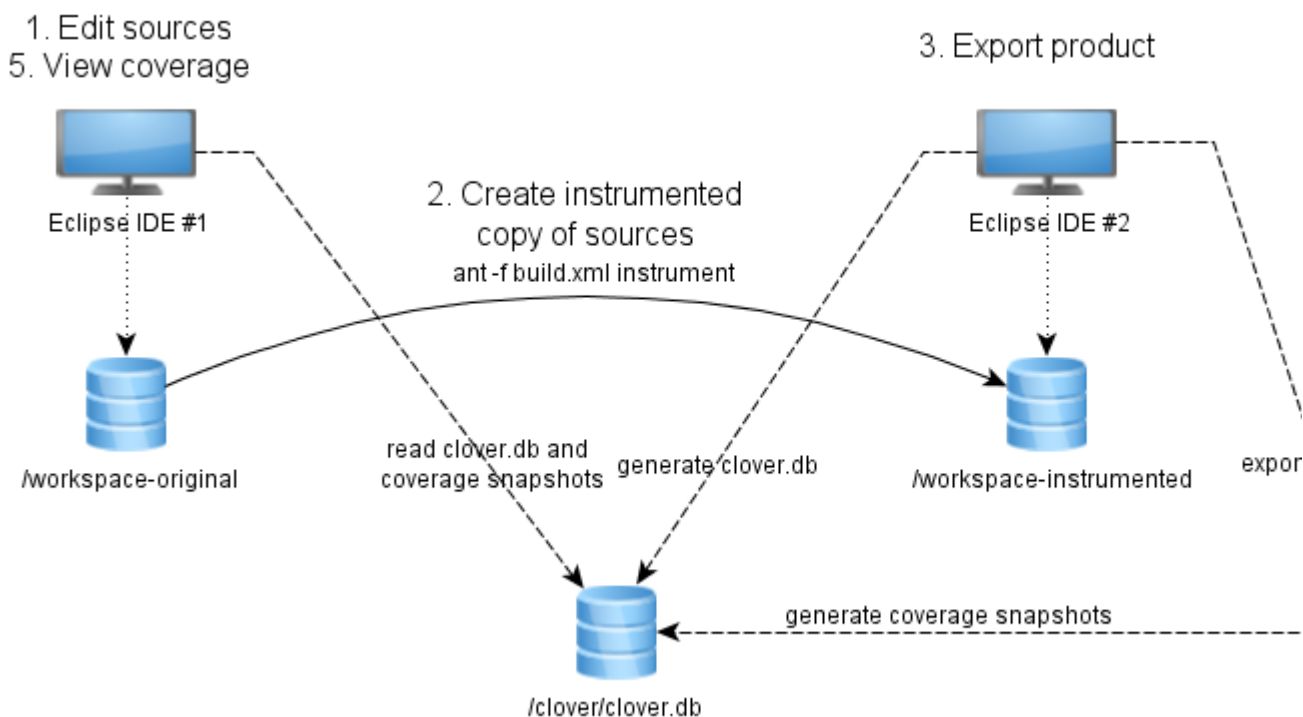
As during instrumentation the absolute path was used for Clover database, your Eclipse IDE should automatically fetch coverage files so that you can see results in Coverage Explorer.

You can also generate it manually e.g. using `<clover-report>` Ant task. See the "report" target in Appendix 1.

#### Sample workbench configuration

The diagram below shows how work with manually instrumented sources (Approach #1) can be organized. A location of Clover database configured in Eclipse IDE as well as in Ant script which instruments sources is the same and points to an absolute path. Thanks to this, after running product, coverage results can be fetched automatically into Eclipse IDE #1.





## Appendix 1

Sample Ant script which instruments all \*.java files from *project.original.dir* and puts them into *project.instrumented.dir*, preserving original directory structure. It copies also all non-java files as well.

```
<project default="instrument">
  <property name="clover.jar" location="${user.home}/clover.jar"/>
  <property name="ant-contrib.jar"
location="${user.home}/ant-contrib-1.0b3.jar"/>
  <property name="project.original.dir" location="original_project"/>
  <property name="project.instrumented.dir" location="instr_project"/>
  <property name="project.clover.db"
location="${project.instrumented.dir}/.clover/clover.db"/>

  <taskdef resource="cloverlib.xml" classpath="${clover.jar}"/>
  <taskdef resource="net/sf/antcontrib/antlib.xml"
classpath="${ant-contrib.jar}"/>

  <target name="_instrument-dir">
    <!-- Use double-slash for windows paths -->
    <propertyregex property="original.dir.quoted"
input="${project.original.dir}" regexp="\" replace="\\\\\\\\\\\\\\\\" global="true"/>
    <propertyregex property="relative.dir" input="${source.dir}"
regexp="${original.dir.quoted}(.*)" select="\1"/>
    <echo message="Instrumenting ${source.dir} into
${project.instrumented.dir}${relative.dir}"/>
    <echo message="Clover database is ${project.clover.db}"/>
    <clover-instr destdir="${project.instrumented.dir}${relative.dir}"
initstring="${project.clover.db}">
      <fileset dir="${project.original.dir}${relative.dir}">
        <include name="**/*.java"/>
      </fileset>
    </clover-instr>
  </target>
</project>
```

```

<target name="instrument">
  <!-- Cleanup from previous run -->
  <delete dir="${project.instrumented.dir}"/>
  <!-- Find all source directories, for each of them call clover-instr.
Please note that we cannot use sth like:
  <clover-instr srcdir="${project.original.dir}"
destdir="${project.instrumented.dir}" initstring="${project.clover.db}">
  directly, because clover-instr does not recreate original directory
structure, but puts everything
  under one destdir root.
  -->
  <foreach target="_instrument-dir" param="source.dir" inheritall="true"
inheritrefs="true">
    <path>
      <!-- Define all package roots here -->
      <dirset dir="${project.original.dir}">
        <include name="**/src"/>
        <include name="**/test"/>
      </dirset>
    </path>
  </foreach>

  <!-- Copy all other non-java files as well -->
  <echo message="Copying other files from ${project.original.dir} to
${project.instrumented.dir}"/>
  <copy todir="${project.instrumented.dir}">
    <fileset dir="${project.original.dir}">
      <exclude name="**/*.java"/>
    </fileset>
  </copy>

  <!-- Now we can build it under PDE. Don't even try to read instrumented
sources ;-) -->
  <echo message="INSTRUMENTATION DONE. Run Eclipse and open project from
${project.instrumented.dir}"/>
</target>

<target name="report">
  <clover-report initstring="${project.clover.db}">
    <current outfile="current.html">
      <format type="html"/>
    </current>
    <current outfile="current.xml">
      <format type="xml"/>
    </current>
  </clover-report>
</target>

```

```

    </clover-report>
  </target>
</project>

```

## Performance Tuning in Clover for Eclipse

Since every project varies in size and speed, Clover may need to be configured to work best for your project.

### Clover-for-Eclipse Memory Allocation

Tracking code coverage for a project, particularly per-test code coverage for large projects can consume a good deal of memory. If things are running too slowly with Clover enabled, consider boosting the memory allocated to your Eclipse installation.

If your workspace contains a lot of projects, we recommend you incrementally enable Clover on them rather than enabling it on all of them at once. Doing it in stages will allow you to determine how many your current memory settings can accommodate.

By default Clover-for-Eclipse will keep its memory usage as low as possible but this may cause code coverage to take a bit longer to be updated after a test run. If you believe you have sufficient memory to load all the per-test coverage data into memory and get faster coverage feedback, consider switching on the **'Keep per-test coverage data in memory'** setting, as seen in the following screenshot:

	Cov%	Av Me Cpx	Cpx
	70,5%	2,0	4 398,0
	86,4%	1,8	14,0
	49,1%	1,8	1 425,0
	78,5%	2,3	2 431,0
	94,2%	1,3	152,0
	71,1%	2,8	83,0
	81,2%	2,2	289,0
	0,0%	1,0	4,0

**Workspace Settings**

- Aggregate coverage generated since the last cle
- Track per-test coverage
- Keep per-test coverage data fully in memory
- Look for updated coverage every: 2s

Statements: 8 493  
Branches: 3 792

If you use Clover-for-Eclipse purely for Test Optimization purposes and not for coverage reporting, you can reduce the granularity of Clover instrumentation from statement to method level. This will speed up instrumentation times, compilation times and test run times. To make this change, click on a project in the Coverage Explorer and alter its instrumentation level. A full rebuild is required after making this change.

The screenshot shows the Eclipse IDE's Coverage Explorer. At the top, there are tabs for Problems, Javadoc, Declaration, Console, Progress, Coverage Explorer, and Test Run Explorer. Below the tabs, there is a 'Show:' dropdown menu set to 'All classes'. The main area displays a table with columns for 'Elem', 'Cov%', 'Av Me Cpx', and 'C'. The table lists several packages and classes, including 'backport-util-concurrent', '(default package)', 'edu.emory.mathcs.backport.util.concurrent', and 'sun.misc'. A context menu is open over the 'backport-util-concurrent' package, showing the following options:

- Instrument and compile at statement level
- Apply

Elem	Cov%	Av Me Cpx	C
backport-util-concurrent		2,0	4 39
(default package)		1,8	1
edu.emory.mathcs.backport.util.concurrent		1,8	1 42
edu.emory.mathcs.backport.util.concurrent		2,3	2 43
edu.emory.mathcs.backport.util.concurrent		1,3	15
edu.emory.mathcs.backport.java.util.concurrent	71,1%	2,8	8
edu.emory.mathcs.backport.java.util.concurrent	81,2%	2,2	28
sun.misc	0,0%	1,0	

### Related Links

Clover Performance Tuning for Ant

## Clover-for-Eclipse Installation Guide

- Requirements
- Known issues
- Installation
  - 1a. Installing the plug-in from the live Clover Eclipse update site
  - 1b. Installing the plug-in from a downloaded archive of the Clover Eclipse update site
  - 1c. Installing the plug-in from the Eclipse Marketplace
  - 2. Installing the license
- Uninstallation

### Requirements

The Clover-for-Eclipse system requirements are as follows:

<b>Eclipse Projects</b>	Your Eclipse projects must use the built-in Java Builder for compilation of source code. We do not support AspectJ-based projects.
<b>JDK Version</b>	
<b>Eclipse Version</b>	See <a href="#">Supported Platforms</a>
<b>Operating System</b>	

**You need a valid Clover license file to run Clover.** You can obtain a free 30 day evaluation license, purchase a commercial license or apply for a free open source license at <http://www.atlassian.com>.

### Known issues

- Projects initially instrumented/built using an earlier Clover plugin version or build may appear as grey in the Coverage Explorer. To resolve this, perform a clean build on your Clover-instrumented classes and select "Yes" when asked to delete coverage data.

### Installation

You can either install the Clover-for-Eclipse from the live Clover Eclipse update site, Eclipse Marketplace or from a zipped archive downloaded manually.

#### 1a. Installing the plug-in from the live Clover Eclipse update site


1. Select from the menu "Help > Install New Software ...".
2. Click "Add...", enter <http://update.atlassian.com/eclipse/clover> in the "Location" field. Click "OK".

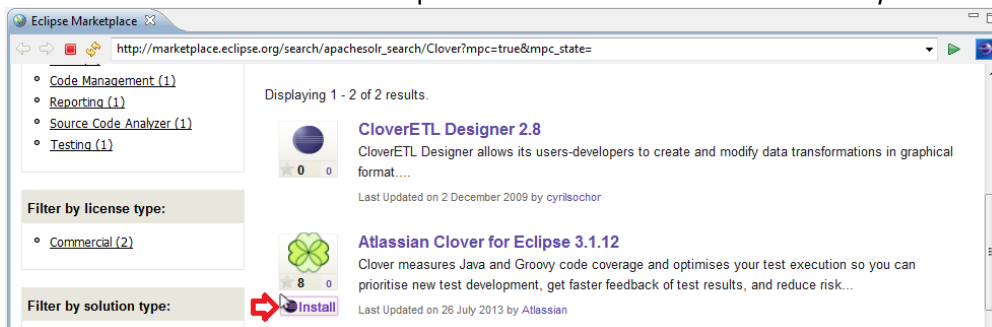
3. Select the entry "*Clover <version> (Eclipse 3.6 - 4.4 and RAD 8.0 - 9.0)*".
4. Deselect the checkbox "*Contact all update sites ...*" for faster installation.
5. Click "*Next*" twice, select the radio button next to "*I accept the license agreement*" and click the "*Finish*".
6. Installation process will start.
  - a. You will be asked to install unsigned content. Click "*OK*".
  - b. You will be asked to restart Eclipse. Click "*Restart now*" to complete the installation.

### 1b. Installing the plug-in from a downloaded archive of the Clover Eclipse update site

1. Download the most recent Clover Eclipse update site archive from the [Clover downloads page](#).
2. Select from the menu "*Help > Install New Software ...*".
3. Click "*Add...*", next click "*Archive...*" and point to the location of the downloaded archive. Click "*OK*".
4. Select the entry "*Clover <version> (Eclipse 3.6 - 4.3. and RAD 8.0 - 9.0)*".
5. Deselect the checkbox "*Contact all update sites ...*" for faster installation.
6. Click "*Next*" twice, select the radio button next to "*I accept the license agreement*" and click the "*Finish*".
7. Installation process will start.
  - a. You will be asked to install unsigned content. Click "*OK*".
  - b. You will be asked to restart Eclipse. Click "*Restart now*" to complete the installation.

### 1c. Installing the plug-in from the Eclipse Marketplace

1. Select from the menu "*Help > Eclipse Marketplace ...*".
2. On the "*Search*" tab, in the "*Find*" field type "*Clover*" and click "*Go*".
3. You shall see a message "*0 matches. Browse for more solutions.*" Click on this link.  
 *Clover is not listed in this dialogue due to license restrictions applied by Eclipse Marketplace.*
4. An internal web browser view will open. Click the "*Atlassian Clover for Eclipse*" on a list of results.



5. Plug-in page will open. Click the "*Install*" button.
6. A dialogue with a list of features to install will open. Check that "*Clover 4*" and "*Clover 4 Ant Support*" are selected and click "*Next*".
7. A license agreement will be displayed. Select the radio button next to "*I accept the license agreement*" and click the "*Finish*".
8. Installation process will start.
  - a. You will be asked to install unsigned content. Click "*OK*".
  - b. You will be asked to restart Eclipse. Click "*Restart now*" to complete the installation.

## 2. Installing the license

1. Obtain a valid trial, purchased or open source license from <http://my.atlassian.com>
2. Within Eclipse, select from the menu "*Window > Preferences*" and click on "*Clover > License*".
3. Paste the contents of your license file into the license text area or select your license file by clicking "*Load ...*".
4. Click "*Apply*". The license summary should now display status, type and message consistent with the type of license you entered.
5. Click "*OK*" to close the window.

## Uninstallation

To uninstall the plug-in:

1. Select from the menu "*Help > About Eclipse > Installation Details*".
2. Select "Installed Software" tab.
3. Find the Clover features in the list - there should be two: "*Clover 4*" and "*Clover 4 Ant Support*".

4. Select both, right click and select *"Uninstall..."*. Click *"Finish"* to confirm you want to proceed.
5. Uninstallation process will start. You will be asked if you want to restart Eclipse. Click *"Restart Now"*.

## Installing Clover-for-Eclipse

### Installation

#### 1. Installing the plugin

You can either install the Clover-for-Eclipse from the live Clover Eclipse update site or from a zipped archive downloaded manually.

##### 1a. Installing the plugin from the live Clover Eclipse update site

In Eclipse 3.4 or Later:

1. Select from the menu "Help | Software Updates" to start the installation process.
2. Click "Available Software"
3. Select "Add Site..." and enter <http://update.atlassian.com/eclipse/clover> then click OK. This will point Eclipse to the Clover update site from where it will download the plugin and an entry to the sites in the
4. Expand the entry "http://update.atlassian.com/eclipse/clover" and its child entry "Clover 2.3.2" until you see "Clover 2 (for Eclipse 3.2/3.3/3.4)" and "Clover 2 Ant Support (3.2/3.3/3.4)".
5. Click the checkboxes next to "Clover 2 (for Eclipse 3.2/3.3/3.4)" and "Clover 2 Ant Support (3.2/3.3/3.4)" and then click "Install..."
6. The "Install" dialog should now show the two features to install. Click "Finish".
7. You may be asked if you agree to the license terms. If you agree, click the radio button next to **'I accept the license agreement'** and click the "Next" button.
8. Finally, you will be asked if you want to restart Eclipse after installing the plugin. Click "Yes" to restart and complete the installation.

##### 1b. Installing the plugin from a downloaded archive of the Clover Eclipse update site

1. Download the most recent Clover Eclipse update site archive from the [Clover downloads page](#)

In Eclipse 3.4 or Later:

1. Select from the menu "Help | Software Updates" to start the installation process.
2. Click "Available Software"
3. Select "Add Site...", click "Archive..." and point Eclipse to the location of the Clover Eclipse update site archive. Click OK. This will point Eclipse to the Clover update site from where it will download the plugin and an entry to the sites in the
4. Expand the entry that refers to the archive you've just added and then expand its child entry "Clover 2.3.2" until you see "Clover 2 (for Eclipse 3.2/3.3/3.4)" and "Clover 2 Ant Support (3.2/3.3/3.4)".
5. Click the checkboxes next to "Clover 2 (for Eclipse 3.2/3.3/3.4)" and "Clover 2 Ant Support (3.2/3.3/3.4)" and then click "Install..."
6. The "Install" dialog should now show the two features to install. Click "Finish".
7. You may be asked if you agree to the license terms. If you agree, click the radio button next to **'I accept the license agreement'** and click the "Next" button.
8. Finally, you will be asked if you want to restart Eclipse after installing the plugin. Click "Yes" to restart and complete the installation.

#### 2. Installing the license

1. Obtain a valid trial, purchased or opensource license for Clover 2. Licenses can be obtained at <http://my.atlassian.com>
2. Within Eclipse, select from the menu "Window | Preferences" and click on Clover > License.
3. Paste the contents of your license file into the license text area or select your license file by clicking "Load..."
4. Click Apply. The license summary should now display status, type and message consistent with the type of license you entered.
5. Click OK to close the window.

## Uninstallation

In Eclipse 3.4:

1. To uninstall the plugin, from the Eclipse menu select "Help | Software Updates".
2. Select "Installed Software".
3. Find the Clover features in the list - there should be two: "Clover 2 (for Eclipse 3.2/3.3/3.4)" and "Clover 2 Ant Support (for Eclipse 3.2/3.3/3.4)".
4. Select both, right click and select "Uninstall...".
5. Click "Finish" to confirm you want to proceed with the uninstall and disable the features.
6. Finally, you will be asked if you want to restart Eclipse after installing the plugin. Click "Yes" to restart to complete uninstallation.

## Clover-for-Eclipse Upgrade Guide

### General instructions

#### 1. Upgrade Clover-for-Eclipse version

Click "*Help -> Check for updates*" and select "Clover" and "Clover Ant Support" features. Follow the wizard instructions.

#### 2. Update license key (optional)

Installing new license key is necessary when you're installing a Clover version released after end of support date of your current license. Open the "*Window -> Preferences -> Clover -> License*" and paste a new key.

#### 3. Rebuild workspace (optional)

Clover's database format may change in newer versions. In such case you'll get a build error with a message informing about database incompatibility. In such case you have to delete old database files - rebuild your workspace and answer "Yes" on a question "*You are doing a rebuild, do you want to delete the old coverage data for project ... ?*".

### Upgrading from specific releases

Please see the [Clover Release Notes](#) and the [Clover-for-Eclipse Changelog](#) for version-specific upgrade instructions.

### Upgrading from Clover 3.3 to Clover 4.0

In order to upgrade from Clover 3.3 to Clover 4.0 you have to:

- disable Clover on your projects ("*Package Explorer -> context menu -> Clover -> Enable/Disable on...*") - this is necessary to remove "Clover Pre-Java Builder" and "Clover Post-Java Builder",
- **uninstall** previous version of Clover and next install the Clover-for-Eclipse 4.x (un-installation is necessary because Clover's features and plugins have been renamed from *com.cenqua.\*\*\** to *com.atlassian.\*\*\**),
- enable Clover on your projects.

### Clover-for-Eclipse Changelog

See also [Clover-for-Ant Changelog](#).

### Clover-for-Eclipse Changelog

Changes for the latest major version are as follows:

#### Changes in Clover-for-Eclipse 4.0.0

July 11, 2014



This is a feature release with new HTML report with the ADG (Atlassian Design Guidelines) look. See the [Clover 4.0 Release Notes](#).

#### Implemented features and fixes

T	Key	Summary	P	Status	Fix Version/s
	CLOV-1345	Apply ADG in the HTML report	↑	CLOSED	4.0.0

1 issue

Please also see the [Clover-for-Ant Changelog](#) for all bugs fixed in the Clover product.

#### Known bugs

T	Key	Summary	P	Fix Version/s	Status
---	-----	---------	---	---------------	--------

No issues found


## Changes in Clover-for-Eclipse 3.3.0

### March 31, 2014

This is a feature release with dedicated support for Spock framework and JUnit4 Parametrized Tests.

Please note that Clover-for-Eclipse plug-in does not support Groovy in the IDE (see the [Supported Platforms](#) page), so you'll have to use Clover's Ant, Maven or Grails plug-in in order to instrument Spock tests.

#### Implemented features and fixes

T	Key	Summary	P	Status	Fix Version/s
	CLOV-1382	Add lambda toggle to report wizards in Eclipse and IDEA	↑	CLOSED	3.3.0

1 issue

Please also see the [Clover-for-Ant Changelog](#) for all bugs fixed in the Clover product.

#### Known bugs

T	Key	Summary	P	Fix Version/s	Status
---	-----	---------	---	---------------	--------

No issues found

#### Older versions

Looking for older versions? See [Clover-for-Eclipse Changelog](#) for Clover 3.2.

## Changes in Clover-for-Eclipse 4.0.0

### Changes in Clover-for-Eclipse 4.0.0

#### July 11, 2014

This is a feature release with new HTML report with the ADG (Atlassian Design Guidelines) look. See the [Clover 4.0 Release Notes](#).



**Implemented features and fixes**

T	Key	Summary	P	Status	Fix Version/s
	CLOV-1345	Apply ADG in the HTML report	↑	CLOSED	4.0.0

1 issue

Please also see the [Clover-for-Ant Changelog](#) for all bugs fixed in the Clover product.

**Known bugs**

T	Key	Summary	P	Fix Version/s	Status
---	-----	---------	---	---------------	--------

No issues found

## Changes in Clover-for-Eclipse 3.3.0


### Changes in Clover-for-Eclipse 3.3.0

**March 31, 2014**

This is a feature release with dedicated support for Spock framework and JUnit4 Parametrized Tests.

Please note that Clover-for-Eclipse plug-in does not support Groovy in the IDE (see the [Supported Platforms](#) page), so you'll have to use Clover's Ant, Maven or Grails plug-in in order to instrument Spock tests.

**Implemented features and fixes**

T	Key	Summary	P	Status	Fix Version/s
	CLOV-1382	Add lambda toggle to report wizards in Eclipse and IDEA	↑	CLOSED	3.3.0

1 issue

Please also see the [Clover-for-Ant Changelog](#) for all bugs fixed in the Clover product.

**Known bugs**

T	Key	Summary	P	Fix Version/s	Status
---	-----	---------	---	---------------	--------

No issues found

## Clover-for-Eclipse Glossary of Terms

This page contains definitions of terms used in the Clover-for-Eclipse plugin documentation.

On this page:

- [Avg Method Cmp](#)
- [Block Contexts](#)
- [Clover Working Set](#)
- [Cmp](#)
- [Cmp Density](#)
- [Columns](#)
- [Coverage Explorer \(view\)](#)
- [Coverage Explorer Column Chooser](#)
- [Coverage Explorer Custom Column Builder](#)
- [Coverage Cloud \(report\)](#)
- [Coverage Report](#)

- Coverage Treemap (report)
- Flush Policy (setting)
- Initstring (setting)
- Instrumentation
- LOC
- NC LOC
- Perspective (Eclipse concept)
- Project Explorer (view)
- Summary Panel
- Test Contributions (view)
- Test Run Explorer (view)
- Workspace (Eclipse setting)
- Workbench (Eclipse concept)

#### **Avg Method Cmp**

The average method complexity of code in the given context.

#### **Block Contexts**

Refers to common Java coding constructs or idioms such as the body of if statements; static initialiser blocks; or property style methods. These are pre-defined by Clover.

#### **Clover Working Set**

A specified set of files, directories and projects that Clover will report on. Especially useful when working with large projects.

#### **Cmp**

Cyclomatic coMPLexity of code in the given context.

#### **Cmp Density**

The complexity density of code in the given context.

#### **Columns**

Individual data sources that comprise part of a chart, visualisation or report.

#### **Coverage Explorer (view)**

The Coverage Explorer allows you to view and control Clover's instrumentation of your Java projects, and shows you the coverage statistics for each project based on recent test runs or application runs.

#### **Coverage Explorer Column Chooser**

A configuration screen that allows the user to set what is displayed in the Coverage Explorer, by selecting from the 24 columns available in Clover. The Column Chooser can be summoned by selecting "Columns..." in the Coverage Explorer view menu.

#### **Coverage Explorer Custom Column Builder**

A dialog that allows the user to define custom columns that show computed values from the data points used in Clover's reporting framework.

#### **Coverage Cloud (report)**

A Clover report visualisation that prints class names to the screen, coloured to show their level of code coverage and scaled in size to illustrate their complexity.

#### **Coverage Report**

Coverage reports are generated by Clover as PDF, HTML or XML, showing Clover's output in a readily digestible format for the user. These reports can be generated for single or multiple projects.

### Coverage Treemap (report)

A Clover report visualisation that shows packages and classes as coloured squares. The square's respective colour indicates the level of code coverage and they are scaled in size to illustrate their complexity (largest is most complex).

### Flush Policy (setting)

The Flush Policy controls when Clover writes coverage data to disk as your application runs.

### Initstring (setting)

This controls where the Clover plugin stores (and looks for) the coverage database.

### Instrumentation

in order to track the code coverage of your projects, Clover must insert special code into your programs at compilation time. This special code is collectively called instrumentation.

### LOC

Lines Of Code (including comment lines).

### NC LOC

Non-Commented Lines Of Code. Lines of code that contain comments are not counted in this metric, leaving only the actual functioning code itself.

### Perspective (Eclipse concept)

In the Eclipse IDE, each window in the desktop development environment contains one or more perspectives. Perspectives are containers for views and editors which control the content of the navigation user interface and controls.

### Project Explorer (view)

In Eclipse, the Project Explorer view shows a hierarchical view of the resources in the Workbench.

### Summary Panel

The Summary Panel is a set of metrics that are displayed alongside the tree for the selected project, package, file, class or method in the tree.

### Test Contributions (view)

The Test Contributions view shows unit tests and methods that generated coverage for the currently opened and selected Java source file.

### Test Run Explorer (view)

The Test Run Explorer view, like several popular plugins such as the JUnit Plugin or TestNG Plugin, lets you explore your recently run tests - showing whether they passed or failed, their duration and any error messages that they generated. Clover-for-Eclipse takes this one step further and allows you to explore the code coverage caused by an individual test, a test class, a package or even your entire project.

### Workspace (Eclipse setting)

The '**General > Workspace**' page is a configuration screen that is used to access IDE preferences in Eclipse.

### Workbench (Eclipse concept)

A general term for the Eclipse desktop development environment.

## Clover-for-Eclipse FAQ

### Clover Eclipse Plugin FAQ

- [I only need instrumented classes for unit testing and I don't want to risk publishing them to my production environment. How can I do this with Clover?](#) — Clover supports writing both instrumented and uninstrumented class files to different directories during a build.
- [Is Clover supported on IBM's RAD?](#) — Yes, IBM RAD is supported. See Supported Platforms page for more details.
- [I store my plugins and features in an Eclipse extension area. Does Clover support this?](#) — The "Clover 4" and "Clover 4 Ant Support" features can be placed in any extension location.
- [Why can I only see coverage data for the last test case I executed?](#) — Clover can be set to only display the coverage information gathered since your last compile — full build or auto build. The default behaviour is to include all coverage data found.

## Clover-for-IDEA

### Clover-for-IDEA Documentation

#### What is Clover-for-IDEA?

Clover-for-IDEA brings the industry-leading code coverage tool, [Atlassian Clover](#) to the IntelliJ IDEA integrated development environment. Clover-for-IDEA allows you to easily measure the coverage of your unit tests, enabling targeted work in unit testing resulting in stability and enhanced quality code with maximal efficiency of effort.

#### Getting Started with Clover for IDEA

[Installation Guide](#)

[Quick Start Guide](#)

[Changelog for latest version of Clover-for-IDEA](#)

#### Using Clover for IDEA

[User's Guide](#)

[Installation & Configuration Guide](#)

#### Resources and Support

[Atlassian Answers](#)

[FAQ](#)

[Technical Support](#)

#### Offline Documentation

You can download the Clover documentation in PDF, HTML or XML format.

### Recently Updated

 [Clover Road Map](#)

Aug 12, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)



[Upgrading third party libraries](#)

Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)



[Updating optimization snapshot file](#)

Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)



[Hacking Clover](#)

Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)



[Part 4 - Test Optimization Tutorial](#)

Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)



[Part 3 - Automating Coverage Checks](#)

Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)



[Part 2 - Historical Reporting](#)

Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)



[Part 1 - Measuring Coverage](#)

Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)



[Clover 4.0 Release Notes](#)

Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)



[A side-by-side comparison of the Classic and the ADG HTML report](#)

Aug 11, 2014 • updated by [Marek Parfianowicz \[Atlassian\]](#) • [view change](#)

## Clover-for-IDEA User's Guide

This page contains all user documentation for Clover-for-IDEA. Click a heading in the table below to jump to that section.

*On this page:*

- [Overview](#)
- [Using the plugin](#)
- [FAQ](#)

### Overview

The Clover IDEA Plugin allows you to instrument your Java code easily from within the [IntelliJ IDEA Java IDE](#), and then view your coverage results inside IDEA.

*Screenshot: Clover for IDEA Plugin*

Element	Coverage	Cplx	Lines	#Uncovered
com	94.7%	79	397	14
cenqua	94.7%	79	397	14
samples	94.7%	79	397	14
money	94.7%	79	397	14
IMoney	-	0	36	0
Money	89.4%	18	67	5
add	100%	1	3	0
addMoney	100%	2	5	0
addMoney	100%	1	3	0
amount	100%	1	3	0
appendTo	100%	1	3	0
currency	100%	1	3	0
equals	61.5%	4	11	5
hashCode	100%	1	3	0
isZero	100%	1	3	0
Money	100%	1	4	0
multiply	100%	1	3	0
negate	100%	1	3	0

Coverage and Tests	
Methods:	- / - -
Statements:	5 / 7 71.4%
Conditionals:	3 / 6 50%
<b>TOTAL:</b>	<b>61.5%</b>
Tests passed:	16 / 16 100%

Metrics	
Lines of Code:	-
NC Lines of Code:	-
Total Complexity:	4
Avg Complexity:	-
Complexity Density:	0.57
Recorded Test Cases:	0
Conditionals:	6
Statements:	7
Methods:	-
Classes:	-
Files:	-
Packages:	-
Test Methods:	0

## Using the plugin

We recommend starting your adventure with Clover for IDEA with a following lecture:

- 1. Clover for IDEA in 10 minutes

If you need more details how given features work, or how to efficiently work with Clover, you can read about:

- 2. Exploration of coverage in IDEA
- 3. Exploration of test results in IDEA
- 4. Scope of instrumentation in IDEA
- 5. IDEA configuration options
- 6. Generating reports in IDEA
- 7. Test Optimization for IDEA

For more advanced topics, like performance tuning, see:

- 9. IDEA Advanced topics











## FAQ

See the [Clover for IDEA Plugin FAQ](#). You can also search posts with the **clover** tag on [Atlassian Answers](#).

## 1. Clover for IDEA in 10 minutes

### Getting Started

This getting started guide will take you through the steps required to generate Clover coverage for your project.

1. Ensure that the clover plugin jar has been added to your project library path.
2. Enable Clover, by selecting the 'Enable Clover' check box in the "File | Settings | Project | Clover" interface.
3. Turn on clover instrumentation by selecting the  toolbar item.
4. Rebuild your project using any of the build mechanisms provided by IDEA.
5. Run your project by running the unit tests or some other means.
6. Refresh the latest coverage data by clicking the  toolbar item.
7. Highlight coverage in the source code editor by selecting the  toolbar item.  
Available highlighting options:
  -  highlight covered code (in green) and code with no coverage (in red),
  -  only highlight code with no coverage,
  -  turn code coverage highlighting off.
  -  this enables little gray and green clovers in package explorer. These indicate the toggled state of the exclusion annotation.
8. When  option is selected only coverage from passed unit tests contributes to the coverage percentage.
9. View the TreeMap report for the current project using the  button.
10. View the Cloud report for current project using  button.

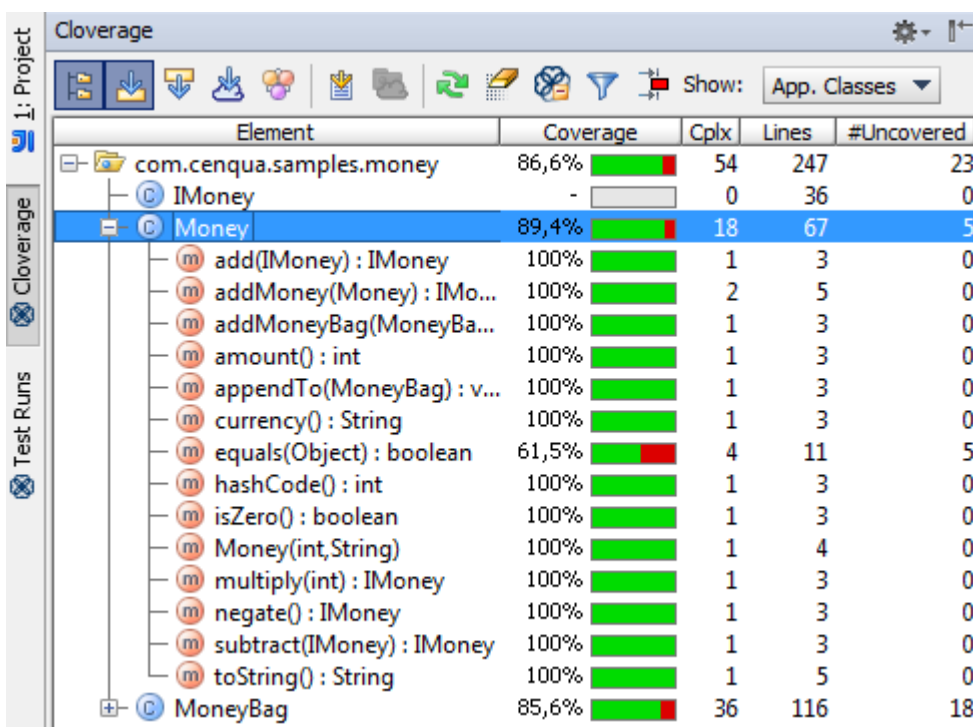
Congratulations! You now know basics of *Clover for IDEA* plugin - enough to start your daily work with it. If you want to learn more, read the [2. Exploration of coverage in IDEA](#) chapter.

## 2. Exploration of coverage in IDEA

### Viewing Coverage Results

Clover will record the code coverage information each time you run your application or a unit-test. This coverage information is available for viewing using IDEA.













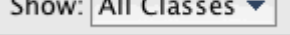
The coverage information can be browsed using the "*Cloverage*" window. The upper portion of the tool window contains a project class browser with inline coverage information:



Element	Coverage	Cplx	Lines	#Uncovered
com.cenqua.samples.money	86,6%	54	247	23
IMoney	-	0	36	0
Money	89,4%	18	67	5
add(IMoney) : IMoney	100%	1	3	0
addMoney(Money) : IMo...	100%	2	5	0
addMoneyBag(MoneyBa...	100%	1	3	0
amount() : int	100%	1	3	0
appendTo(MoneyBag) : v...	100%	1	3	0
currency() : String	100%	1	3	0
equals(Object) : boolean	61,5%	4	11	5
hashCode() : int	100%	1	3	0
isZero() : boolean	100%	1	3	0
Money(int,String)	100%	1	4	0
multiply(int) : IMoney	100%	1	3	0
negate() : IMoney	100%	1	3	0
subtract(IMoney) : IMoney	100%	1	3	0
toString() : String	100%	1	5	0
MoneyBag	85,6%	36	116	18



The tool bar at the top of the browser contains the following buttons:

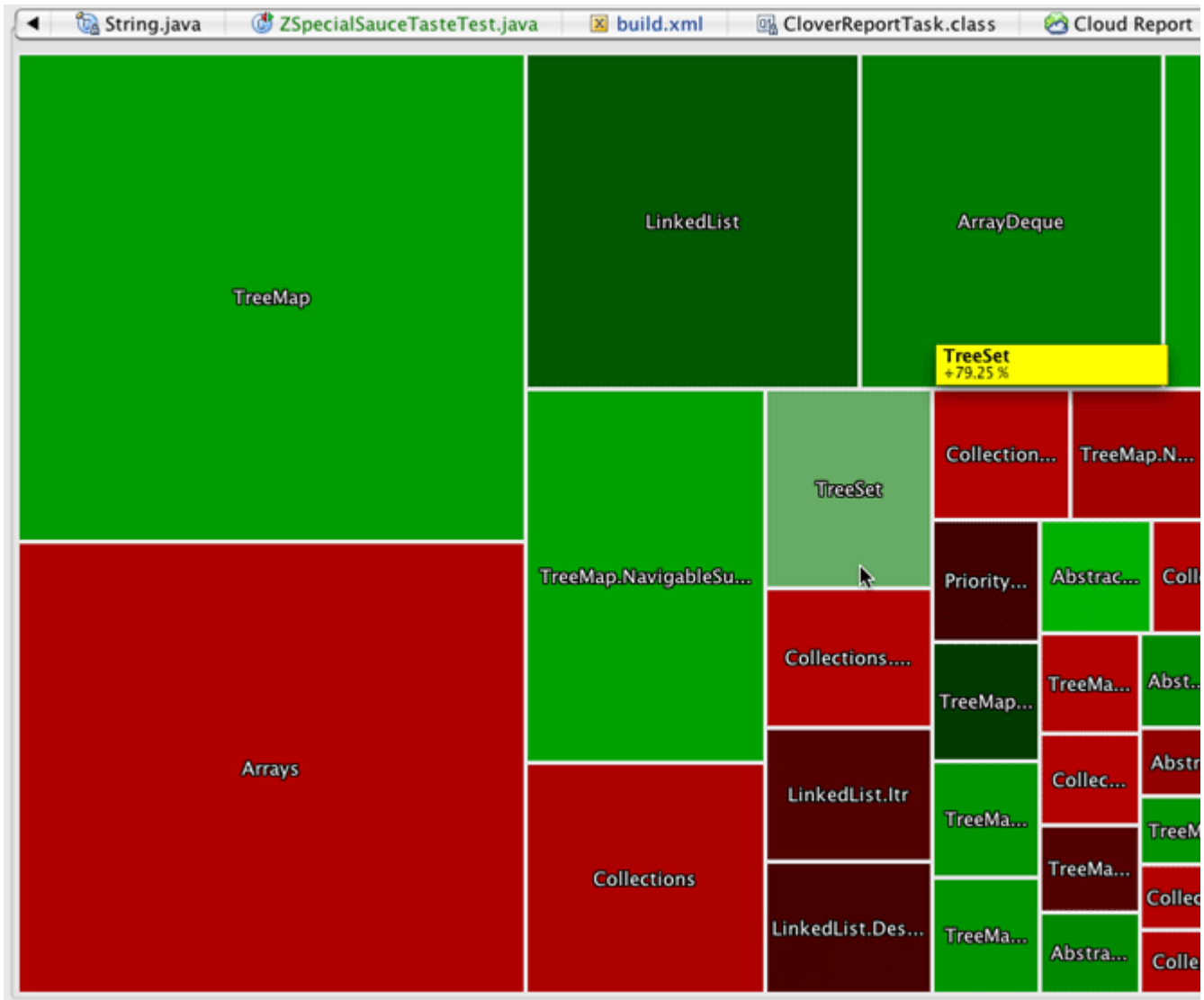
-  **Flatten Packages.** With this selected, only concrete packages are shown in the browser.
-  **Autoscroll to Source.** With this selected, a single click on a class in the browser will load the corresponding source file in an editor pane, with coverage info overlaid.
-  **Autoscroll from Source.** With this selected, the coverage browser will track the currently active source file in the editor pane.
-  **Always view package in the Cloud report.** When this is selected the Cloud view will automatically show a package selected in this tree.
-  **Show Coverage Summary.** With this selected, the Coverage metrics (see below) will be visible.
-  **Generate Report.** Launches a dialog to create a coverage report in HTML, XML or PDF format.
-  **Cloud report for selected package.** Opens Cloud report for selected package.
-  **Refresh.** Reloads coverage data.
-  **Clean Coverage.** Cleans gathered coverage data without deleting the instrumentation database.
-  **Delete.** Delete the current coverage database.
-  **Set Context Filter.** Launches a dialog to set the context filter.
-  **Hide Fully Covered Elements.** Removes elements with 100% coverage from view.
-  **Set Coverage Scope.** Choose which classes should be included in the Clover Coverage View - only application classes, only test classes or all classes.

#### Coverage tree map reports


The coverage tree map report allows simultaneous comparison of classes and package by complexity and by code coverage. The tree map is divided by a package (labelled) and then further divided by a class (unlabelled). The size of the package or the class indicates its complexity (larger squares indicate great complexity, while smaller squares indicate less complexity). Colours indicate the level of coverage, as follows:

- Bright green (most covered)
- Dark green (more coverage)
- Black (around 50% coverage)
- Dark Red (little coverage)
- Bright Red (uncovered)





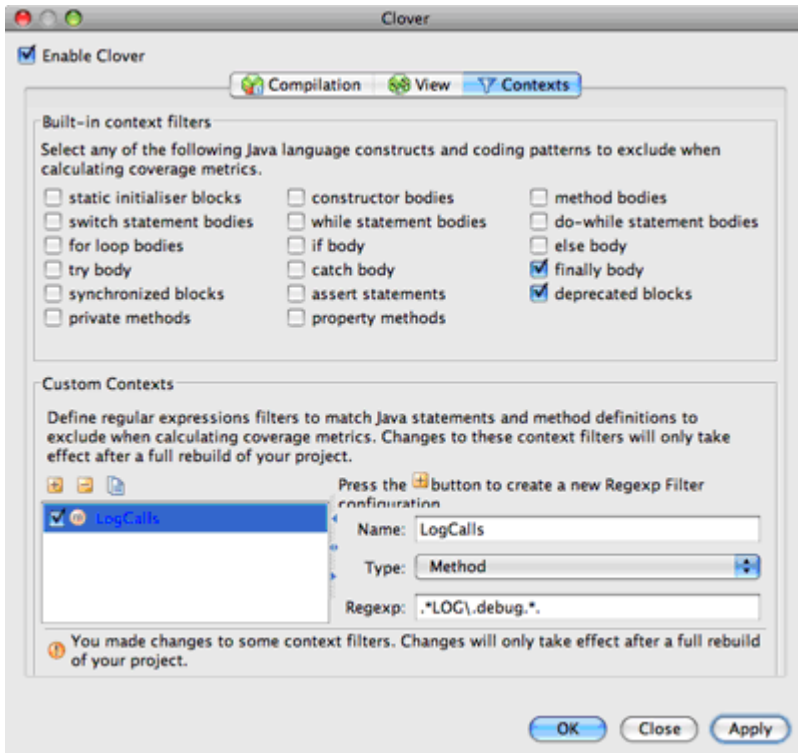
The percentage shown in the yellow box is the code coverage for the class currently under the mouse.

View the TreeMap report for current project using the  button.

### Setting Context Filters

The types of classes you want included in the Clover Coverage View can be set with Context Filters:

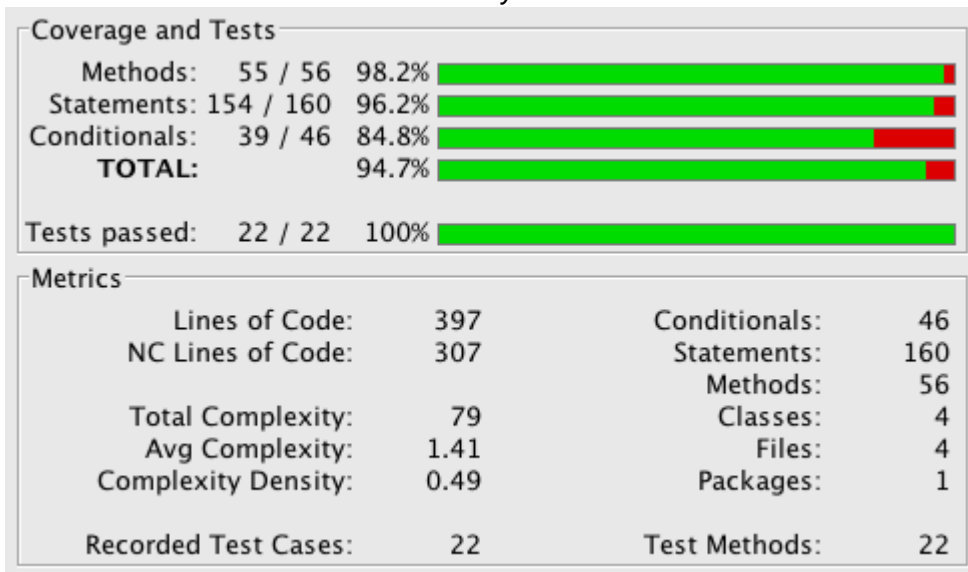
*Screenshot: Setting Context Filters*




### Using the Tool Window

The lower portion of the Tool Window contains various Metrics for the currently selected node in the browser:

Screenshot: Clover Tool Window Summary



### Showing Coverage with Annotations


In addition, the plugin can annotate the Java code with the coverage information. This can be turned on by pressing the Show Coverage  toolbar button.

Screenshot: Annotated Java Code

```

47         if (sum.isZero())
48             return;
49         fMonies.addElement(sum);
50     }
51     public boolean equals(Object anObject) {
52         if (isZero())
53             if (anObject instanceof IMoney)
54                 return ((IMoney)anObject).isZero();
55
56         if (anObject instanceof MoneyBag) {
57             MoneyBag aMoneyBag= (MoneyBag)anObject;
58             if (aMoneyBag.fMonies.size() != fMonies.size())
59                 return false;
60
61             for (Enumeration e= fMonies.elements(); e.hasMoreElements(); ) {
62                 Money m= (Money) e.nextElement();

```

 If you do not have "Auto Coverage Refresh" enabled, you will need to press the Refresh Button in the Main Toolbar or the Clover Tool Window to see the updated coverage information.




If a source file has changed since a Clover build, then a warning will be displayed alerting you to fact that the inline coverage information may not be accurate. The coverage highlighting will be yellow, rather than the red shown above.

Now you perfectly know how your application is covered. But how to efficiently navigate between tests and application code? Read the [3. Exploration of test results in IDEA](#) chapter.

### 3. Exploration of test results in IDEA

#### Test Runs Explorer

The Test Runs Explorer displays recently run tests in your Clover-instrumented project.

The upper panel displays test cases as a flat list (  ), grouped by package (  ) or by source root folder (  ).

The lower panel displays classes and methods covered by the test case selected in the upper panel. Two metrics are displayed for each class and method:

- **Contributed Coverage** indicates the percentage of statements that have been covered by selected test case,
- **Unique Coverage** indicates the percentage of statements that have been covered by selected test case only.

Test Case	Started	Status	Time	Contrib. Coverage	Unique Coverage
testMoneyBagEquals	8 sie 22:08:12	PASS	0,008s	39,5%	1,1%
testMoneyBagHash	8 sie 22:08:12	PASS	0s	26%	4,5%
testMoneyEquals	8 sie 22:08:12	PASS	0s	11,3%	0%
testNormalize2	8 sie 22:08:12	PASS	0s	41,2%	0%
testNormalize3	8 sie 22:08:12	PASS	0,001s	47,5%	0%
testNormalize4	8 sie 22:08:12	PASS	0s	45,2%	0%
testSimpleBagAdd	8 sie 22:08:12	PASS	0s	49,7%	0%
testSimpleMultiply	8 sie 22:08:12	PASS	0s	10,2%	0%
testSimpleNegate	8 sie 22:08:12	PASS	0s	10,2%	0%
testSimpleSubtract	8 sie 22:08:12	PASS	0s	14,7%	0%
testBagSimpleAdd	8 sie 22:08:12	PASS	0s	52%	0%
testBagSubtract	8 sie 22:08:12	PASS	0s	57,6%	0%
testBagNotEquals	8 sie 22:08:12	PASS	0s	35,6%	1,1%
testMixedSimpleAdd	8 sie 22:08:12	PASS	0s	35,6%	0%
testBagMultiply	8 sie 22:08:12	FAIL	0,002s	47,5%	10,7%

Element	Contrib. Coverage	Unique Coverage
default-pkg		
Money	55,3%	0%
Money(int,String)	100%	0%
add(IMoney) : IMoney	100%	0%
addMoney(Money) : IMoney	60%	0%
amount() : int	100%	0%
currency() : String	100%	0%
equals(Object) : boolean	46,2%	0%
isZero() : boolean	100%	0%
negate() : IMoney	100%	0%
subtract(IMoney) : IMoney	100%	0%

#### Unique Actions in Test Run Explorer

- / / : Click to choose test cases layout.
- : Toggle calculating test coverage to display in the upper panel. If enabled, the plugin will provide information about given test contribution to the selected target scope (see below). Note that enabling this option seriously slows down the Clover database refresh.
- **Target scope:** Only tests touching selected scope would be shown.
  - *All tests:* All recorded tests in the project,
  - *File tests:* All tests touching the currently displayed file,
  - *Class / Method / Statement at cursor:* All tests involving the class, method or statement under the cursor, respectively.

#### Select In -> Clover (Alt-F1 menu)

It is possible to view currently selected element in Clover using the Alt-F1 menu.

If the cursor is inside a recognized test case, it would be displayed in Test Runs Explorer, listing methods touched by the test in the lower panel.

Otherwise the element under cursor is displayed in the Coverage view.

You have already learned how to navigate through code coverage and test results. But don't you have a feeling

that your coverage reports could be more accurate when focused on *really important* areas of your application? If you do so, don't hesitate to read [4. Scope of instrumentation in IDEA](#) to learn how to configure instrumentation scope from whole project down to a single line of code.

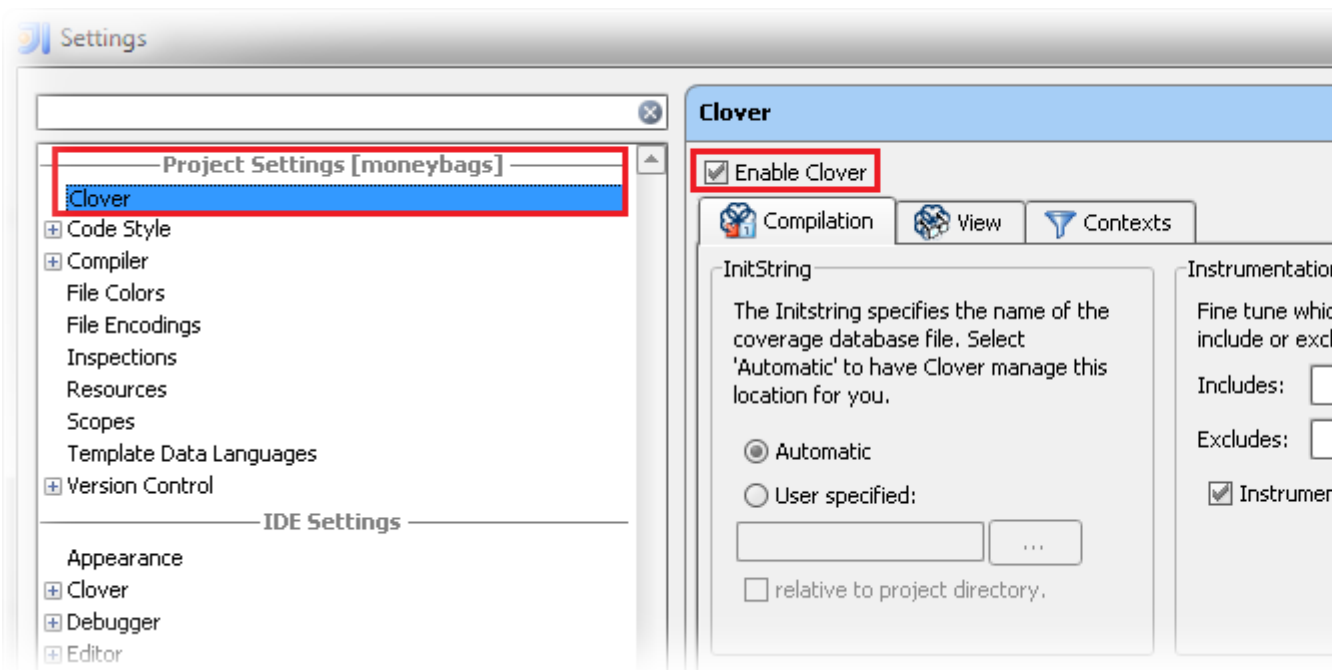
## 4. Scope of instrumentation in IDEA

Clover provides many ways to fine-tune instrumentation scope, which gives you an ability to concentrate your work on the most important code.

- Enabling and disabling Clover for a project
- Enabling and disabling build with Clover
- Excluding and including modules
- Excluding and including packages
- Excluding and including files
- Excluding certain blocks of code
- Excluding methods and statements matching regular expression
- Excluding arbitrary lines of code
- Showing Clover coverage annotations in Java source editors

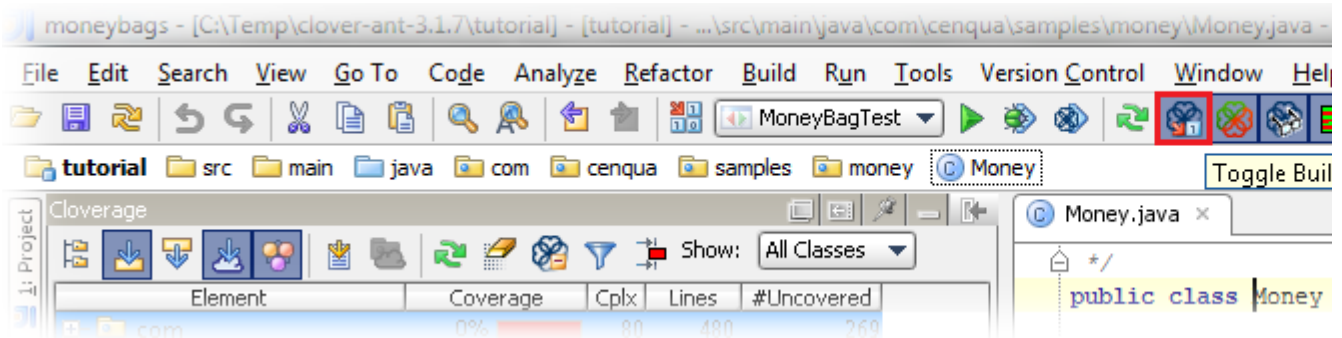
### Enabling and disabling Clover for a project

If you want to completely disable Clover support for a project (and remove all Clover data etc), then open *"File > Settings > Project Settings > Clover"* and deselect the *"Enable Clover"* checkbox.



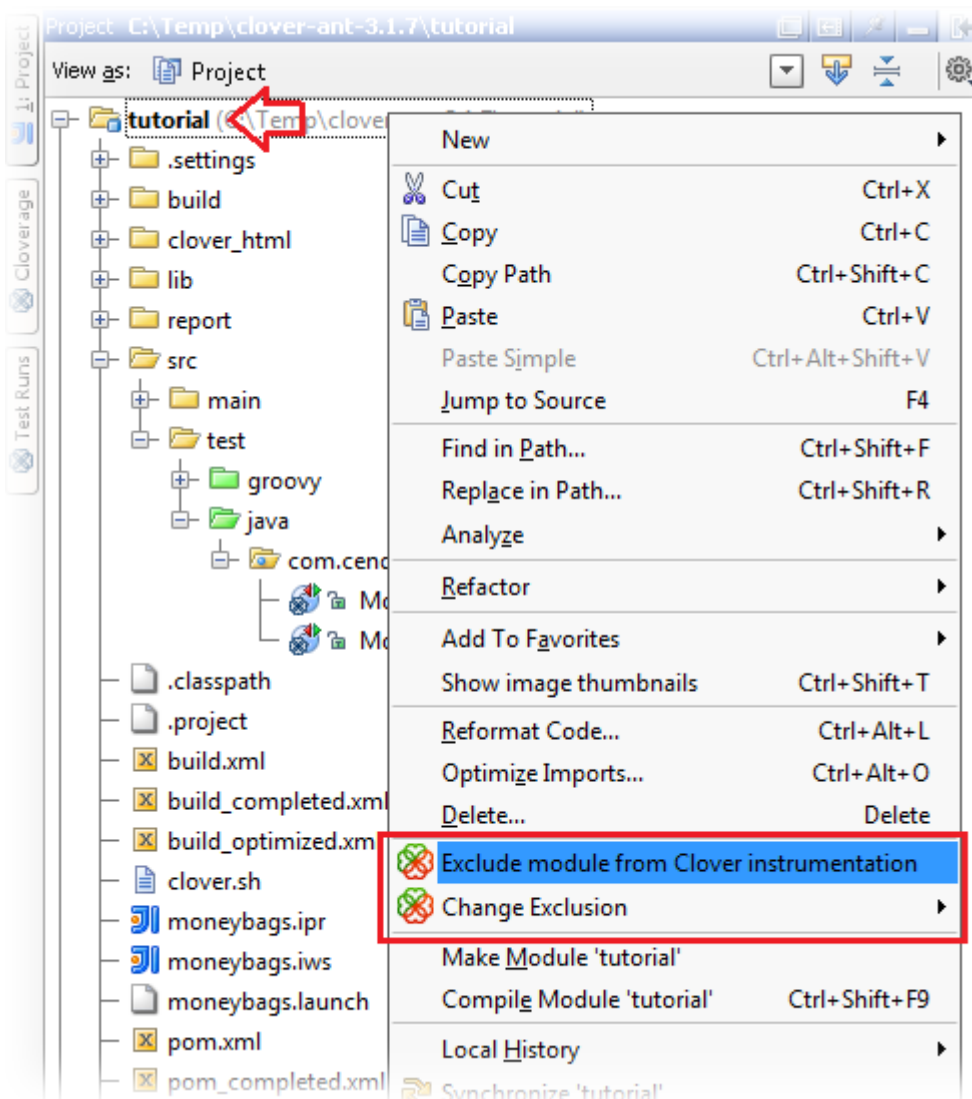
### Enabling and disabling build with Clover

In order to track the code coverage of your projects, Clover must insert special code into your program at compilation time - called instrumentation - to record this coverage. When Clover is enabled on your project, Clover will automatically perform this task for every file you compile in the project. You can tell Clover not to instrument your project by clicking "Toggle build with Clover" button (on a main bar).



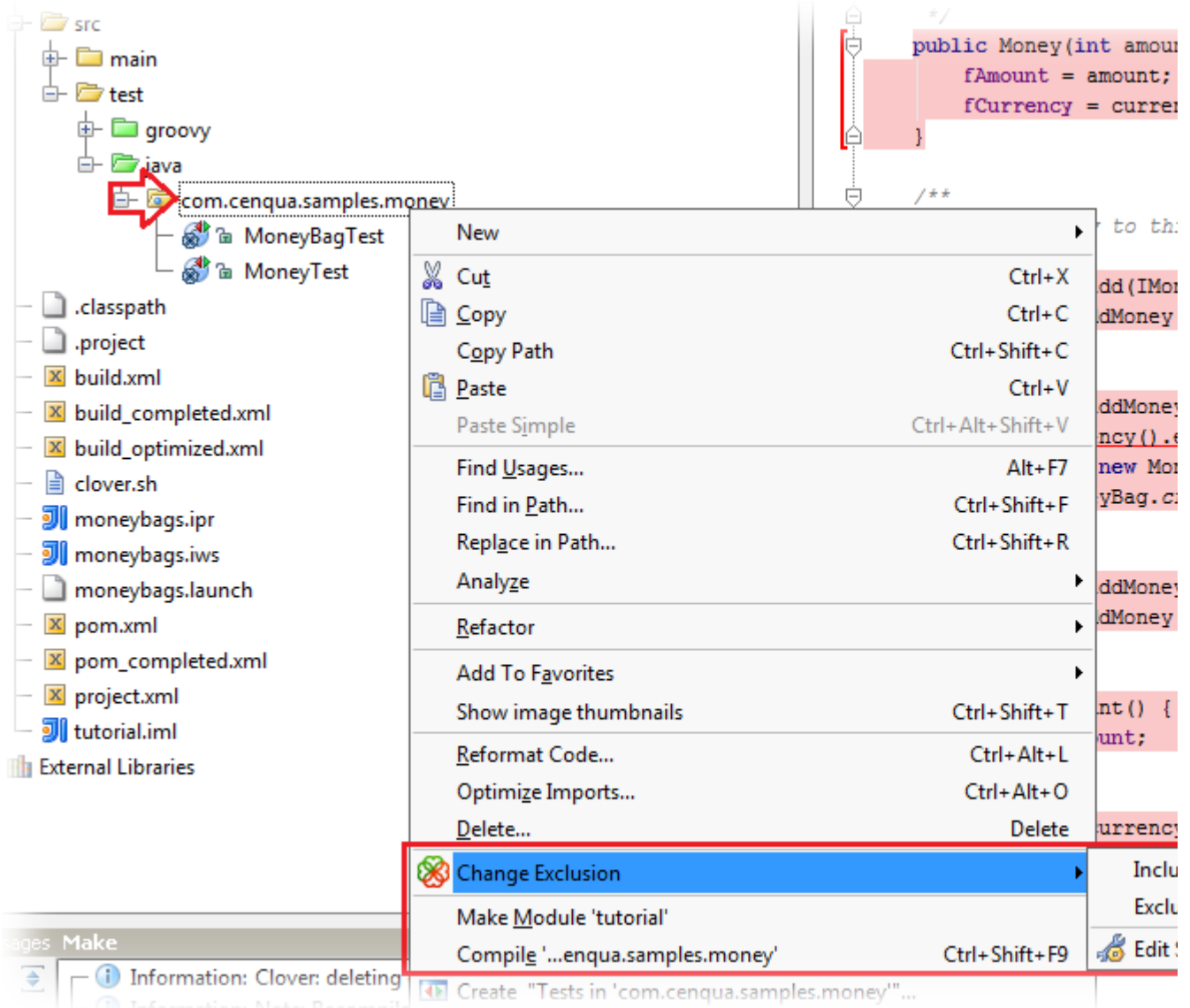
### Excluding and including modules

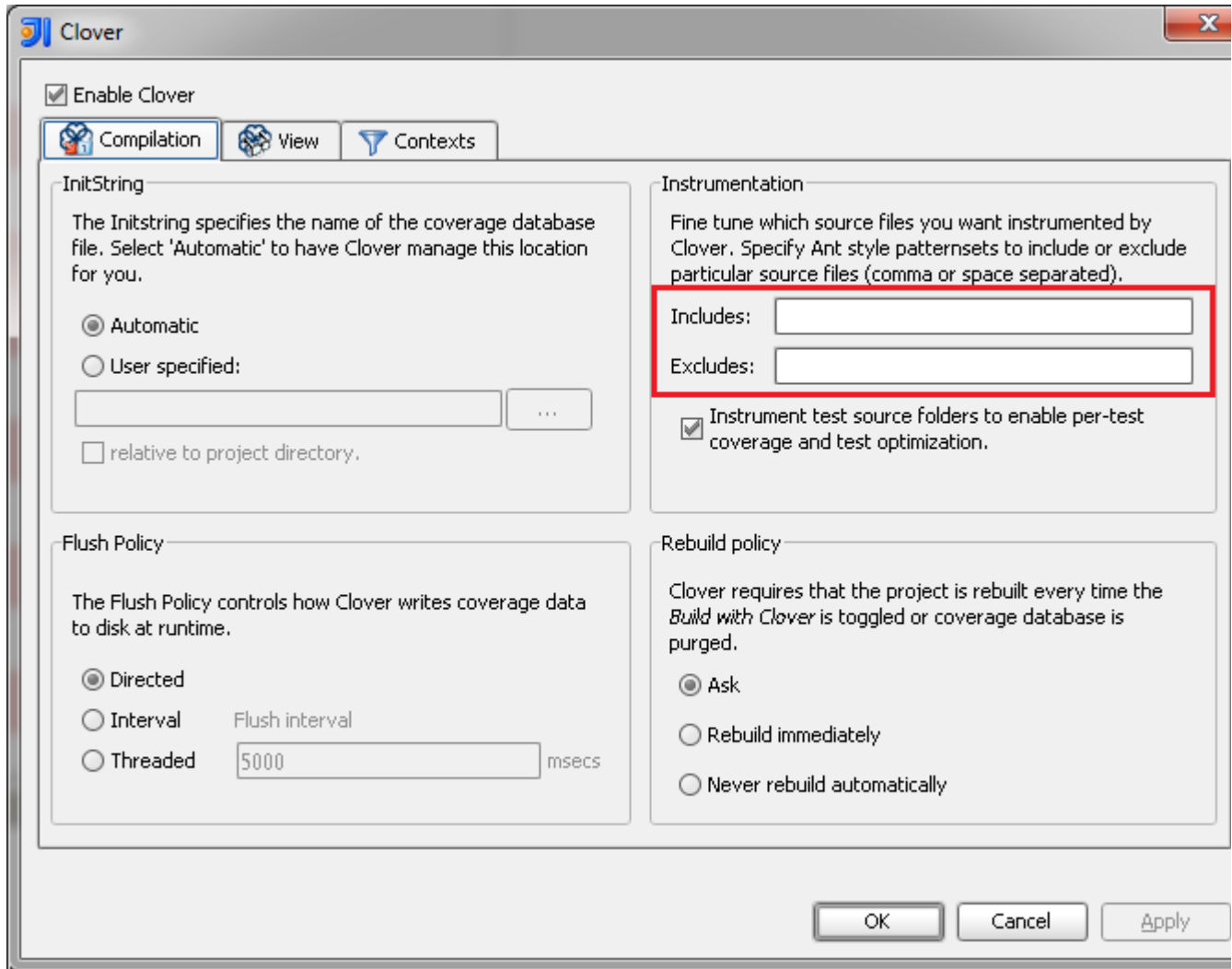
Right click on a module in Project view and select "Exclude module from Clover instrumentation" from context menu.



### Excluding and including packages

Right click on a package in Project view, select "Change exclusion" and next one of "Include ..." / "Exclude ..." / "Edit settings ..." from context menu.





### Excluding and including files

Right click on a file in Project view, select "Change exclusion" and next one of "Include ..." / "Exclude ..." / "Edit settings ..." from context menu.

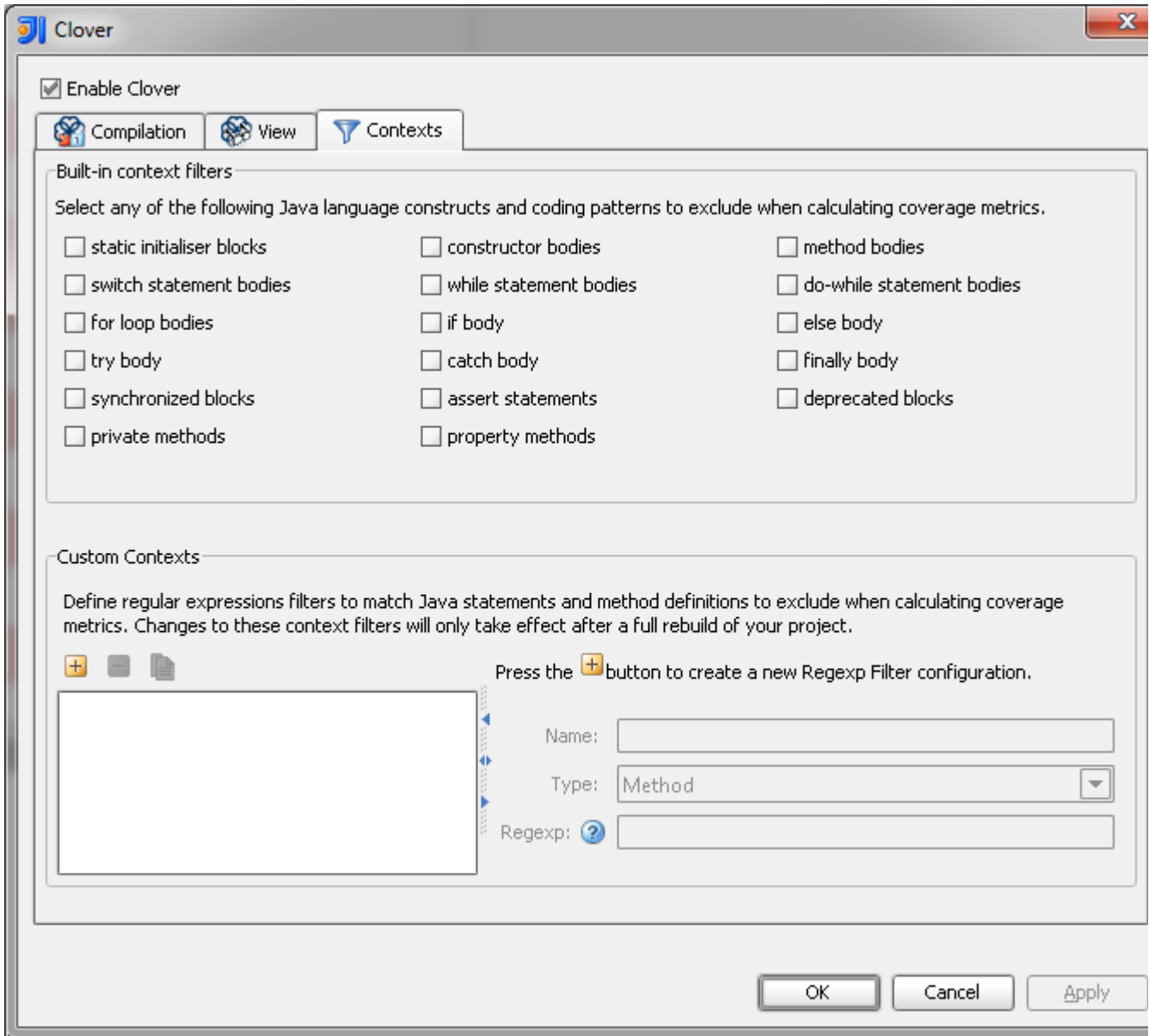
### Excluding certain blocks of code

Open "File > Settings > Project Settings > Clover" page. Open "Contexts" tab.

In the "Built-in context filters" box you can choose Java language constructs or coding patterns to be excluded. The most interesting are:

- assert statements
- catch body
- finally body
- private methods (all methods having private keyword)
- property methods (all methods having name like getXyz() / setXyz() / isXyz(), being public and having no arguments for isXyz()/getXyz())

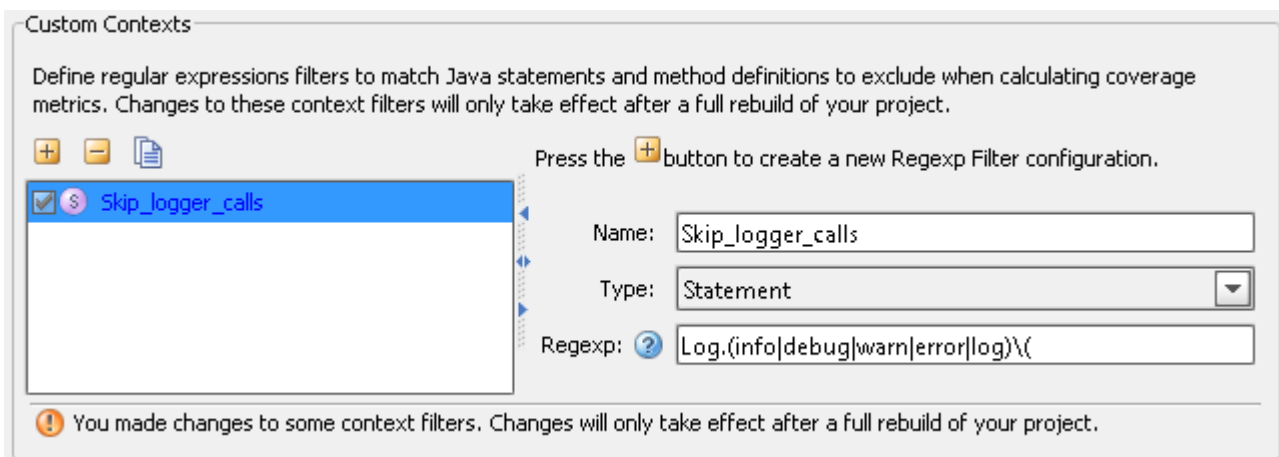




### Excluding methods and statements matching regular expression

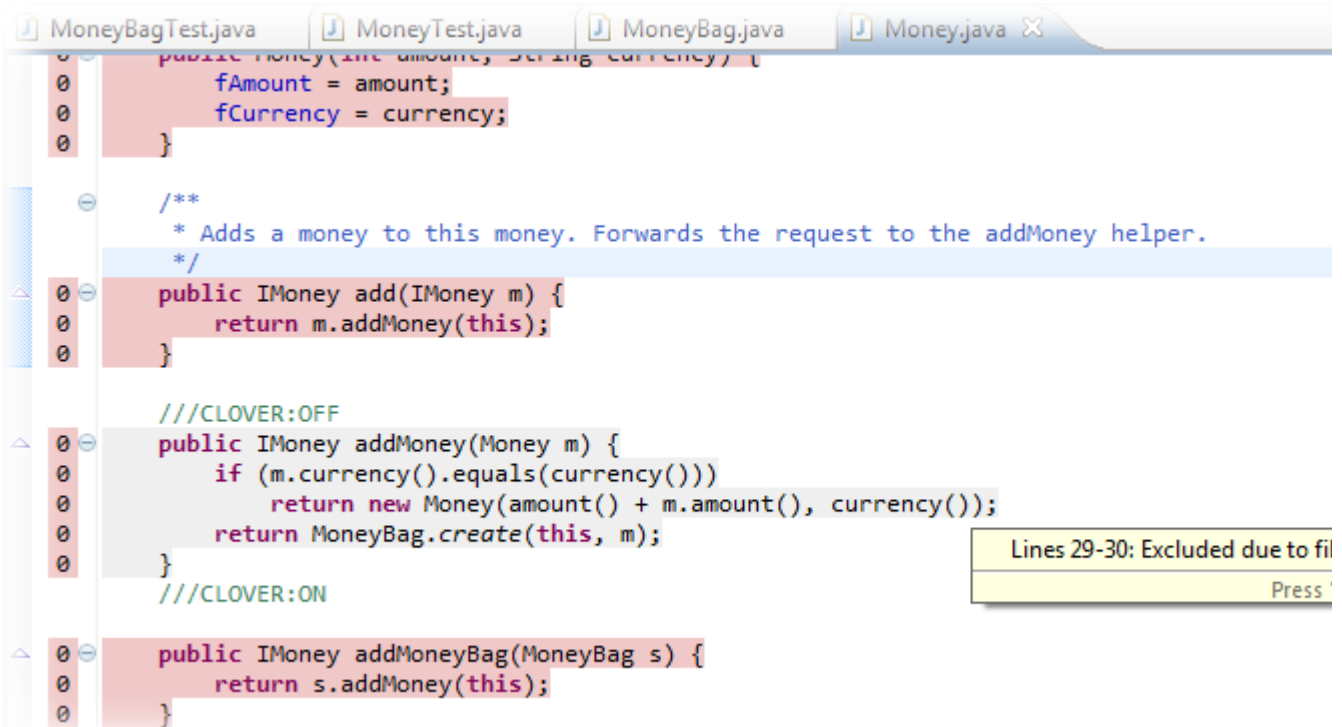
Open "File > Settings > Project Settings > Clover" page. Open "Contexts" tab.

In the "Custom Contexts" box you can define regular expressions for method signatures and statements.



### Excluding arbitrary lines of code

Put `///CLOVER:OFF` and `///CLOVER:ON` in source code (note that three slashes are used) to exclude given sections.

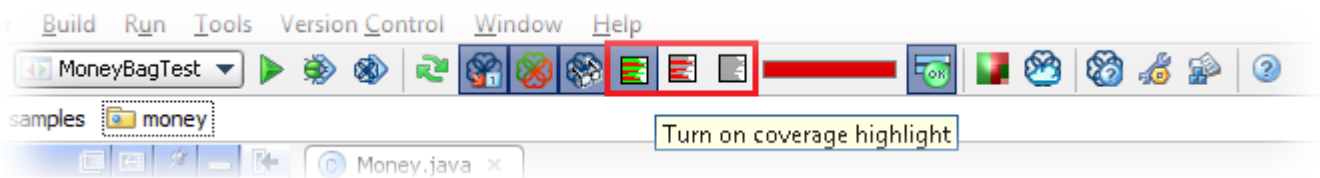


### Showing Clover coverage annotations in Java source editors

If you wish to temporarily disable the red/green code coverage annotations in your Java source editors (but wish to continue using Clover on your projects), you can simply toggle one of three toggles:

- "Turn on coverage highlight"
- "Turn on coverage highlight for uncovered code only"
- "Turn off coverage highlight"

These buttons are available on main menu bar.



Now you have your project instrumentation tuned to your needs. Are you looking for more tweaks? Read the [5. IDEA configuration options](#) chapter.

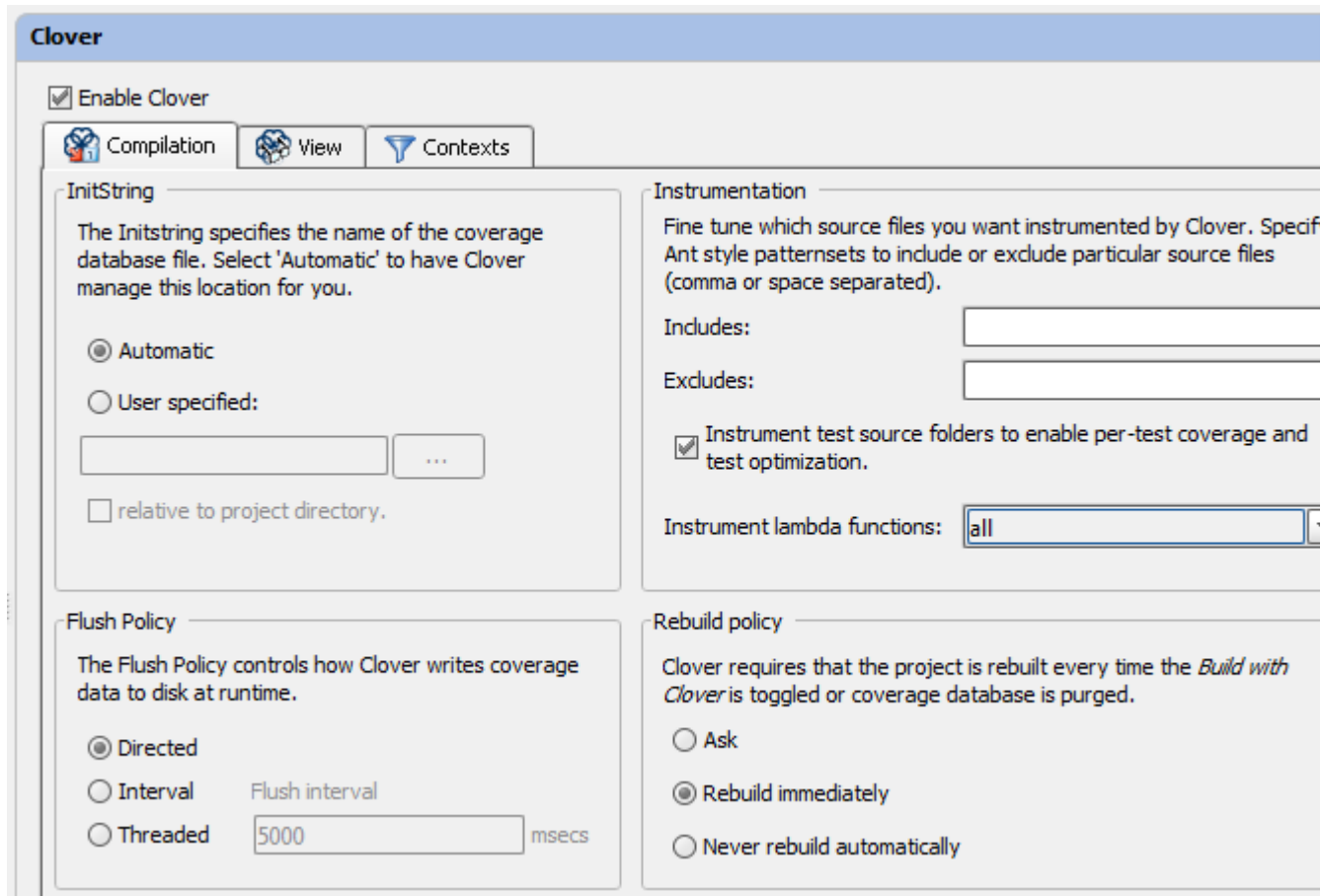
## 5. IDEA configuration options

### Configuration Options

#### Compilation Options

Configuration options for Clover are accessible on the Clover panel of the Project Properties dialog. The first Tab on this panel provides compilation options:

*Screenshot: Clover for IDEA Compilation Options*



#### Initstring

This section controls where the Clover coverage database will be stored. Select 'Automatic' to have Clover manage this location for you (relative to your project directory). Select 'User Specified' to nominate the path to the Clover coverage database. This is useful if you want to use the plugin in conjunction with an Ant build that already sets the location of the Clover coverage database.

#### Flush Policy

The Flush Policy controls how Clover writes coverage data to disk at runtime. See [Flush Policies](#).

#### Instrumentation

- **Includes / Excludes**

Allows you to specify a comma separated list of set of Ant Patterns that describe which files to include and exclude in instrumentation. These options are the same as those described in the [<clover-setup>](#) task. You can also specify whether source in test folders should be also instrumented.

For example, by using an "Excludes" value of `**/remote/*.java` you will stop instrumentation of files in the "remote" folder of your project.

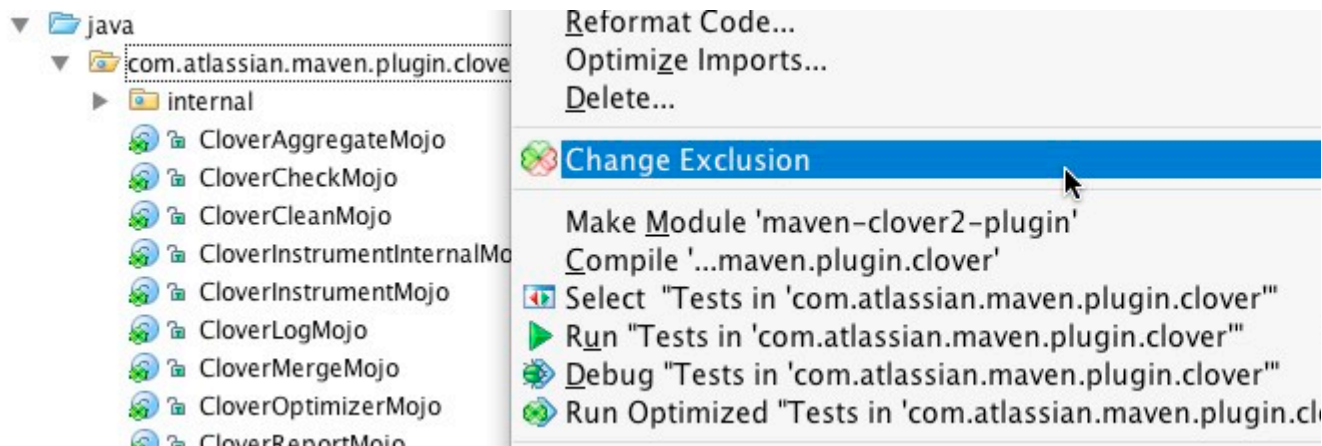
It is now possible to change exclusion/inclusion patterns directly from the Project Explorer.

Right click on a package or file, click the Change Exclusion context menu and select whether a pattern corresponding to the selected element should be added (or removed) from your includes or excludes list.

Files in the Project Explorer are annotated by gray or green clover when they are currently excluded or included by the current Clover configuration. A package is annotated when it is explicitly excluded or included.

**i** When your Includes/Excludes are edited manually via the settings, the package annotation may not work correctly (manual pattern may be not recognized), but the file annotation always reflects what the Clover instrumenter will use.

*Screenshot: Right-Click Context Menu for Setting Includes and Excludes*



- **Instrument test source folders ...** - when selected, sources from test folders will be instrumented and treated as test code
- **Instrument lambda functions:** whether Java 8 lambda functions shall be instrumented. If instrumented, they're treated like normal methods (and can be shown in HTML report and considered in code metrics, for example). Possible values:
  - *none* - do not instrument lambda functions,
  - *expression* - instrument lambdas in expression-like form, e.g. "(a, b) -> a + b",
  - *block* - instrument lambdas in code blocks, e.g. "(a, b) -> { return a + b; }"
  - *all* - instrument all lambda functions.

⚠ Due to Clover's restrictions related with code instrumentation and javac compiler's type inference capabilities, you may get compilation errors when expression-like lambda functions are passed to generic methods or types. In such case disable instrumentation of expression-like form (i.e. use the *none* or *block* setting). See the [Java 8 code instrumented by Clover fails to compile](#) Knowledge Base article for more details.

#### Rebuild Policy

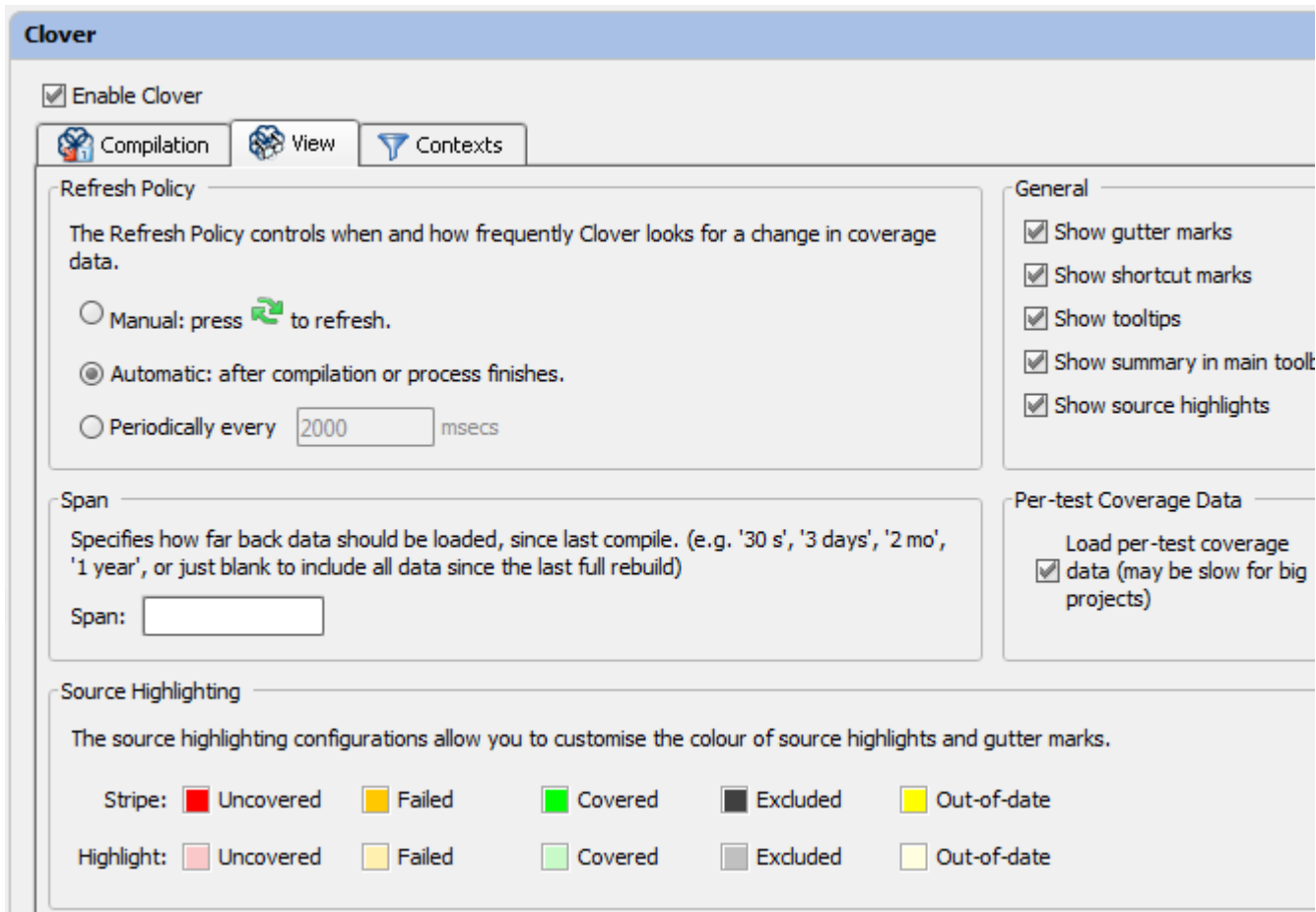
The Clover database becomes obsolete after certain operations (such as toggling *Build with Clover*). The Rebuild Policy setting allows defining the plugin's behavior in the case of such an event.

- **Ask:** The plugin will show a dialog window asking for confirmation.
- **Rebuild Immediately:** The plugin will rebuild the project automatically, without asking the user.
- **Never Rebuild Automatically:** The plugin will neither rebuild nor display a dialog — with this setting the user is responsible for rebuilding the project manually.

#### View Options

The second Tab on the configuration panel provides view options;

*Screenshot: Clover for IDEA View Options*



#### Refresh Policy

The Refresh Policy controls how the Clover Plugin monitors the Coverage Database for new data.

"Manual" means that you have to click  button to refresh the coverage data.

"Automatic" means that the Clover Plugin will refresh the database on:

- project open,
- finished compilation,
- termination of unit test or application run.

"Periodically" means that the Clover Plugin will periodically check for new coverage data for you.

#### General

Allows you to customize where coverage data is displayed within the IntelliJ IDE. Gutter marks appear in the left hand gutter of the Java Source Editor. Source highlights appear directly over your source code. Shortcut marks appear in the right hand gutter and allow you to navigate directly to uncovered regions of code.

#### Span

See [Using Spans](#).

#### Per-test Coverage Data

Per-test coverage can be disabled to boost performance. If you disable this feature, the following information will not be available:

- Test results.
- Code has been uniquely covered by specific tests.
- Code has been covered by failed tests only.

However, the coverage data will load faster as a result.

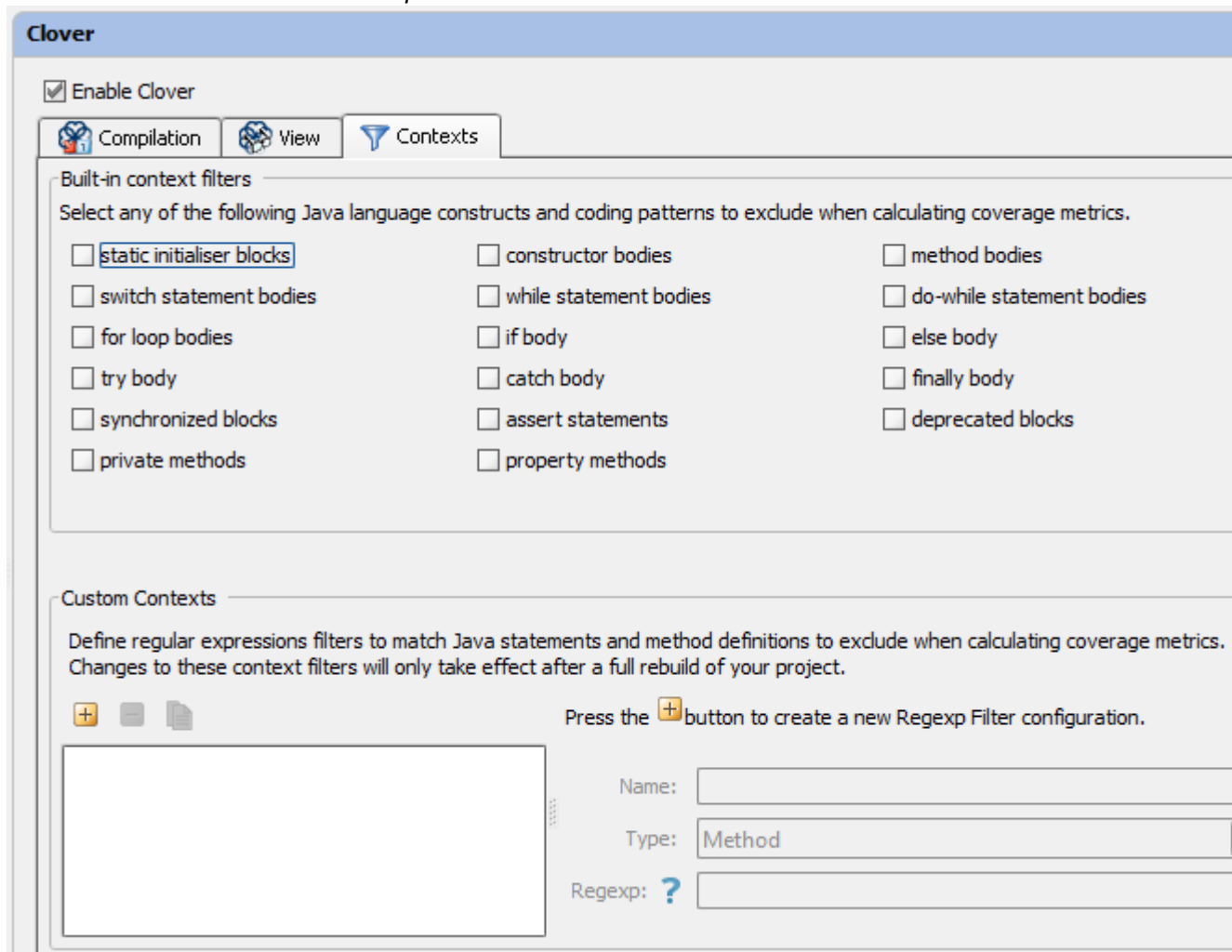
#### Source Highlighting

Allows you to fine tune the colours used Clover in its coverage reporting. The 'highlight colour' for each item is used for Source Highlights and the 'stripe colour' for each item is used for Gutter and Shortcut marks. You can click on a given colour to change it.

### Filter Options

The third Tab on the configuration panel provides filter options;

Screenshot: Clover for IDEA Filter Options







#### Built-in Context Filters

Allows you to specify contexts to *ignore* when viewing coverage information. For example, selecting the *finally body* context will remove 'finally' block bodies ('block' syntactic constructs in the Java language) from the reported coverage. For more information, see [Coverage Contexts](#) in the Clover Core documentation.

#### Custom Contexts

This allow you to define custom contexts to *ignore* when viewing coverage information.



#### Working with regexp filters:

- Use ,  or  to Create, Delete or Copy respectively the selected filter.
- All new and edited regexp filters will be shown in 'blue', indicating that they are currently unavailable.
- To make a new/edited filter active, you need to delete the existing coverage database using the  button and rebuild your project/module.

 See [Coverage Contexts](#) for more information.

### Example: Creating a regexp context filter

For the sake of this example, let us assume that we want to remove all private methods from the coverage reports. How would we go about this?

- Open the configuration panel "Settings | Clover | Filters".
- Select  to create a new Regexp Context Filter.
- Set the name to private.
- Since we are creating this filter to filter private 'methods', specify the Method type.
- We now need to define regular expression that will match all private method signatures. That is, a regexp that will match any method with the private modifier. An example of such a regexp is `(.*)?private .*`. Enter this regexp in the regexp field.
- You will notice that the name of this new filter appears in blue. Blue is used to indicate that the filter is either new or recently edited and therefore 'unavailable'. To make this new filter available, select  from the Main Toolbar and recompile your project. Once active, you will notice the private filter appear in the Context Filter Dialog. You will now be able to filter private methods out of your Clover coverage calculations and reports.

---

Now you have tweaked and hacked Clover according to your developer needs. But you would like to share information about code coverage with you colleagues or present it to management? If yes, read next chapter: [6. Generating reports in IDEA](#).

## Clover-for-IDEA Auto-Updates

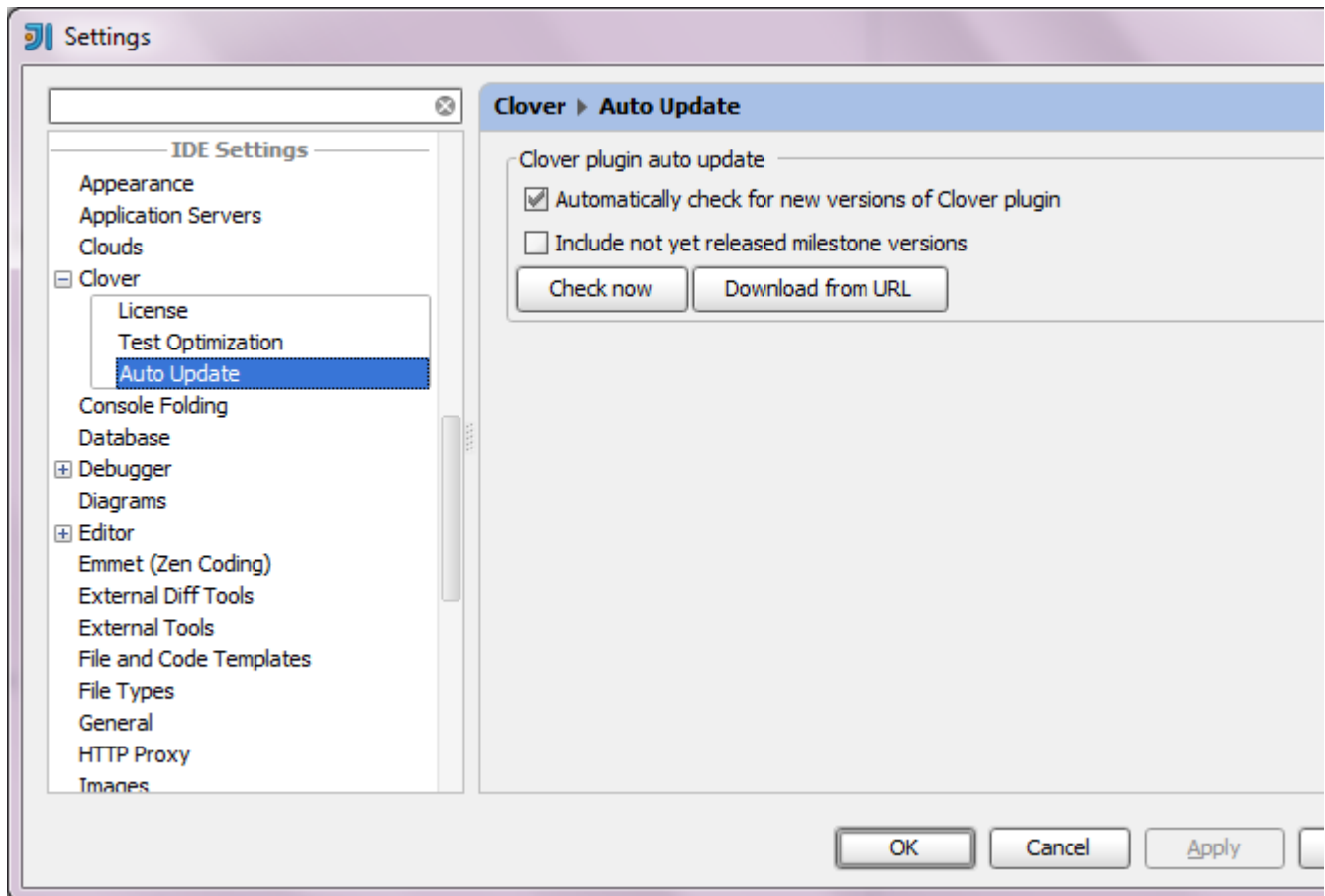
Clover-for-IDEA can check whether a newer version of the application is available with its '**Auto Update**' feature.

To configure this setting in IDEA, open '**Settings**', '**IDE Settings**', '**Clover**', '**Auto Update**'.

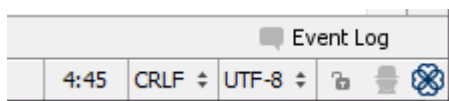
You can select the option '**Automatically check for new versions of Clover plugin**'. Another setting, '**Include not yet released milestone versions**' allows you to see information about upcoming releases of Clover-for-IDEA. Once configured, the plugin will do a daily check for a new version of the program.

You can check for the newest version by clicking '**Check now**'. You can also update the program from a specific location by clicking '**Download from URL**'.

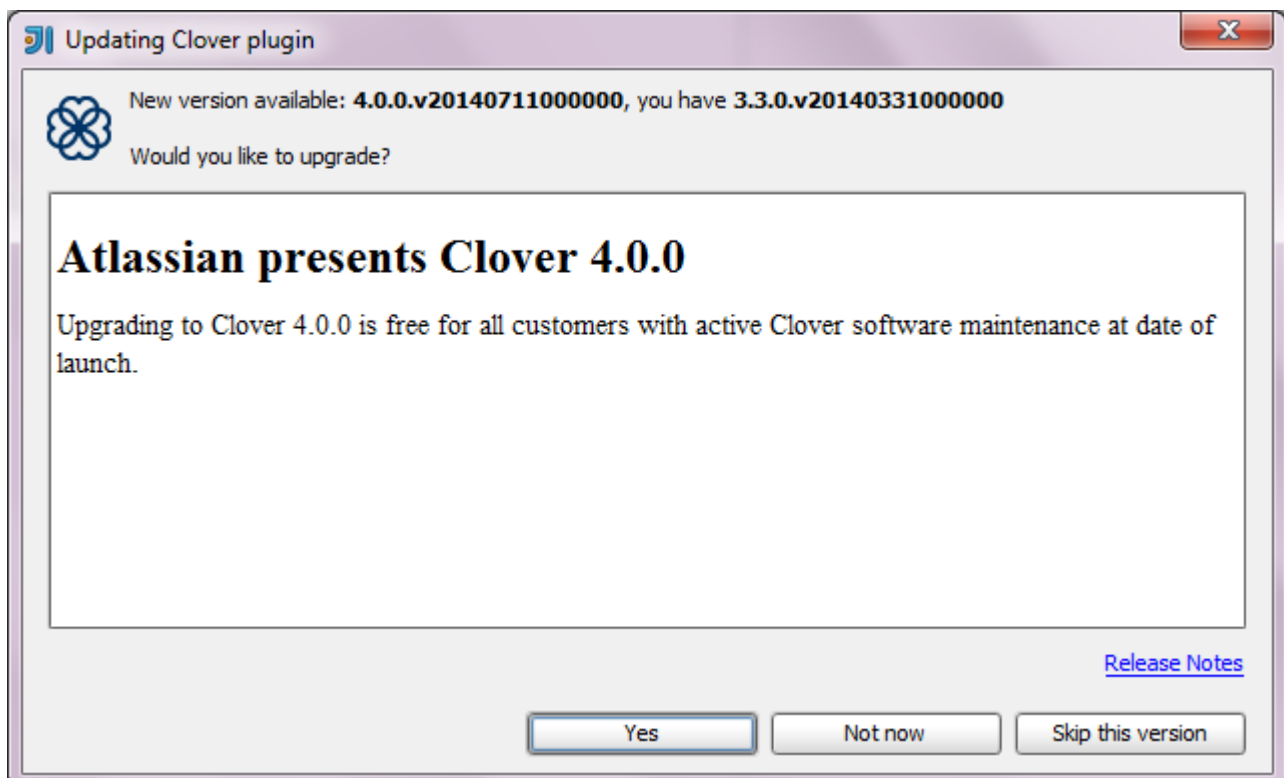




When an update to Clover-for-IDEA is available, a blinking Clover icon appears on the IDEA status bar:



You can click this icon, which will produce a dialog window showing new version information. You can also choose to upgrade Clover-for-IDEA directly from that window.





## 6. Generating reports in IDEA

- Introduction
- Generating a Report
  - Select an Output Format
  - Configure General Settings for the Report
  - Configure context filters
  - Finalise the Report
- Opening the Generated Report

### Introduction

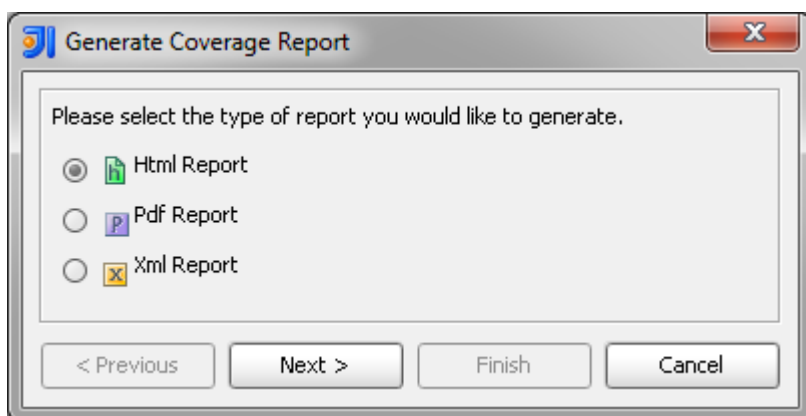
The Clover for IDEA plugin allows you to generate HTML, PDF or XML reports for your project.

### Generating a Report

To create a report, open the "Cloverage" window and click on the "Generate Clover Report" button. The 'Generate Coverage Report' dialog opens.

#### Select an Output Format

For your report, you can select an output format of HTML, PDF or XML. To select the desired output format, click the corresponding radio button in the 'Generate Coverage Report' dialog and click 'Next'.

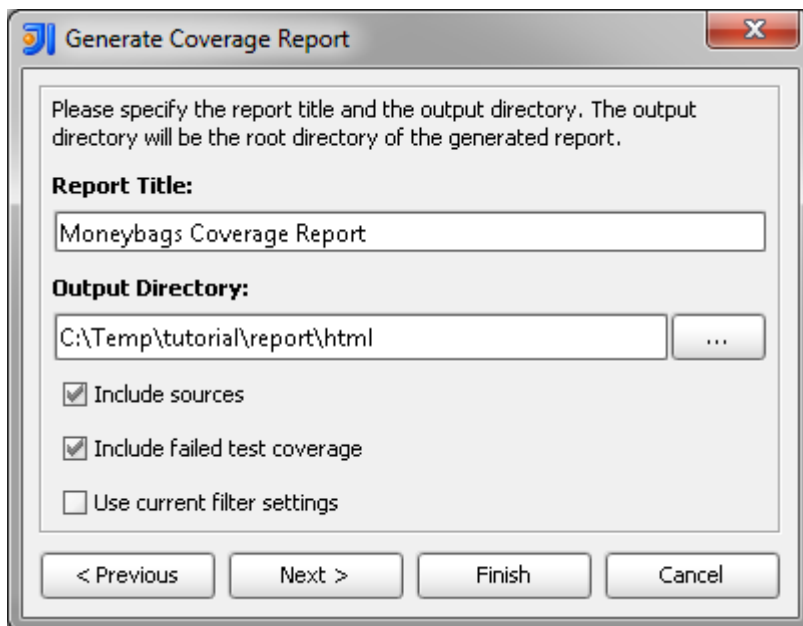


#### Configure General Settings for the Report

Your report can make use of the following settings:

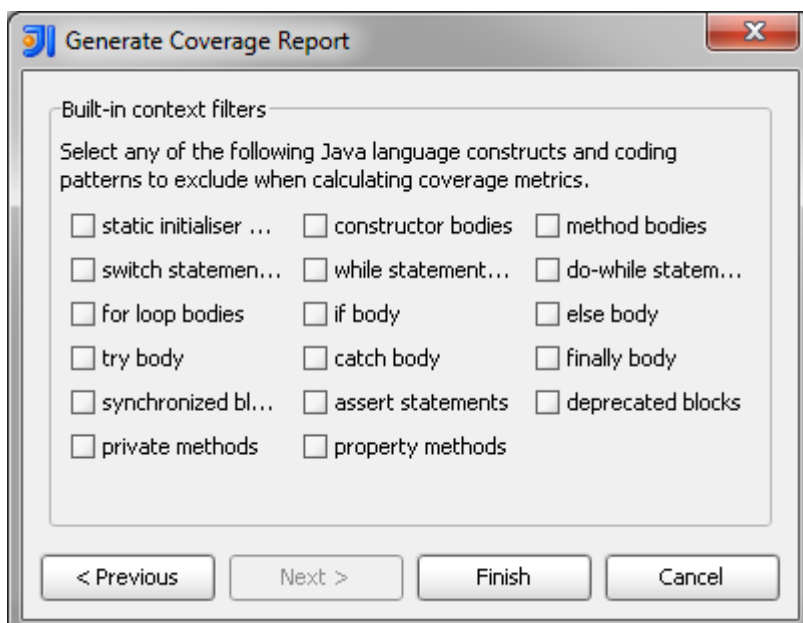
Setting	Default	Description
Report Title	<project name> Coverage Report	Report title.
Output Directory	<project dir>\report\html	Directory where report will be written to. In case when report already exists in this location, appropriate warning will be displayed.
Include sources (only for HTML reports)	True	Whether to include source code in the HTML reports. Not including source will mean users can't see per-line coverage information but report generation will run faster.
Include line info (only for XML reports)	False	Whether line by line coverage information is added to the report.

Include failed test coverage (only for HTML reports)	True	Tests from failed tests are included by default but can be excluded if they wish to discount this as worthy of being reported.
Use current filter settings	False	If the user un-checks this they will be given the opportunity to set a custom context.



### Configure context filters

Filter configuration page is only shown if you choose not to accept default filter settings. This lets you select any of the predefined pre-defined filters.

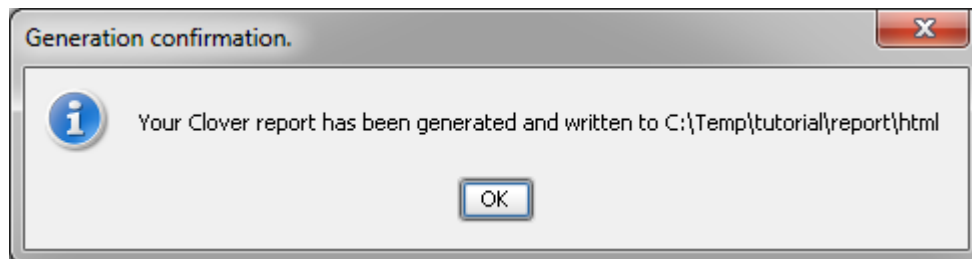


### Finalise the Report

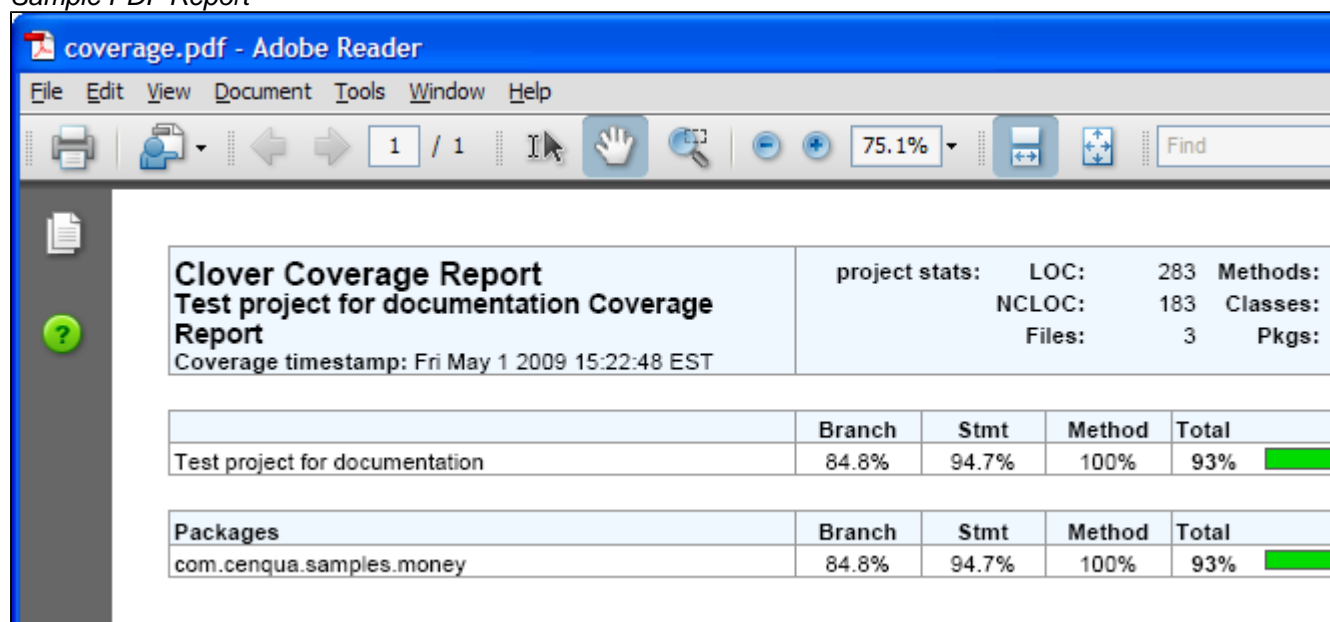
Clicking 'Finish' will start the report generation process. Progress will be displayed on status bar.

### Opening the Generated Report

When report generation is complete, a pop-up will display a location of generated report. You have to open this report manually.



### Sample PDF Report



Next chapter: [7. Test Optimization for IDEA.](#)

## 7. Test Optimization for IDEA

This page explains how to set up Clover's Test Optimization feature in the IDEA development environment.

On this page:

- [Before You Begin](#)
- [Launching Test Optimization](#)
- [Measuring Test Optimization Results](#)
- [Test Optimization Settings](#)
  - [Setting Global Preferences](#)
  - [Setting Per-launcher Preferences](#)
- [Configurations Unsuitable For Test Optimization](#)
  - [Limitations with Test Suites](#)
  - [Limitation with test methods](#)
- [Troubleshooting](#)

### Before You Begin

Before using Test Optimization with Clover-for-IDEA, be aware of the following.

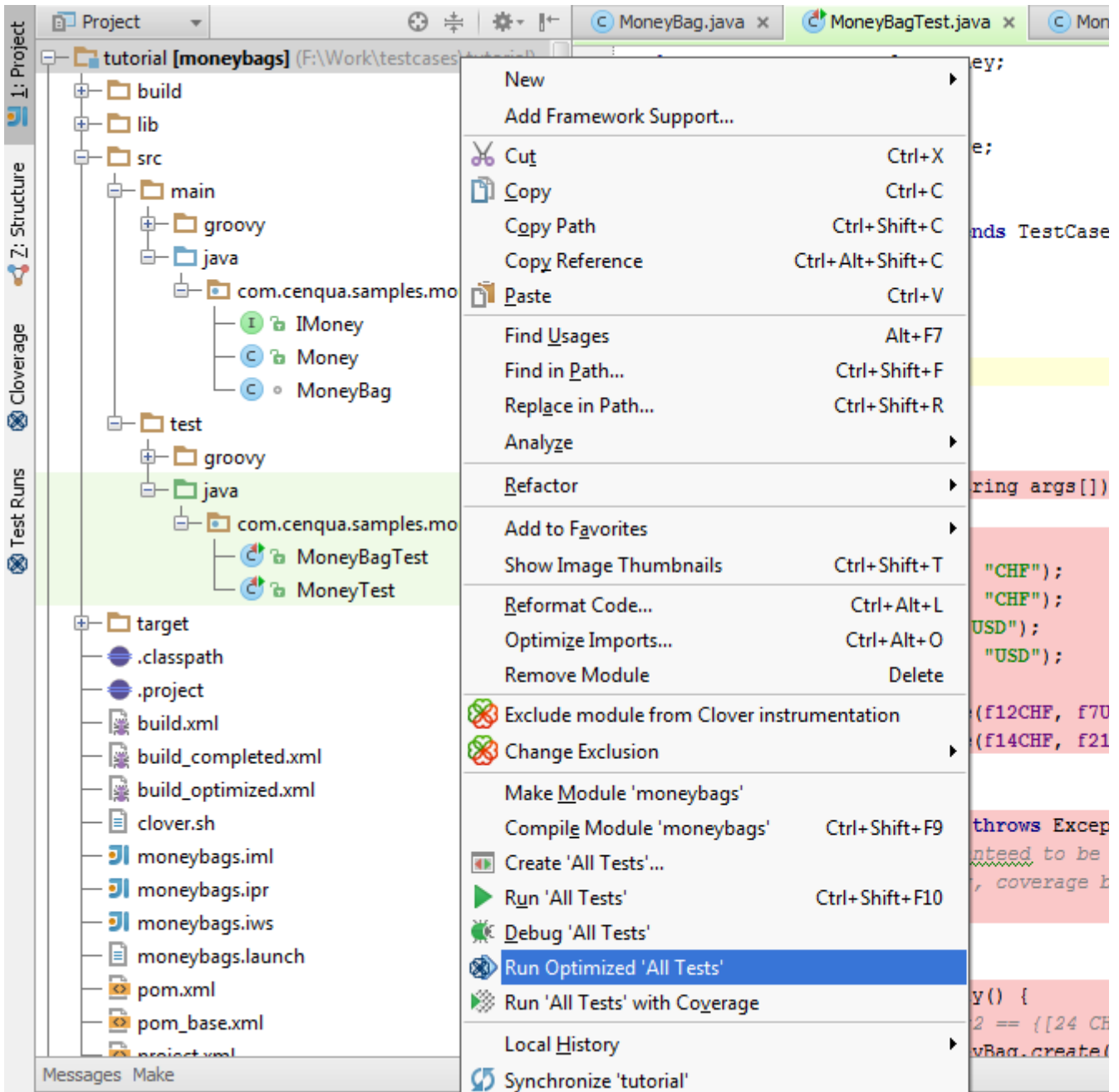
- Test Optimization is available as a '**Run Optimized**' command, similar to '**Run**' or '**Debug**'.
- Test Optimization supports JUnit launch configurations only.
- Ensure you have Clover enabled on the project; when there is no Clover instrumentation, there is no Test Optimization.

## Launching Test Optimization

To establish Test Optimization in Clover-for-IDEA, carry out one of the following actions:

- Right-click on a folder or package containing test classes and select 'Run Optimized', OR

Screen shot: launching a build with test optimization from the context menu



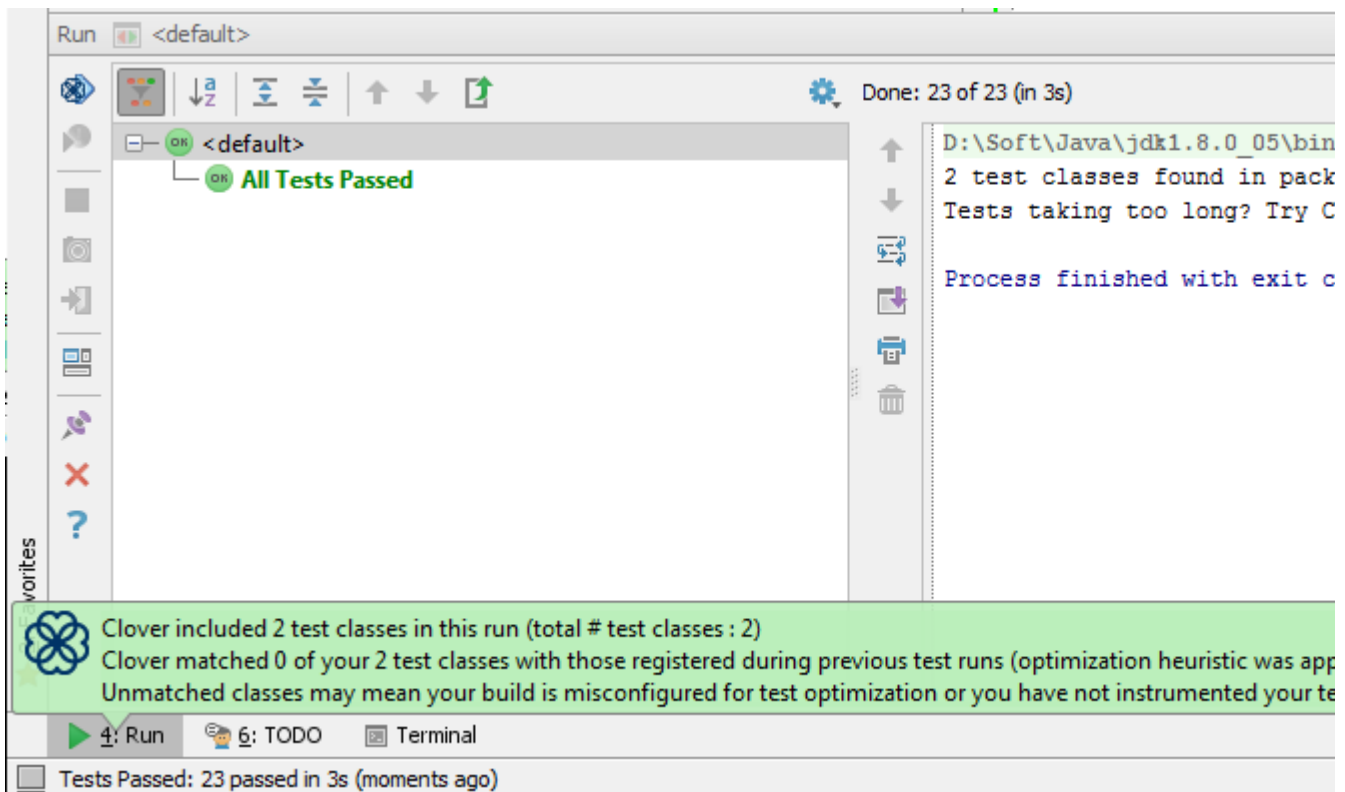
- Select an existing launch configuration in the **Run Configurations** drop-down menu and press  icon.


Screenshot: Launching a Build with Test Optimization from the Drop-Down Menu



## Measuring Test Optimization Results

When optimized tests are being run, Clover displays additional information about it.



After Optimized tests run, Clover saves a snapshot file with coverage information that is used to optimize the following test runs. This file may be deleted using the Delete Snapshot icon  in the Clover tool bar. The Delete Snapshot icon is invisible when the project does not have the snapshot file. Test Optimization would run all tests (no optimization) when the snapshot file is deleted or absent.

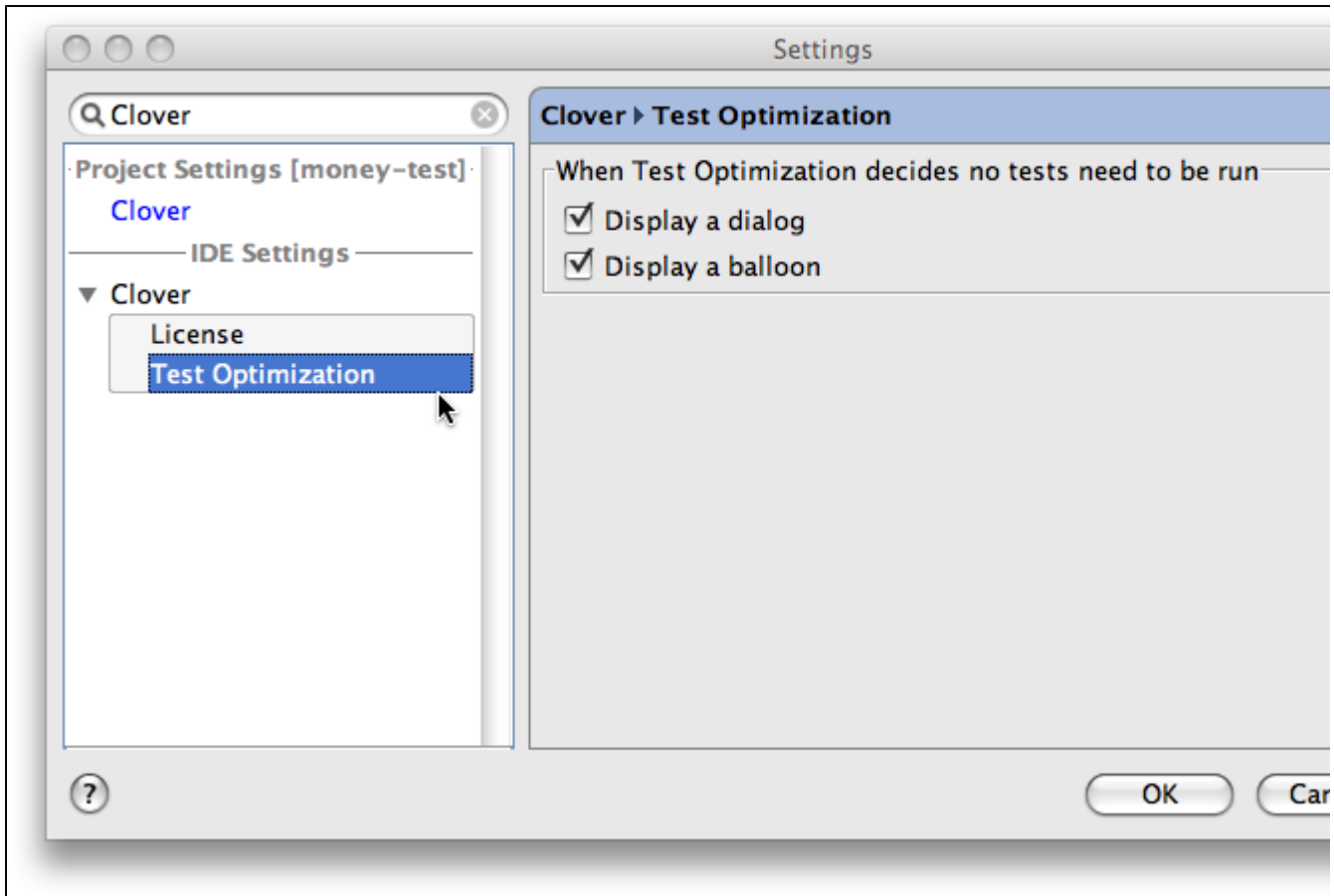
## Test Optimization Settings

The list below shows the settings available for Test Optimization.

- Discard snapshot every X compiles: When enabled, snapshot is re-generated every X compiles. This is the equivalent of Ant's '**fullrunevery**' setting (See [Clover-for-Ant documentation](#)).
- Minimize tests: main functionality. When disabled Clover only reorders tests, all of them are always run.
- Test reordering:
  - Do not reorder (means NOOP if Minimize Tests is also off)
  - Failing tests first: Re-order tests so that the ones which failed (the last time Optimized Test was run) are run first.
  - Random order.

## Setting Global Preferences

*Screenshot: Setting Global Preferences*

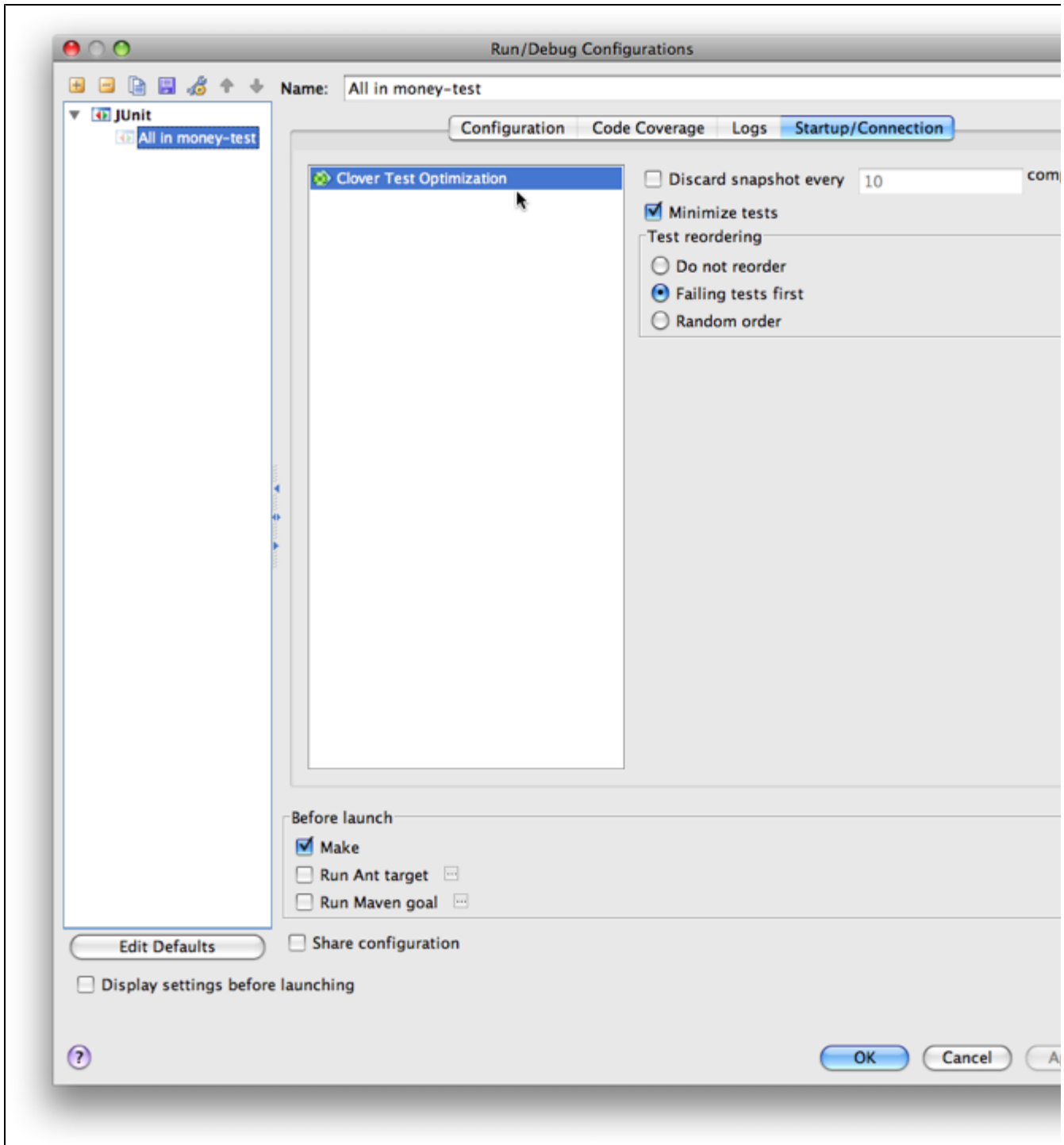


These are global (IDE-scope) preferences of Test Optimization where it is defined how Clover notifies about empty test runs, i.e. ones that have all tests optimized out.

- Display a dialog: Shows a dialog box that requires the user to close it manually before test run can proceed,
- Display a balloon: Pops up a notification balloon.

#### Setting Per-launcher Preferences

*Screenshot: Running Optimized Configurations*



Test Optimization specific configuration options for specific JUnit configurations. The defaults copied to new configurations may be set using Edit Defaults button.

### Configurations Unsuitable For Test Optimization

Unfortunately not all configurations are suitable for Test Optimization. Please see the following points for specific details.

#### Limitations with Test Suites

Clover does not recognise test suites as entities that should be optimized away. As the result test suites are always run (never optimized).

If your test launch configuration includes both test suite and the test case (which is probably an incorrect configuration), then the test case would be run twice (normal behavior) or once (via test suite) when the test case is optimized away.

Resolution: Do not include test suites in launch configuration, add test cases directly.

For more information, see this JIRA issue: [CIJ-249](#).

#### Limitation with test methods

Due to bug [CLOV-1084](#), in case when you execute optimized run and next add new test method to a class, such class will be always executed in next optimized runs. In order to fix it, you have to delete optimization snapshot (see button in tool bar menu).

#### Troubleshooting

To troubleshoot Test Optimization in Clover-for-IDEA, check through the following solutions:

1. If Clover is disabled for the project or generally Clover does not work for the project;
  - Ensure that the Clover icons are visible.
  - Check whether Coverage Explorer shows any coverage for the project.
2. If Clover has the test source settings wrong;
  - Check whether the Test Runs tool window shows any tests.
  - Ensure that Clover | Compilation | *Instrument test source folders* option is enabled.
3. If your test case is run twice, or not optimized at all;
  - Clover does not support test suites. Make sure you don't try to run one, launch test cases directly ([CIJ-249](#)).

---

Next chapter: [9. IDEA Advanced topics](#).

## 8. Launching Ant build from IDEA

## 9. IDEA Advanced topics

- [Performance Tuning in Clover for IDEA](#)

### Performance Tuning in Clover for IDEA

#### *Boosting Allocated Memory*

Tracking code coverage for a project, particularly per-test code coverage for large projects can consume a good deal of memory. If things are running a bit slowly with Clover enabled, consider [boosting the allocated memory](#) to your IDEA installation.

## Clover-for-IDEA Installation Guide

### Installing the plugin from IDEA (recommended)

The easiest way to install Clover-for-IDEA is to select it from IDEA's plugin menu.

1. Launch IntelliJ IDEA.
2. Go to '**File > Settings > IDE Settings > Plugins > Browse repositories**'. Select 'Atlassian Clover for IDEA' and click 'Install plugin'.
3. Close and re-start IDEA.

You can now begin [using Clover for IDEA](#).

Screen shot: *installing Clover from the IDEA plugin menu*



The screenshot shows the 'Browse Repositories' window in IntelliJ IDEA. The 'Atlassian Clover for IDEA' plugin is selected and highlighted in blue. The plugin details on the right include:

- Category:** TOOLS INTEGRATION
- Install plugin:** A green button with a download icon.
- Rating:** 5 stars (★★★★★)
- Downloads:** 63784
- Updated:** 2014-07-11
- Version:** ver idea-4.0.0.v20140711000000
- Description:** Clover is an award-winning Java code coverage analysis instruments source code and then records precisely wh tests are run. The detailed test coverage reports help de areas where the testing is weak, enabling them to write Clover fits into a developer's environment, be it Ant, Maven command line. Quality Assurance and Project Managers quality metrics over time via html and pdf reports. Clover to thousands of companies and open source projects a
- Documentation:** Documentation is available online from <http://confluence.atlassian.com/display/CLOVER/Clover>
- Activation:** To activate this plugin, you will require a Clover license f [Clover license page](http://confluence.atlassian.com/display/CLOVER/Clover) on atlassian.com
- Change Notes:** Clover 4.0 brings a redesigned HTML report, following th Guidelines. See: <https://confluence.atlassian.com/display/CLOVER/Clove>
- Vendor:** Atlassian

## Installing the plugin manually

You can download the Clover-for-IDEA plugin from [Atlassian Downloads](#) and install it manually as follows:

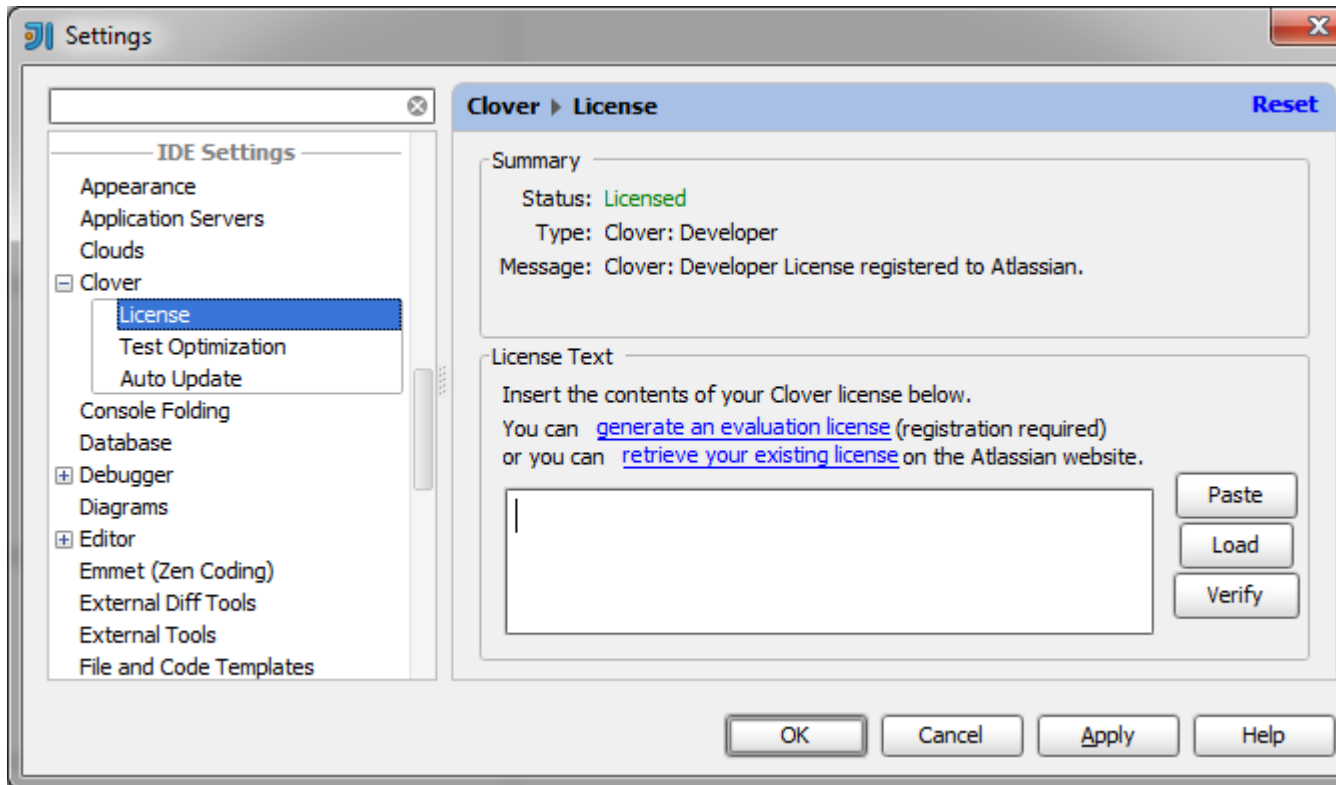
1. Shut down any running instances of IDEA.
2. Remove any previous versions of the the Clover-for-IDEA plugin .jar file from the following plugin installation locations:
  - a. IDEA\_HOME/plugins (all platforms)
  - b. IDEA\_HOME/config/plugins (all platforms excepting Mac OS X)
  - c. USER\_HOME/Library/Application\ Support/IntelliJIDEA70 (Mac OS X only)
3. Copy the downloaded .jar file into the relevant location for your operating system:
  - a. IDEA\_HOME/plugins (all platforms)
  - b. IDEA\_HOME/config/plugins (all platforms excepting Mac OS X)
  - c. USER\_HOME/Library/Application\ Support/IntelliJIDEA70 (Mac OS X only - create the directory if it doesn't exist)
4. Re-start IDEA.

You will need a valid license to activate your plugin.

- Download your clover.license file from <http://www.atlassian.com/clover/>. Evaluation licenses are available free of charge.
- Open the Clover license dialog in IDEA. Go to '**File > Settings > IDE Settings > Clover > Clover**

**License**, click **Load** and select the 'clover.license' file you just downloaded.

Screen shot: the Clover license dialog in IDEA



## Configuring compiler settings

### **i** IntelliJ IDEA 12

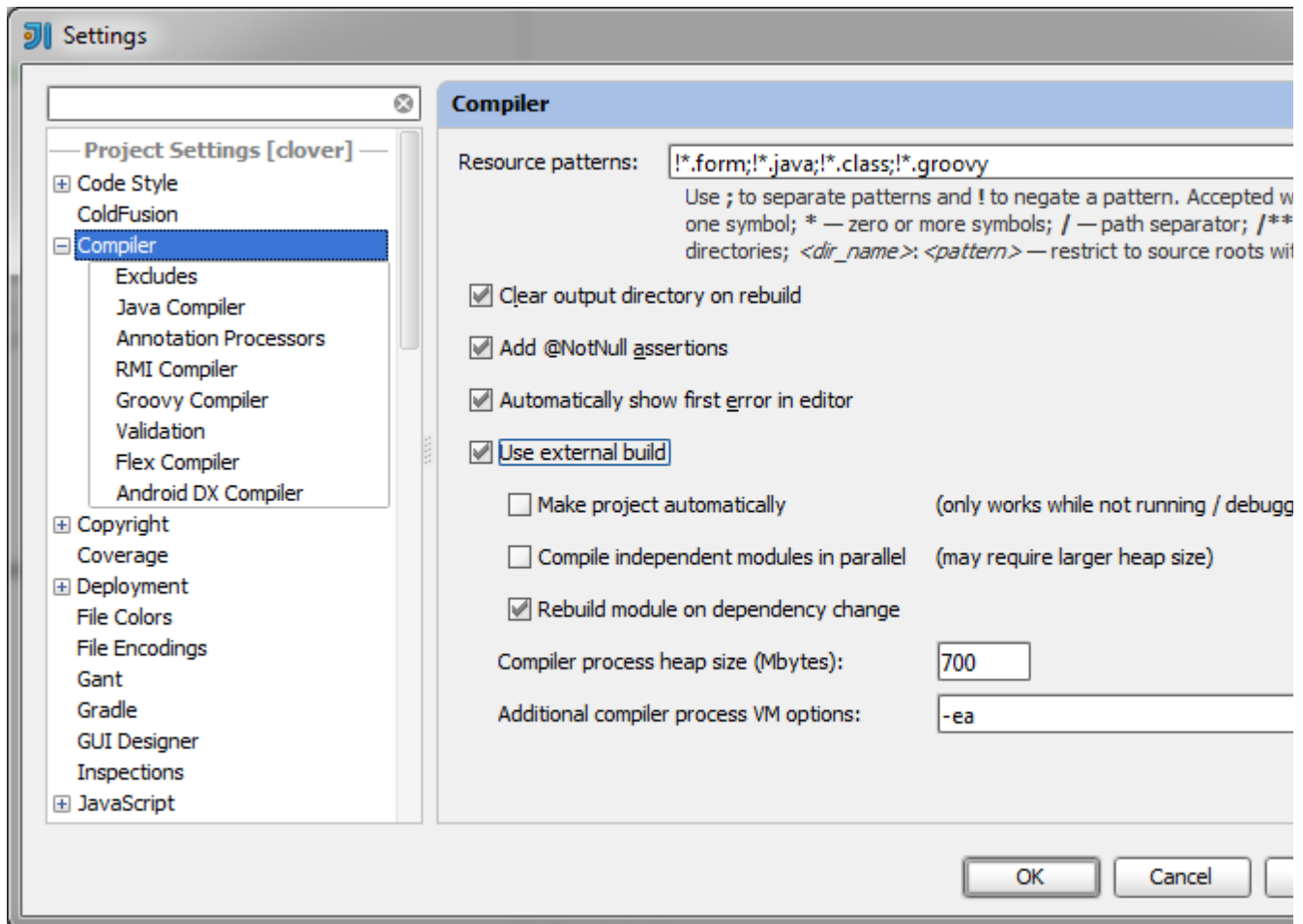
JetBrains IDEA 12.0 has a new compiler feature named "external build", which is enabled by default in project settings (*File > Settings > Project Settings > Compiler > "Use external build"* toggle). The IDEA 12 is supported since Clover version 3.1.8, however the support for "external build" feature is available since Clover version 3.1.12 and IDEA 12.1.1. For older versions, i.e. the combination of Clover 3.1.8-3.1.12 and IDEA 12.0.0-12.1.0 please keep the "external build" feature disabled.

### **i** IntelliJ IDEA 13

In the IDEA 13.0 the "external build" feature has become default setting and thus there's no 'Use external build' toggle available.

## Enabling the "external build" feature

Open the *File > Settings > Project Settings > Compiler* page.



Use the following state for toggles:

- *Use external build* - it can be either enabled (the new IDEA12 feature is used) or disabled (the "classic" IDEA build is used)
- *Make project automatically* - it's strongly recommended to disable this option; enabling it causes frequent compilation and faster growth of Clover database, which might affect performance
- *Compile independent modules in parallel* - it must be disabled; Clover does not support parallel build; follow the [CLOV-1293](#) for future updates

## Known Issues

- If you are using the Maven build tool, you should avoid using the same IntelliJ output directory as Maven does. As Maven uses the `target/classes` and `target/test-classes` directories, avoid specifying these ones. The `clover.db` location for IntelliJ should also be distinct from that used by Maven.

## Uninstalling the Plugin

The easiest way to uninstall it is via **'File > Settings > IDE > Plugins'**. Just select the Clover IDEA Plugin from the list and click **'Uninstall Plugin'**. The removal will take place after you restart IDEA.

To uninstall Clover-for-IDEA manually:

1. Shut down any running instances of IDEA.
2. Delete the 'clover-idea.jar' file from its installation directory, either `IDEA_HOME/config/plugins` (all platforms), `IDEA_HOME/plugins` (all platforms except Mac OS X) or `USER_HOME/Library/Application Support/IntelliJIDEA70` (Mac OS X only)
3. Restart IDEA.

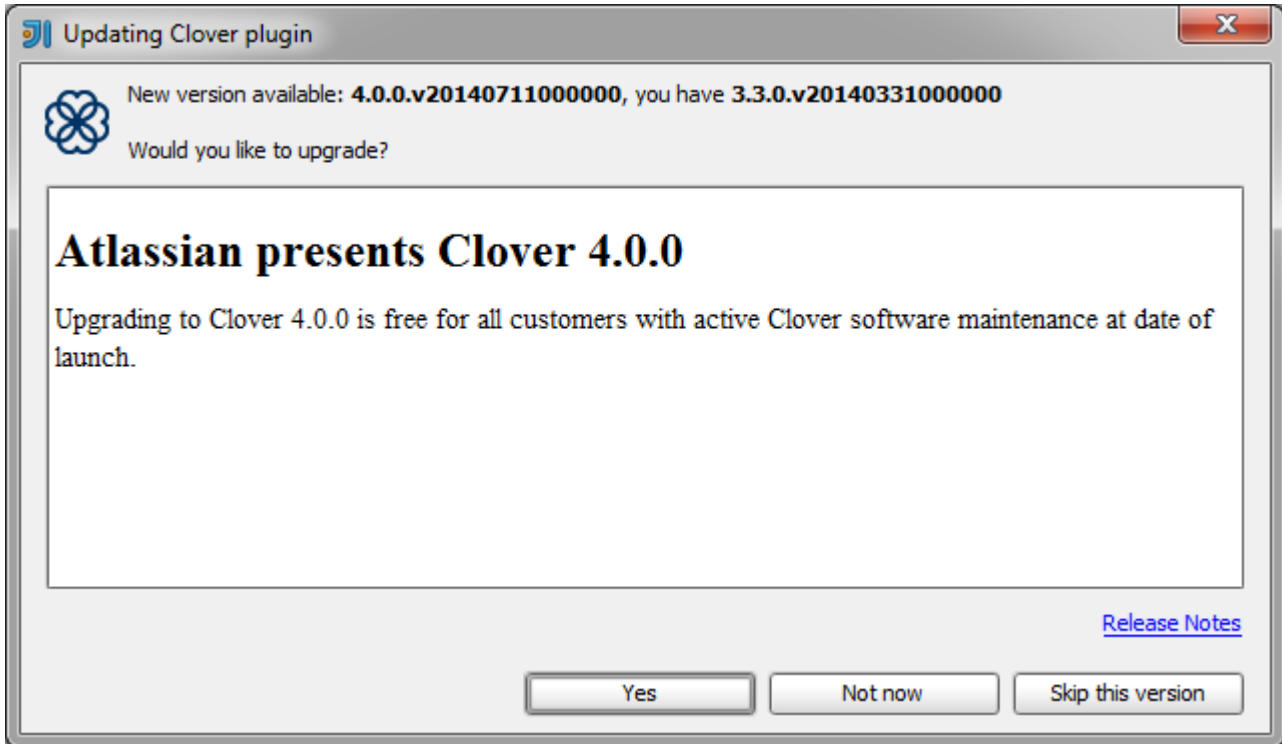
## Clover-for-IDEA Upgrade Guide

## General instructions

By default Clover automatically checks for updates. When new version becomes available, the Clover icon will blink in the status bar. In order to update:

1. Click on the blinking Clover icon.
2. An update dialog will pop up. Click 'Yes'.
3. Restart IDEA after installation.

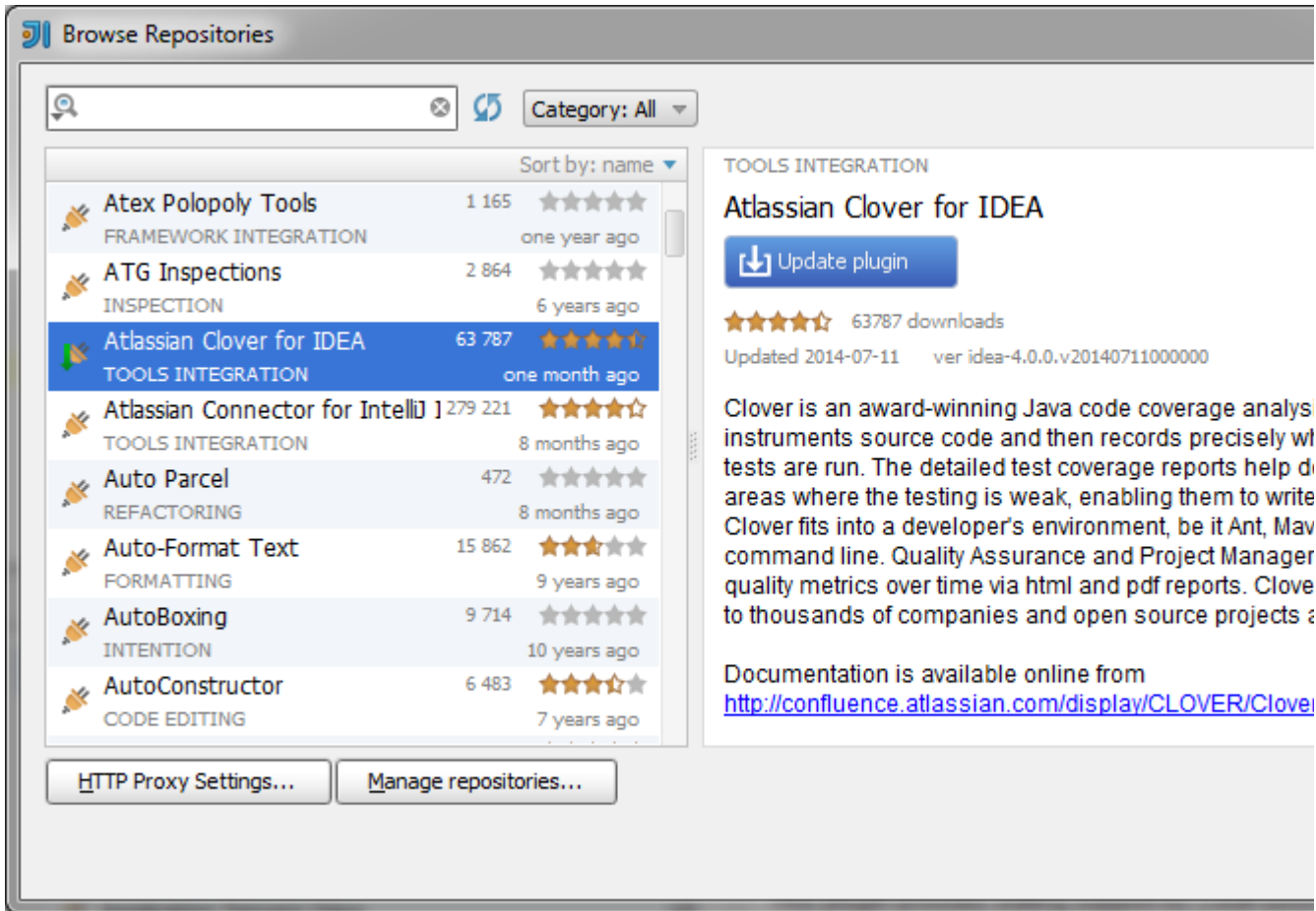
Screen shot: *the update dialog.*



In order to update the plugin manually:

1. Open 'File > Settings > IDE Settings > Plugins > Browse repositories'.
2. Find 'Atlassian Clover for IDEA' and click 'Update plugin':
3. Restart IDEA after installation.

Screen shot: *the 'Browse Repositories' dialog.*



### Upgrading from specific releases

Please see the [Clover Release Notes](#) and the [Clover-for-IDEA Changelog](#) for version-specific upgrade instructions.

### Clover-for-IDEA Changelog

Please also refer to the [Clover-for-Ant Changelog](#).

### Clover-for-IDEA Changelog

The changes for the latest versions are as follows:

#### Changes in Clover-for-IDEA 4.0.0

July 11, 2014

This is a feature release with new HTML report with the ADG (Atlassian Design Guidelines) look. See the [Clover 4.0 Release Notes](#).

#### Implemented features and fixes

T	Key	Summary	P
	CLOV-1345	Apply ADG in the HTML report	

1 issue

See also the [Clover-for-Ant Changelog](#).

#### Known major bugs

T	Key	Summary	P	Fix Version/s	Status
No issues found					







## Changes in Clover-for-IDEA 3.3.0

**March 31, 2014**

This is a feature release with a dedicated support for Spock framework and JUnit4 Parameterized Tests.

Please note that Clover-for-IDEA plug-in does not support Groovy in the IDE (see the [Supported Platforms](#) page), so you'll have to use Clover's Ant, Maven or Grails plug-in in order to instrument Spock tests.

### Implemented features and fixes

T	Key	Summary	P
	CLOV-1462	ClassNotFoundException when running tests in IDEA 13.1 RC with Clover enabled	
	CLOV-1382	Add lambda toggle to report wizards in Eclipse and IDEA	
	CLOV-1441	Clover plugin doesn't load on IDEA 13 Startup	

3 issues

See also the [Clover-for-Ant Changelog](#).

### Known major bugs

T	Key	Summary	P	Fix Version/s	Status
No issues found					

### Older versions

Looking for older versions? See [Clover-for-IDEA Changelog](#) for Clover 3.2.



## Changes in Clover-for-IDEA 4.0.0

### Changes in Clover-for-IDEA 4.0.0

**July 11, 2014**

This is a feature release with new HTML report with the ADG (Atlassian Design Guidelines) look. See the [Clover 4.0 Release Notes](#).

### Implemented features and fixes

T	Key	Summary	P
	CLOV-1345	Apply ADG in the HTML report	

1 issue

See also the [Clover-for-Ant Changelog](#).

### Known major bugs

T	Key	Summary	P	Fix Version/s	Status
---	-----	---------	---	---------------	--------

No issues found

## Changes in Clover-for-IDEA 3.3.0







### Changes in Clover-for-IDEA 3.3.0

**March 31, 2014**

This is a feature release with a dedicated support for Spock framework and JUnit4 Parameterized Tests.

Please note that Clover-for-IDEA plug-in does not support Groovy in the IDE (see the [Supported Platforms](#) page), so you'll have to use Clover's Ant, Maven or Grails plug-in in order to instrument Spock tests.

#### Implemented features and fixes

T	Key	Summary	P
	CLOV-1462	ClassNotFoundException when running tests in IDEA 13.1 RC with Clover enabled	
	CLOV-1382	Add lambda toggle to report wizards in Eclipse and IDEA	
	CLOV-1441	Clover plugin doesn't load on IDEA 13 Startup	

3 issues

See also the [Clover-for-Ant Changelog](#).

#### Known major bugs

T	Key	Summary	P	Fix Version/s	Status
---	-----	---------	---	---------------	--------

No issues found

## Clover-for-IDEA Glossary of Terms

### Block Contexts

Refers to common Java coding constructs or idioms such as the body of if statements; static initialiser blocks; or property style methods. These are pre-defined by Clover.

### Coverage (view)

The Coverage view allows you to view and control Clover's instrumentation of your Java projects, and shows you the coverage statistics for each project based on recent test runs or application runs.

### Coverage Cloud (report)

A Clover report visualisation that prints class names to the screen, coloured to show their level of code coverage and scaled in size to illustrate their complexity.

### Coverage Report

Coverage reports are generated by Clover as PDF, HTML or XML, showing Clover's output in a readily digestible format for the user.

### Coverage Treemap (report)

A Clover report visualisation that shows packages and classes as coloured squares. The square's respective colour indicates the level of code coverage and they are scaled in size to illustrate their complexity (largest is most complex).

### Flush Policy (setting)

The Flush Policy controls when Clover writes coverage data to disk as your application runs.

### Initstring (setting)

This controls where the Clover plugin stores (and looks for) the coverage database.

### Instrumentation

In order to track the code coverage of your projects, Clover must insert special code into your programs at compilation time. This special code is collectively called instrumentation.

### Project (view)

The Project view is a navigation side bar in IDEA that allows you to view the project tree and drill down into elements of the project structure visually, in order to select or edit them.

### Summary Panel

The Summary Panel is part of Cloverage view with a set of metrics that are displayed alongside the tree for the selected project, package, file, class or method in the tree.

### Test Runs (view)

The Test Run Explorer view, like several popular plugins such as the JUnit Plugin or TestNG Plugin, lets you explore your recently run tests - showing whether they passed or failed, their duration and any error messages that they generated. Clover-for-IDEA takes this one step further and allows you to explore the code coverage caused by an individual test, a test class, a package or even your entire project.

## Clover-for-IDEA FAQ

**Clover IDEA Plugin FAQ**

- [I've run my tests, but coverage information does not show in IDEA](#)
- [What does enabling Instrument Test Source Folders do?](#)
- [Where does IDEA write its log file?](#)

## Clover-for-Grails



## Clover-for-Grails Documentation

### What is Clover-for-Grails?

Clover-for-Grails integrates the industry-leading code coverage tool, [Atlassia n Clover](#) with the Grails web application development framework. Clover-for-Grails allows you to easily measure the coverage of your unit tests, enabling targeted work in unit testing — resulting in stability and enhanced quality code with maximal efficiency of effort.

### Getting Started with Clover-for-Grails

[Download Clover-for-Grails](#)

[Installation Guide](#)

[Change log for Clover-for-Grails](#)

### Using Clover for Grails

[Installation and Upgrade Guide](#)

[Quick Start Guide](#)

[User's Guide](#)

### Resources and Support











[Atlassian Answers](#)






[Support](#)

### Offline Documentation

You can download the Clover documentation in PDF, HTML or XML format.

## Recently Updated

-  [Clover Road Map](#)  
 Aug 12, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Upgrading third party libraries](#)  
 Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Updating optimization snapshot file](#)  
 Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Hacking Clover](#)  
 Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Part 4 - Test Optimization Tutorial](#)  
 Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Part 3 - Automating Coverage Checks](#)  
 Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Part 2 - Historical Reporting](#)  
 Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Part 1 - Measuring Coverage](#)  
 Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Clover 4.0 Release Notes](#)  
 Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [A side-by-side comparison of the Classic and the ADG HTML report](#)  
 Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)

-  [Clover Release Notes](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Sonar Clover Plugin](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Clover Command Line Tools](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Clover-for-Grails Changelog](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)
-  [Configuring method context filters](#)  
Aug 11, 2014 • updated by Marek Parfianowicz [Atlassian] • [view change](#)

## Clover-for-Grails Quick Start Guide

### On this page:


- [Generating Clover Reports](#)
  - [Install the Clover-for-Grails plugin](#)
  - [Generating a basic Clover coverage report](#)
  - [Passing the location of your clover.license file to the grails command line](#)
- [Troubleshooting](#)
- [Further reading](#)

## Generating Clover Reports

### Install the Clover-for-Grails plugin

Install the Clover-for-Grails plugin by running the following Grails command in the root of your Grails project directory.


```
grails install-plugin clover
```

 The 'grails install-plugin' command has been removed in Grails 2.3.5 upwards. In such case we recommend adding the Clover plugin to your project's BuildConfig.groovy or pom.xml. For more installation options, please see [Clover-for-Grails Installation Guide](#).

### Generating a basic Clover coverage report

To generate a basic Clover code coverage report, you need to add the Clover option `-clover.on` to the `grails test-app` command line target for running unit tests against your Grails project.

```
grails test-app -clover.on -clover.view
```

 Adding the Clover option `-clover.view` to this Grails command makes the report open in a browser window immediately after generation. If you omit this command line option, Clover will generate a report that you can then open manually.

### Passing the location of your clover.license file to the grails command line

If you have not placed your `clover.license` file within your Grails project or user home directory (as indicated in the [Installation Guide](#)), you can pass the license file's location to the `grails` command line by adding the Clover option `-clover.license.path=/path/to/clover.license`:

```
grails test-app -clover.on -clover.license.path=/path/to/clover.license
```

❗ On Windows platform you must put the entire `"-clover.license.path=c:\path\to\clover.license"` in double quotes (it's a Grails "feature" which tries to evaluate a non-quoted value to boolean instead of string).

## Troubleshooting

If you find that Clover-for-Grails runs out of memory, try increasing the Grails PermGen allocation by either setting the `JAVA_OPTS` environment variable:

### Linux/UNIX/Mac OS X:

```
export JAVA_OPTS="-XX:MaxPermSize=192m"
```

### Windows:

```
set JAVA_OPTS="-XX:MaxPermSize=192m"
```

Alternatively, you can define this variable in the `startGrails` (Linux/UNIX/Mac OS X) or `startGrails.bat` (Windows) script in the `<Grails Home Directory>/bin` directory.

## Further reading

- [Configuration options](#)
- [Clover-for-Grails User's Guide](#)

## Clover-for-Grails User's Guide

### Configuration options

Clover-for-Grails supports the configuration options outlined in the code sample below. All of these configuration options are defined in a single `clover {}` code block, which itself is defined within the Groovy build configuration file (`BuildConfig.groovy`) of your Grails project.

ℹ The `BuildConfig.groovy` file is located in the `grails-app/conf` subdirectory of your Grails project's root directory.

```

clover {

  on = true|false // a boolean value indicating whether or not clover is enabled

  optimize = true|false // whether test optimization is enabled
                    // (by setting optimize=true there's not need to set
on=true)

  license.path = "/path/to/clover.license" // the location of the clover license
file,
                    // if not in one of the default locations

  debug = true|false // a boolean to toggle debugging on or off

  verbose = true|false // a boolean to toggle verbose output on or off; can be
overridden by debug=true; since 3.1.6

  initstring = "" // the location to use for Clover to write out its database

  instrumentLambda = "none|expression|block|all" // whether to instrument Java 8
lambda functions,
                    // see the <clover-setup> Ant task for more details

  forceClean = true|false // whether to clean build directory (to force compilation
of classes with Clover),
                    // true by default

  srcDirs = [] // an array of Strings of source directories to including in
instrumentation

  includes = [] // an array of String Ant Glob Patterns to include for
instrumentation

  excludes = [] // an array of String Ant Glob Patterns to exclude for
instrumentation

  setuptask = {} // Gant script to be called instead of the default clover-setup

  reporttask = {} // Gant script to be called when tests have finished

  historypointtask = {} // Gant script to be called when clover history point has to
be created

  reportStyle = "adg|classic" // style of the report - since Clover 4.0

  json = true|false // a boolean to enable generation of json output in report;
used if custom 'reporttask' script is not defined

  title = "" // a report title

  view = true|false // launch generated report in a web browser

}

```

More details for:

- [setuptask](#)
- [reporttask](#)

## Advanced report configuration

Define a `reporttask` closure in your `clover {}` code block to configure advanced report generation options for

your Groovy project's build process. Here, you can define various attributes and elements of the `clover-report` task. In fact, any [Clover Ant tasks](#) and their attributes and elements may be used in this closure.

**i** You would not normally include `clover-setup` tasks in the `reporttask` closure because the latter is executed after the `clover-setup` tasks have executed.

The `reporttask` closure is passed the following parameters:

- `ant` — an instance of a `org.codehaus.gant.GantBuilder`
- `binding` — the groovy binding for accessing project variables
- `plugin` — the clover grails plugin that invoked this closure

The syntax used to define your `clover-report` tasks or any other valid Ant task in the `clover {}` code block is [Gant](#).

#### **Example: custom reports in PDF, HTML, XML and JSON formats**

The following example `clover {}` code block and `reporttask` definition in your `BuildConfig.groovy` file will:

- generate a Clover report in both PDF and HTML formats and
- place the results in the `build/clover/report` subdirectory of your Grails project directory.

```

clover {

    // reports.dir defines the location of your Clover report output
    // within your Grails project.

    reports.dir = "build/clover/report"

    // The reporttask closure is invoked after all tests have run.

    reporttask = { ant, binding, plugin ->

        ant.mkdir(dir: "${clover.reports.dir}")

        ant.'clover-report' {

            ant.current(outfile: "${clover.reports.dir}/clover.pdf", summary: true) {
                format(type: "pdf")
            }

            ant.current(outfile: "${clover.reports.dir}") {
                format(type: "html")
                ant.columns {
                    lineCount()
                    complexity()
                    filteredElements(format:"bar")
                    uncoveredElements(format: "raw")
                    totalElements(format: "raw")
                    totalPercentageCovered()
                }
            }

            ant.current(outfile: "${clover.reports.dir}/clover.xml") {
                format(type: "xml")
            }

            ant.current(outfile: "${clover.reports.dir}") {
                format(type: "json")
            }

        }

        plugin.launchReport(clover.reports.dir)

    }
}

```

The *reporttask* closure could be also used to run other post-build activities, like clover-check or clover-log, for example.

**Example: failing build if code coverage is too low**

```

reporttask = { ant, binding, self ->
    ant.'clover-check'(target: "80%", haltOnFailure: true) { }
}

```

## Advanced setup configuration

Define a *setuptask* 'closure' in your `clover {}` code block to configure advanced options for your Groovy

project's build processes. Here, you can define various attributes and elements of the `clover-setup` task.

The `setuptask` closure is passed the following parameters:

- `ant` — an instance of a `org.codehaus.gant.GantBuilder`
- `binding` — the groovy binding for accessing project variables
- `plugin` — the clover grails plugin that invoked this closure

The syntax used to define your clover-setup tasks in the `clover {}` code block is [Gant](#).



Please be aware that some attributes and sub-elements of the `clover-setup` task do not support Groovy. Therefore, if your Grails project makes substantial use of Groovy code (as opposed to pure Java code, which is likely to be the case), not all features of the `clover-setup` task will be available to you. Refer to the `clover-setup` topic for details.

#### **Example: using custom clover.db location and set of files to be instrumented**

```
clover {
    // Example setuptask closure that will be invoked to configure clover.
    // Any Clover initialisation tasks should be defined here.
    // All attributes on the ant clover-setup task, which are
    // supported by your source code, can be defined here.

    setuptask = { ant, binding, plugin ->
        ant.'clover-setup'(initstring:
"${binding.projectWorkDir}/clover/custom/clover.db") {
            ant.fileset(dir: "grails-app", includes: "**/*.groovy") { }
        }
    }
}
```

#### **Example: enabling distributed coverage**

```
clover {
    setuptask = { ant, binding, plugin ->
        ant.'clover-setup'(initstring:
"${binding.projectWorkDir}/clover/db/clover.db",
            tmpDir: "${binding.projectWorkDir}/clover/tmp") {
            distributedCoverage(port: 7777, host: "localhost", timeout: 5000,
numClients: 0)
        }
    }
}
```

#### **Example: using shared coverage recorder**

Use the Shared Coverage Recorder only in case when you have performance problem related with creation of thousands of coverage recording files in `clover.db` directory. See [Coverage Recorders](#) for more details.

```
clover {
  setuptask = { ant, binding, plugin ->
    ant.delete(dir: "target/clover/db")
    ant.delete(dir: "target/clover/tmp")
    ant.'clover-setup'(initstring: "target/clover/db/clover.db",
      tmpDir: "target/clover/tmp") {
      ant.profiles {
        ant.profile(name: "default", coverageRecorder: "SHARED")
      }
    }
  }
  reporttask = { ant, binding, plugin ->
    ant.delete(dir: "target/clover/report")
    ant.'clover-report'(initstring: "target/clover/db/clover.db") {
      ant.current(outfile: "target/clover/report/clover.xml") {
        ant.format(type: "xml")
      }
      ant.current(outfile: "target/clover/report") {
        ant.format(type: "html")
      }
    }
  }
}
```

## Configuring method context filters

You can filter-out less important code sections using context filters for: methods (groovy+java), statements (java) and code blocks (java). See [Using Coverage Contexts](#) for more details.

A report can use custom set of columns - see [<clover-report> / <columns>](#) tag documentation.

### Example:




```
setuptask = { ant, binding, plugin ->
    ant.'clover-setup'(initstring: "target/clover/db/clover.db") {
        ant.fileset(dir: "grails-app") { }
        ant.fileset(dir: "src/groovy") { }
        ant.fileset(dir: "src/java") { }
        ant.fileset(dir: "test") { }

        // Ignore private constructors in Groovy - using normalized constructor
        signature
        ant.methodContext(name: 'private-constructors', regexp: 'private void
<init>\\(\\(\\)') { }

        // Ignore logging statements. Note that statement context is currently
        supported only for Java.
        ant.statementContext(name: 'log',    regexp: '^log\\..*')
        ant.statementContext(name: 'if-log', regexp: '^if.*\\(log\\..*')
    }
}

reporttask = { ant, binding, plugin ->
    ant.'clover-report'(initstring: "target/clover/db/clover.db") {
        ant.current(outfile: "target/clover/html-report", title: "API Report") {
            // NOTE: you have to add the 'filter' attribute with a list of filters from
            clover-setup task
            ant.format(type: "html", filter: "private-constructors,log,if-log") { }
            ant.fileset(dir: "grails-app") { }
            ant.fileset(dir: "src/groovy") { }
            ant.fileset(dir: "src/java") { }
            ant.fileset(dir: "test") { }
            ant.columns {
                lineCount()
                filteredElements()
                uncoveredElements()
                totalPercentageCovered()
            }
        }
    }
}
```

## Test Optimization with Clover-for-Grails

 This feature is available since **Clover-for-Grails 3.1.10.1**.

Follow the steps in this document to set up Clover's Test Optimization, which allows targeted testing of only the code which has changed since the last build. This page contains the basic steps for adding Clover's Test Optimization to a Grails application.

### Command line quick start

The quickest possible way to start using Test Optimization in Clover-for-Grails is to run tests with the **-clover.optimize** option, for instance:

```
grails test-app -clover.optimize
```

The **-clover.optimize** does the following:

- enables Clover instrumentation
- disables clean (note that **-clover.on** performs full clean by default, unless the **-clover.forceClean=false** is used)
- analyses which application and test classes were modified since the last build and selects appropriate set of tests
  - it's achieved by overwriting value of the `testTargetPatterns` variable from `_GrailsTest.groovy` script
- stores optimization snapshot after test phase

By default, the snapshot file gets saved to "**`\${projectWorkDir}/clover.snapshot`**" (the ``${projectWorkDir}`` is `"~/grails/X.X.X/projects/project_name"` by default).

This file is needed to optimize subsequent builds. You can also specify an alternative location in **clover.snapshotLocation**, which can be defined in `BuildConfig.groovy` or passed from command line, for instance:

```
clover {
    snapshotLocation = "/path/to/clover.snapshot"
}
```

### Test Optimization in action

For the first time you should run a full build in order to make sure that the whole code is instrumented by Clover. You can run:

```
grails clean
grails test-app -clover.optimize
```

or

```
grails test-app -clover.on    # clover.on forces full clean
grails test-app -clover.optimize
```

The first time Clover Test Optimization is used a full test run will be done. You should see the following log message appear in the console:

```

C:\Windows\System32\cmd.exe
c:\Work\clover\grails\testcases\petclinic203>grails test-app -clover.optimize
! Environment set to test....
Clover: Clover is enabled. Configuration: [license:[path:C:\Users\Marek\clover.license], optin
se:[:], on:[:]]
Clover: Using Clover license path: C:\Users\Marek\clover.license
! Environment set to test.....
Clover: using default clover-setup configuration.
Clover:
    directories: [src/java, src/groovy, test/unit, test/integration, grails-app]
    includes:    [**/*.groovy, **/*.java]
    excludes    [**/conf/**, **/plugins/**]
! Compiling 19 source files
! Compiling 3 source files
! Compiling 3 source files.
Clover: Test source compilation phase ended
Clover: Configuring test optimization with options OptimizationOptions{enabled=true, minimize=
bug=true, maxCompilesBeforeStaleSnapshot=10, optimizableName='test', initString='C:\Users\Mare
etclinic203\clover\dh\clover_dh', snapshot=C:\Users\Marek\grails\2.0.3\projects\petclinic203
Clover: Test Optimization selected 3 out of 3 tests for execution
! Completed 8 unit tests, 0 failed in 1352ms
! Compiling 1 source files.
Clover: Test source compilation phase ended
Clover: Configuring test optimization with options OptimizationOptions{enabled=true, minimize=
bug=true, maxCompilesBeforeStaleSnapshot=10, optimizableName='test', initString='C:\Users\Mare
etclinic203\clover\dh\clover_dh', snapshot=C:\Users\Marek\grails\2.0.3\projects\petclinic203
Clover: Test Optimization selected 1 out of 1 tests for execution
! Completed 1 integration test, 0 failed in 127ms
! Tests FINISHED - view reports in C:\Work\clover\grails\testcases\petclinic203\target\test-rep
Clover: Tests ended. Generating reports
Clover: Generating report using default 'clover-report' task
Clover: Saving optimization snapshot

```

If you then rerun the build, without modifying any source files (and ensuring the snapshot file is not deleted) you should see the following:

```

C:\Windows\System32\cmd.exe
c:\Work\clover\grails\testcases\petclinic203>grails test-app -clover.optimize
! Environment set to test....
Clover: Clover is enabled. Configuration: [license:[path:C:\Users\Marek\clover.license], optin
se:[:], on:[:]]
Clover: Using Clover license path: C:\Users\Marek\clover.license
! Environment set to test.....
Clover: using default clover-setup configuration.
Clover:
    directories: [src/java, src/groovy, test/unit, test/integration, grails-app]
    includes:    [**/*.groovy, **/*.java]
    excludes    [**/conf/**, **/plugins/**]
! Compiling 1 source files.....
Clover: Test source compilation phase ended
Clover: Configuring test optimization with options OptimizationOptions{enabled=true, minimize=
bug=true, maxCompilesBeforeStaleSnapshot=10, optimizableName='test', initString='C:\Users\Mare
etclinic203\clover\dh\clover_dh', snapshot=C:\Users\Marek\grails\2.0.3\projects\petclinic203
Clover: Test Optimization selected 0 out of 3 tests for execution
! Compiling 1 source files.....
Clover: Test source compilation phase ended
Clover: Configuring test optimization with options OptimizationOptions{enabled=true, minimize=
bug=true, maxCompilesBeforeStaleSnapshot=10, optimizableName='test', initString='C:\Users\Mare
etclinic203\clover\dh\clover_dh', snapshot=C:\Users\Marek\grails\2.0.3\projects\petclinic203
Clover: Test Optimization selected 0 out of 1 tests for execution
! Tests FINISHED - view reports in C:\Work\clover\grails\testcases\petclinic203\target\test-rep
Clover: Tests ended. Generating reports
Clover: Generating report using default 'clover-report' task
Clover: Saving optimization snapshot

```

If a source file (application or test class) is modified in any way (including whitespace changes), and you re-run the build, only test cases that cover the modified file will be run:

```

C:\Windows\System32\cmd.exe
c:\Work\clover\grails\testcases\petclinic203>grails test-app -clover.optimize
! Environment set to test....
Clover: Clover is enabled. Configuration: [license:[path:C:\Users\Marek\clover.license], optin
se:[:], on:[:]]
Clover: Using Clover license path: C:\Users\Marek\clover.license
! Environment set to test.....
Clover: using default clover-setup configuration.
Clover:
    directories: [src/java, src/groovy, test/unit, test/integration, grails-app]
    includes:    [**/*.groovy, **/*.java]
    excludes:   [**/conf/**, **/plugins/**]
! Compiling 2 source files.....
Clover: test source compilation phase ended
Clover: Configuring test optimization with options OptimizationOptions{enabled=true, minimize=
bug=true, maxCompilesBeforeStaleSnapshot=10, optimizableName='test', initString='C:\Users\Mare
etclinic203\clover\db\clover.db', snapshot=C:\Users\Marek\grails\2.0.3\projects\petclinic203\
Clover: Test Optimization selected 1 out of 3 tests for execution
! Completed 6 unit tests, 0 failed in 1199ms
! Compiling 1 source files.....
Clover: Test source compilation phase ended
Clover: Configuring test optimization with options OptimizationOptions{enabled=true, minimize=
bug=true, maxCompilesBeforeStaleSnapshot=10, optimizableName='test', initString='C:\Users\Mare
etclinic203\clover/db\clover.db', snapshot=C:\Users\Marek\grails\2.0.3\projects\petclinic203\
Clover: Test Optimization selected 0 out of 1 tests for execution
! Tests PASSED - view reports in C:\Work\clover\grails\testcases\petclinic203\target\test-rep
Clover: Tests ended. Generating reports
Clover: Generating report using default 'clover-report' task
Clover: Saving optimization snapshot

```

By default, the same snapshot file is updated for 10 consecutive builds. On the 10th build, the snapshot file is deleted and recreated, which triggers a full test run.

You can also delete this file manually to force a full test run (note that if snapshot file is stored in default location, the 'grails clean' command does not remove this file).

## References

- [Overview of Test Optimization](#)
- [Test Optimization Technical Details](#)

## Clover-for-Grails Installation Guide

This page provides instructions for all available Clover-for-Grails plug-in installation options.

### **i** Clover License for Clover-for-Grails

We recommend purchasing Clover Server license for Grails-based projects.

*A reason is that in order to see coverage results for a project built and tested from a command line, an HTML report must be generated (and report generation is not available in the Clover Desktop edition).*

- [Overview](#)
- [Installation of Clover for Grails plug-in](#)
  - [Using the BuildConfig.groovy](#)
  - [Using the pom.xml](#)
  - [Using the 'install-plugin' command](#)
- [Installing the Clover license file](#)
- [Related Topics](#)

## Overview

The documentation below assumes that you have already installed Grails and have configured your PATH enviro

ment variable to point to the `bin` directory of your Grails installation.

### Installing Clover-for-Grails

There are several ways to install Clover-for-Grails:

- Declaring dependency in `BuildConfig.groovy`
- Declaring dependency in `pom.xml`
- Using the 'install-plugin' command
  - installing from the grails central repository
  - installing from the given web address
  - installing from the downloaded installation file

### Installing the Clover license

Clover requires a license file in order to run. Once you have installed the Clover-for-Grails plugin, you will need to download and install the Clover license file `clover.license`. You can generate a 30-day evaluation Clover license file by logging in to <https://my.atlassian.com> and following the instructions on the site. After 30 days, you need to purchase a commercial Clover license file from this site to continue running Clover. Refer to the [instructions below](#) for installing your Clover license file.

### Upgrading Clover-for-Grails

Upgrading the Clover-for-Grails plugin is also very easy, as [indicated here](#).

## Installation of Clover for Grails plug-in

### Using the BuildConfig.groovy

Edit your `grails-app/conf/BuildConfig.groovy` file and add a section like below:

#### grails-app/conf/BuildConfig.groovy

```
grails.project.dependency.resolution = {
  plugins {
    compile "org.grails.plugins:clover:4.0.0"
  }
  // For Grails 2.2 and later you have to add a dependency to a Clover core:
  dependencies {
    compile "com.atlassian.clover:clover:4.0.0" // com.cenqua.clover:clover
  }
  for 3.x.x
  {
    // or use a legacy dependency resolution mechanism:
    // legacyResolve true
  }
}
```

### Using the pom.xml

**Grails 2.1** has introduced new dependency management based on Maven's `pom.xml` instead of `BuildConfig.groovy` file. In order to use this, declare `"pom true"` in `BuildConfig.groovy`:

#### grails-app/conf/BuildConfig.groovy

```
grails.project.dependency.resolution = {
  pom true
}
```

and add dependencies to `org.grails.plugins:clover` (and `com.atlassian.clover:clover`) in `pom.xml`:


**pom.xml**

```
<dependency>
  <groupId>org.grails.plugins</groupId>
  <artifactId>clover</artifactId>
  <version>4.0.0</version>
  <scope>compile</scope>
  <type>zip</type>
</dependency>
<!-- For *Grails 2.2* or later you must also add a dependency to the Clover Core
-->
<dependency>
  <groupId>com.atlassian.clover</groupId> <!-- com.cenqua.clover for Clover 3.x
-->
  <artifactId>clover</artifactId>
  <version>4.0.0</version>
  <scope>compile</scope>
  <type>jar</type>
</dependency>
```

## Using the 'install-plugin' command

```
grails install-plugin clover 4.0.0
```

 The 'install-plugin' command has been deprecated in Grails 2.1. It's recommended to use BuildConfig.groovy or pom.xml.

 If you're using Grails 1.3.7 and earlier, you might need to add a following entry to your list of repositories.

```
grailsRepo "http://plugins.grails.org"
```


 If you experience problems using this method, try installing the plugin directly from its web address:

```
grails install-plugin
http://plugins.grails.org/grails-clover/tags/RELEASE_x_y_z/grails-clover-x.y.z.zip
```

Where x, y and z (optional) refer to the latest version of the Clover-for-Grails plugin to be installed.

It's also possible to install it from a downloaded file:

```
grails install-plugin /path/to/grails-clover-x.y.zip
```


 Due to the nature of Grails' plugin installation architecture, you will need to install the Clover-for-Grails plugin into each Grails project whose source code you wish to test with Clover.

## Installing the Clover license file

Once you have obtained your 30 day evaluation or commercial `clover.license` file, you need to install it so that your Clover-for-Grails plugin can acknowledge its existence.

You have the following options for installing the Clover license file `clover.license`:

Move or copy the <code>clover.license</code> file into this location:	Notes about this option:
The root directory of your Grails project. <ul style="list-style-type: none"> <li>For e.g. <code>&lt;Grails Installation Directory&gt;/samples/petclinic</code></li> </ul>	This option installs the Clover license file to this Grails project only. You will need to install this file into each of your other Grails projects with the Grails-for-Clover plugin too.
The <code>etc</code> directory within your Grails project. <ul style="list-style-type: none"> <li>For e.g. <code>&lt;Grails Installation Directory&gt;/samples/petclinic/etc</code></li> </ul>	This is similar to the previous option but 'hides' the <code>clover.license</code> file from the root directory of your Grails project.
Your user home directory. <ul style="list-style-type: none"> <li>Linux/UNIX/Mac OS X:               <ul style="list-style-type: none"> <li>e.g. <code>/home/alice/</code> or <code>~</code></li> </ul> </li> <li>Windows:               <ul style="list-style-type: none"> <li>e.g. <code>C:\Users\Alice</code></li> </ul> </li> </ul>	This will prevent you having to install the Clover license file into every Grails project on your computer as the license file will apply to every Grails project developed on your computer and login account.

 You can also place the `clover.license` anywhere else that's accessible to the Clover-for-Grails plugin and reference it from either:

- The [Grails command line](#)
- The [BuildConfig.groovy](#) file.

## Related Topics

- [Clover-for-Grails Quick Start Guide](#)
- [does-clover-grails-plugin-work-with-grails-2-2-1](#)

## Clover-for-Grails Upgrade Guide

### Upgrading the Clover-for-Grails plugin

Upgrading the Clover-for-Grails plugin is easy. All you need to do is to 're-install' the Clover-for-Grails plugin by following one of the three methods above. During the installation process, Grails will prompt you with a message similar to the following:

```
You currently already have a version of the plugin installed
[clover-x.y].
Do you want to upgrade this version? (y, n)
```

Type 'y', then press enter, then Grails will remove the old version of the Clover-for-Grails plugin and replace it with the newer version.

## Upgrading Grails framework

Clover-for-Grails only supports the version of Grails indicated on the [Supported Platforms](#) page. Hence, if your Grails project has been developed using an older version of Grails, you will need to:

1. Download and install a newer version of Grails that Clover supports.
2. Update your `PATH` environment variable to point to the `bin` directory of the newly installed version of Grails.
3. Change directory to the root of your Grails project directory.
4. Run the command:

```
grails upgrade
```

This upgrades your Grails project to the new version of Grails that you just installed.

## About Clover-for-Grails

### Overview

The Clover-for-Grails plugin allows you to produce Clover code coverage reports from the Grails web application development framework. It provides detailed information to highlight areas of low coverage in your project, helping to guide your unit-testing activities.

### Open Source status

This plugin is open source, under the Apache 2.0 license. Source code is available here: <https://bitbucket.org/atlassian/grails-clover-plugin>

### License

The plugin includes a built in 30-day evaluation license. You can buy full license here: <https://my.atlassian.com/purchase>

### Support

- [Atlassian Answers](#) (the 'clover' tag) is the best place to search first
- if you still got stuck, feel free to raise a support ticket at [Atlassian Support](#) (under the "Clover Support" project)
- if you've found a bug or would like to raise a feature request, create new issue in [Clover's issue tracker](#)

### Downloads

The easiest way is to define dependency to "clover" in your application's `BuildConfig.groovy` file. See [installation guide](#) for more details.

The latest binaries can be also downloaded from <http://grails.org/plugin/clover> page.

## Clover-for-Grails Changelog

Please also refer to the [Clover-for-Ant Changelog](#).

The changes for the latest versions are as follows:


### Changes in Clover-for-Grails 4.0.0

#### July 14, 2014

This is a feature release with new HTML report with the ADG (Atlassian Design Guidelines) look.

#### Implemented features and fixes



Key	Summary	T	P
CLOV-1345	Apply ADG in the HTML report		

1 issue





See also change log for Clover-for-Ant, Clover-for-Maven, Clover-for-Eclipse, Clover-for-IDEA.

## Changes in Clover-for-Grails 3.3.0

**April 1, 2014**

This is a feature release with dedicated support for the Spock framework and JUnit4 parameterized tests.

### Implemented features and fixes

Key	Summary	T	P
CLOV-1256	as a developer I'd like to instrument tests written in the Spock framework		
CLOV-1458	Grails Clover Plugin Causing ConcurrentModification Error		

2 issues

See also change log for Clover-for-Ant, Clover-for-Maven, Clover-for-Eclipse, Clover-for-IDEA.

### Older versions

Looking for older versions? See [Clover-for-Grails Changelog for Clover 3.2.](#)


## Changes in Clover-for-Grails 4.0.0

### Changes in Clover-for-Grails 4.0.0

**July 14, 2014**

This is a feature release with new HTML report with the ADG (Atlassian Design Guidelines) look.

### Implemented features and fixes

Key	Summary	T	P
CLOV-1345	Apply ADG in the HTML report		

1 issue

See also change log for Clover-for-Ant, Clover-for-Maven, Clover-for-Eclipse, Clover-for-IDEA.



## Changes in Clover-for-Grails 3.3.0

### Changes in Clover-for-Grails 3.3.0

**April 1, 2014**

This is a feature release with dedicated support for the Spock framework and JUnit4 parameterized tests.

### Implemented features and fixes

Key	Summary	T	P
CLOV-1256	as a developer I'd like to instrument tests written in the Spock framework		



## 2 issues

See also change log for Clover-for-Ant, Clover-for-Maven, Clover-for-Eclipse, Clover-for-IDEA.

# Clover Command Line Tools

Clover provides a set of Command line tools for integration with legacy build systems such as Make, or custom build scripts. **If you use Jakarta Ant to build your project, a set of [Clover Ant Tasks](#) provides easier Ant integration.**

To use the tools in your build system, the synopsis is:

1. Copy and instrument your source files using [CloverInstr](#).
2. Compile the instrumented source files using a standard java compiler.
3. Execute your tests using whatever framework.
4. (Optional) If you have multiple separate coverage databases, merge them using [CloverMerge](#).
5. Use either the [XMLReporter](#), [HtmlReporter](#), [PDFReporter](#), [JSONReporter](#) or [ConsoleReporter](#) to view the measured coverage results.

## Command line tools:

<a href="#">CloverInstr</a>	Copies and instruments individual Java source files, or a directory of source files. Please note that this tool does not instrument Groovy.
<a href="#">CloverMerge</a>	Merges existing Clover databases to allow for combined reports to be generated.
<a href="#">XMLReporter</a>	Produces coverage reports in XML.
<a href="#">HtmlReporter</a>	Produces coverage reports in HTML.
<a href="#">JSONReporter</a>	Produces coverage reports in JSON format.
<a href="#">PDFReporter</a>	Produces coverage reports in PDF format.
<a href="#">ConsoleReporter</a>	Reports coverage results to the console.

## Troubleshooting

### Troubleshooting License Problems

When running Clover Command Line Tools you may come across the following error:

```
ERROR: No license file found.
Exception in thread "main" java.lang.RuntimeException: Invalid or
missing License.. Please visit http://my.atlassian.com to obtain a valid
license.
at com.atlassian.clover.CloverStartup.loadLicense(CloverStartup.java:58)
at com.atlassian.clover.CloverStartup.loadLicense(CloverStartup.java:25)
```

Please ensure that your `clover.license` file is in the same directory as the `clover.jar` file, or use the `clover.license.path` parameter when running Clover Command Line Tools. For example:

```
java -Dclover.license.path=/path/to/clover.license -cp
/path/to/clover.jar com.atlassian.clover.CloverInstr ...
```

## CloverInstr

This tool copies and instruments a set of Java source files specified on the command line. The output of the instrumentation process is **instrumented java source**; you will then need to compile the instrumented source using a standard Java compiler.

### Usage

```
java com.atlassian.clover.CloverInstr [OPTIONS] PARAMS [FILES...]
```

*Note: in Clover 3.1.x and older a class was named `com.cenqua.clover.CloverInstr`.*

### Params

<code>-i, --initstring &lt;file&gt;</code>	Clover initstring. This is the full path to the dbfile that will be used to construct/update to store coverage data.
<code>-s, --srcdir &lt;dir&gt;</code>	Directory containing source files to be instrumented. If omitted individual source files should be specified on the command line.
<code>-d, --destdir &lt;dir&gt;</code>	Directory where Clover should place the instrumented sources. <b>Note that files will be overwritten in the destination directory.</b>

### Options

<code>-dc, --distributedCoverage &lt;string&gt;</code>	Configuration for recording distributed pre-test coverage. Valid keys and default values are: ON   OFF
<code>--dontFullyQualifyJavaLang</code>	If set, then <code>java.lang</code> will not be used in instrumented source.
<code>-e, --encoding &lt;encoding&gt;</code>	Specify the file encoding for source files. If not specified, the platform default encoding is used.
<code>-f, --flushinterval &lt;int&gt;</code>	Tell Clover how often to flush coverage data when using either "interval" or "threaded" flushpolicy. Value in milliseconds.
<code>--instrumentation &lt;policy&gt;</code>	Set the instrumentation strategy. Valid values are "field" and "class". Default is "class".
<code>--instrlevel &lt;string&gt;</code>	Set the instrumentation level. Valid values are "statement" and "method". Default is "statement".
<code>--instrlambda &lt;string&gt;</code>	Whether to instrument lambda functions. Valid values are: <ul style="list-style-type: none"> <li>"none" - lambda functions will not be visible as "methods", code statements from a lambda body will become a part of an enclosing method</li> <li>"expression" - only lambda functions in an expression-like form (e.g. "<code>(a, b) -&gt; a + b</code>") are instrumented</li> <li>"block" - only lambda functions written as code blocks (e.g. "<code>() -&gt; { return 123; }</code>") are instrumented</li> <li>"all" - instrument all lambda functions</li> </ul> <i>Since Clover 3.2.2.</i>
<code>-p, --flushpolicy &lt;policy&gt;</code>	Tell Clover which flushpolicy to use when flushing coverage data to disk. Valid values are "directed", "interval" and "threaded". With "interval" or "threaded", you must also specify a flushinterval using -f. The default value is "directed".
<code>-mc --methodContext &lt;name&gt;=&lt;regexp&gt;</code>	Defines a single custom method context. May be supplied more than once. (The <code>\</code> may be needed to prevent shell expansion)

<code>-sc --statementContext &lt;name&gt;=&lt;regexp&gt;</code>	Defines a single custom statement context. May be supplied more than once. (The <code>\</code> may be needed to prevent shell expansion)
<code>-r, --relative</code>	If specified, the <code>initstring</code> is treated as a relative path, rather than being converted to an absolute path. This is useful for distributed testing environments.
<code>--recordTestResults &lt;true false&gt;</code>	If set to "false", test results will not be recorded; instead, results can be added via the <code>&lt;testResults&gt;</code> fileset at report time. For more details please see ' <a href="#">Advanced Usage</a> '.
<code>--source &lt;level&gt;</code>	Set the language level for Clover to use when parsing files.
<code>--sourceRoot &lt;string&gt;</code>	Source root path prefix that will be ignored when evaluating the test inclusion patterns. This parameter is optional; it specifies what is trimmed from the beginning of the file path before the tests Include/Exclude Pattern is evaluated (see parameters below). For example, if you specify <code>--sourceRoot /home/user/project/src</code> , then pattern <code>'test/*.'</code> would match a file in this location: <code>'/home/user/project/src/test/Test.java'</code> . If you leave the <code>--sourceRoot</code> option out, the pattern would need to start with <b>* or specify the full path</b> <code>'/home/user/project/src/test/.'</code>
<code>-v, --verbose</code>	Enable verbose logging.

## API Usage

CloverInstr provides a simple API that accepts an array of strings representing the command line arguments and returns an integer result code. The following fragment illustrates use of the API:

```
import com.atlassian.clover.CloverInstr;

...

String [] cliArgs =
    { "-jdk14", "-i", "clover.db", "-d", "build/instr", "Money.java" }
    ;
int result = CloverInstr.mainImpl(cliArgs);
if (result != 0) {
    // problem during instrumentation
}
```

## Examples

```
java com.atlassian.clover.CloverInstr -i clover.db -s src -d build/instr
```

Find all java source files in the directory "src", copy and instrument them into the directory "build/instr", which will be constructed if it does not exist. Coverage database "clover.db" is initialised.

```
java com.atlassian.clover.CloverInstr --source 1.4 -i clover.db -d
../../build/instr \
    Money.java IMoney.java
```

Copy and instrument the source files "Money.java" and "IMoney.java" into the directory `"../../build/instr"`. Use the JDK1.4 grammar (i.e. support the 'assert' keyword).

## CloverMerge

This tool merges existing Clover databases to allow for combined reports to be generated.

### Usage

```
java com.atlassian.clover.CloverMerge [OPTIONS] PARAMS [DBFILES...]
```

Note: in Clover 3.1.x and older a class was named `com.cenqua.clover.CloverMerge`.

### Parameters

Parameter	Description	Required
<code>-i, --initstring &lt;file&gt;</code>	Clover initstring. Clover initstring. This is the path where the new merged database will be written.	Yes.

### Options

Option	Description
<code>-d, --debug</code>	Enable debug logging.
<code>-s, --span interval</code>	Specifies the span to use when reading subsequent databases to be merged. This option can be specified more than once and applies to all databases specified after the option, or until another span is specified
<code>-u, --update interval</code>	If specified, any existing database specified by <code>-i</code> will be included in the merge. If interval is specified, it is used as the span when reading the existing database.
<code>-v, --verbose</code>	Enable verbose logging.

### API Usage

CloverMerge provides a simple API that accepts an array of strings representing the command line arguments and returns an integer result code. The following fragment illustrates use of the API:

```
import com.atlassian.clover.CloverMerge;

...

String [] cliArgs = { "-i", "new.db", "proj1.db", "proj2.db", "-s", "10s",
"proj3.db" };
int result = CloverMerge.mainImpl(cliArgs);
if (result != 0) {
    // problem during instrumentation
}
```


### Examples

```
java com.atlassian.clover.CloverMerge -i new.db proj1.db proj2.db
```

Merges `proj1.db` and `proj2.db` into the new database `new.db`. A span of zero seconds is used.

```
java com.atlassian.clover.CloverMerge -i new.db proj1.db -s 30s proj2.db proj3.db
```

Merges proj1.db, proj2.db and proj3.db into the new database new.db. A span of zero seconds is used for proj1.db, and a span of 30 seconds is used for proj2.db and proj3.db.

 The files named coverage.dbxxxx\_xxx\_xxx are not db files. Please see the [clover knowledge base](#). You only need to merge files called coverage.db (there should be one per a directory) and not the recording files coverage.dbxxxx\_xxx\_xxx (there could be hundreds of these)

## ConsoleReporter

Reports Code Coverage for the given coverage database to the console.

### Usage

```
java com.atlassian.clover.reporters.console.ConsoleReporter [OPTIONS] PARAMS
```

*Note: in Clover 3.1.x and older a class was named com.cenqua.clover.reporters.console.ConsoleReporter.*

### Params

-i, --initstring <file>	The initstring of the coverage database.
-------------------------	--

### Options

-t, --title <string>	Report title
-l, --level <string>	The level of detail to report. Valid values are "summary", "class", "method", "statement". Default value is "summary".
-p, --sourcepath <path>	The source path to search when looking for source files.
-si, --showinner	<b>Since 3.2.0:</b> Show inner functions in the report (like a lambda function inside a method).
-sl, --showlambda	<b>Since 3.2.0:</b> Show lambda functions in the report.
-s, --span <interval>	Specifies how far back in time to include coverage recordings from since the last Clover build. See <a href="#">Using Spans</a> . Default includes "all coverage data found".
-u, --unittests	<b>Since 3.1.6:</b> Show unit tests results summary. By default summary is not listed.
-c, --codetype	<b>Since 3.1.6:</b> The type of code to report on. Valid values are: APPLICATION, TEST, ALL. Default value: APPLICATION

### Return code

The ConsoleReporter.main() calls System.exit() and returns a non-zero value in case of error during report generation.

### API Usage

ConsoleReporter provides a simple API that accepts an array of strings representing the command line arguments and returns an integer result code. The following fragment illustrates use of the API:

```
import com.atlassian.clover.reporters.console.ConsoleReporter;

...

String [] cliArgs = { "-l", "method", "-t", "Method Coverage", "-i",
"clover.db" };
int result = ConsoleReporter.mainImpl(cliArgs);
if (result != 0) {
    // problem during report generation
}
```

## Examples

```
java com.atlassian.clover.reporters.console.ConsoleReporter -i clover.db
```

Reads coverage for the Clover database "clover.db", and produces a summary report to the console.

```
java com.atlassian.clover.reporters.console.ConsoleReporter -l "method" -t "Method Coverage" -i clover.db
```

Produces the same report as above, but includes method-level coverage information, and a report title.

## HtmlReporter

Produces an HTML report of Code Coverage for the given coverage database.

### Usage

```
java com.atlassian.clover.reporters.html.HtmlReporter [OPTIONS] PARAMS
```

*Note: in Clover 3.1.x and older a class was named `com.cenqua.clover.reporters.html.HtmlReporter`.*

### Params

<code>-i, --initstring &lt;file&gt;</code>	The initstring of the coverage database.
<code>-o, --outputdir &lt;dir&gt;</code>	The directory to write the report to. Will be created if it doesn't exist.

### Options

<code>-a, --alwaysreport</code>	Forces a report to be generated, even if there is no coverage data. Defaults to 'false', i.e. a report with no coverage will abort generation.
<code>-b, --hidebars</code>	Don't render coverage bars.
<code>-bw</code>	Don't colour syntax-highlighted source — smaller HTML output.

-c, --orderby <compname>	Comparator to use when listing packages and classes. Default is PcCoveredAsc. Valid values are: <ul style="list-style-type: none"> <li>• Alpha — Alphabetical.</li> <li>• PcCoveredAsc — Percent total coverage, ascending.</li> <li>• PcCoveredDesc — Percent total coverage, descending.</li> <li>• ElementsCoveredAsc — Total elements covered, ascending.</li> <li>• ElementsCoveredDesc — Total elements covered, descending.</li> <li>• ElementsUncoveredAsc — Total elements uncovered, ascending.</li> <li>• ElementsUncoveredDesc — Total elements uncovered, descending.</li> </ul>
-d, --debug	Switch logging level to debug.
-e, --showempty	Show classes/packages even if they don't have any statements, methods or conditionals. default is false
-f, --filter <string>	Comma or space separated list of contexts to ignore when generating coverage reports. Most useful one is "catch". Valid values are "assert", "static", "instance", "constructor", "method", "switch", "while", "do", "for", "if", "else", "try", "catch", "finally", "sync", or the name of a user-defined Context. See <a href="#">Using Contexts</a> .
-h, --hidesrc	Don't render source level coverage.
-if --includefailcoverage	Specifies whether or not to include coverage attributed to a test that has failed. If omitted, failed test coverage is not included. Default setting is 'false'.
-n, --nocache	Insert no-cache browser directives in html output.
-p, --sourcepath <path>	The source path to search when looking for source files.
-s, --span <interval>	Specifies how far back in time to include coverage recordings from. See <a href="#">Using Spans</a> . Default includes "all coverage data found".
-si, --showwinner	<b>Since 3.2.0:</b> Show inner functions in the report (like a lambda function inside a method).
-sl, --showlambda	<b>Since 3.2.0:</b> Show lambda functions in the report.
-su, --showunique	<b>Since 3.1.5:</b> Calculate and show unique per-test coverage (for large projects can consume much memory and take a significant amount of time). Defaults to false.
--style <string>	<b>Since 4.0.0:</b> Style of the report: "adg" - following the <a href="#">Atlassian Design Guidelines</a> (default) "classic" - JavaDoc-like (deprecated, will be removed in future)
-t, --title <string>	Report title.
-tc, --threadcount <int>	Number of <u>additional</u> threads to be allocated to report generation. Default is 0.
-tw, --tabwidth <int>	The number of spaces to substitute TAB characters with. Defaults to 4.
-v, --verbose	Switch logging level to verbose.

### Return code

The `HtmlReporter.main()` calls `System.exit()` and returns a non-zero value in case of error during HTML report generation.



## API Usage

HtmlReporter provides a simple API that accepts an array of strings representing the command line arguments and returns an integer result code. The following fragment illustrates use of the API:

```
import com.atlassian.clover.reporters.html.HtmlReporter;
import com.atlassian.clover.CloverStartup;
import com.atlassian.clover.Logger;

public class HtmlReportRunner {
    public void runReport() {
        CloverStartup.loadLicense(Logger.getInstance());
        String [] cliArgs = { "-i", "clover.db", "-o", "clover_html" };
        int result = HtmlReporter.runReport(cliArgs);
        if (result != 0) {
            // problem during report generation
        }
    }
}
```

## Examples

```
java com.atlassian.clover.reporters.html.HtmlReporter -i clover.db -o clover_html
```

Reads coverage for the Clover database "clover.db", and produces a report in the directory "clover\_html".

```
java com.atlassian.clover.reporters.html.HtmlReporter -c ElementsCoveredAsc
-t "My Coverage" -i clover.db -o clover_html
```

Produces the same report as above, but includes a report title, and orders lists by total elements covered rather than percentage covered.

## JSONReporter

Produces a JSON report of Code Coverage for the given coverage database.

### Usage

```
java com.atlassian.clover.reporters.json.JSONReporter [OPTIONS] PARAMS
```

*Note: in Clover 3.1.x and older a class was named `com.cenqua.clover.reporters.json.JSONReporter`.*

### Parameters

-i, --initstring <file>	The initstring of the coverage database.
-o, --outputdir <dir>	The output directory for generated JSON.

### Options

-a, --alwaysreport	Forces a report to be generated, even if there is no coverage data. Defaults to 'false', i.e. a report with no coverage will abort generation.
-d, --debug	Switch logging level to debug.
-if --includefailcoverage	Specifies whether or not to include coverage attributed to a test that has failed. If omitted, failed test coverage is not included. Default setting is 'false'.
-si, --showwinner	<b>Since 3.2.0:</b> Show inner functions in the report (like a lambda function inside a method).
-sl, --showlambda	<b>Since 3.2.0:</b> Show lambda functions in the report.
-tc, --threadcount <int>	Number of <u>additional</u> threads to be allocated to report generation. Default is 0.
-v, --verbose	Switch logging level to verbose.

### Return code

The `JSONReporter.main()` calls `System.exit()` and returns non-zero return code in case of error during JSON report generation.

### API Usage

JSONReporter provides a simple API that accepts an array of strings representing the command line arguments and returns an integer result code. The following fragment illustrates use of the API:

```
import com.atlassian.clover.CloverStartup;
import com.atlassian.clover.Logger;
import com.atlassian.clover.reporters.json.JSONReporter;

public class JsonReportRunner {
    public void runReport() {
        CloverStartup.loadLicense(Logger.getInstance());
        String [] cliArgs = { "-i", "clover.db", "-o", "clover_json" };
        int result = JSONReporter.runReport(cliArgs);
        if (result != 0) {
            // problem during report generation
        }
    }
}
```

### Examples

```
java com.atlassian.clover.reporters.json.JSONReporter -i clover.db -o clover_json
```

Read coverage for the Clover database "clover.db" and produce a report in the directory "clover\_json".

## PDFReporter

Produces a PDF summary report of Code Coverage for the given coverage database.

### Usage

```
java com.atlassian.clover.reporters.pdf.PDFReporter [OPTIONS] PARAMS
```

Note: in Clover 3.1.x and older a class was named `com.cenqua.clover.reporters.pdf.PDFReporter`.

### Params

<code>-i, --initstring &lt;file&gt;</code>	The initstring of the coverage database.
<code>-o, --outfile &lt;file&gt;</code>	The file to write the report to. <i>Note: In Clover 3.1.x and older the longer option was named '--outputfile'</i>

### Options

<code>-a, --alwaysreport</code>	Forces a report to be generated, even if there is no coverage data. Defaults to 'false', i.e. a report with no coverage will abort generation.
<code>-b, --hidebars</code>	Don't render coverage bars.
<code>-c, --orderby &lt;compname&gt;</code>	Comparator to use when listing packages and classes. Default is <code>PcCoveredAsc</code> . Valid values are: <ul style="list-style-type: none"> <li>• Alpha — Alphabetical.</li> <li>• <code>PcCoveredAsc</code> — Percent total coverage, ascending.</li> <li>• <code>PcCoveredDesc</code> — Percent total coverage, descending.</li> <li>• <code>ElementsCoveredAsc</code> — Total elements covered, ascending.</li> <li>• <code>ElementsCoveredDesc</code> — Total elements covered, descending.</li> <li>• <code>ElementsUncoveredAsc</code> — Total elements uncovered, ascending.</li> <li>• <code>ElementsUncoveredDesc</code> — Total elements uncovered, descending.</li> </ul>
<code>-d, --debug</code>	Switch logging level to debug
<code>-e, --showempty</code>	Show classes/packages even if they don't have any statements, methods or conditionals. default is false.
<code>-f, --filter &lt;string&gt;</code>	Comma or space separated list of contexts to ignore when generating coverage reports. Most useful one is "catch". Valid values are "assert", "static", "instance", "constructor", "method", "switch", "while", "do", "for", "if", "else", "try", "catch", "finally", "sync", or the name of a user-defined Context. See <a href="#">Using Contexts</a> .
<code>-if</code> <code>--includefailcoverage</code>	Specifies whether or not to include coverage attributed to a test that has failed. If omitted, failed test coverage is not included. Default setting is 'false'.
<code>-p, --pagesize &lt;size&gt;</code>	Specify the page size to render. Valid values are "Letter" and "A4". Default is "A4".
<code>-s, --span &lt;interval&gt;</code>	Specifies how far back in time to include coverage recordings from. See <a href="#">Using Spans</a> . Default includes all coverage data found.
<code>-t, --title &lt;string&gt;</code>	Report title
<code>-tc, --threadcount &lt;int&gt;</code>	Number of <u>additional</u> threads to be allocated to report generation. Default is 0.
<code>-v, --verbose</code>	Switch logging level to verbose

### Return code

The `PDFReporter.main()` calls `System.exit()` and returns non-zero return code in case of error during PDF report

generation.

## API Usage

PDFReporter provides a simple API that accepts an array of strings representing the command line arguments and returns an integer result code. The following fragment illustrates use of the API:

```
import com.atlassian.clover.CloverStartup;
import com.atlassian.clover.Logger;
import com.atlassian.clover.reporters.pdf.PDFReporter;

public class PdfReportRunner {
    public void runReport() {
        CloverStartup.loadLicense(Logger.getInstance());
        String [] cliArgs = { "-i", "clover.db", "-o", "clover.pdf" };
        int result = PDFReporter.runReport(cliArgs);
        if (result != 0) {
            // problem during report generation
        }
    }
}
```

## Examples

```
java com.atlassian.clover.reporters.pdf.PDFReporter -i clover.db -o coverage.pdf
```

Reads coverage for the Clover database "clover.db", and produces a pdf report in the file "coverage.pdf".

```
java com.atlassian.clover.reporters.pdf.PDFReporter -c ElementsCoveredAsc
-t "My Coverage" -i clover.db -o coverage.pdf
```

Produces the same report as above, but includes a report title, and orders lists by total elements covered rather than percentage covered.

## XMLReporter

Produces an XML report of Code Coverage for the given coverage database.

## Usage

```
java com.atlassian.clover.reporters.xml.XMLReporter [OPTIONS] PARAMS
```

*Note: in Clover 3.1.x and older a class was named `com.cenqua.clover.reporters.xml.XMLReporter`.*

## Parameters

-i, --initstring <file>	The initstring of the coverage database.
-o, --outfile <file>	The file to write XML output to.

## Options

-a, --alwaysreport	Forces a report to be generated, even if there is no coverage data. Defaults to 'false', i.e. a report with no coverage will abort generation.
-d, --debug	Switch logging level to debug.
-f, --filter <string>	Comma or space separated list of contexts to ignore when generating coverage reports. Most useful one is "catch". Valid values are "assert", "static", "instance", "constructor", "method", "switch", "while", "do", "for", "if", "else", "try", "catch", "finally", "sync", or the name of a user-defined Context. See <a href="#">Using Contexts</a> .
-if --includefailcoverage	Specifies whether or not to include coverage attributed to a test that has failed. If omitted, failed test coverage is not included. Default setting is 'false'.
-l, --lineinfo	Include source-level coverage info.
-s, --span <interval>	Specifies how far back in time to include coverage recordings from. . See <a href="#">Using Spans</a> . Default includes "all coverage data found".
-si, --showwinner	<b>Since 3.2.0:</b> Show inner functions in the report (like a lambda function inside a method).
-sl, --showlambda	<b>Since 3.2.0:</b> Show lambda functions in the report.
-tc, --threadcount <int>	Number of <u>additional</u> threads to be allocated to report generation. Default is 0.
-t, --title <string>	Report title.
-v, --verbose	Switch logging level to verbose.

### Return code

@since Clover 3.1.12: The XMLReporter.main() calls System.exit() and returns non-zero return code in case of error during XML report generation.

### API Usage

XMLReporter provides a simple API that accepts an array of strings representing the command line arguments and returns an integer result code. The following fragment illustrates use of the API:

```
import com.atlassian.clover.CloverStartup;
import com.atlassian.clover.Logger;
import com.atlassian.clover.reporters.xml.XMLReporter;

public class XmlReportRunner {
    public void runReport() {
        CloverStartup.loadLicense(Logger.getInstance());
        String [] cliArgs = { "-i", "clover.db", "-o", "clover.xml" };
        int result = XMLReporter.runReport(cliArgs);
        if (result != 0) {
            // problem during report generation
        }
    }
}
```

### Examples

```
java com.atlassian.clover.reporters.xml.XMLReporter -i clover.db -o coverage.xml
```

Read coverage for the Clover database "clover.db", and produce a report in the file "coverage.xml"

```
java com.atlassian.clover.reporters.xml.XMLReporter -l -t "My Coverage" -i  
clover.db -o coverage.xml
```

Produce the same report as above, but include source-level coverage information, and a report title.

## Bamboo Clover Plugin

### Bamboo Clover Plugin

Bamboo Clover Plugin provides a "single-click" integration with your Ant-based or Maven-based builds in Bamboo.

See [Bamboo Documentation Home](#):

- [Enabling the Clover add-on](#)
- [Using Bamboo with Clover](#)
  - [Getting gcov results in Clover coverage summary](#)
- [Viewing the Clover code-coverage for a build](#)
- [Viewing the Clover code-coverage for a plan](#)

More Bamboo tool integrations:

- [Integrating Bamboo with Atlassian applications](#)

## Gradle Clover Plugin

 This is an open-source extension and Atlassian does not provide technical support for it.

[Gradle](#) framework can automate the building, testing, publishing, deployment and more of software packages or other types of projects such as generated static websites or generated documentation.

### Gradle Cookbook

Gradle cookbook presents a simple script with basic Clover integration:

<http://wiki.gradle.org/display/GRADLE/Cookbook#Cookbook-usingClover>

### Gradle Clover Plugin

A quite functional Clover plugin written by Benjamin Muschko:

<https://github.com/bmuschko/gradle-clover-plugin>

Known issues:

- <https://github.com/bmuschko/gradle-clover-plugin/issues>
- plug-in requires presence of "main" source directory (e.g. src/main/groovy)

- a default test inclusion pattern is "\*\*/\*Test.java" and "\*\*/\*Test.groovy", so in case you have other test naming convention (for instance, Spock framework has "\*\*Spec.groovy") you have to declare `clover.testIncludes` property in build.gradle

## Griffon Clover Plugin

 This is an open-source extension and Atlassian does not provide technical support for it.

[Griffon Clover Plugin home page](#)

## Hudson Clover Plugin

 This is an open-source extension and Atlassian does not provide technical support for it.

Integration with Hudson continuous integration system. This plug-in allows you to capture code coverage reports from Clover. Hudson will generate and track code coverage across time.

Home page: <http://wiki.hudson-ci.org/display/HUDSON/Clover+Plugin>

## Jenkins Clover Plugin

 This is an open-source extension and Atlassian does not provide technical support for it.

Integration with Jenkins continuous integration system. This plug-in allows you to capture code coverage reports from Clover. Jenkins will generate and track code coverage across time.

Home page: <https://wiki.jenkins-ci.org/display/JENKINS/Clover+Plugin>

## Sonar Clover Plugin

 This is an open-source extension and Atlassian does not provide technical support for it.

Integration with [Sonar](#) - an open source quality management platform, dedicated to continuously analyse and measure technical quality, from project portfolio to a method.

Home page: <http://docs.codehaus.org/display/SONAR/Clover+Plugin>

## Clover Release Notes

### Release Summary

#### Clover 4.0

11 July 2014

- HTML ADG report

Read the [Clover 4.0 release notes](#).

## Clover 3.3

### 31 March 2014

- Spock framework support
- JUnit4 parameterized tests
- Lambda toggle in report wizards (Eclipse, IDEA)

Read the [Clover 3.3 release notes](#).

## Clover 3.2

### 21 October 2013

- Java 1.8 Support
- Support for Eclipse 4.3 and RAD 8.5
- Dropped support for JDK1.4
- Enhanced reports

Read the [Clover 3.2 release notes](#).

## Clover 3.1

### 31 May 2011

- Java 1.7 Support
- Maven 3 Support
- Groovy 1.6-1.8 Support for Ant, Maven 2 and Grails
- Clover-for-Grails Plugin for Grails 1.3.7
- Support for Eclipse 3.6 and 3.7
- Support for IntelliJ 10.5 and 11
- Bug fixes and improvements

Read the [Clover 3.1 release notes](#).

## Clover 3.0

### 31 March 2010

- Groovy Support for Ant, Maven 2 and Grails
- New Clover-for-Grails Plugin
- Per-Test Coverage Viewer for Eclipse
- New Dashboard View in Eclipse
- Updated Tutorial with Groovy Code
- Other Enhancements and Improvements
- Over 50 bug fixes and improvements

Read the [Clover 3.0 release notes](#).

## Clover 2.6

### 9 Sept 2009

- New Clover Editions
- Eclipse Plugin Performance Improvements
- IDE Plugin Ease-of-Use Features
- 100% Coverage filter
- New HTML Tree Map
- New API for Optimizing Tests Programmatically
- Clover Auto-Update Feature for IDEA

Read the [Clover 2.6 release notes](#).

## Clover 2.5



**11 May 2009**

- Test Optimization in Clover for Eclipse and IDEA
- Distributed Per-Test Coverage
- Performance Improvements

Read the [Clover 2.5 release notes](#).

**Clover 2.4****5 Nov 2008**

- Test Optimization
- Easier Integration
- Reporting Improvements
- Clover-for-IDEA final release

Read the [Clover 2.4 release notes](#).

**Clover 2.3****9 May 2008**

- New options for Movers report
- New <added> tag
- 15 fixes and revisions

Read the [Clover 2.3 release notes](#).

**Clover 2.2****10 April 2008**

- New visualisations for Dashboard
- Stack trace navigation in reports
- Better cross-referencing in reports
- Configure Clover to warn you or fail your build when your coverage drops

Read the [Clover 2.2 release notes](#).

**Clover 2.1****14 Feb 2008**

- Configurable metrics reporting
- Per-package coverage clouds
- Historical charting
- Enhanced 'movers' section
- Per-test coverage for merged databases
- Greatly improved performance

Read the [Clover 2.1 release notes](#): [Ant](#) and [Eclipse](#)

**Clover 2.0****17 Oct 2007**

- Coverage by test case
- Test results integrated with reports
- 'Coverage Cloud' reports
- Linked, cross-referenced reports
- Context filters
- Method-level metrics
- Sortable columns
- Streamlined Ant integration and simplified Ant tasks
- Eclipse plugin and Maven 2 plugin

- [Inline help](#)

Read the [Clover 2.0 release notes](#)

## Upgrade Guides

- [Clover-for-Ant Upgrade Guide](#)
- [Clover-for-Eclipse Upgrade Guide](#)
- [Clover-for-IDEA Upgrade Guide](#)
- [Clover-for-Maven 2 and 3 Upgrade Guide](#)

## Changelogs

*Information about changes in minor versions can be found here:*

- [Clover-for-Ant Changelog](#)
- [Clover-for-Eclipse Changelog](#)
- [Clover-for-IDEA Changelog](#)
- [Clover-for-Maven 2 and 3 Changelog](#)
- [Clover-for-Grails Changelog](#)

## Clover 4.0 Release Notes

**11 July 2014**

We are happy to announce **Clover 4.0**.

**Clover 4.0** brings completely redesigned HTML report following the [Atlassian Design Guidelines \(ADG\)](#).

Upgrading to Clover 4.0 is free for all customers with active Clover software maintenance at the date of launch and of subsequent updates.

### Highlights of Clover 4.0:

- [New ADG HTML report](#)



### Changes in Clover 4.0.x bug-fix versions

This page describes new features of Clover 4.0. For changes in 4.0.x bug-fix releases, please refer to the relevant documentation:

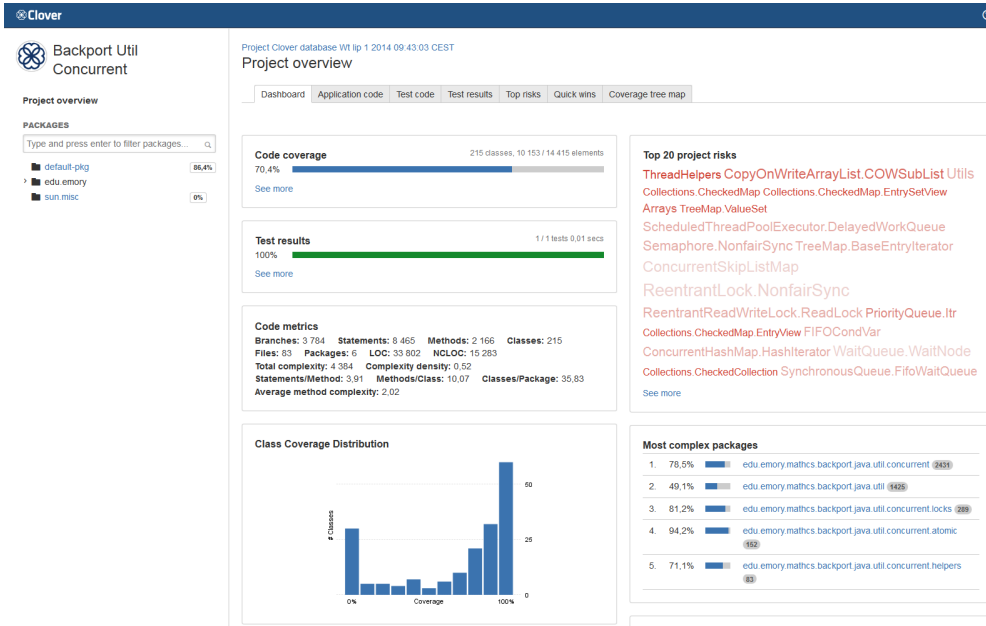
- [Clover-for-Ant | Changelog](#)
- [Clover-for-Maven 2 and 3 | Changelog](#)
- [Clover-for-IDEA | Changelog](#)
- [Clover-for-Eclipse | Changelog](#)
- [Clover-for-Grails | Changelog](#)

## Highlights of Clover 4.0

### New ADG HTML report

Clover 4.0 brings completely redesigned HTML report with new page layout, navigation, fonts and colours. All according to the [Atlassian Design Guidelines](#).

[Project overview - dashboard](#)

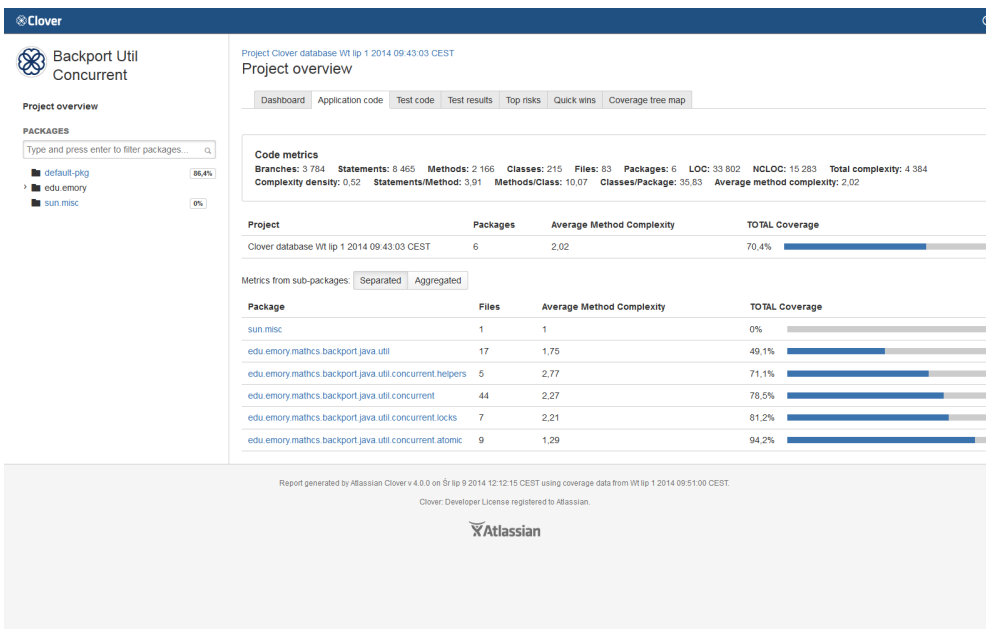


On the project overview page you will find several tabs, thanks to which you can quickly learn about your project:

- Dashboard - contains several widgets with statistics and most critical issues
- Application code - browse through application classes
- Test code - browse through test classes
- Test results - contains results from your unit tests
- Top risks - the most complex and the least covered classes
- Quick wins - "low hanging fruits"
- Coverage tree map

You can also use a package tree view with a search box to quickly jump to a package you're interested in.

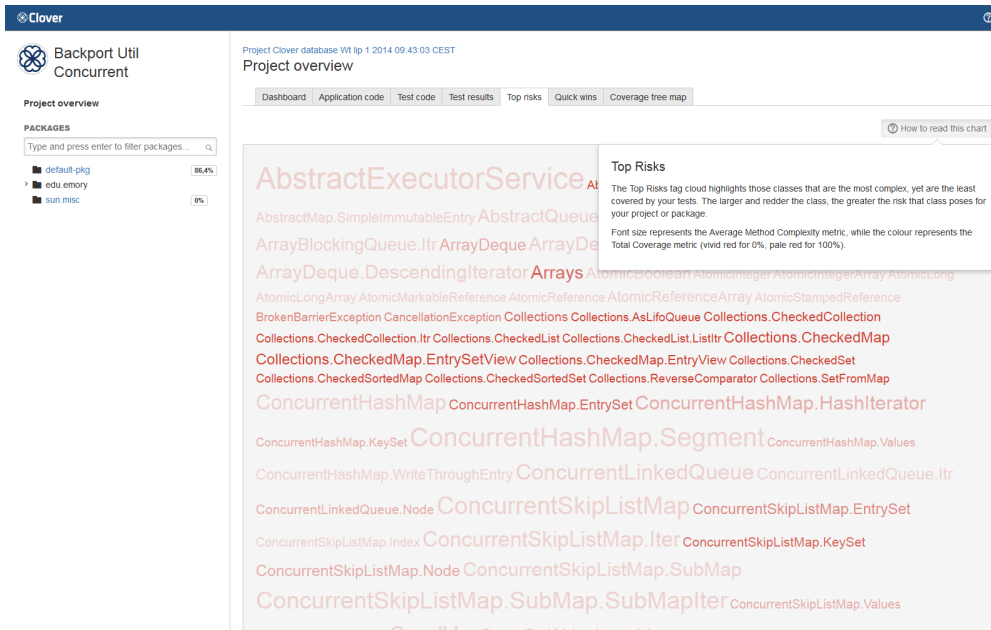
### Project overview and a package overview pages



A content is similar to the old report.

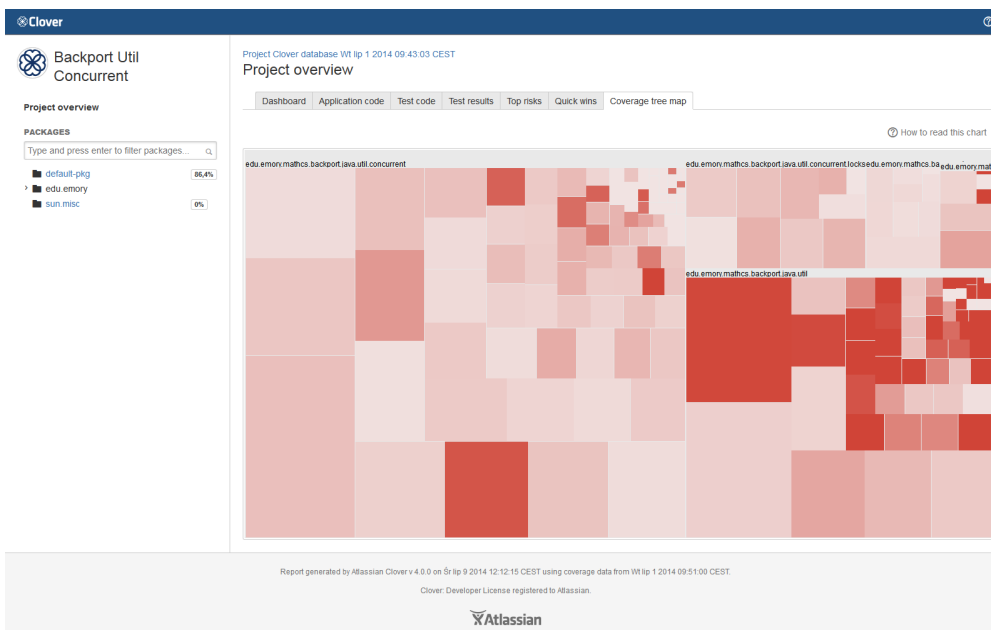
In addition, you can easily switch between the separated and aggregated code metrics (on a project overview page).

### 'Top risks' and 'Quick wins' tag clouds



Tags have a new colour palette (shades of red instead of a red-blue gradient). You can also open a help box explaining how code metrics are visualized.

### Coverage tree map



Tree map also has a new colour palette (shades of red instead of a green-black-red gradient).

### Source file view

Project Clover database: Wt lip 1 2014 09:43:03 CEST / Package: edu.emory.mathcs.backport.java.util

File: LinkedList.java

Coverage histogram: 78% of files have more coverage

Code metrics: Branches: 138, Statements: 279, Methods: 75, Classes: 4, LOC: 535, NLOC: 453, Total complexity: 141, Complexity density: 0.51, Statements/Method: 3.72, Methods/Class: 18.75, Average method complexity: 1.88

Class	Line #	Total Statements	Complexity	TOTAL Coverage	Actions
LinkedList	13	194	100	61.8%	Show methods
LinkedList.Entry	24	1	1	100%	Show methods
LinkedList.Itr	361	42	20	39.4%	Show methods
LinkedList.DescItr	432	42	20	42.3%	Show methods

Contributing tests: This file is covered by 47 tests. Select tests to highlight the test coverage.

Source view: Collapse all methods, Show legend

```

1 package edu.emory.mathcs.backport.java.util;
2
3 import java.io.*;
4 import java.util.List;
5 import java.util.Iterator;
6 import java.util.Collection;
7 import java.util.ListIterator;
8 import java.util.AbstractSequentialList;
9 import java.lang.reflect.Array;
    
```

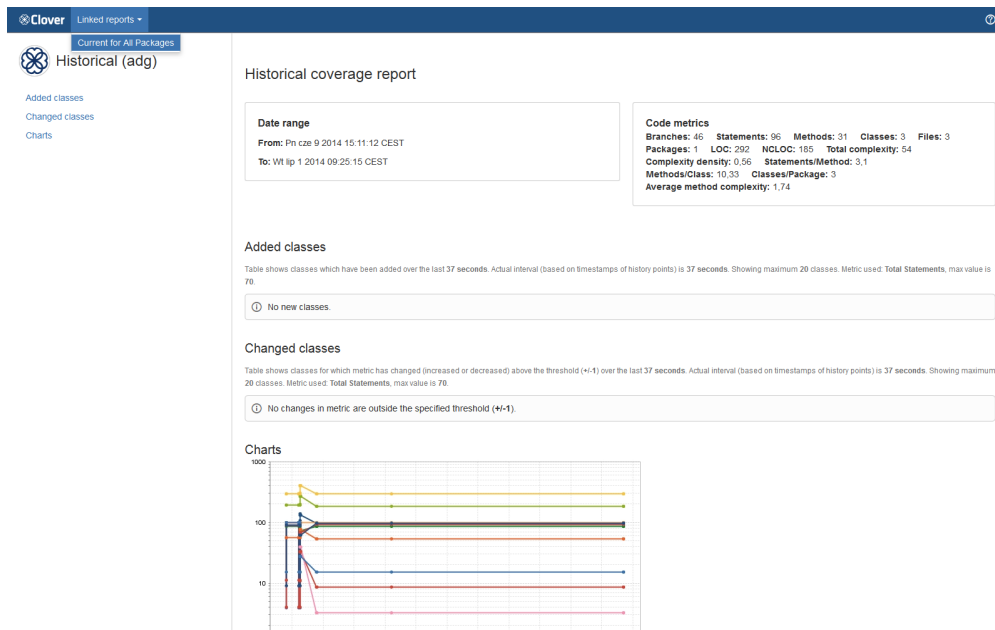
The "Show methods" link opens a modal dialog with class details. In the "Select tests to highlight the test coverage" dialog you can see all tests hitting given source file; you can also select them to see per-test coverage. In these dialog boxes you can also quickly filter methods and tests matching given string:

Contributing tests

Test contribution	Test	Result	Actions
<input type="checkbox"/>	testRemoveAll	PASS	
<input type="checkbox"/>	testIteratorRemove	PASS	
<input type="checkbox"/>	testDescendingIteratorRemove	PASS	
<input type="checkbox"/>	testRemoveFirstOccurrence	PASS	
<input type="checkbox"/>	testRemoveElement	PASS	
<input type="checkbox"/>	testRemoveLastOccurrence	PASS	
<input type="checkbox"/>	testRemove	PASS	

Select tests to highlight code covered by them in the source view. Apply

Historical report



The historical report has been "ADG-ified" as well.

In case you generate several linked reports, you can easily switch between them via the "Linked reports" drop down.

### Learning new ADG report

If you'd like to learn where elements from an old report are placed in the new one and how to navigate between them, see:

[A side-by-side comparison of the Classic and the ADG HTML report.](#)

## Upgrading from Clover 3.3 to Clover 4.0

When upgrading from Clover 3.3 to Clover 4.0, you may need to make few changes in your build scripts.

### 1) Clover JAR has been renamed from *com.cenqua.clover:clover* to *com.atlassian.clover:clover*

This JAR is being used for compilation and runtime. It's a dependency of Clover-for-Maven plugin (*com.atlassian.maven.plugins:maven-clover2-plugin*) and Clover-for-Grails plugin (*org.grails.plugins:clover*). However, your Maven or Grails scripts may have a dependency to this Clover JAR declared explicitly - especially when you're running in-container tests or spawning new JVMs in a build.

You have to use a following dependency in Maven's pom.xml:

```
<dependency>
  <groupId>com.atlassian.clover</groupId> <!-- com.cenqua.clover for Clover 3.x
-->
  <artifactId>clover</artifactId>
  <version>4.0.0</version>
</dependency>
```

or in Grails' `grails-app/conf/BuildConfig.groovy`:

```
grails.project.dependency.resolution = {
  plugins {
    compile 'org.grails.plugins:clover:4.0.0'
  }
  dependencies {
    compile 'com.atlassian.clover:clover:4.0.0' // com.cenqua.clover for Clover
  }
  3.x
}
```

## 2) Clover's database file format has changed.

You may need to delete existing Clover database files. In most typical environment configurations it will be done automatically (for instance, by Maven's *clean* goal). There is **no need** to delete [test optimization snapshots](#) or [history points](#).

## 3) Clover no longer supports JDK5.

You have to use at least JDK6 for both compilation and runtime. Please note, that Clover still supports Java 1.3-1.5 language levels (`<clover-setup source="1.4"/>`, for instance).

## 4) Deprecated Ant types have been removed

Since Clover 3.2, the Ant 1.6.5 is no longer supported. In Clover 4.0, we've deleted Ant 1.6-specific types, such as the `<clover-optimized-selector>` - use the `<clover-optimized-testset>` instead.

## 5) The `cloverjunitlib.xml` has been merged into `cloverlib.xml`

This file contained definitions of Ant 1.7+ types, such as `<clover-optimized-testset>`. You can find these definitions in `cloverlib.xml` now. Example:

```
<taskdef resource="cloverlib.xml" classpath="clover.jar"/>
```

## 6) Clover-for-Eclipse features and plugins have new IDs.

They have been renamed from `com.cenqua.***` to `com.atlassian.***`. Because of this, when upgrading, you have to:

- disable Clover on your projects ("*Package Explorer* -> *context menu* -> *Clover* -> *Enable/Disable on...*") - this is necessary to remove "Clover Pre-Java Builder" and "Clover Post-Java Builder",
- **uninstall** previous version of Clover and next install the Clover-for-Eclipse 4.x (otherwise you will end up with two Clover versions in one Eclipse),
- enable Clover on your projects.

## A side-by-side comparison of the Classic and the ADG HTML report

This page shows a side-by-side comparison of Clover's HTML report generated with a "classic" and "adg" style setting.

This page may be helpful for you to quickly learn how to navigate through a new ADG report and how to find information you were used to see in the old "Classic" one.

### Project overview

The screenshot displays the Clover Coverage Report interface. On the left, a navigation menu includes 'Dashboard' (1), 'Coverage Reports' (2), 'Coverage (Aggregate)' (3), 'Test Code (Aggregate)' (4), and 'Test Results' (5). Below this is the 'Application Packages' section with tabs for 'Classes', 'Tests', and 'Results'. A table lists 'Class' and 'Coverage' with a value of '(87.3%)'. The main content area shows 'Clover Coverage Report' with a 'Coverage timestamp: Wt lip 1 2014 09:25:15 CEST'. It features an 'Overview' tab and 'FRAMES NO FRAMES SHOW HELP' options. A 'Coverage' section shows '3 classes, 158 / 173 elements' and a '91,3%' progress bar. Below is the 'Class Coverage Distribution' section. At the bottom, a 'Project overview' section includes a search bar for 'PACKAGES' with the result '> com.cenqua'. A secondary navigation bar at the bottom right shows 'Dashboard' (1), 'Application code' (2), and 'Test code' (3).

Most of links from the top-left pane have been moved to tabs. Two tabs ("Top risks" and "Quick wins") which were accessible through a cloud icon (5, 6) in the old report are now accessible directly.

### Package view



### Backport Util Concurrent Clover Coverage Report

- Dashboard
- Coverage Reports  
- Coverage (Aggregate)
- Test Code (Aggregate)
- Test Results

#### Application Packages <sup>1</sup>

- edu.emory.mathcs.backport.java.util (49,1%)
- edu.emory.mathcs.backport.java.util.concurrent (78,5%)
- edu.emory.mathcs.backport.java.util.concurrent.atomic (94,2%)
- edu.emory.mathcs.backport.java.util.concurrent.helpers (71,1%)
- edu.emory.mathcs.backport.java.util.concurrent.locks (81,2%)
- sun.misc (0%)

edu.emory.mathcs.backport.java.util.concurrent

Classes Tests Results

Class	Cov
AbstractExecutorService <sup>2</sup>	(94,
ArrayBlockingQueue	(97,
ArrayBlockingQueue.Itr	(88,
BrokenBarrierException	(33,

Clover Coverage Report - Backport Util Conc

Cover <sup>6</sup> timestamp: Wt lip 1 2014 09:51:00 CEST

App <sup>6</sup> Clouds <sup>4</sup>

Overview **Package** <sup>3</sup> File

FRAMES NO FRAMES SHOW HELP

#### Package <sup>7</sup>

edu.emory.mathcs.backport.java.util.concu

- Class <sup>2</sup>
- CopyOnWriteArrayList.COWSubIterator
  - CopyOnWriteArrayList.COWSubList
  - RejectedExecutionException
  - ConcurrentSkipListMap.KeySet
  - ConcurrentHashMap.EntrySet
  - ConcurrentSkipListMap.EntrySet
  - ExecutionException
  - BrokenBarrierException
  - CancellationException
  - TimeoutException
  - Executors.DelegatedExecutorService



## Backport Util Concurrent

Project overview

#### PACKAGES <sup>1</sup>

Type and press enter to filter packages...

- default-pkg 86,4%
- edu.emory
  - mathcs.backport
    - java.util 49,1%
      - concurrent 78,5%
  - sun.misc 0%

### Package edu.emory.mathc

- Application code <sup>6</sup>
- Top risks <sup>4</sup>
- Quick wi

#### Code metrics <sup>5</sup>

Branches: 2 276    Statements: 5 163  
 Complexity density: 0,47    Statement:

#### Package <sup>7</sup>

edu.emory.mathcs.backport.java.util.concur

#### Class <sup>2</sup>

- CopyOnWriteArrayList.COWSubIterator
- CopyOnWriteArrayList.COWSubList

## RejectedExecutionException

---

The bottom-left frame with "Classes", "Tests" and "Results" tabs has been removed - instead of this, "Application code", "Test code" and "Test results" tabs are available in the main view (2, 6). A flat list of packages has been replaced by a package tree (1). A "Clouds" link and icon became "Top risks" and "Quick wins" tabs (4).

### Source file view

### Clover Coverage Report

Dashboard

Coverage Reports  

Coverage (Aggregate)

Test Code (Aggregate)

Test Results

#### Application Packages

com.cenqua.samples.money (91,3%)

Classes Tests Results

Class	Coverage
Money	(83,3%)
MoneyBag	(94,4%)


### Clover Coverage Report

Coverage timestamp: Wt lip 1 2014 09:25:15 CEST

Overview Package **File** 1

FRAMES NO FRAMES SHOW HELP

Expand All 4

 **MoneyBag** Line # 16  
5

Show Tests (15) Select All **Deselect All** 6

```

Collapse All 7
1 package com.cenqua.samples.mon
2
3 import java.util.*;
4
5 /**
6  * A MoneyBag defers exchange .
7  * 12 Swiss Francs to 14 US Do
8  * containing the two Monies 1
9  * 10 Swiss francs gives a bag
10 * the deferred exchange rate
11 * MoneyBag with different exc

```





## Coverage Report

Project overview

#### PACKAGES

Type and press enter to filter packages... 

- ▼  com.cenqua
  - >  **samples.money** 91,3%

Project Clover database Wt lip 1 2014 09:25:  
**File MoneyBag.java** 1

#### Coverage histogram 2



#### Classes 4

Class	Line #	Total
MoneyBag	16	70

#### Contributing tests

This file is covered by 15 tests. [Select tests to](#)

## Source view

[Collapse all methods](#) **7**

```
1 package com.cenqua.samp
2
3 import java.util.*;
4
5 /**
6  * A MoneyBag defers ex
7  * 12 Swiss Francs to i
8  * containing the two l
```

The "Overview Package File" links are replaced by the breadcrumbs (1). List of methods of a certain class as well as list of tests hitting the source file are shown in dialogs, which you can open by clicking "Show methods" and "Select tests ... " links (5,6).

## Test results view

### Clover Coverage Report

Dashboard

Coverage Reports  

Coverage (Aggregate)

Test Code (Aggregate)

Test Results 3

#### Application Packages

com.cenqua.samples.money (91,3%)

### Clover Test Report

Coverage timestamp: Wt lip 1 2014 09:25:15 CEST

Overview **Package** 1 2 File Test

FRAMES NO FRAMES SHOW HELP


#### Package

com.cenqua.samples.money.test

#### Test Classes

MoneyBagTest

MoneyTest

 Report generated by Clover Code Coverage v4.0.1  
Pn sie 11 2014 13:33:52 CEST.

#### com.cenqua.samples.money.test

Classes Tests Results


Test Case	% Success
MoneyBagTest	(90,9%)
MoneyTest	(100%)






## Coverage Report

Project overview 3

#### PACKAGES

Type and press enter to filter packages... 

- ▼  com.cenqua
  - ▼  samples.money 91,3%
  -  test 90,6%

Project Clover database Wt lip 1 2014 09:25:

### Package com.cenqua.sam

1 2  
Test code Test results

#### Package

com.cenqua.samples.money.test

#### Test Classes

MoneyBagTest

MoneyTest

Navigation between a test class (source code) and test results (e.g. from unit tests) has been improved - you can simply switch tabs (1, 2). Test results for an entire project are available on the "Project overview" page, under the "Test results" tab (3).

## Clover 3.3 Release Notes

31 March 2014

We are happy to announce **Clover 3.3**.

**Clover 3.3** adds support for the Spock framework and JUnit4's parameterized tests.

Upgrading to Clover 3.3 is free for all customers with active Clover software maintenance at the date of launch and of subsequent updates.

#### Highlights of Clover 3.3:

- Spock framework support
- JUnit4 parameterized tests support
- New toggle in report wizards in Eclipse and IDEA



#### Changes in Clover 3.3.x bug-fix versions

This page describes new features of Clover 3.3. For changes in 3.3.x bug-fix releases, please refer to the relevant documentation:

- Clover-for-Ant | [Changelog](#)
- Clover-for-Maven 2 and 3 | [Changelog](#)
- Clover-for-IDEA | [Changelog](#)
- Clover-for-Eclipse | [Changelog](#)
- Clover-for-Grails | [Changelog](#)

## Highlights of Clover 3.3

### Spock framework support

The [Spock framework](#) is one of the best unit testing frameworks compatible with JUnit and based on the Groovy language. You can write beautiful yet powerful tests, including but not limited to: data series, mocking, behaviour testing and detailed reporting about failed assertions.

#### Automatic detection of Spock's Specifications

Clover 3.3 comes with a new test detector, which handles Spock's Specifications. It means that you will see Specifications on the "Tests" tab and you will no longer have to configure a [custom test pattern](#) to treat them as test code.

#### Handling static test names as declared in the code

In the Spock framework tests are defined using a descriptive caption, such as: `def "this is my test" { ... }` Under the hood, Spock transforms such code into methods named like `"$spock_feature_n_m"`. Clover 3.3 handles these statically-defined names, so you can see them on a method list.

*Screen shot: Clover displays descriptive test name instead of the cryptic "\$spock\_feature\_n\_m".*

Clover Coverage Report				Statistics for f	
Coverage timestamp: Wt mar 11 2014 11:12:07 CET				Stmts: 3	
<a href="#">Overview</a> <a href="#">Package</a> <a href="#">File</a>				Branches: 2	
<a href="#">FRAMES</a> <a href="#">NO FRAMES</a> <a href="#">SHOW HELP</a>				Methods: 2	
				Classes: 2	

Expand All

[-] UnrollWithVarsWithSelectors	Line # 19	Total Statements 2	Complexity 1	TOTAL
#person.name is a #sex.toLowerCase() person	29	2	1	
[-] UnrollWithVarsWithSelectors\$Person	Line # 22	Total Statements 1	Complexity 3	TOTAL
getSex() : String	24	1	3	

```

19  @Unroll
20  class UnrollWithVarsWithSelectors extends Specification {
21  >>
22      static class Person {
23          String name
24      >>
25          String getSex() {
26              name == "Fred" ? "Male" : "Female"
27          }
28      }
29  >>
30  def "#person.name is a #sex.toLowerCase() person"() {
31      expect:
32      person.getSex() == sex
33      println "call: $person.name $person.sex"
34  }
35  where:
36      person || sex
37      new Person(name: "Fred") || "Male"
38      new Person(name: "Wilma") || "Female"
39  }

```

#### Handling runtime test names and the @Unroll annotation

The Spock framework allows to define a data series and to run the same test multiple times using different inputs. But how to detect which iteration has failed among hundreds of test iterations which were run? The Spock comes with a solution – the `@Unroll` annotation appends a sequence number to a test name.

Furthermore, you can define variables or even use Groovy selectors in it. For instance:

```

def "minimum of #a and #b is #c"() { ... }
def "#person.name is a #sex.toLowerCase() person"() { ... }

```

Clover 3.3 tightly integrates with Spock's test runner so that in the HTML report you can see test names exactly as they were evaluated by Spock.

Screen shot: Clover 3.3 knows test names at runtime as evaluated by Spock for each test iteration.

```

19 @Unroll
20 class UnrollWithVarsWithSelectors extends Specification {
21
22     static class Person {
23         String name
24
25         String getSex() {
26             name == "Fred" ? "Male" : "Female"
27         }
28     }
29
30     def "#person.name is a #sex.toLowerCase() person"() {
31         expect:
32         person.getSex() == sex

```

Tests that hit line # 31

Highlight	Test Contribution	Test	Result
↑	<input type="checkbox"/>	UnrollWithVarsWithSelectors.Wilma is a female person	PASS
↑	<input type="checkbox"/>	UnrollWithVarsWithSelectors.Fred is a male person	PASS

## JUnit4 parameterized tests support

Similarly to the Spock support, Clover 3.3 can record runtime test names for JUnit4 test classes annotated with `@Parameterized` annotation. This integration is not automatic, however. See the [Integrating Clover with JUnit4 Parameterized Tests](#) article.

## New toggle in report wizards in Eclipse and IDEA

The 'Show lambda functions' toggle allows to show Java 8 lambda functions in HTML and XML reports.

### HTML Report Configuration

Fine tune the available coverage report options.



Report Title:

Output Directory:  
 ...

Use the current filter settings from this project

Include failed test coverage

thread(s) allocated to report generation

Include source

Show lambda functions:

Note that other Clover integrations (such as with Ant, Maven or Grails) have this feature available since Clover 3.2, but it has to be configured using different options (`showLambdaFunctions` and `showInnerFunctions`).

## Upgrading from Clover 3.2 to Clover 3.3



Clover 3.3 is backward-compatible with Clover 3.2. When upgrading, you may need to delete existing Clover database files due to changes in the database format. In most typical environment configurations it will be done automatically (for instance, by Maven's *clean* goal).

## Clover 3.2 Release Notes

21 October 2013

We are happy to announce **Clover 3.2**.

**Clover 3.2** adds support for Java 1.8 language constructs, including instrumentation of lambda expressions and virtual extension methods (the "default" methods) in interfaces. New language constructs can be shown in HTML, XML, PDF or JSON report.

Upgrading to Clover 3.2 is free for all customers with active Clover software maintenance at the date of launch and of subsequent updates.

### Highlights of Clover 3.2:

- Java 1.8 Support
- Extended HTML and XML reports
- Extended API for plug-in developers
- Dropped support for Eclipse 3.4, 3.5 and RAD 7.5.
- Added support for Eclipse 4.3 (Kepler) and RAD 8.5
- Dropped support for Ant 1.6.x



### Changes in Clover 3.2.x bug-fix versions

This page describes new features of Clover 3.2. For changes in 3.2.x bug-fix releases, please refer to the relevant documentation:

- Clover-for-Ant | [Changelog](#)
- Clover-for-Maven 2 and 3 | [Changelog](#)
- Clover-for-IDEA | [Changelog](#)
- Clover-for-Eclipse | [Changelog](#)
- Clover-for-Grails | [Changelog](#)

## Highlights of Clover 3.2

### Java 1.8 Support


Clover 3.2 now provides support for the [Java 1.8 language features](#) such as lambda functions, method references, virtual extension methods or repeating annotations. It is possible to instrument lambda functions and method references and measure code coverage for them. It is also possible to see lambda functions in HTML and XML reports as well as in a source code editor window in the IntelliJ IDEA (version 12 or later). Code metrics (such as number of statements, number of methods, cyclomatic complexity etc) also take into account lambda expressions.

### Extended HTML and XML reports

In Clover 3.2 you can see lambda functions declared in a method body or as a class field in HTML report as well as see a lambda function signature in a `<line>` tag in XML report. Just use `<clover-report>` task with `showLambdaFunctions=true` and `showInnerFunctions=true` options.

*Screen shot: an HTML report with lambda functions declared inside methods (`showLambdaFunctions=true` and `showInnerFunctions=true`):*

**Clover Coverage Report - JAVA 1.8**  
 Coverage timestamp: Pt paź 18 2013 14:28:54 CEST  
[Overview](#) [Package](#) [File](#)  
 FRAMES NO FRAMES SHOW HELP



44% of files have more coverage

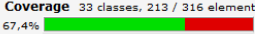
**Statistics for file LambdaInContexts.java:**  
 Stmts: 75 LOC: 209 Total cmp: 49 Stmts/Method: 2,21  
 Branches: 24 NCLOC: 105 Cmp density: 0,65 Methods/Class: 6,8  
 Methods: 34 Avg method cmp: 1,44  
 Classes: 5

Expand All

LambdaInContexts	Line #	Total Statements	Complexity	TOTAL Coverage	
lambdaInVariableDeclarationAndAssignment() : void	43	9	8	64,7%	<div style="width: 64.7%; height: 10px; background-color: #28a745;"></div>
\$lam_x#0(<inferred>)	44	1	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
\$lam#1()	52	1	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
\$lam_n#2(<inferred>)	55	1	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
produceN(Produce<Integer[]>.int) : Integer[]	59	1	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
lambdaInReturnStatement() : Map<Integer>	67	2	2	50%	<div style="width: 50%; height: 10px; background-color: #28a745;"></div>
\$lam_x#3(<inferred>)	68	1	1	0%	<div style="width: 0%; height: 10px; background-color: #28a745;"></div>
lambdaAsMethodArgument() : void	79	7	4	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
\$lam#4()	81	2	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
\$lam#5()	84	1	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
\$lam#6()	87	1	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
print(Printer) : void	90	1	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
lambdaInLambda() : void	101	6	4	66,7%	<div style="width: 66.7%; height: 10px; background-color: #28a745;"></div>
\$lam#7()	103	2	2	50%	<div style="width: 50%; height: 10px; background-color: #28a745;"></div>
\$lam#8()	103	1	1	0%	<div style="width: 0%; height: 10px; background-color: #28a745;"></div>
call() : Integer	120	1	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
lambdaInTernaryExpression(boolean) : void	113	9	7	61,5%	<div style="width: 61.5%; height: 10px; background-color: #28a745;"></div>
\$lam#9()	116	1	1	0%	<div style="width: 0%; height: 10px; background-color: #28a745;"></div>
\$lam#10()	116	1	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
\$lam#11()	119	1	1	0%	<div style="width: 0%; height: 10px; background-color: #28a745;"></div>
lambdaWithCastExpression() : void	135	12	5	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
\$lam#12()	140	1	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
\$lam#13()	144	1	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
\$lam#14()	148	1	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>
\$lam#15()	152	1	1	100%	<div style="width: 100%; height: 10px; background-color: #28a745;"></div>

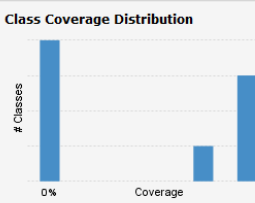
Screen shot: a dashboard showing a lambda function in the "Least Tested Methods" section (showLambdaFunctions=true):

**Coverage** 33 classes, 213 / 316 elements  
67.4%



**Test Results** 0 / 0 tests 0 secs  
No test results could be found. Please ensure that you have instrumented your unit tests correctly.

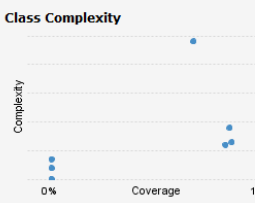
**Class Coverage Distribution**




**Top 14 Project Risks**

- AnnotationsOnJavaTypes LambdaInContexts
- RepeatingAnnotations LambdaOneLinersAndBlocks
- LambdaAndMethodReferences
- LambdaArgumentAndReturnTypes
- RepeatingAnnotations.Bits RepeatingAnnotations.Bit
- AnnotationsOnJavaTypes.ReadOnly AnnotationsOnJavaTypes.NonNull AnnotationsOnJavaTypes.NonEmpty
- AnnotationsOnJavaTypes.UnmodifiableList AnnotationsOnJavaTypes.TemperatureException
- AnnotationsOnJavaTypes.Critical

**Class Complexity**



**Coverage Tree Map**



**Most Complex Packages**

- 67.4% default-pkg (113)

**Least Tested Methods**

- 0% lambdaVsOperatorPriority.\$lam\_y#16(<inferred>) (2)
- 0% lambdaVsOperatorPriority.\$lam\_y#18(<inferred>) (2)
- 0% AnnotationsOnJavaTypes.annotationInTypeCast(Object) : void (1)
- 0% AnnotationsOnJavaTypes.annotationInObjectCreation() : void (1)
- 0% lambdaInReturnStatement.\$lam\_x#3(<inferred>) (1)
- 0% \$lam#7.\$lam#8() (1)
- 0% lambdaInTernaryExpression.\$lam#9() (1)
- 0% lambdaInTernaryExpression.\$lam#11() (1)

**Most Complex Classes**

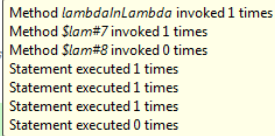
- 66.2% LambdaInContexts (49)
- 83.9% LambdaArgumentAndReturnTypes (19)
- 84.1% LambdaOneLinersAndBlocks (14)
- 81.4% LambdaAndMethodReferences (13)
- 0% AnnotationsOnJavaTypes (5)

Screen shot: code coverage highlighting and tool-tip markers for lambda functions in IntelliJ IDEA 12

```

/**
 * Lambda expression bodies, for which the target_type is the type expected for the body, which is derived in
 * turn from the outer target type.
 *
 * Callable<Runnable> has "Runnable call() throws
 * Runnable has "void run()"
 */
public void lambdaInLambda() {
    try {
        Callable<Runnable> call = () -> () -> { System.out.println("Callable calls Runnable which calls run"); };
        System.out.println("lambdaInLambda " + call.call());
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```



## Extended API for plug-in developers

Clover 3.2 comes with an extended Clover API (and even more will come in next minor releases). It allows to navigate more easily through the Clover database structure. See <https://docs.atlassian.com/atlassian-clover/latest/JavaDoc> documentation.

## Dropped support for Eclipse 3.4, 3.5 and RAD 7.5.

Due to fact that Eclipse 3.6 is the lowest Eclipse version officially tested on JDK1.5, we decided to drop support for older Eclipse versions. Clover 3.2 may still work on them, but it is not guaranteed.

## Added support for Eclipse 4.3 (Kepler) and RAD 8.5

The latest Eclipse and RAD versions available at the date of the Clover 3.2 release are supported.

Please note that Eclipse 4.3 does not support Java 8 yet - the most probably it will be available in Eclipse 4.4.

## Dropped support for Ant 1.6.x

Clover 3.2.0 does no longer support the Ant 1.6.x officially. It may still work with Ant 1.6.x, but it is not guaranteed.

## Upgrading from Clover 3.1 to Clover 3.2

Clover 3.2 is generally backward-compatible with Clover 3.1, and the migration should be straightforward.

However, there are few issues breaking the backward compatibility:

### 1) CloverInstr

Deprecated command line options `-jdk14`, `-jdk15` and `-jdk17` have been removed. Use the `--source=1.x` instead of this.

### 2) HTMLReporter

Deprecated command line option `-nu/--nunique` has been removed. Use `--showunique/-su` in order to see unique per-test coverage in the report. Deprecated option `-r/--resultsdir` has been removed.

### 3) PDFReporter

The `--outfile` option was renamed to `--outfile` to make it consistent with the XMLReporter.

4) Clover classes were renamed from **com.cenqua.\*** to **com.atlassian.\***.

It affects all classes in Clover core (`com.cenqua.clover.clover`). This class rename should be transparent to build scripts (Ant / Maven / Grails) unless your build scripts or tools are referencing these classes directly. In such case rename references as follows:

- Ant tasks - from `com.cenqua.clover.tasks.*` to `com.atlassian.clover.ant.tasks.*`
- Ant types - from `com.cenqua.clover.types.*` to `com.atlassian.clover.ant.types.*`
- CloverInstr, CloverMerge - from `com.cenqua.clover.*` to `com.atlassian.clover.*`
- HtmlReporter, XMLReporter, JSONReporter, PDFReporter - from `com.cenqua.clover.reporters.*` to `com.atlassian.clover.reporters.*`

- SnapshotPrinter - from `com.cenqua.clover.test.optimization.SnapshotPrinter` to `com.atlassian.clover.optimization.SnapshotPrinter`.

## Clover 3.1 Release Notes

31 May 2011

With great pleasure Atlassian presents **Clover 3.1**.

**Clover 3.1** adds support for Java 1.7 language constructs, Maven 3, Groovy 1.7 & 1.8, Grails 1.3.7, Eclipse 3.6 and 3.7 and IntelliJ IDEA 10.5 and 11.

Upgrading to Clover 3.1 is free for all customers with active Clover software maintenance at the date of launch and of subsequent updates.

### Highlights of Clover 3.1:

- Java 1.7 Support
- Maven 3 Support
- Groovy 1.6-1.8 Support for Ant, Maven 2 and Grails
- Clover-for-Grails Plugin for Grails 1.3.7
- Support for Eclipse 3.6 and 3.7
- Support for IntelliJ 10.5 and 11
- Bug fixes and improvements



### Changes in Clover 3.1.x bug-fix versions

This page describes new features of Clover 3.1. For changes in 3.1.x bug-fix releases, please refer to the relevant documentation:

- Clover-for-Ant | [Changelog](#)
- Clover-for-Maven 2 and 3 | [Changelog](#)
- Clover-for-IDEA | [Changelog](#)
- Clover-for-Eclipse | [Changelog](#)
- Clover-for-Grails | [Changelog](#)

## Highlights of Clover 3.1

1

### Java 1.7 Support

Clover 3.1 now provides support for the [Java 1.7 language features](#) including try resource blocks, multi-catch statements, diamond generics syntax, binary numeric literals and numeric literals with underscores.

2

### Maven 3 Support

Clover 3.1 now supports recording and reporting code coverage in all your Maven 3 projects. Check out the [Clover-for-Maven](#) plugin.

## 3

## Groovy 1.6-1.8 Support for Ant, Maven 2 and Grails

Ant and Maven 2 plugins now provide support for Groovy 1.6 through to 1.8. Most Clover-for-Ant tasks will work on Groovy code and the Clover-for-Maven 2 plugin now supports Groovy code compilation and report generation.

**Read more:** [Clover-for-Ant Upgrade Guide](#) and [Upgrade Notes for Clover-for-Maven 2 Groovy Integration](#).

**i** Unless otherwise indicated, all tasks described in the [Clover-for-Ant User's Guide](#) work with Groovy code.

### Groovy Code Coverage Reporting

Clover's reporting features support Groovy code, which includes [per-test coverage](#) and other reporting features available in [Ant](#) and [Maven 2](#).

Furthermore:

- Clover will only report a line that contains Groovy's safe operator as covered if the check evaluated to both true and false.
- Clover also supports filtering specified Groovy methods.

*Screenshot: Clover Groovy Code Coverage*

```

104      }
105  81      if (!spaceEntries) {
106      0          log.warn "No permissions set for space with alias [$spaceAlias], and no default space per
107      0          return false // No permissions set for the "any" space
108      }
109      }
110
111      // If the uri is null this means we are checking for permissions to access the SPACE rather than
112      // eg we use the default permissions for the space
113  143     if (uri == null) {
114      7         uri = DEFAULT_POLICY_URI
115     }
116
117  143     if (!uri.endsWith('/')) {
118  113         uri += '/'
119     }
120
121     // Assume these are sorted in descending order so that we find longest matches first
122  143     def uriPermCandidates = (spaceEntries?.findAll { k, v ->
123  234         (k == DEFAULT_POLICY_URI) || uri.startsWith(k)
124     })
125
126  143     if (log.debugEnabled) {
127      0         log.debug "Found policy permissions that could apply for uri [$uri]: $uriPermCandidates"
128     }
129
130  143     def explicitMatch
131  143     uriPermCandidates*.value.find { uriPerms ->
132  148         roleList.find { role ->
133  153             def permsForRole = uriPerms?.get(role)
134
135  153             def explicitGrant
136  153             permissionList.each { permission ->
137  167                 def grant = permsForRole?.get(permission)
138  167                 if (grant != null) {
139  113                     if (explicitGrant != null) {
140  14         explicitGrant &= grant
141                 } else {
142  99         explicitGrant = grant
143                 }
144             }
145         }
146     }

```

Line 122, Col 34: true branch executed 143 times, false branch executed 0 times.

*Screenshot: Clover Filtering Specified Groovy Methods*

```
10 14  Map<String, IMoney> pocket = [:]
11
12  def insert(int amt, String currency) {
13     final IMoney existingMoney = pocket[currency]
14     final Money newMoney = new Money(amt, currency)
15     if (existingMoney) {
16         pocket[currency] = existingMoney.add(newMoney);
17     } else {
18         pocket[currency] = newMoney;
19     }
20 }
21
```

## 4

### Clover-for-Grails Plugin for Grails 1.3.7

Clover 3.1 incorporates a plugin for the [Grails web application development framework](#) supporting Grails version 1.3.7. Grails project developers can test their Groovy code using Clover to generate coverage reports.

The Clover-for-Grails plugin is very easy to install and upgrade, with multiple installation options that can be issued from a single Grails command. Upgrading is as easy as reinstalling the Clover-for-Grails plugin.

You can configure the Clover-for-Grails plugin on the command line or by including Clover-for-Ant-based ([Gant](#)) instructions directly inside the `BuildConfig.groovy` file.

**Read more:** [Clover-for-Grails](#), [Clover-for-Grails Installation Guide](#)

*Screenshot: Clover Report Dashboard of a Grails Project*

**Clover Coverage Report**  
 Dashboard  
 Coverage Reports  
 Coverage (Aggregate)  
 Test Code (Aggregate)  
 Test Results

**Application Packages**  
 com.jcatalog.wiki (0%)  
 com.jcatalog.wiki.block (0%)  
 org.weceem.blog (70.6%)  
 org.weceem.content (87.6%)  
 org.weceem.controllers (11.4%)  
 org.weceem.css (71.4%)  
 org.weceem.event (-)  
 org.weceem.export (68.5%)  
 org.weceem.files (29.5%)  
 org.weceem.html (87%)  
 org.weceem.jobs (8.3%)  
 org.weceem.js (69.2%)  
 org.weceem.security (88.3%)

**Classes Tests Results**

Class	Coverage
AccessDeniedException	(0%)
AdminTagLib	(20%)
Blog	(66.7%)
BlogEntry	(73.7%)
CacheService	(18%)
CacheTagLib	(11.6%)
Comment	(80%)
ConfluenceSpaceImporter	(79.3%)
Content	(94%)
ContentController	(7.1%)
ContentDirectory	(37.9%)
ContentFile	(24.8%)
ContentRepositoryService	(48.3%)
ContentVersion	(100%)
ContentVersionService	(0%)
DefaultSpaceExporter	(0%)
DefaultSpaceImporter	(69.3%)
DiffUtils	(0%)
DomainConverter	(0%)
EditorController	(9.7%)
EditorFieldTagLib	(27.4%)
EditorsService	(89.8%)
EditorsBuilder	(100%)
EventsService	(100%)
ExternalLink	(54.5%)
ExternalLinksJob	(8.3%)
Folder	(100%)
HTMLContent	(87%)
ImportException	(0%)
ImportExportConverter	(52.9%)
ImportExportService	(61.9%)
JavaScript	(69.2%)
ParagraphBlock	(0%)
PortalController	(54.5%)
RelatedContent	(25%)
RepositoryController	(13.6%)
SAXConfluenceParser	(97.8%)
SecurityPermissionsBuilder	(93.3%)
SecurityPolicyBuilder	(100%)

**Clover Coverage Report -**  
 Coverage timestamp: Mon Feb 15 2010 15:29:05 EST  
 Overview Package File  
 FRAMES NO FRAMES SHOW HELP

**Statistics for project Clover database Mon F**  
 Stmt: 2,700 LOC: 7,139  
 Branches: 1,344 NCLOC: 74  
 Methods: 510 Files: 60 Avg me  
 Classes: 61 Packages: 17

**Coverage** 61 classes, 1,919 / 4,554 elements  
 42.1%

**Test Results** 58 / 58 tests 8.86 secs  
 100%

**Class Coverage Distribution**  
 # Classes vs Coverage

**Class Complexity**  
 Complexity vs Coverage

**Most Complex Packages**

- 52.1% org.weceem.services (560)
- 11.4% org.weceem.controllers (424)
- 25.7% org.weceem.tags (342)
- 68.5% org.weceem.export (195)
- 87.6% org.weceem.content (101)

**Most Complex Classes**

- 48.3% ContentRepositoryService (444)
- 29.5% WeceemTagLib (251)
- 13.6% RepositoryController (226)
- 94% Content (58)
- 79.7% SimpleSpaceImporter (54)

**Top 20 Project Risks**  
 DiffUtils WidgetTagLib ParagraphBlock ContentController SynchronizationController RepositoryController DefaultSpaceExportContentRepositoryService ContentDirectory WikiItemRenderEditorController DefaultSpaceImporter SpaceController ContentCacheService WeceemDialect

**Coverage Tree Map**  
 org.weceem.services org.weceem.controllers org.weceem.export org.weceem.tags

**Least Tested Methods**

- 0% ContentController.field show() (42)
- 0% WeceemTagLib.field menu() (49)
- 0% ContentRepositoryService.createContentFile(def) : def (31)
- 0% RepositoryController.field searchRequest() (31)
- 0% WidgetTagLib.field widget() (29)
- 0% DefaultSpaceExporter.execute(Space) : File (7)
- 0% ContentRepositoryService.updateNode(Content, def) : def (25)
- 0% RepositoryController.field beforeInterceptor() (19)
- 0% DiffUtils.diffString(def, def) : def (15)
- 0% SynchronizationController.field linkContentFile() (13)
- 0% ContentDirectory.deleteContent() : Boolean (17)
- 0% SynchronizationController.field deleteContent() (13)
- 0% DiffUtils.diff(def, def) : def (11)
- 0% ContentRepositoryService.synchronizeSpace(def) : def (13)
- 0% WeceemTagLib.field eachSibling() (17)
- 0% RepositoryController.field uploadFile() (15)
- 0% CacheTagLib.field cache() (13)
- 0% CacheService.getOrPut(def, boolean, def, def) : def (13)
- 0% WeceemTagLib.field humanDate() (11)
- 0% WeceemTagLib.field breadcrumb() (11)

5

Support for Eclipse 3.6 and 3.7

The Clover-for-Eclipse plugin now supports Eclipse 3.6 and 3.7. Eclipse does not yet support Java 1.7 language features but Clover-for-Eclipse is ready once it does.

6

Support for IntelliJ 10.5 and 11

The Clover-for-IDEA plugin now supports IntelliJ 10.5 and 11 including support for Java 1.7 language features.

Screenshot: Clover-for-IDEA with a project using Java 1.7 language features



Element	Coverage	Cplx	Lines	#Uncovered
default-pkg	32.3%	11	40	21
Foo	25.9%	8	40	20
foo() : void	100%	1	3	0
main(String[]) : void	16.7%	7	31	20
Foo.A	100%	1	1	0
close() : void	-	1	1	0
Foo.B	66.7%	2	1	1

# 7

## Bug fixes and improvements

See the [changelog](#) for details.

## Clover 3.0 Release Notes

31 March 2010

### Atlassian presents Clover 3.0

**Clover 3.0** is a major release which adds Groovy support for Ant, Maven 2 and Grails. We also offer the new Clover-for-Grails plugin that allows you to use Clover's code coverage capabilities directly inside your Grails project. Furthermore, we add a new per-test coverage viewer and completely new dashboard view to Eclipse.

Upgrading to Clover 3.0 is free for all customers with active Clover software maintenance at the date of launch.

#### Highlights of Clover 3.0:

- Groovy Support for Ant, Maven 2 and Grails



- New Clover-for-Grails Plugin
- Per-Test Coverage Viewer for Eclipse
- New Dashboard View in Eclipse
- Updated Tutorial with Groovy Code
- Other Enhancements and Improvements
- Over 50 bug fixes and improvements

**Thank you for your feedback:**

Your votes and issues help us keep improving our products, and are much appreciated.

**Upgrading to Clover 2.6**

Clover 2.6 can be downloaded from the [Clover Download Centre](#). Before upgrading, please refer to the relevant documentation:

- [Clover-for-Ant | Changelog](#)
- [Clover-for-Maven 2 and 3 | Changelog](#)
- [Clover-for-IDEA | Changelog](#)
- [Clover-for-Eclipse | Changelog](#)

**Highlights of Clover 3.0****Groovy Support for Ant, Maven 2 and Grails**

Clover 3.0 Ant and Maven 2 plugins now provide support for [Groovy](#). Most [Clover-for-Ant](#) tasks will now work on Groovy code and the [Clover-for-Maven 2](#) plugin now supports Groovy code compilation and report generation.

**Read more:** [Clover-for-Ant Upgrade Guide](#) and [Upgrade Notes for Clover-for-Maven 2 and 3 Groovy Integration](#).

**i** Unless otherwise indicated, all tasks described in the [Clover-for-Ant User's Guide](#) work with Groovy code.

**Groovy Code Coverage Reporting**

Clover's reporting features support Groovy code, which includes [per-test coverage](#) and other reporting features available in [Ant and Maven 2](#).

Furthermore:

- Clover will only report a line that contains Groovy's safe operator as covered if the check evaluated to both true and false.
- Clover also supports filtering specified Groovy methods.

[Screenshot: Clover Groovy Code Coverage](#)

```

104     }
105 81    if (!spaceEntries) {
106 0     log.warn "No permissions set for space with alias [$spaceAlias], and no default space per
107 0     return false // No permissions set for the "any" space
108     }
109     }
110
111     // If the uri is null this means we are checking for permissions to access the SPACE rather than
112     // eg we use the default permissions for the space
113 143   if (uri == null) {
114 7     uri = DEFAULT_POLICY_URI
115     }
116
117 143   if (!uri.endsWith('/')) {
118 113   uri += '/'
119     }
120
121     // Assume these are sorted in descending order so that we find longest matches first
122 143   def uriPermCandidates = (spaceEntries?.findAll { k, v ->
123 234   (k == DEFAULT_POLICY_URI) || uri.startsWith(k)
124     })
125
126 143   if (log.debugEnabled) {
127 0     log.debug "Found policy permissions that could apply for uri [$uri]: $uriPermCandidates"
128     }
129
130 143   def explicitMatch
131 143   uriPermCandidates*.value.find { uriPerms ->
132 148   roleList.find { role ->
133 153   def permsForRole = uriPerms?.get(role)
134
135 153   def explicitGrant
136 153   permissionList.each { permission ->
137 167   def grant = permsForRole?.get(permission)
138 167   if (grant != null) {
139 113   if (explicitGrant != null) {
140 14   explicitGrant &= grant
141   } else {
142 99   explicitGrant = grant
143   }
144   }
145     }
146     }

```

Line 122, Col 34: true branch executed 143 times, false branch executed 0 times.

Screenshot: Clover Filtering Specified Groovy Methods

```

10 14  Map<String, IMoney> pocket = [:]
11
12  def insert(int amt, String currency) {
13     final IMoney existingMoney = pocket[currency]
14     final Money newMoney = new Money(amt, currency)
15     if (existingMoney) {
16         pocket[currency] = existingMoney.add(newMoney);
17     } else {
18         pocket[currency] = newMoney;
19     }
20 }
21

```

Filtered by: insert

## 2

### New Clover-for-Grails Plugin

Clover 3.0 now incorporates a new plugin for the [Grails web application development framework](#). Grails project developers can now test their Groovy code using Clover to generate coverage reports.

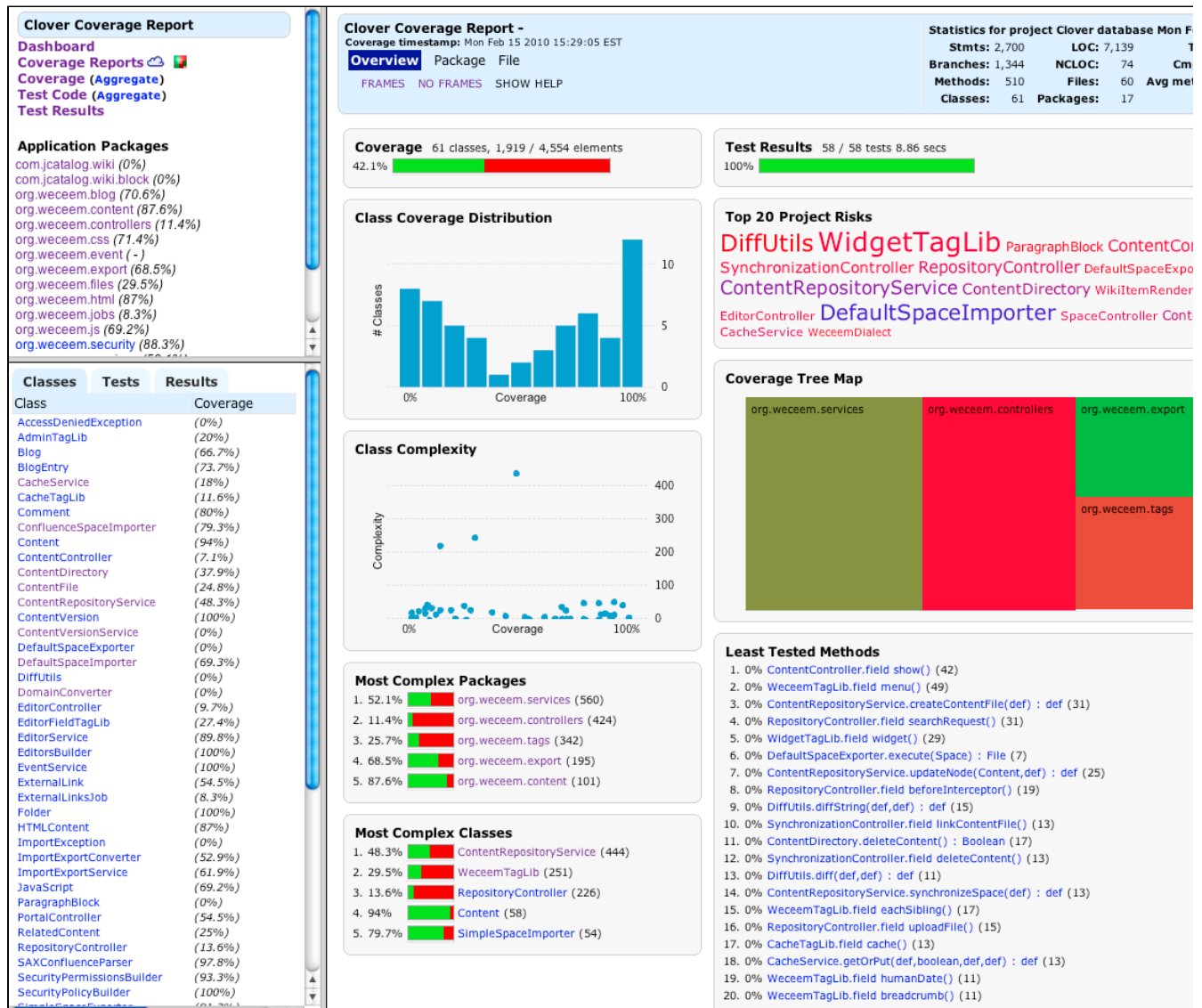
The Clover-for-Grails plugin is very easy to install and upgrade, with multiple installation options that can be issued from a single Grails command. Upgrading is as easy as reinstalling the Clover-for-Grails plugin.

You can configure the Clover-for-Grails plugin on the command line or by including

Clover-for-Ant-based (Gant) instructions directly inside the `BuildConfig.groovy` file.

**Read more:** [Clover-for-Grails, Clover-for-Grails Installation Guide](#)

*Screenshot: Clover Report Dashboard of a Grails Project*



3

## Per-Test Coverage Viewer for Eclipse

The Clover-for-Eclipse plugin includes a new inline per-test coverage viewer. Hovering over the gutter now displays a pop-up with a list of tests that hit that line. Clicking on a test will take you directly to the test source code.

*Screenshot: New Clover-for-Eclipse per-test coverage pop-up*

```

26
27 6 public IMoney add(IMoney m) {
28 6     return m.addMoneyBag(this);
29 6 }
Lines 27-29: Method executed 6 times.
Covered by 6 test runs
30
31 4 com.cenqua.samples.money.MoneyBagTest
32 4   testBagSumAdd
33 4   testBagSimpleAdd
34   testNormalize2
35 4   testIsZero
36 4   testNormalize3
37 4   testBagSubtract
38
39 12
40 12
41 24     appendMoney((Money) e.nextElement());
42 12 }
43

```

Double-click the test name

## 4

### New Dashboard View in Eclipse

For the first time, the Clover Report Dashboard is now available in Eclipse!

The screenshot shows the Eclipse IDE with the Clover Dashboard open. The dashboard provides a comprehensive overview of the project's test coverage and quality metrics.

**Coverage** 3 classes, 160 / 172 elements  
93%

**Test Results** 24 / 24 tests 3.02 secs  
100%

**Most Complex Packages**  
1. 93% com.cenqua.samples.money (54)

**Most Complex Classes**  
1. 94.4% MoneyBag (36)  
2. 89.4% Money (18)

**Top 2 Project Risks**  
**Money MoneyBag**

**Least Tested Methods**  
1. 61.5% Money.equals(Object) : boolean (4)  
2. 72% MoneyBag.equals(Object) : boolean (7)

## 5

## Updated Tutorial with Groovy Code

The existing 'Money Demo' tutorial (located in `CLOVER_HOME/tutorial`) has been updated with additional Groovy code for Ant and Maven 2. The Ant (`build_completed.xml`) and Maven 2 (`pom_completed.xml`) tutorial solution files contain examples of Groovy integration.

For Ant builds, your Groovy code will automatically be compiled if the `GROOVY_HOME` environment variable has been set and points to the location of your Groovy directory.

**Read more:** [Clover-for-Ant tutorial](#)

## 6

## Other Enhancements and Improvements

### Clover-for-Maven 2

The `clover2:check` task now has two new parameters: `methodPercentage` and `conditionalPercentage`. These parameters allow you to set target percentages that define when to fail the build for methods and conditions, respectively.

**i** Be aware that these will only be used if `maven.clover.targetPercentage` has been set.

For more information, please refer to the [Clover-for-Maven 2 and 3 User's Guide](#).

### Clover XML Reports

The Clover XML report now includes test results and Clover ships with an XML schema (XSD) for these reports.

## 7

## Over 50 bug fixes and improvements

See the [changelog](#) for details.

## Clover 2.6 Release Notes

**9 September 2009**

### Atlassian presents Clover 2.6

**Clover 2.6** is a major release which adds an affordable Desktop Edition to the Eclipse and IDEA plugins. It also brings a new coverage filter, auto-update functionality in IDEA, adds a visual treemap report to Clover-for-Ant and Maven2, as well as significant performance enhancements for Eclipse users.

Upgrading to Clover 2.6 is free for all customers with active Clover software maintenance at date of launch.

#### Highlights of Clover 2.6:

- [New Clover Editions](#)
- [Eclipse Plugin Performance Improvements](#)
- [IDE Plugin Ease-of-Use Features](#)
- [100% Coverage filter](#)
- [New HTML Tree Map](#)
- [New API for Optimizing Tests Programmatically](#)
- [Clover Auto-Update Feature for IDEA](#)
- [Over 80 bug fixes and improvements](#)

**Thank you for your feedback:**

Your votes and issues help us keep improving our products, and are much appreciated.

**Upgrading to Clover 2.6**

Clover 2.6 can be downloaded from the [Clover Download Centre](#). Before upgrading, please refer to the relevant documentation:

- [Clover-for-Ant | Changelog](#)
- [Clover-for-Maven 2 and 3 | Changelog](#)
- [Clover-for-IDEA | Changelog](#)
- [Clover-for-Eclipse | Changelog](#)

**Highlights of Clover 2.6****1****New Clover Editions**

Clover licences now add an affordable Desktop Edition to the line-up. Clover Desktop Edition is for individual developers and provides code coverage analysis and also the use of test optimization for developers working in isolation. Clover Server edition contains the full suite of report generation and Continuous Integration features for coding in a team environment.

**Read more:** [About Clover Editions](#)

**2****Eclipse Plugin Performance Improvements**

The Clover database format has been completely rewritten to make running Clover in an IDE as fast as possible. Large code bases, such as JIRA and Confluence, have shown a significant reduction in the time it takes to compile source code and run tests with Clover enabled - so much so that you will hardly know that Clover is there!

**3****IDE Plugin Ease-of-Use Features**

A new context menu has been added to the package explorer to make including or excluding files and packages quick and easy. For large projects, this makes it possible to turn Clover on or off for a specific package or class, run the tests, view the coverage and then turn Clover off again.

**Read more:** [Clover-for-IDEA User's Guide](#)

**4****100% Coverage filter**

When some of your files have reached 100% coverage, you can remove them from view using this new filter. This allows you to reduce the clutter in the display and see only those files that are not completely covered by unit tests.

**Read more:** [Clover-for-IDEA User's Guide](#)



5

## New HTML Tree Map

The popular Tree Map feature from Clover for Eclipse and Clover for IDEA is now available in Clover for Ant and Clover for Maven, showing a geometric view of coverage in coloured squares, with size indicating the scale of each file. The new HTML Tree Map allows you to easily spot not only poorly tested, large classes but also identify clusters of untested code.

**Read more:** [Treemap Charts](#)

6

## New API for Optimizing Tests Programatically

This new addition to Clover will allow you to take advantage of Clover's Test Optimization feature, only running those tests for which code has changed, even if you are using a testing framework other than Junit or TestNG.

**Read more:** [Clover Development Hub](#)

7

## Clover Auto-Update Feature for IDEA

Clover for IDEA will now automatically check for new version updates. This allows you to easily stay on the leading edge of Clover's latest features. When a new version is available, a Clover icon appears in the status bar, allowing you to install it by clicking.

**Read more:** [Clover-for-IDEA Auto-Updates](#)

8

## Over 80 bug fixes and improvements

See the [changelog](#) for details.

## Clover 2.5 Release Notes

11 May 2009

### Atlassian presents Clover 2.5

**Clover 2.5** is a major release with a key new feature called Distributed Per-Test Coverage, which will allow you to optimize your functional tests. It also brings the breakthrough recent feature Test Optimization to the Eclipse and IntelliJ IDEA development environments.

Upgrading to Clover 2.5 is free for all customers with active Clover software maintenance at date of launch.

#### Highlights of Clover 2.5:

- Test Optimization in Clover for Eclipse and IDEA
- Distributed Per-Test Coverage
- Performance Improvements
- Over 20 bug fixes and improvements

#### Thank you for your feedback:

*Your votes and issues help us keep improving our products, and are much appreciated.*



### Upgrading to Clover 2.5

Clover 2.5 can be downloaded from the [Clover Download Centre](#). Before upgrading, please refer to the relevant documentation:

- [Clover-for-Ant | Changelog](#)
- [Clover-for-Maven 2 and 3 | Changelog](#)
- [Clover-for-IDEA | Changelog](#)
- [Clover-for-Eclipse | Changelog](#)

## Highlights of Clover 2.5

1

### Test Optimization in Clover for Eclipse and IDEA

Clover's much-lauded feature called Test Optimization is now available in Clover for Eclipse and Clover for IntelliJ IDEA, bringing the ability to run only the tests that have been affected by changes to program code. In many cases this will cut down the running time of your test phases and allow you to run them far more often. This is in sharp contrast with the traditional 'shotgun testing' which indiscriminately runs every single test, regardless of whether any code within has changed.

**Read more:** [About Test Optimization](#)

2

### Distributed Per-Test Coverage

With the new Distributed Per-Test Coverage feature, Clover now has the ability to record per-test coverage from tests that are running in separate test JVMs, which may be co-sited or distributed around a network. This allows you to roll together results from unit and functional tests, from JVMs running different test frameworks, possibly in remote locations, yet resulting in a single unified view of your project's code coverage.

This feature also allows you to run Clover's famous Test Optimization on your functional tests. A battery of functional tests (being generally more time-consuming than unit tests) strongly benefits from the ability to run only the tests on code which has changed.

**Read more:** [About Distributed Per-Test Coverage](#)

3

### Performance Improvements

Clover 2.5 is significantly faster than previous versions, supporting a range of performance-enhancing settings. Clover can now be configured to gather information only at the level at which you need it, making it faster.

**Read more:** [Clover Performance Tuning](#)

4

### Over 20 bug fixes and improvements

See the [changelog](#) for details.

## Clover 2.4 Release Notes

**5 November 2008**



## Atlassian presents Clover 2.4

**Clover 2.4** is a major release with a significant new feature called Test Optimization, along with a number of improvements and bug fixes.

Upgrading to Clover 2.4 is free for all customers with active Clover software maintenance at date of launch.

### Highlights of Clover 2.4:

- Test Optimization
- Easier Integration
- Reporting Improvements
- Clover for IDEA
- Over 35 bug fixes and improvements

### Thank you for your feedback:

*Your votes and issues help us keep improving our products, and are much appreciated.*



### Upgrading to Clover 2.4

Clover 2.4 can be downloaded from the [Clover Download Centre](#). Before upgrading, please refer to the relevant documentation:

- [Clover-for-Ant | Changelog](#)
- [Clover-for-Maven 2 and 3 | Changelog](#)
- [Clover-for-IDEA | Changelog](#)
- [Clover-for-Eclipse | Changelog](#)

## Highlights of Clover 2.4

1

### Test Optimization

In the new Test Optimization feature, Clover now has the ability to optimise test runs, which greatly reduces the time taken to test a code change. Traditionally, the full suite of tests is run whenever a small code change is made. Now, for a given edit, Clover works out the optimal subset of tests that will exercise that change. Running the optimal subset is in general dramatically faster than running the full test suite. This means that developers are more likely to run tests prior to committing, and Continuous Integration servers can get through far more build and test cycles. This means faster feedback to developers when their code changes break tests.

Read more: [Ant](#), [Maven2](#)

2

### Easier Integration

Integration with Ant is now as simple as adding:

```
<taskdef resource="cloverlib.xml" classpath="/path/to/clover.jar"/>
<clover-env/>
```

to your build.xml and ensuring that the clover.jar is on your test classpath. `<clover-env/>` automati

cally defines all the targets you need to have Clover seamlessly integrated with your build.

The clover-maven2-plugin can be run outside the forked-lifecycle - removing the need to have unit tests run twice.

3

## Reporting Improvements

New columns have been added: **PercentageCoveredContribution** and **PercentageUncoveredContribution** - showing the percentage of coverage a particular class contributes to the overall project. Also the new **FilteredElements** column shows the amount of code that has been filtered from a coverage report. Custom contexts, allowing arbitrary blocks of code to be filtered from reports, are now supported by the maven-clover2-plugin. Historical Charts will now auto-scale depending on the data and have had a color change - making them look less like something rendered in a 1986 era arcade game.

4

## Clover for IDEA

Clover2 for IDEA is now fully featured and final. Easily find where to add your next test using Clover2's per-test coverage. Spot potential project risks by using the 'coverage clouds' and 'coverage tree map' visualisations - directly in your IDE. **ALT-F1** to select the current class you are viewing in the Coverage Explorer.

5

## Over 35 bug fixes and improvements

See the [changelog](#) for details.

## Clover 2.3 Release Notes

9 May 2008

## Atlassian presents Clover 2.3

This is a release with a number of new features and bug fixes.

Upgrading to Clover 2.3 is free for all customers with active Clover software maintenance at date of launch.

### Highlights of Clover 2.3:

- New options for Movers report
- New <added> tag
- 15 fixes and revisions

### Thank you for your feedback:

*Your votes and issues help us keep improving our products, and are much appreciated.*



### Upgrading to Clover 2.3

Clover 2.3 can be downloaded from the [Clover Download Centre](#). Before upgrading, please refer to the relevant documentation:

- [Clover-for-Ant Upgrade Guide | Changelog](#)

## Highlights of Clover 2.3

1

### New options for Movers report

More options are now available for generation of the Clover 'movers' report. Users can now define arbitrary metrics to detect movers.

[Read more.](#)

2

### New <added> tag

This new tag is for viewing coverage of newly added classes. It's now possible to keep track of classes newly added to your project, via the historical report.

3

### 15 fixes and revisions

See the [Changelog](#) for details.

## Clover 2.2 Release Notes

10 April 2008

## Atlassian presents Clover 2.2

Clover 2.2 allows you to improve your coverage reporting with stack trace navigation, better cross-referencing and Dashboard visualisations. The IDE and build-tool variants of Clover will also see a prominent upgrade on release.

Upgrading to Clover 2.2 is free for all customers with active Clover software maintenance at date of launch.

### Highlights of Clover 2.2:

- New visualisations for Dashboard
- Stack trace navigation in reports
- Better cross-referencing in reports
- Don't go backwards

### Thank you for your feedback:

*Your votes and issues help us keep improving our products, and are much appreciated.*



**Upgrading to Clover 2.2**

Clover 2.2 can be downloaded from the [Clover Download Centre](#). Before upgrading, please refer to the relevant documentation:

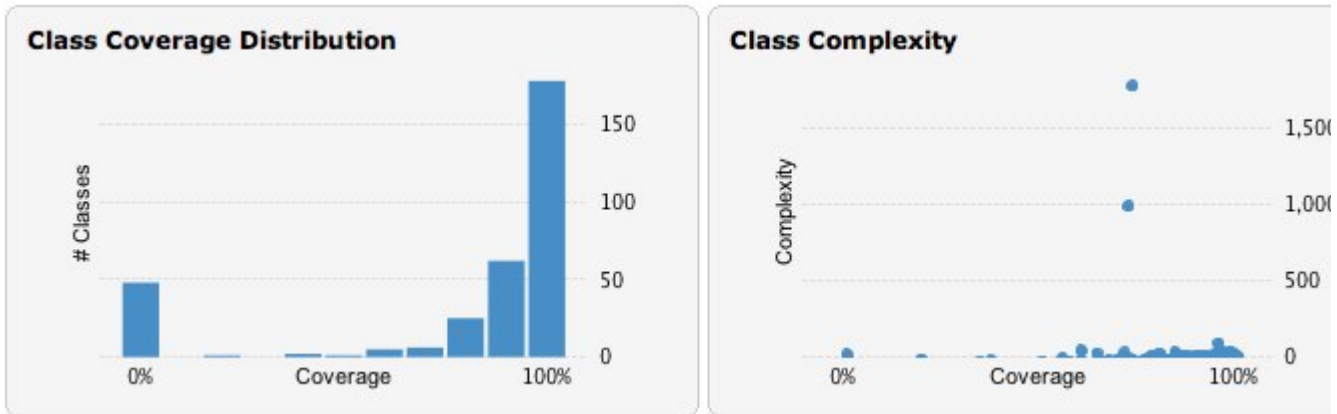
- [Clover-for-Ant Upgrade Guide | Changelog](#)
- [Clover-for-Eclipse Upgrade Guide | Changelog](#)
- [Clover-for-Maven 1 Upgrade Guide | Changelog](#)

## Highlights of Clover 2.2



### New visualisations for Dashboard

Clover 2.2 adds new histogram charts, which show coverage distribution across all your project's classes. A new scatter plot shows complexity against coverage, making outlying classes easier to spot. A sparkline version of the histogram chart is also included.



### Stack trace navigation in reports

This feature will help you diagnose test failures and the lines of code where the failure occurred. Relevant lines of source files are annotated, marking them to indicate that a failure occurred at that point. A pop-up dialog shows the full trace or a few lines of context (stack lines) on either side, plus information about which test(s) failed there.

```

81 405 [E] protected void verify(Configuration aConfig, String aFileName, String[] aExpec
82                                     throws Exception
83                                     {
84 405 [F] verify(createChecker(aConfig), aFileName, aFileName, aExpected);
{ Test failures at line 84
{ test:
junit.framework.ComparisonFailure: error message 0 expected: <...HashMap, InnerClass, HashSet...> but was: <...InnerClass, HashSet, HashM
at com.puppcrawl.tools.checkstyle.BaseCheckTestCase.verify(BaseCheckTestCase.java:122)
at com.puppcrawl.tools.checkstyle.BaseCheckTestCase.verify(BaseCheckTestCase.java:99)
at com.puppcrawl.tools.checkstyle.BaseCheckTestCase.verify(BaseCheckTestCase.java:84)
at com.puppcrawl.tools.checkstyle.checks.metrics.ClassDataAbstractionCouplingCheckTest.test(ClassDataAbstractionCouplingCheckTes
at org.junit.internal.runners.JUnit38ClassRunner.run(JUnit38ClassRunner.java:81)
at junit.framework.JUnit4TestAdapter.run(JUnit4TestAdapter.java:36)
92
93 440 [E] protected void verify(Checker aC,

```



### Better cross-referencing in reports

Source reports now have much better cross-referencing, providing class identifier linking.

53		
54	39	<code>while (child != null) {</code>
55	33	<code>  if (child.getType() == TokenTypes.METHOD_DEF) {</code>
56	13	<code>    hasMethod = true;</code>
57	13	<code>    final DetailAST modifiers =</code>
58		<code>      child.findFirstToken(TokenTypes.MODIFIERS);</code>
59	13	<code>    if (!modifiers.branchContains(TokenTypes.LITERAL_STATIC)) {</code>
60	10	<code>      hasNonStaticMethod = true;</code>
61		<code>    }</code>
62		<code>  }</code>
63	33	<code>  if (child.getType() == TokenTypes.CTOR_DEF) {</code>
64	1	<code>    hasDefaultCtor = false;</code>
65	1	<code>    final DetailAST modifiers =</code>
66		<code>      child.findFirstToken(TokenTypes.MODIFIERS);</code>
67	1	<code>    if (!modifiers.branchContains(TokenTypes.LITERAL_PRIVATE)</code>
68		<code>      &amp;&amp; !modifiers.branchContains(TokenTypes.LITERAL_PROTECTED))</code>
69		<code>    {</code>
70		<code>      // treat package visible as public</code>
71		<code>      // for the purpose of this Check</code>
72	1	<code>      hasPublicCtor = true;</code>

4

## Don't go backwards

Clover can now be configured to warn you (and optionally fail your build) when your coverage drops. In this way, you can ensure that your code coverage is maintained or improves over time.

## Clover 2.1 Release Notes

14 February 2008

### Atlassian Software Systems presents Clover 2.1

Clover 2.1 allows you to tailor your coverage reporting even more closely to your needs. Configurable risk metrics let you choose an algorithm that matches your definition of a project risk. 'Coverage Clouds' are now available for every individual package. Building on the per-test coverage that was introduced in Clover 2.0, in Clover 2.1 reports from merged databases now include per-test coverage data.

Additionally, a whole range of advanced charting options are available. Clover historical reports are now much more configurable, especially the charts and movers section. You can now customise which data gets shown, where.

Upgrading to Clover 2.1 is free for all customers with active Clover software maintenance at date of launch.

#### Highlights of Clover 2.1:

- Per-test coverage for merged databases
- Per-package coverage clouds
- Historical charting
- Enhanced 'movers' section
- Clover expression language
- New SUM metric

#### Thank you for your feedback:

*Your votes and issues help us keep improving our products, and are much appreciated.*



### Upgrading to Clover 2.1

Clover 2.1 can be downloaded from the [Clover Download Centre](#). Before upgrading, please refer to the relevant documentation:

- [Clover-for-Ant Upgrade Guide | Changelog](#)
- [Clover-for-Eclipse Upgrade Guide | Changelog](#)
- [Clover-for-Maven 2 and 3 User's Guide | Changelog](#)

## Highlights of Clover 2.1

1

### Per-test coverage for merged databases

You can now see what tests hit what code, even when generating combined reports across multiple databases.

[Read more](#)

2

### Per-package coverage clouds

[Coverage clouds](#) are now available at the package level, allowing you to compare classes within a specific package. This is specifically aimed at large projects, which have multiple developers working in different packages.

3

### Historical charting

New charts have been added to [historical](#) reports. You now have complete control over series in the charts.

4

### Enhanced 'movers' section

[Movers](#) are classes which have either gained coverage or lost coverage. You can now define multiple movers in a report, allowing you to easily spot them. For example, you could now show classes which have changed in the last day, as well as classes that have changed in the last week.

5

### Clover expression language

The new [Clover Expression Language](#) allows you to combine metrics in interesting ways, to derive your own original metric.

6

### New SUM metric

The new [SUM Metric](#) is a new column that provides a risk ranking, in much the same way that [Crap4 J](#) does.

## Clover 2.0 Release Notes

 Clover 4.0 has now been released. Read the [Clover Release Notes](#) and the latest [Clover documentation](#) for information on the newest Clover release.

17 October 2007

## Atlassian Software Systems presents Clover 2.0

Clover 2 is a major rewrite that adds new and unique functionality to help your team get the most out of their testing effort. We've augmented Clover's award-winning functionality by incorporating both test result and code complexity statistics. The resulting reports give you powerful insight into your testing. You can now see not only what sections of code were covered by your tests, but also **what tests hit what code**. Clover's new 'Cloud reports' let you quickly and easily assess strengths and weaknesses in your testing suite, helping you to prioritise your testing effort.

Clover 2 provides sophisticated source-level HTML reports. Quickly drill down into your test suite to see results of individual tests, along with information about what code each test hit. From there you can click to see the actual source lines hit by the test. In-page controls allow you to toggle the coverage annotations from all tests that hit a particular class. At any line in the source you can click to get a popup showing all the tests that hit that particular line, and their pass/fail status.

Upgrading to Clover 2.0 is free for all customers with active Clover software maintenance at date of launch.

### Highlights of Clover 2.0:

- Coverage by test case
- Test results integrated with reports
- 'Coverage Cloud' reports
- Linked, cross-referenced reports
- Context filters
- Method-level metrics
- Sortable columns
- Streamlined Ant integration and simplified Ant tasks
- Eclipse plugin and Maven 2 plugin
- Inline help

### A big 'Thank You' to our Clover 2 Beta Testers

Your helpful bug reports and feature suggestions have been invaluable in shaping Clover 2. We thank you for your patience during the beta process. If you'd like to request a feature or report a bug, [we'd love to hear from you](#).

*Your votes and issues help us keep improving our products, and are much appreciated.*

### Upgrading to Clover 2.0

Clover 2.0 can be downloaded from the [Clover Download Centre](#). Before upgrading, please refer to the relevant Upgrade Guide(s) and Changelog(s):

- [Clover-for-Ant Upgrade Guide | Changelog](#)
- [Clover-for-Eclipse Upgrade Guide | Changelog](#)

## Highlights of Clover 2.0

1

### Coverage by test case



Each source file displays which tests hit which line of code. When you select a test case, the lines that the test case executed are highlighted. Alternatively, when you click on a source line, the tests that hit that line are displayed. [Read more](#).

**2**

### Test results integrated with reports

Test results (pass/fail/error) are now optionally integrated with the coverage report. Error traces are hyperlinked to the relevant source line. [Read more](#).

**3**

### 'Coverage Cloud' reports

These give an instant overview of specific aspects of your project. All classes in your project are displayed on a single page and highlighted to inform you about project risks or potential coverage improvements. [Read more](#).

**4**

### Linked, cross-referenced reports

Reports produced with the same task are automatically linked to each other. Source code is cross-referenced for easy traversal between classes and up and down package hierarchies.

**5**

### Context filters

Source code excluded by a [context filter](#) is now highlighted grey and project, package and file level statistics may be viewed with filtering either on or off.

**6**

### Method-level metrics

Metrics at the method level are displayed both inline and in each class summary section.

**7**

### Sortable columns

All tables in Clover 2 reports are client-side sortable.

**8**

### Streamlined Ant integration and simplified Ant tasks

The new `<clover-html-report>` and `<clover-pdf-report>` tasks provide sensible defaults to the existing `<clover-report>` task. The Clover `initstring` is now optional. If not specified, Clover 2 will automatically create and manage the coverage database for you.

**9**

### Eclipse plugin and Maven 2 plugin

Fully integrated plugins for [Eclipse](#) and [Maven 2](#) are available for Clover 2.0.



# 10

## Inline help

Help tooltips can be turned on for each page to describe metrics and controls.

## Clover Release Summary

### Clover 2.6 (9-September-2009)

- New Clover Editions
- Eclipse Plugin Performance Improvements
- IDE Plugin Ease-of-Use Features
- 100% Coverage filter
- New HTML Tree Map
- New API for Optimizing Tests Programmatically
- Clover Auto-Update Feature for IDEA

### Clover 2.5 (11-May-2009)

- Test Optimization in Clover for Eclipse and IDEA
- Distributed Per-Test Coverage
- Performance Improvements

### Clover 2.4 (5-Nov-2008)

- Test Optimization
- Easier Integration
- Reporting Improvements
- Clover-for-IDEA final release

### Clover 2.2 (10-Apr-2008)

- New visualisations for Dashboard
- Stack trace navigation in reports
- Better cross-referencing in reports
- Configure Clover to warn you or fail your build when your coverage drops

### Clover 2.1 (14-Feb-08)

- Configurable metrics reporting
- Per-package coverage clouds
- Historical charting
- Enhanced 'movers' section
- Per-test coverage for merged databases
- Greatly improved performance
- More in release notes: [Ant](#) and [Eclipse](#)

### Clover 2.0 (17-Oct-07)


- Coverage by test case
- Test results integrated with reports
- 'Coverage Cloud' reports
- Linked, cross-referenced reports
- Context filters
- Method-level metrics
- Sortable columns
- Streamlined Ant integration and simplified Ant tasks
- Eclipse plugin and Maven 2 plugin
- Inline help
- More in [release notes](#)

# Clover Tutorials

- [Clover-for-Ant tutorial](#) — This tutorial demonstrates how you can use Clover with JUnit to measure the code coverage of a project. It takes you through the process of compiling a sample project and running the unit tests from Ant, then modifying the build file to add Clover targets and properties.
- [Clover-for-Maven tutorials](#)
- [How to configure your Clover license](#)

## Clover-for-Ant tutorial

This tutorial demonstrates how you can use Clover with JUnit to measure the code coverage of a project. It takes you through the process of compiling a sample project and running the unit tests from Ant, then modifying the build file to add Clover targets and properties.

 If you want to quickly see exemplary Clover report and not be bothered by manual configuration, please have a look at:

- [Part 0 - Clover in 10 minutes](#)

### Tutorial Parts

1. [Part 1 - Measuring Coverage](#)
2. [Part 2 - Historical Reporting](#)
3. [Part 3 - Automating Coverage Checks](#)
4. [Part 4 - Test Optimization](#)

The Clover Tutorial describes different features of Clover in a step-by-step approach. Once you've completed the Tutorial, have a look at [Using Clover Interactively](#) and [Using Clover in Automated Builds](#) for examples of how to pull the different aspects of Clover together for your project.

### Before You Start

You will need to [download Clover-for-Ant](#) and [Apache Ant](#), preferably the latest versions, then install them on your system. The tutorial source files and JUnit are bundled with the Clover distribution in the /tutorial directory.


Instructions for installing Ant can be found in the [Apache Ant User Manual](#).

Instructions for installing Clover can be found in the [Clover-for-Ant Installation Guide](#).

The Clover tutorial assumes that you have basic knowledge of creating and modifying Ant build files. The [Apache Ant User Manual](#) provides any additional support you may require in this area. It is also assumed that you have a basic understanding of JUnit. A good introduction to JUnit can be found in the [JUnit Cookbook](#). This Clover tutorial is crafted around the example code described in the [Cookbook](#).

### The Tutorial Work Area

The tutorial source files are located within the 'tutorial' directory (that is, `CLOVER_HOME/tutorial`). In the 'tutorial' directory you will find the initial build file and the directory 'src' which contains the java files that you will be testing. These sample files are shipped with JUnit test files, as described in the [JUnit Cookbook](#). They represent a simple library for dealing with money and provide methods to add, subtract, and collect money etc. The `MoneyTest.java` file contains all the unit tests for the library and utilises the JUnit framework.

 The `build.xml` file is located directly under the 'tutorial' directory.

### NEXT:

Once you have Ant installed and the Clover for Ant package have been downloaded, you are ready to progress to [Part 1 - Measuring Coverage](#) in the tutorial.

## Part 0 - Clover in 10 minutes



### Requirement

This tutorial requires at least Clover 3.1.12 which contains pre-configured tutorial project (specifically Ant `build_quick.xml` file).

**i License**

Make sure that `clover_installation_dir/lib/clover.license` file exists next to `clover.jar` file and contains valid license.

Welcome to the Clover-for-Ant 10 minutes tutorial. This document will show you how to quickly get Clover reports for sample project.

*On this page:*

- [Introduction](#)
- [Generating report](#)
  - [Running Ant command](#)
  - [Opening report](#)
- [Uncovering details](#)

**Introduction**

In this tutorial we will generate Clover HTML report for Money library provided in the `tutorial/src` directory.

**Generating report**

The process of generating report contains few phases, configured to be executed with single Ant command for the purpose of this tutorial:

- Compiling project (and instrumenting its source code with Clover statements).
- Running unit tests (and gathering code coverage).
- Generating code coverage report.

**Running Ant command**

To generate report, use the command `ant -f build_quick.xml`

Output should be similar to the following:

```
$ ant -f build_quick.xml
Buildfile: build_quick.xml

with.clover:
  [clover-setup] Clover Version 3.1.12, built on ...
  [clover-setup] Loaded from: C:\ant\lib\clover.jar
  [clover-setup] Site License registered to ...
  [clover-setup] Clover is enabled with initstring
  'C:\clover\tutorial\clover\clover2_X.db'

code.compile:
  [mkdir] Created dir: C:\clover\tutorial\build
  [javac] Compiling 4 source files to C:\clover\tutorial\build
  [clover] Clover Version 3.1.12, built on ...
  [clover] Loaded from: C:\ant\lib\clover.jar
  [clover] Site License registered to ...
  [clover] Processing files at 1.4 source level.
  [clover] Clover all over. Instrumented 4 files.

test.compile:
  [mkdir] Created dir: C:\clover\tutorial\build\test
  [javac] Compiling 2 source files to C:\clover\tutorial\build\test
  [clover] Clover Version 3.1.12, built on ...
  [clover] Loaded from: C:\ant\lib\clover.jar
  [clover] Site License registered to ...
  [clover] 23 test methods detected.

test.run:
  [mkdir] Created dir: C:\clover\tutorial\build\testresult
  [junit] Running MoneyBugTest
  [junit] Tests run: 22, Failures: 0, Errors: 0, Time elapsed: 0.346 sec

clover.report:
  [clover-html-report] Clover Version 3.1.12, built on ...
  [clover-html-report] Loaded from: C:\ant\lib\clover.jar
  [clover-html-report] Site License registered to ...
  [clover-html-report] Loading coverage database from: ...
  [clover-html-report] Writing report to 'C:\clover\tutorial\clover_html'
  [clover-html-report] Done. Processed 1 packages.

BUILD SUCCESSFUL
Total time: 9 seconds
```

This shows that the java source files have been compiled and instrumented, tests have been executed and Clover report generated.

### Opening report

You can now view the report by opening the file `tutorial/clover_html/index.html` in a web browser. See ['Current Report'](#) for details about interpreting this coverage report.

### Uncovering details

If you want to understand what happened under the hood and how to customize and adjust the process of getting reports, please continue with [Part 1 - Measuring Coverage](#)

## Part 1 - Measuring Coverage

Welcome to the Clover-for-Ant tutorial. This document will walk you through the process of integrating Clover with an Ant build, gradually exploring Clover's more advanced code coverage features along the way.

*On this page:*

- Introduction
- Step 1. Compiling and running
  - Compiling
  - Running the tests
- Step 2. Adding Clover targets
  - Adding Clover task definitions
  - Adding a target to enable Clover
  - Adding Clover to the build classpath
  - Adding <clover-clean> to the Clean Target
- Step 3. Testing with Clover
  - Compile with Clover
  - Running the tests
- Step 4. Creating a report
  - Adding a Clover report target
  - Generating the report
- Step 5. Improving coverage
- NEXT

## Introduction

Part one of this tutorial focuses on the creation and interpretation of Clover 'current' reports. Current reports display graphical and numerical data relating to the most recent coverage data collected for the project. This tutorial covers the initial creation of coverage data before stepping you through how to generate and interpret coverage reports. We'll then look at how to improve the coverage achieved by tests and regenerate the coverage reports. This section covers the very basic features of Clover and is an important first step for all users.

In this tutorial we will compile and unit-test the Money library provided in the `tutorial/src` directory, then use Clover to determine how well the unit tests actually test the library.

In the first step, we will compile the Money library and run tests against it.

## Step 1. Compiling and running

In this step we will compile the library and run the tests against it without using Clover to check that everything is working correctly before including Clover in the next step. In the `tutorial` directory you will find the initial build file which contains targets for compiling, running and cleaning the build.

### Compiling

To compile the java files, use the command `ant code.compile`.

Output should be similar to the following:

```
$ ant code.compile
Buildfile: build.xml

code:
  [mkdir] Created dir: C:\clover\tutorial\build
  [javac] Compiling 4 source files to C:\clover\tutorial\build

BUILD SUCCESSFUL
Total time: 9 seconds
```

This shows that the java source files have been compiled and the class files have been placed in the `C:\clover\tutorial\build` directory.

### Running the tests

To run the JUnit tests, use the command `ant test.run`.

Output should be similar to the following:

```
$ ant test.run
  Buildfile: build.xml

  test:
    [junit] Running MoneyTest
    [junit] Tests run: 22, Failures: 0, Errors: 0, Time elapsed: 0.052 sec

BUILD SUCCESSFUL
Total time: 1 second
```


This shows that all the tests have been run and have passed.

We have now compiled the Money library, and run tests against it. In the next step, we will add Clover targets and properties to the build file to enable measurement of code coverage.

### Step 2. Adding Clover targets


Now that we have compiled the code and run unit tests, we are ready to add Clover targets and properties to the build file so we can measure the code coverage of the tests. Modifying the build file is trivial. Firstly we need to add a target to enable and configure Clover for the build.

#### Adding Clover task definitions

 For this tutorial, ensure that the property `clover.jar` has been defined as the path to your 'clover.jar' file. Hence, if you followed the [Adding to Ant's build.xml](#) instructions and have only added the Clover 'taskdef' resource to your 'build.xml' file, you'll need to redefine this resource to match the format described in this step.

Load the `build.xml` file into your favourite text editor and add the Clover Ant task and type definitions:

```
<property name="clover.jar" location="../lib/clover.jar"/>
<taskdef resource="cloverlib.xml" classpath="{clover.jar}"/>
```

 **Note**  
This assumes that the `clover.jar` is left in the unpacked Clover distribution from which this tutorial is being done. If you have installed the `clover.jar` elsewhere, adjust the path accordingly.

These lines define the Clover Ant tasks which can then be used within the build file.

#### Adding a target to enable Clover

Add a target called `with.clover` which will enable and configure Clover for a build:

```
<target name="with.clover">
  <clover-setup/>
</target>
```

#### Adding Clover to the build classpath

The `clover.jar` needs to be in the runtime classpath when you execute the tests. To achieve this, add the line in marked below to the `build.classpath` Ant path:

```
<path id="build.classpath">
  <pathelement path="{clover.jar}"/> <!-- add this -->
  <pathelement path="{junit.jar}"/>
  <pathelement path="{app.build}"/>
</path>
```

### Adding `<clover-clean>` to the Clean Target

It is advisable to add the `<clover-clean/>` task to the `clean` target. This will delete the Clover database when the `clean` target is executed.

```
<target name="clean" >
  <delete dir="build"/>
  <clover-clean/> <!-- add this -->
</target>
```

Once you have made these changes, save the `build.xml` file. We will add some more Clover targets later to perform coverage reporting, but first we will re-compile the Money library with Clover and re-run the tests to obtain coverage data.

### Step 3. Testing with Clover

We are now ready to measure the coverage of the tests over the Money library.

#### Compile with Clover

Ensure that your build has been cleaned by running `ant clean`. This deletes all class files from previous compilations.

Compile your code with Clover using the command `ant with.clover code.compile`.

You will get output similar to the following:

```
$ ant with.clover code.compile
  Buildfile: build.xml

  with.clover:
  [clover-setup] Clover Version ..., built on ...
  [clover-setup] Clover is enabled with initstring
  'C:\clover\tutorial\.clover\clover2_X.db'

  code:
    [mkdir] Created dir: C:\clover\tutorial\build
    [javac] Compiling 4 source files to C:\clover\tutorial\build
    [clover] Clover Version ..., built on ...
    [clover] Loaded from: C:\ant\lib\clover.jar
    [clover] Site License registered to ...
    [clover] Processing files at 1.3 source level.
    [clover] Clover all over. Instrumented 4 files.
```

The result of this process is that your source files have been instrumented by Clover and then compiled as usual. As part of the instrumentation process, Clover creates a database that will be used during the coverage recording and report process.

#### Running the tests

We now need to run the tests again, using the command `ant test.run`. This command will run the tests, this time measuring coverage. Output from Ant will be the same as a normal test run:

```
$ ant test.run
  Buildfile: build.xml

  test:
    [mkdir] Created dir: C:\clover\tutorial\build\test
    [junit] Running MoneyTest
    [junit] Tests run: 22, Failures: 0, Errors: 0, Time elapsed: 0.346 sec

  BUILD SUCCESSFUL
  Total time: 1 second
```

During this test run, Clover measured the code coverage of the tests and wrote the coverage data to disk.

In the next step we'll generate a coverage report from this data to see how well the tests actually cover the Money library.

#### Step 4. Creating a report

We are now ready to produce a coverage report. This section will focus on producing a Clover HTML report. For information on how to generate a PDF report see the `<clover-pdf-report>` task, or for other types of Clover reports see the `<clover-report>` task.

#### Adding a Clover report target

Open the `build.xml` file in a text editor and add the following target to create a HTML report:

```
<target name="clover.report">
  <clover-html-report outdir="clover_html" title="Clover Demo"/>
</target>
```

The `<clover-html-report>` task is a simplified version of the `<clover-report>` task. As no `historydir` attribute has been specified, it uses the current coverage data. Historical reports, which show the progress of coverage over the life of the project, are discussed later in this tutorial (see [Part 2 - Historical Reporting](#)). The current report is to be in HTML format, written to the directory `clover_html` and with the title "Clover demo". The output directory `clover_html` is relative to the path of the Ant build file. In this case, the directory `clover_html` will be nested within `tutorial` as this is the location of `build.xml`.

#### Generating the report

Create a HTML report with the command `ant clover.report`. You will get output similar to the following:



```

$ ant clover.report

with.clover:
[clover-setup] Clover Version ..., built on ...
[clover-setup] Loaded from: C:\ant\lib\clover.jar
[clover-setup] Site License registered to ...
[clover-setup] Clover is enabled with initstring
'C:\clover\tutorial\clover\clover2_x.db'

clover.report:
[clover-html-report] Clover Version ..., built on ...
[clover-html-report] Loaded from: C:\ant\lib\clover.jar
[clover-html-report] Site License registered to ...
[clover-html-report] processed 22 slices in 22ms (1ms per pair)
[clover-html-report] Writing report to 'C:\clover\tutorial\clover_html'
[clover-html-report] Done. Processed 1 packages.

BUILD SUCCESSFUL
Total time: 1 second

```

You can now view the report by opening the file `tutorial/clover_html/index.html` in a web browser. See ['Current Report'](#) for details about interpreting this coverage report.

In the next step, we will enhance the JUnit tests to improve code coverage of the Money library.

### Step 5. Improving coverage

After having a look at the coverage report, you'll notice that coverage is not 100%. Although not always possible, it is best to get as close to full coverage as you can. Think of it this way: every line that isn't covered could contain a bug that will otherwise make it into production. **You should certainly aim to cover all of the code that will be executed under normal operation of the software.**

One method in the Money library that is not fully covered is the `equals()` method in the Money class (lines 40-42 as seen below). The first few lines of this method handle the special case when the Money value is zero. The coverage report shows that the code to handle this has not been covered by the tests. Line 40 has been executed 27 times, but since it has never evaluated to `true` it has not been fully covered and is therefore in red. It follows then that the two successive lines have never been executed.

Line	Count	Code	Coverage	Covered Elements	Complexity	Complexity Density
38		}				
<div style="border: 1px solid black; padding: 2px; margin-bottom: 5px;"> <span style="display: inline-block; width: 20px; height: 10px; background-color: green; margin-right: 5px;"></span> <span style="display: inline-block; width: 20px; height: 10px; background-color: red; margin-right: 5px;"></span> <span style="font-weight: bold;">61.5%</span> <span style="margin-left: 20px;">Covered Elements: 8</span> <span style="margin-left: 20px;">Complexity: 4</span> <span style="margin-left: 20px;">Complexity Density: 0.57</span> </div>						
39	27	<code>public boolean equals(Object anObject) {</code>				
40	27	<code>if (isZero())</code>				
41	0	<code>if (anObject instanceof IMoney)</code>				
42	0	<code>return ((IMoney)anObject).isZero();</code>				
43	27	<code>if (anObject instanceof Money) {</code>				
44	24	<code>Money aMoney= (Money)anObject;</code>				
45	24	<code>return aMoney.currency().equals(currency())</code>				
46		<code>&amp;&amp; amount() == aMoney.amount();</code>				
47		<code>}</code>				
48	3	<code>return false;</code>				
49		<code>}</code>				

We can now improve the tests so that this section of code is covered. To do this, make the following additions (shown in **bold**) to the `MoneyBagTest.java` file.

Declare the variable `f0USD`:

```

public class MoneyBagTest extends TestCase {
    private Money f12CHF;
    private Money f14CHF;
    private Money f7USD;
    private Money f21USD;
    private Money f0USD;
    ...
}

```

Initialise `f0USD` in the `setUp()` method:

```
protected void setUp() {
    f12CHF = new Money(12, "CHF");
    f14CHF = new Money(14, "CHF");
    f7USD = new Money( 7, "USD");
    f21USD = new Money(21, "USD");
    f0USD = new Money(0, "USD");
    ...
}
```

Finally, the following test needs to be added:

```
public void testMoneyEqualsZero() {
    assertTrue(!f0USD.equals(null));
    IMoney equalMoney = new Money(0, "CHF");
    assertTrue(f0USD.equals(equalMoney));
}
```

After these amendments have been made, compile and run tests again (by running `ant test.run`), then re-generate the HTML report (by running `ant clover.report`). You will see that the `Money` class now has 100% coverage.

NEXT

[Part 2 - Historical Reporting](#)

## Part 2 - Historical Reporting

Part two of this tutorial focuses on the creation and interpretation of Clover 'Historical' reports.

*On this page:*

- [Step 1 - Creating history points](#)
  - [Adding a history point target](#)
  - [Recording a history point](#)
- [Step 2 - Generating historical data](#)
- [Step 3 - Creating historical reports](#)
  - [Add a historical report target](#)
  - [Generating a historical report](#)
- [Step 4 - Customising historical reports](#)
  - [Changing output format](#)
  - [Chart Selection](#)
  - [Chart Configuration](#)
  - ['Movers' Configuration](#)
- [NEXT](#)

Historical reports display graphical and numerical data relating to sets of coverage data collected over time for the project. This tutorial covers the generation of a set of historical data, interpretation of the information displayed in the Historical reports and customisation of the reports for your particular reporting preferences.

In the first step, we will edit the Ant build file to generate a *history point*.

A history point is a snapshot of code coverage and metrics data for the project at a particular point in time. By running tests with Clover over time and creating a series of history points, it is possible to compare code coverage and metrics by viewing results in a single Clover report and enabling you to track the development of your project.

### Step 1 - Creating history points

Step 1 describes how to set up the relevant Ant target and run the command so that a history point can be created. The generation of historical reports is discussed in later steps.

#### **Adding a history point target**

Add the following target to your `build.xml` file:

```
<target name="record.point">
  <clover-historypoint historyDir="clover_history"/>
</target>
```

When this target is run, a history point will be created with the timestamp value of the coverage run.

The value of the `historyDir` parameter is the directory where the history points will be stored. You should create this directory before executing this target.



By default, Clover records the history point with a timestamp of the coverage run. If you wish to override the timestamp value of a history point, you can add `date` and `dateFormat` attributes to the task allowing you to reconstruct coverage history. See documentation for the `<clover-historypoint>` task for details.

### Recording a history point

Ensure that the source code has been instrumented, and the tests run, by using the commands `ant clean` with `.clover` `code.compile` and `ant test.run` respectively.

Run the command `ant record.point`. Output should be similar to the following:

```
$ ant record.point
Buildfile: build.xml

record.point:
 [clover-historypoint] Clover Version ..., built on ...

 [clover-historypoint] Merged results from 2 coverage recordings.
 [clover-historypoint] Writing report to
 'C:\clover\tutorial\clover_history\clover-20030307111326.xml.gz'
 [clover-historypoint] Done.

BUILD SUCCESSFUL
Total time: 2 seconds
```

In the next step we will add more tests to improve coverage of the Money Library, recording Clover history points along the way.

### Step 2 - Generating historical data

In [Part 1 of the tutorial](#) we made additions to the testing suite to improve code coverage. In order to show the historical reporter in use, we will now continue to add tests and periodically record history points which will later be used as code coverage and metrics data by the historical reporter.

The `Money.java` file is at 100% coverage, but there are several sections of code that remain untested in the `MoneyBag.java` file. This step will focus on bringing the coverage of this class to 100%, as well as creating historical data in the form of history points.

Open the source file `MoneyBagTest.java` in your favourite text editor and make the following additions shown in bold:

Declare the variables `f0CHF` and `fMB3`:

```
public class MoneyBagTest extends TestCase {
    private Money f12CHF;
    private Money f14CHF;
    private Money f7USD;
    private Money f21USD;
    private Money f0USD;
    private Money f0CHF;

    private IMoney fMB1;
    private IMoney fMB2;
    private IMoney fMB3;
    ...
}
```

Initialise `f0CHF` and `fMB3` in the `setUp()` method:

```
protected void setUp() {
    f12CHF = new Money(12, "CHF");
    f14CHF = new Money(14, "CHF");
    f7USD = new Money( 7, "USD");
    f21USD = new Money(21, "USD");
    f0USD = new Money(0, "USD");
    f0CHF = new Money(0, "CHF");

    fMB1 = MoneyBag.create(f12CHF, f7USD);
    fMB2 = MoneyBag.create(f14CHF, f21USD);
    fMB3 = MoneyBag.create(f0CHF, f0USD);
    ...
}
```

Add the following test:

```
public void testMoneyBagEqualsZero() {
    assertTrue(!fMB3.equals(null));
    IMoney expected = MoneyBag.create(new Money(0, "CHF"), new Money(0, "USD"));
    assertTrue(fMB3.equals(expected));
}
```

After making the above changes, reinstrument and test your code by running `ant clean with.clover code.compile` and `ant test.run` respectively. Then record a new history point by running `ant record.point`. By recording a history point now, Clover will capture the new state of code coverage and metrics for comparison with past or future runs.

Add the following tests to bring the coverage of the Money project to 100%:

```
public void testToString() {
    String expected="{[12 CHF][7 USD]}";
    assertEquals(expected, fMB1.toString());
}
public void testVectorSize() {
    IMoney other = MoneyBag.create(new Money(2, "CHF"), new Money(2, "USD"));
    assertTrue(!other.equals(fMB3));
}
```

Once again, re-instrument your code, test and record a new history point.

We have now created a series of history points for the Money library. The next section discusses how to generate a Clover historical report which will display the historical data that has been collected.

### Step 3 - Creating historical reports

Now that we have recorded several history points, the next step is to add a target to the build file which will call the historical reporter and generate a historical report.

#### *Add a historical report target*

Add the following target to `build.xml`:

```
<target name="hist.report">
  <clover-report>
    <historical outfile="historical.pdf"
      historyDir="clover_history">
      <format type="pdf"/>
    </historical>
  </clover-report>
</target>
```

The `hist.report` target is similar to the `report.html` target defined in Part 1. The main differences are that the nested element specifies `<historical>` rather than `<current>` and there is no specification of the output format as `html`.

The historical reporter needs to be able to find the coverage history files in order to create the report; so the `historyDir` value must be the same as the `historyDir` defined for the history points. The format of the report can be either PDF or HTML, as specified by the `<format>` element `<clover-report>`. The `<format>` element is optional and is not included in the example above. When the `<format>` element is omitted, a PDF report is produced by default. Depending on the chosen format, the `outfile` value may represent a single file as in the case of the PDF format, or the name of a directory (in the case of the HTML format).

### Generating a historical report

Create a historical report by using the command `ant hist.report`. Output should be similar to the following:

```
$ ant hist.report
Buildfile: build.xml

hist.report:
[clover-report] Clover Version ..., built on ...
[clover-report] Writing report to 'C:\clover\tutorial\historical.pdf'
[clover-report] Merged results from 2 coverage recordings.
[clover-report] Done. Processed 1 packages.
[clover-report] Writing historical report to 'C:\clover\tutorial\historical.pdf'
[clover-report] Read 3 history points.
[clover-report] Done.

BUILD SUCCESSFUL
Total time: 8 seconds
```

The report can now be viewed by opening the file `tutorial/historical.pdf` in a PDF viewer such as [Adobe Acrobat Reader](#).



To interpret this history report, see [Understanding a Historical Report](#)

### Step 4 - Customising historical reports

In the previous sections of this tutorial we have looked at how to create and interpret a basic historical report. In addition to basic reporting, the historical reporter is highly configurable and this section will detail some of the options you can use. For a full list of the report configuration options, see the documentation for the `<clover-report>` task.

#### Changing output format

The default historical report type is PDF, although an HTML report can also be produced. To create an HTML report, add a nested `<format>` element with type specified as `html` to your `<clover-report>` element. Try adding the following target to your `build.xml` file and executing the command `ant hist.report.html`:

```

<target name="hist.report.html">
  <clover-report>
    <historical outfile="clover_html/historical"
      title="My Project"
      historyDir="clover_history">
      <format type="html"/>
    </historical>
  </clover-report>
</target>

```

A custom title can also be displayed for your report by using the `title` attribute in the `<historical>` element as above.

### Chart Selection

The historical reporter allows you to specify which charts to include in your report, and also allows you to configure further options in the charts themselves.

The default reporting mode is to include all four report elements: `<overview>`, `<coverage>`, `<metrics>` and `<movers>`. But to include some and not the others is a simple matter of nesting the desired elements within the `<historical>` element. Try adding the following target to your `build.xml` file as an example:

```

<target name="hist.report.coverage">
  <clover-report>
    <historical outfile="histCoverage.pdf"
      title="My Project"
      historyDir="clover_history">
      <overview/>
      <coverage/>
      <format type="pdf"/>
    </historical>
  </clover-report>
</target>

```

The above code will produce a historical PDF report with the title 'My Project' which includes only two sections: the 'Overview' and the 'Coverage over time' charts.

### Chart Configuration

Clover presents flexible charting configuration, allowing you to present information exactly as you like it. The `<chart>` element allows you to define a custom chart and fill it with specific data with the `<columns>` element.

```

<target name="hist.report.select">
  <clover-report>
    <historical outfile="histSelect.pdf"
      title="My Project"
      historyDir="clover_history">
      <chart/>
      <metrics/>
      <format type="pdf"/>
    </historical>
  </clover-report>
</target>

```

This will produce a PDF file with the filename `histSelect.pdf` with two sections: the a custom chart with total coverage information; and the 'Metrics over time' chart. You can also specify whether or not a chart uses a log scale by adding the `logscale` attribute:

```

<metrics logscale="false"/>

```

### 'Movers' Configuration

The 'Movers' section of the historical report shows you the classes whose coverage has changed the most recently. This is useful for spotting classes that have had sudden changes in coverage, perhaps the unintended result of changes to the unit test suite.

The 'Movers' chart allows you to specify the threshold of point change a class must satisfy, the maximum number of gainers and losers to display and the period across which the gains and losses are calculated. Add the following target to your `build.xml` file as an example of this feature in use:

```
<target name="hist.report.movers">
  <clover-report>
    <historical outfile="histMovers.pdf"
      title="My Project"
      historyDir="clover_history">
      <movers threshold="5%" range="20" interval="2w"/>
    </historical>
  </clover-report>
</target>
```

In this case, the configuration values selected state that classes must have a change in coverage of at least 5 percentage points to be included in the chart, a maximum of 20 gainers and 20 losers can be displayed, and the initial valuation point for class coverage is 2 weeks prior to the most recent history point. Should there be greater than 20 gainers in this period, then the classes with the biggest percentage point gain will be displayed, and the same for the losers.

See [Interval Format](#) for details on the syntax for specifying interval values.

The next section of this tutorial will discuss how you can automate the coverage checking of your project.

NEXT

[Part 3 - Automating Coverage Checks](#)

## Part 3 - Automating Coverage Checks

This section of the tutorial looks at some advanced features of Clover.

The `<clover-check>` task provides a useful mechanism for automating your coverage checking and gives you the option of failing your build if the specified coverage percentage is not met. It is easily integrated into your build system.

On this page:

- [Adding coverage checking](#)
- [Failing the build if coverage criteria not met](#)
- [Adding Package-level coverage criteria](#)
- [Context filtering](#)
- [NEXT](#)

### Adding coverage checking

Ensure that you have current Clover coverage data so that you can check the coverage percentage for your project. Clover coverage data is generated as described in [Part 1](#) of the Tutorial.

Add the `<clover-check>` task to your build by specifying a target similar to the following:

```
<target name="clover.check" depends="with.clover">
  <clover-check target="80%"/>
</target>
```

This configuration sets an overall project target of 80% coverage.

Use the command `ant clover.check` to run the check. If your test coverage satisfies the target coverage percentage, output will be similar to the following:



```
$ ant clover.check
Buildfile: build.xml

with.clover:

clover.check:
 [clover-check] Merged results from 1 coverage recording.
 [clover-check] Coverage check PASSED.

BUILD SUCCESSFUL
Total time: 2 seconds
```

If your coverage percentage does not reach the coverage target, you'll get something like this instead:

```
$ ant clover.check
Buildfile: build.xml

with.clover:

clover.check:
 [clover-check] Merged results from 1 coverage recording.
 [clover-check] Coverage check FAILED
 [clover-check] The following coverage targets were not met:
 [clover-check] Overall coverage of 74% did not meet target of 80%

BUILD SUCCESSFUL
Total time: 2 seconds
```

In order to try this out on the Money Library used in this tutorial, try commenting out some of the tests in the `MoneyTest.java` file to create a situation where the code coverage does not reach 80%.

#### **Failing the build if coverage criteria not met**

In the above situation where the target is not met, after the message has been written to output, the build for the specified target will continue as normal.

Adding the `haltOnFailure` attribute allows you to specify whether or not you want the build to fail automatically if the coverage target is not met. The default for `haltOnFailure` is `false`.

```
<target name="clover.check.haltontail" depends="with.clover">
  <clover-check target="80%" haltOnFailure="true"/>
</target>
```

The `failureProperty` attribute of the `<clover-check>` task allows you to set a specified property if the target of the project is not met:

```
<target name="clover.check.setproperty" depends="with.clover">
  <clover-check target="80%" failureProperty="coverageFailed"/>
</target>
```

In this case, if the coverage does not reach 80%, the property `coverageFailed` will have its value set to the coverage summary message "Overall coverage of \*% did not meet target of 80%". This allows you to check the value of this property in other Ant targets and manage the outcome accordingly. For an example on managing the resulting actions for a project which does not meet its coverage target, see [Using Clover in Automated Builds](#)



### Adding Package-level coverage criteria

The `<clover-check>` task also allows you to specify the desired percentage covered for different packages, which is useful if you have certain packages that have more or less stringent coverage requirements than the rest of the project. This is done by adding nested 'package' elements like the following:

```
<target name="clover.check.packages" depends="with.clover">
  <clover-check target="80%">
    <package name="com.clover.example.one" target="70%" />
    <package name="com.clover.example.two" target="40%" />
  </clover-check>
</target>
```

### Context filtering

The `<clover-check>` task allows you to prescribe a filter that excludes coverage of certain block-types from overall coverage calculations. See [Coverage Contexts](#) for more information. The `filter` attribute accepts a comma separated list of the contexts to exclude from coverage calculations.

```
<target name="clover.check.nocatch" depends="with.clover">
  <clover-check target="80%" filter="catch" />
</target>
```

This will run the Clover coverage percentage check as normal, but will calculate coverage with omission of all 'catch' blocks.

### NEXT

[Part 4 - Test Optimization Tutorial](#)

## Part 4 - Test Optimization Tutorial

This section of the tutorial walks through the process of setting up Clover [Test Optimization](#), which efficiently runs only the tests for code which has changed since the last build.

On this page:

- [Adding Test Optimization Tasks to build.xml](#)
  - [1. Adding Paths to Resources](#)
  - [2. Choosing a Location for the Snapshot File](#)
  - [3. Adding a new Ant Target to Generate the Optimized Test 'Snapshot'](#)
  - [4. Editing the JUnit Task to Add the 'clover-optimized-testset' Element](#)
- [Demonstrating that Test Optimization is Working](#)
  - [5. Running the Test Optimized Build](#)
  - [6. Running an 'Empty' Optimized Build](#)
  - [7. Editing a Java File in the Project](#)
  - [8. Rebuilding the Project with Test Optimization](#)
- [Related Links](#)


This tutorial assumes you have completed the other [Clover Tutorial](#) steps and have fully set up Ant in those steps to build and test the MoneyBag Java project, with Clover testing the JUnit code coverage. We will make use of the Ant tasks set up in `build.xml` from the previous tutorial chapters, here. Knowing that, read on.

The process described here will change your build file to always run in Test Optimization mode.

### Adding Test Optimization Tasks to build.xml

#### 1. Adding Paths to Resources

Open your `build.xml` file.

 You should already have this line included:

```
<taskdef resource="cloverlib.xml" classpath="${clover.jar}"/>
```

from the earlier Clover-for-Ant tutorial steps.

Edit around this line to add in one additional line of code, as shown below:

```
<taskdef resource="cloverlib.xml" classpath="${clover.jar}"/>
<clover-env/>
```

## 2. Choosing a Location for the Snapshot File

Test Optimization uses the concept of a **'snapshot'** file. This is a file that records information about the previous build, as a point of comparison. This is what allows Clover to run Optimized tests, by comparing the data in the snapshot file with the current build that you are intending to launch.

For the purposes of this tutorial, leave the snapshot file in its default location, (next to the clover database `clover.db`) here:

```
<PROJECT_DIR>/ .clover/coverage.db.snapshot
```

## 3. Adding a new Ant Target to Generate the Optimized Test 'Snapshot'

Now we'll add a new Ant target to generate the test snapshot:

```
<target name="clover.snapshot" depends="with.clover">
  <clover-snapshot file="${clover.snapshot.file}"/>
</target>
```

## 4. Editing the JUnit Task to Add the 'clover-optimized-testset' Element

In your `build.xml` file, edit the `test.run` target, specifically the `junit` portion and its sub-element, `batchtest`.

Having just completed the earlier steps in the Clover-for-Ant tutorial, your code inside `batchtest` will look like so:

```
<junit fork="yes" printsummary="true">
  <classpath refid="testbuild.classpath"/>
  <formatter type="xml"/>
  <batchtest fork="yes" todir="${test.result}">
    <fileset dir="${test.src}" includes="**/*Test.java"/>
  </batchtest>
</junit>
```

To add Test Optimization to the build: add a new element, `clover-optimized-testset`, move the `fileset` element inside the new `clover-optimized-testset` element.

Edit your new `batchtest` code block until it is the same as the following:

```
<junit fork="yes" printsummary="true">
  <classpath refid="testbuild.classpath"/>
  <formatter type="xml"/>
  <batchtest fork="yes" todir="${test.result}">
    <clover-optimized-testset fullrunevery="10"
      enabled="true"
      ordering="failfast"
      minimize="true"
      snapshotfile="${clover.snapshot.file}">
      <fileset dir="${test.src}" includes="**/*Test.java"/>
    </clover-optimized-testset>
  </batchtest>
</junit>
```

Your Test Optimization configuration is now complete.

### **Demonstrating that Test Optimization is Working**


Finally, we will build our project, creating the essential Clover 'snapshot' file. Next, we will edit one of the Java files in the 'Money' project, commenting out one of the unit tests. When we run the Test Optimized build a second time, Clover will compare the snapshot file against the new coverage database and then run targeted tests which incorporate only those files which have changed, (which in this case will be only one, the Java file that we edited). This saves valuable time, which is the key advantage of Test Optimization.

### **5. Running the Test Optimized Build**

Run your build with the following command:

```
ant with.clover clean test.run clover.snapshot
```

Adding the `clover.snapshot` target here will create the additional snapshot database, which is used as a point of comparison for the Test Optimization.

 This is essential for enabling Test Optimization of future builds.

Clover will output this text to the console, showing that it has created the all-important snapshot file:

```
test.run:
  [mkdir] Created dir: /tutorial/build/testresult
  [junit] Running com.cenqua.samples.money.MoneyBagTest
  [junit] Tests run: 26, Failures: 0, Errors: 0, Time elapsed: 0.141 sec
  [junit] Running com.cenqua.samples.money.MoneyTest
  [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 3.079 sec

...

clover.snapshot:
[clover-snapshot] Clover Version ..., built on ...
[clover-snapshot] Loaded from: /tutorial/lib/clover.jar
[clover-snapshot] Clover: Commercial License registered to Atlassian.
[clover-snapshot] Snapshot file not found, creating new file at
/tutorial/.clover/clover.db.snapshot
```

### **6. Running an 'Empty' Optimized Build**

If we re-run the same Ant build, Clover will detect that none of the source files have changed. Because we are running a Test Optimized build, Clover won't build or test anything, the result is zero tests.

With a Test-Optimized build, Clover will output the following if no files have changed since the last build:

```
test.run:
  [junit] Clover estimates saving around 3 seconds on this optimized test run.
  [junit] Clover is including 0 test classes in this run (total test classes: 2)
```

This is desired behaviour, especially in a continuous integration environment where builds are automated and run regularly.

### 7. Editing a Java File in the Project

To show how Test Optimization works, we'll change one of the files in the project. When we run the Test Optimized build, Clover will detect that this file has changed and build it exclusively (rather than rebuilding everything).

Edit the file `MoneyBag.java` from the tutorial project. For the purposes of this demonstration, add a `System.out.println()` in the `add` method on line #27:

```
public IMoney add(IMoney m) {
    System.out.println("Adding: " + m);
    return m.addMoneyBag(this);
}
```

Now save the file.

### 8. Rebuilding the Project with Test Optimization

Now having changed a file in the project, we will run the same Ant tasks again.

```
ant with.clover test.run clover.snapshot
```

Clover will detect that the source file has changed, rebuilding and only running the tests for that file specifically. We can see this illustrated in the console output:

```
test.run:
  [junit] Clover estimates saving around 3 seconds on this optimized test run.
  [junit] Clover is including 1 test class in this run (total test classes: 2)
  [junit] Running com.cenqua.samples.money.MoneyBagTest
  [junit] Tests run: 26, Failures: 0, Errors: 0, Time elapsed: 0.146 sec
```

This Clover output shows that only one of two test classes was included. Note that all the test methods in this one test class were run, since Clover currently optimizes to the class level only. Clover also estimates the time saved in this particular build and test run. In this case the saving is only seconds, but in more complex projects it could well be multiple minutes or hours.

That concludes the Clover-for-Ant Test Optimization tutorial. For more information on integrating Test Optimization, see the related links below.

#### Related Links

- [Overview of Test Optimization](#)
- [Test Optimization Quick Start Guide](#)
- [Test Optimization Technical Details](#)
- [Test Optimization Quick Start for Maven 2](#)
- [Clover for Maven 2 - Test Optimization Best Practices](#)

## Clover-for-Maven tutorials

### Code examples

Checkout the <https://bitbucket.org/atlassian/maven-clover2-plugin> Mercurial repository.

In the **src/it** directory you'll find a number code examples showing integrations with various frameworks, inclusion filters, compilation of Java 8 and Groovy code etc.

## How to configure your Clover license

### Configuring your Clover license

This page contains instructions for configuring the licence file in all versions of Clover.

#### Installing a licence for Clover-for-Ant

1. You need a valid Clover license file to run Clover. You can obtain a free 30 day evaluation license or purchase a commercial license at <http://www.atlassian.com>.
2. To configure your clover.license file, do one of the following:
  - Place your clover.license file in CLOVER\_HOME/lib; or
  - Place the license file on the Java Classpath that will be used to run Clover; or
  - Place the license file on the file system somewhere, and then set the Java System Property clover.license.path to the absolute path of the license file.
3. If you are not finished, carry on with the [Clover-for-Ant installation](#). Enjoy using Clover.

#### Installing a licence for the Clover-for-Maven 2 plugin

1. The plug-in does not include a built-in evaluation license - you will need to download a license from [Atlassian](#).
2. Configure your license. You can either:
  - a. add it in your **.m2/settings.xml** file (so that it will become available for all projects running on a given machine):

```
<profiles>
  <profile>
    <id>my-clover-profile</id>
    <activation>
      <activeByDefault>>true</activeByDefault>
    </activation>
    <properties>
      <!-- You can define the path to a license file: -->

      <maven.clover.licenseLocation>/path/to/clover.license</maven.clover.licenseLocation>

      <!-- Or you can embed license key (remember to keep newline
      characters): -->
      <maven.clover.license><![CDATA[
        ...
      ]]></maven.clover.license>
    </properties>
  </profile>
</profiles>
```

- b. or add it in your **pom.xml** file:

```
<build>
  <plugins>
    <plugin>
      <groupId>com.atlassian.maven.plugins</groupId>
      <artifactId>maven-clover2-plugin</artifactId>
      <!-- Define a path to a license file: -->
      <licenseLocation>/path/to/clover.license</licenseLocation>

      <!-- Or embed a license key (remember to keep newline
characters): -->
      <license><![CDATA[
          ...
        ]]></license>
    </plugin>
  </plugins>
</build>
```

3. If you are not finished, carry on with the [Clover-for-Maven configuration](#). Enjoy using Clover.

## Installing a licence for the Clover-for-Eclipse plug-in

1. You need a valid Clover license file to run Clover. You can obtain a free 30 day evaluation license or purchase a commercial license at <http://www.atlassian.com>.
2. Open your valid trial, purchased or Open Source license file for Clover.
3. Within Eclipse, select from the menu "Window | Preferences" and click on Clover > License.
4. Paste the contents of your license file into the license text area or select your license file by clicking "Load...".
5. Click Apply. The license summary should now display status, type and message consistent with the type of license you entered.
6. Click OK to close the window.
7. If you are not finished, carry on with the [Clover-for-Eclipse Installation](#). Enjoy using Clover.

## Installing a license for the Clover-for-IDEA plug-in

1. Download Clover license file from <http://my.atlassian.com/>. Evaluation licenses are available free of charge.
2. Open the Clover license dialog in IDEA. Go to '**File > Settings > IDE Settings > Clover License**'. Click '**Load**' and select the 'clover.license' file you just downloaded. Close the window.
3. You're ready to use Clover-for-IDEA. See [Clover-for-IDEA User's Guide](#) to learn Clover's features.

## Hacking Clover

These pages show how it is possible to "hack" Clover and use it in a non-standard way, beyond the scope it was designed for.

- [Clover-for-Android](#)
- [Clover-for-Scala](#)
- [Converting XML to database format](#)
- [Measuring per-test coverage for manual tests](#)
- [Updating optimization snapshot file](#)
- [Using Clover for other programming languages](#)
  - [Instrumenting JSP files](#)
  - [Using Clover for PHP](#)

## Clover-for-Android

 The Clover for Android is in alpha stage and therefore it **is not officially supported** by Atlassian. The

following page was created for all *Clover-lovers* 😊 who'd love to use our tool on the Android platform.

Please **do not** raise Android-related issues on [Atlassian Support](#) - instead of this add comments to this page or raise questions on [Atlassian Answers](#) - we will review them and try to help in spare time.

Feel free to download and use the experimental Clover-for-Android version. Feel free to contribute by extending this manual.



We're proud to inform that Android support was initially created during one of our Atlassian [ShipIt](#) days.



### **New to Clover?**

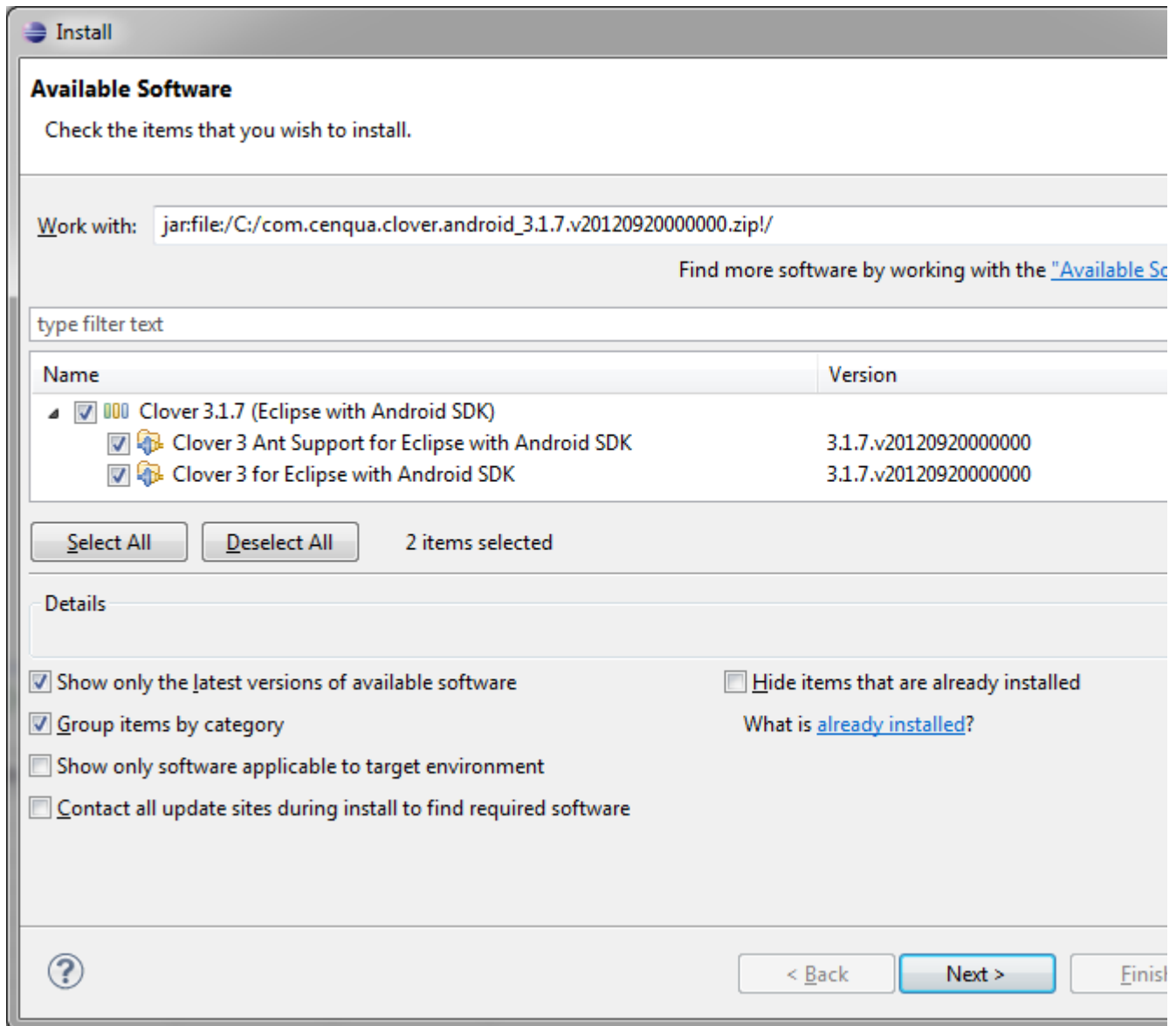
If you haven't used Clover before, we strongly recommend spending few minutes to learn its basic features:

[Clover for Eclipse in 10 minutes](#)

## **Installation**

*A prototype has been tested using following software versions, but it should work on other versions as well.*

- Download and install Eclipse for Java Developers from <http://www.eclipse.org/downloads> (recommended 4.2; use at least 3.6.2)
- Download and install Android SDK from <http://developer.android.com/sdk/index.html> (version 20.0.3; use at least version 8)
- Download Clover-for-Android from [http://www.atlassian.com/software/clover/downloads/binary/com.cenqua.clover.android\\_3.1.7.v20120920000000.zip](http://www.atlassian.com/software/clover/downloads/binary/com.cenqua.clover.android_3.1.7.v20120920000000.zip)
- Run Eclipse and install (*Help > Install New Software > Add ...*)
  - Google ADT from <https://dl-ssl.google.com/android/eclipse/> update site (version 20.0.3); don't select NDK Plugins to avoid download of Eclipse CDT
  - Clover-for-Android from downloaded zip file



**Note:** Clover-for-Android is based on the Clover-for-Eclipse 3.1.7.

**Note:** Clover-for-Android is using 'adb' application to fetch coverage data from a device. It expects to find it in:

- `<sdk>/platform-tools/adb.exe` (on Windows)
- `<sdk>/platform-tools/adb` (on other operating systems)

Please note that location of 'adb' has changed from `<sdk>/tools` to `<sdk>/platform-tools` since SDK v 8, so please don't use older SDK versions. Value of `<sdk>` is being fetched from 'com.android.ide.eclipse.adt.sdk' property in Eclipse Preferences ("Window > Preferences > Android > Android SDK").

**Note:** Clover-for-Android has been tested on a following configuration:

- Windows7 (64-bit) + Eclipse 4.2 for Java + JDK7 + ADT 20.0.3

## Clover-for-Android vs Clover-for-Eclipse

Clover for Android has following differences, compared to base Clover for Eclipse:

- The "initstring" can be expressed as URI
- The "Refresh Coverage Data" button fetches coverage data from Android device
- The "Delete Coverage Data" button deletes coverage data from Android device
- The CLOVER\_RUNTIME library (plugins/com.cenqua.clover.runtime\*.jar) size has been drastically



reduced in order to make it deployable on android devices

- There is no need to copy clover.db and clover.jar for application execution

A following features were not tested on Clover for Android:

- Distributed Coverage feature (i.e. sending coverage data via network socket from android device)
- Test Optimization

A following features do not work currently:

- Test Coverage Contribution view does not work when tests are kept in a separate project
- Test optimization does not work if tests are kept in a separate project (this does not work in Clover-for-Eclipse too, because Clover does not support cross-project test optimization)

## Usage

### Building Android application with Clover

- 1) Run Eclipse and open your Android project.
- 2) Right click on the project, select "*Clover > Enable on this project*" from context menu.
- 3) Right click on the project, select "Properties > Clover". Toggle on "Enable Clover in this project". Next select "Custom value" radio button and enter a path to Clover database in a following format:

**clover+remote:file:///<path to database file on desktop>?localCoverageDir=<path to coverage directory on device>**

Click OK. All Clover views shall be added to current perspective. You can also open them from "*Window > Show view > Other ... > Clover*".

### Possible InitString formats:

- plain path (absolute or relative) - for Java
- URI: **clover+remote:file:///<path to database file on desktop>?localCoverageDir=<path to coverage directory on device>** - for Android
- URI: **clover+local:file:///<path to database file on desktop>** - for Java

### Keep in mind that for URIs:

- on Windows platform you must encode ":" and "\" characters to make the initstring URI-compliant
  - : = %3A
  - \ = %5C
- you must have **file:///** with three slashes (because in URI format after two slashes we have a host name, which is not our case)

### Examples:


- **clover+remote:file:///c%3A%5CTemp%5Candroid.db?localCoverageDir=/data/data/com.example.android.notepad/clover**  
with "Relative to project dir" disabled  
remote coverage, database c:\Temp\android.db on Windows, coverage files in /data/data/com.example.android.notepad/clover on Android device,
- **clover+remote:file:///home/alice/workspace/android.db?localCoverageDir=/data/data/com.example.android.notepad/clover**  
with "Relative to project dir" disabled  
remote coverage, database /home/alice/android.db on Unix/MacOS, coverage files in /data/data/com.example.android.notepad/clover on Android device,
- **clover+local:file:///c%3A%5CTemp%5Cclover.db**

with "Relative to project dir" disabled  
local coverage, database c:\Temp\clover.db on Windows

- [clover/clover.db](#)  
with "Relative to project dir" enabled  
local coverage, database <project\_directory>/clover/clover.db
- [c:\Temp\clover.db](#)  
with "Relative to project dir" disabled  
local coverage, database c:\Temp\clover.db on Windows

4) Change the flush policy to *"At set intervals from a Clover thread"* or *"At set intervals"* and define interval (1000 ms for example).


Note that when you close application on Android device, JVM is still running, thus the *"At JVM shutdown ..."* option is not preferred.


 You can also trigger Clover flush programatically - just put `///CLOVER:FLUSH` inline comment in your code (for example in `Activity.onDestroy()` method).

5) Open *"Project Properties > Java Build Path > Order and Export"*. You will find a CLOVER\_RUNTIME library on a list. Enable the checkbox so that Clover library will be exported. Perform a full rebuild of the project. You should see "red coverage" in source files and in Coverage Explorer.

#### Running instrumented application on a device or emulator

1) Select *"Run as ... > Android application"* from main menu or Package Explorer context menu. You can execute your application or unit tests on a real device or simulator. Choose device you wish to use.

 There is no need to copy the **clover.db** to a device (yay!). Clover-for-Android is using a special coverage recorder version, which does not require presence of this database.

 There is also no need to copy **clover.jar** to a device (yay!). Google Android Toolkit will automatically package CLOVER\_RUNTIME jar file into Dalvik image during packaging.

#### Running unit tests on a device or emulator

##### Option #1

Unit tests for Android are kept in a separate Eclipse project.

For such scenario we have found a following configuration which works:

- add application project to Java Build Path in the test project (*"Project Properties > Java Build Path > Projects > Add ..."*)
- enable Clover in the test project (*"Project Properties > Clover > Enable on this project toggle"*)
- configure InitString (*"Project Properties > Clover > Instrumentation tab > Initstring box"*)
  - use `"clover+remote:"` URI as for application project
  - use `?localCoverageDir=` value **the same** as for application project (for example `"/data/data/com.my.app"`)
    - we've found some problems with permissions when trying to write to default data directory for test application (for example `"/data/data/com.my.app.tests"`)
  - use database name **different** than used for application project (e.g. `"/tmp/clover-tests.db"`)
    - you cannot use the same database name in two Eclipse projects
- configure flush policy (*"Project Properties > Clover > Instrumentation tab > Flush Policy box"*)
  - choose *"At JVM shutdown and on special instruction"*
- add the `///CLOVER:FLUSH` inline comment in `tearDown()` method for all JUnit test cases (or at least in the last test case executed) - see example below

- perform full rebuild of both projects

Right click on the test project and select "Run as ... > Android JUnit Test". After tests are finished, select both projects in "Coverage Explorer" view and click "Refresh Coverage Data" button for each of them.

You shall see coverage results for both projects.

## Option #2

Unit tests are kept together with application code in one Eclipse project.

*Configuration not tested yet.*

### Retrieving coverage data from a device

Just click on the "Refresh Coverage Data" button 😊

⚠ Clover is using "adb" command to retrieve coverage files from the **default** device. Make sure that you're running one device only, otherwise you might fetch coverage snapshots from wrong device.

### Viewing coverage results

As soon coverage data is fetched from device you can browse them using *Coverage Explorer* or *Test Run Explorer* view. It is also possible to generate HTML/XML/PDF reports, as usual.

### Cleaning coverage data

When you click on the "Delete Coverage Recordings", Clover will remove files from desktop as well as from android device.

⚠ Clover is deleting coverage files from a location defined in `initstring`. In case when `initstring` has changed, you have to remove files manually - open the "File Explorer" view from "DDMS" perspective and navigate to a directory where coverage data was stored (usually it will be `/data/data/com.my.application.name/clover`).

⚠ Clover is using "adb" command to delete coverage files from the **default** device. Make sure that you're running one device only, otherwise you might delete coverage snapshots from wrong device.

## Example 1 - manual testing

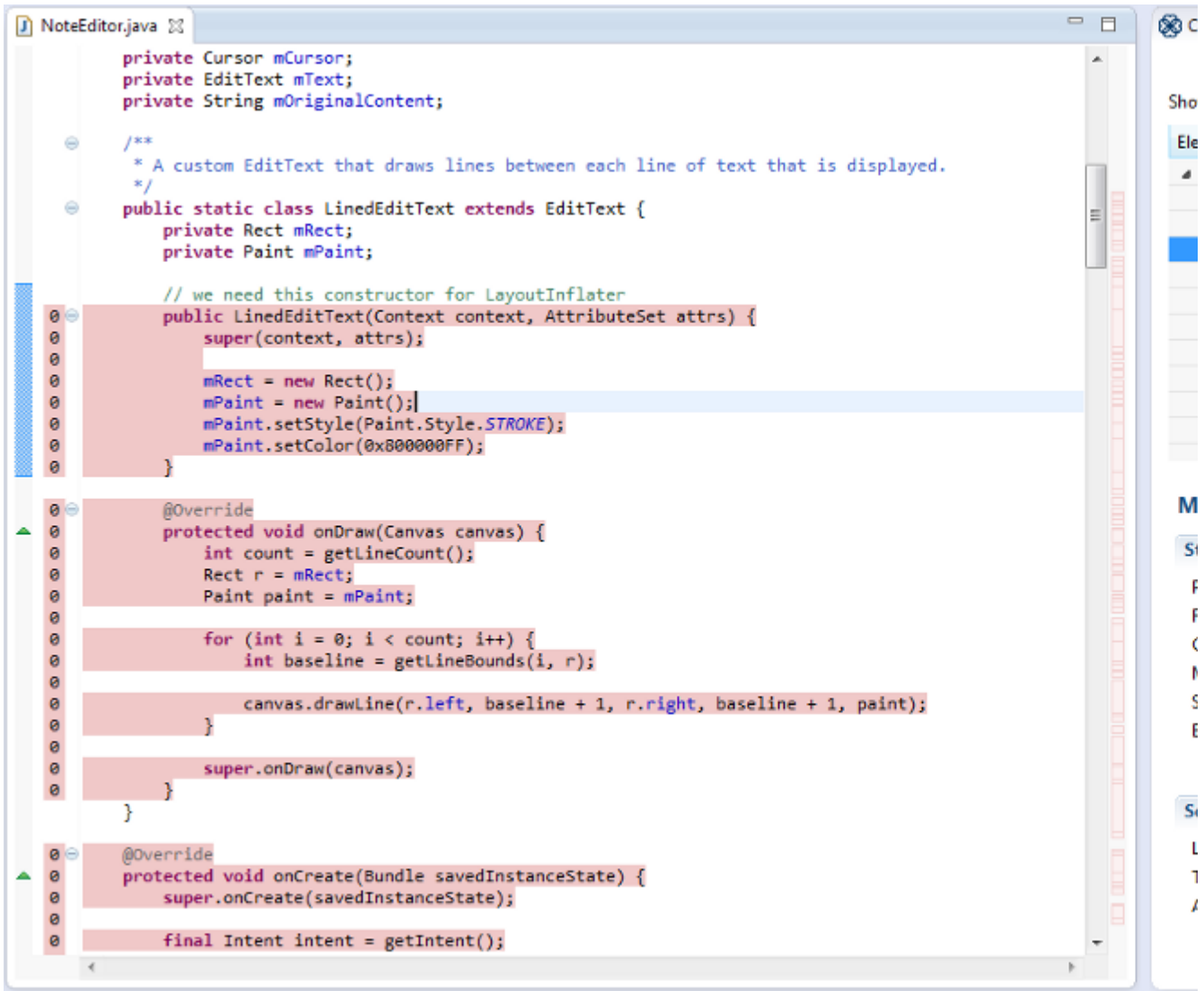
Open "File > New ... > Project ... > Android > Android Sample Project" and click "Next". Choose "Android 2.3.3" build target and click "Next". Choose "Notepad" sample, name it "NotePad" and click Finish.

Right click on the project, select "Clover > Enable on this project" from context menu. All Clover views shall be added to current perspective. You can also open them from "Window > Show view > Other ... > Clover".

Open "Project Properties > Java Build Path > Order and Export". You will find a CLOVER\_RUNTIME on a list. Tick the checkbox so that library will be exported.

Open "Project Properties > Clover". Set "`clover+remote:file:///c%3A%5CTemp%5Candroid.db?localCoverageDir=/data/data/com.example.android.notepad/clover`" `initstring`. Disable "Relative to project dir" checkbox. Set flush policy to "At set intervals from a Clover thread" with 1000 ms interval. Click OK.

Perform full rebuild. You should see red coverage in *Coverage Explorer* and text editors as on picture below.



```
NoteEditor.java
private Cursor mCursor;
private EditText mText;
private String mOriginalContent;

/**
 * A custom EditText that draws lines between each line of text that is displayed.
 */
public static class LinedEditText extends EditText {
    private Rect mRect;
    private Paint mPaint;

    // we need this constructor for LayoutInflater
    public LinedEditText(Context context, AttributeSet attrs) {
        super(context, attrs);
        mRect = new Rect();
        mPaint = new Paint();
        mPaint.setStyle(Paint.Style.STROKE);
        mPaint.setColor(0x800000FF);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        int count = getLineCount();
        Rect r = mRect;
        Paint paint = mPaint;

        for (int i = 0; i < count; i++) {
            int baseline = getLineBounds(i, r);

            canvas.drawLine(r.left, baseline + 1, r.right, baseline + 1, paint);
        }

        super.onDraw(canvas);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

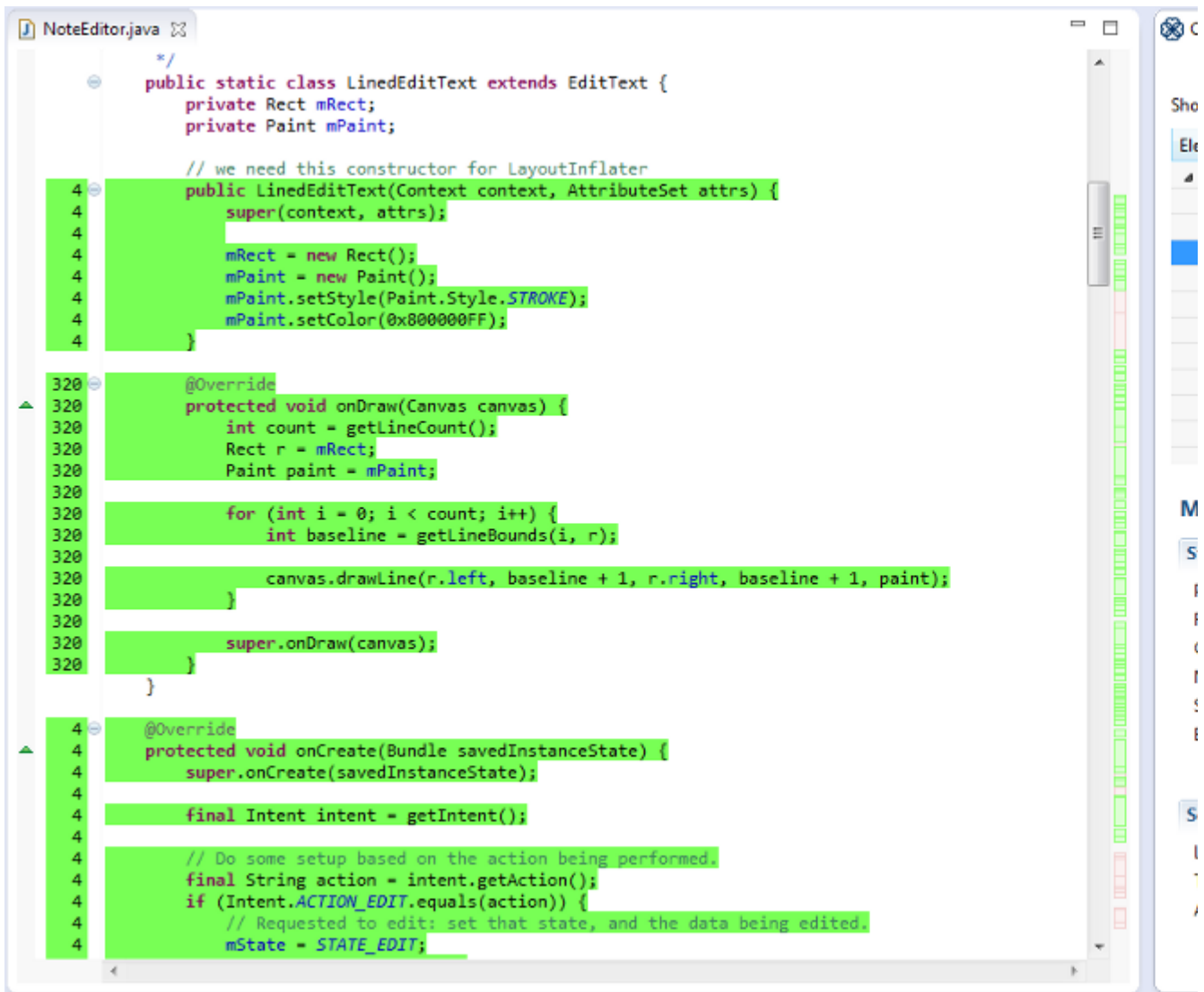
    final Intent intent = getIntent();
}
```

Now let's run the application on Android device. Right click on the project, select "Run as > Android application". Wait few minutes until emulator starts and installs our app.

Add new notes, delete it, change title etc. Exit application.



Click "Refresh Coverage Data" button in Coverage Explorer view. After few seconds you shall see coverage data like on picture below:

A screenshot of an IDE window titled 'NoteEditor.java'. The code defines a class 'LinedEditText' extending 'EditText'. It includes a constructor and two overridden methods: 'onDraw' and 'onCreate'. The code is annotated with green highlights, indicating code coverage. The IDE interface includes a scrollbar on the right and a 'Show Elements' panel on the far right.

```
public static class LinedEditText extends EditText {
    private Rect mRect;
    private Paint mPaint;

    // we need this constructor for LayoutInflater
    public LinedEditText(Context context, AttributeSet attrs) {
        super(context, attrs);
        mRect = new Rect();
        mPaint = new Paint();
        mPaint.setStyle(Paint.Style.STROKE);
        mPaint.setColor(0x800000FF);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        int count = getLineCount();
        Rect r = mRect;
        Paint paint = mPaint;

        for (int i = 0; i < count; i++) {
            int baseline = getLineBounds(i, r);
            canvas.drawLine(r.left, baseline + 1, r.right, baseline + 1, paint);
        }

        super.onDraw(canvas);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        final Intent intent = getIntent();

        // Do some setup based on the action being performed.
        final String action = intent.getAction();
        if (Intent.ACTION_EDIT.equals(action)) {
            // Requested to edit; set that state, and the data being edited.
            mState = STATE_EDIT;
        }
    }
}
```

You can browse coverage in IDE as well as generate XML/PDF/HTML reports, for example:

**NotePad Coverage Report**  
**Clover Coverage Report**

**Dashboard**  
 Coverage Reports  
 Coverage (Aggregate)  
 Test Code (Aggregate)  
 Test Results

**Application Packages**  
 com.example.android.notepad (45,3%)

Classes	Tests	Results
Class		Coverage
NoteEditor		(48,5%)
NoteEditor.LinedEditText		(100%)
NotePad		(0%)
NotePad.NoteColumns		(0%)
NotePadProvider		(42,7%)
NotePadProvider.DatabaseHelper		(25%)
NotesList		(25,4%)
NotesLiveFolder		(0%)
TitleEditor		(92,3%)

**Clover Coverage Report - NotePad Coverage Report**  
 Coverage timestamp: 5r wrz 19 2012 09:22:36 CEST

**Overview** Package File  
 FRAMES NO FRAMES SHOW HELP

**Statistics for pro**  
 Stmts: 313  
 Branches: 74  
 Methods: 35  
 Classes: 17

**Coverage** 17 classes, 191 / 422 elements  
 45,3%

**Class Coverage Distribution**

**Class Complexity**

**Most Complex Packages**

**Test Results** 0 / 0 tests 0 secs  
 No test results could be found. Please ensu

**Top 16 Project Risks**  
 NotesLiveFolder NotePa  
 NoteEditor NotePad.NoteC  
 TitleEditor R.R.attr R.drawable R.id R.layout

**Coverage Tree Map**  
 com.example.android.notepad

**Least Tested Methods**

- 0% NotePadProvider.insert(Uri, Conte
- 0% NoteEditor.cancelNote() : void (4
- 0% NotePadProvider.databaseHelper

## Example 2 - unit testing

Prerequisite: "NotePad" project configured as in Example 1.

Open "File > New ... > Project ... > Android > Android Sample Project" and click "Next". Choose "Android 2.3.3" build target and click "Next". Choose "Notepad > tests" sample, name it "NotePadTests" and click Finish.

Right click on the NotePadTests project in Package Explorer. Open "Project Properties > Java Build Path > Projects". Add "NotePad" project to the list.

Click on the "Order and Export" tab, select NotePad checkbox. Click OK.

Open "Project Properties > Clover". Select "Enable Clover in this project" checkbox.

Set "clover+remote:file:///c%3A%5CTemp%5Candroid-tests.db?localCoverageDir=/data/data/com.example.android.notepad/clover" initstring. Disable "Relative to project dir" checkbox.

Set flush policy to "At JVM shutdown and on special instruction". Click OK and close Properties window.

Note that there is no need to export CLOVER\_RUNTIME library, because it's already exported in the NotePad project.



Open NotePadTest class and add a tearDown() method like below:

```
public void tearDown() throws Exception {
    ///CLOVER:FLUSH
    super.tearDown();
}
```

Open NotePad/AndroidManifest.xml file and increase required API Level to 8 (it's required by ActivityInstrumentationTestCase2):

```
<uses-sdk android:minSdkVersion="8" android:targetSdkVersion="8"/>
```

Perform full rebuild.

Right click on the NotePadTest project, select "Run as ... > Android JUnit Test" from context menu. Wait until tests are finished.

Select NotePad and NotePadTests project in Coverage Explorer and for each of them click on the "Refresh Coverage Data" button.

After few seconds you shall see coverage results similar to those:

The screenshot displays the Eclipse IDE with the following components:

- Editor:** Shows the `NotePadTest.java` file. The code is highlighted in green, indicating 100% coverage. The code includes:
 

```
package com.example.android.notepad;
import android.test.ActivityInstrumentationTestCase2;
/**
 * Make sure that the main launcher activity opens up properly, which will be
 * verified by {@link #testActivityTestCaseSetUpProperly}.
 */
public class NotePadTest extends ActivityInstrumentationTestCase2<NotesList> {
    /**
     * Creates an {@link ActivityInstrumentationTestCase2} for the {@link Note:
    public NotePadTest() {
        super(NotesList.class);
    }
    /**
     * Verifies that the activity under test can be launched.
    public void testActivityTestCaseSetUpProperly() {
        NotesList activity = getActivity();
        assertNotNull("activity should be launched successfully", activity);
    }
    public void tearDown() throws Exception {
        System.out.println("Flushing Clover");
        ///CLOVER:FLUSH
        super.tearDown();
    }
}
```
- Coverage Explorer:** Shows the project structure with coverage data for various classes. The `NotesList` class is highlighted in blue, and the `NotesList` class is highlighted in green, indicating 100% coverage. The `NotePadTests` project is also highlighted in blue.



## Troubleshooting

In case of problems, you can search for more information in following places:

Open "*Window > Preferences > Clover*". Set "*Clover plugin logging output level*" to Debug or Verbose. Open Error Log console ("*Window > Show view > Error Log*"). You can track Clover messages like:

- retrieval of coverage data from device

```
Executing 'D:\Soft\Android\android-sdk\platform-tools\adb.exe pull
/data/data/com.example.android.notepad/clover/android-test.dbh3wa8ogtz4s
i_0_hut67c_h7aira7g.s
c:\Temp\android-test.dbh3wa8ogtz4si_0_hut67c_h7aira7g.s'
```

- removal of outdated files

```
deleting out of date coverage recording file:
android-test.dbhut67c_h7air9v5, timestamp < 1348064838148
```

- loading coverage snapshots

```
Read header for "c:\Temp\android-test.dbhut7iw_h7aj0xtu":
Header[dbVersion=1348064838148, writeTimeStamp=1348064886310, format=0]
```


- loading per-test coverage snapshots

```
Recording data for file
"c:\Temp\android-test.dbh4ffemvct3pe_0_hut7iw_h7aj0y4l.s":
PerTestRecordingTranscript[coverage.size=64,
testTypeName='com.example.android.notepad.NotePadTest',
testMethodName='com.example.android.notepad.NotePadTest.testActivityTest
CaseSetUpProperly',
exitMessage='null', stackTrace='null', exitStatus='Normal',
start=1348064884677]
```

Open "*Window > Open perspective > DDMS*".

- check messages in the *LogCat* view
- check if coverage files are written as specified in the *FileExplorer* view

## Known bugs

-  **CLOV-1194** - Implement `GrowablePerTestRecorder` and use it for Android instrumentation [OPEN](#)

## Android Project

We are waiting for your feedback! Feel free to vote on implementing full Android support in:

 **CLOV-569** - Android for Clover-for-Ant [OPEN](#) Clover for Ant

[+ CLOV-570](#) - Android for Clover-for-Maven [OPEN](#) Clover for Maven 2&3


[+ CLOV-1122](#) - Android for Clover-for-Eclipse [OPEN](#) Clover for Eclipse

[+ CLOV-1127](#) - Android for Clover-for-IDEA [OPEN](#) Clover for IDEA

[+ CLOV-1126](#) - Android for Clover Commandline Tools [OPEN](#) Clover Commandline Tools

## Clover-for-Scala

### DRAFT

 The Clover for Scala is the prototype and therefore it **is not officially supported** by Atlassian. The following page was created for all *Clover-lovers* 😊 who'd love to use our tool with the Scala language.

Please **do not** raise Scala-related issues on [Atlassian Support](#) - instead of this add comments to this page or raise questions on [Atlassian Answers](#) - we will review them and try to help in spare time.

Feel free to download and use the experimental version. Feel free to contribute by extending this manual. Feel free to vote on and comment Scala feature request in JIRA project. We're waiting for your feedback!!!

We're proud to inform that Android support was initially created during one of our Atlassian [Shiplt](#) days.

#### New to Clover?

If you haven't used Clover before, we strongly recommend spending few minutes to learn its basic features:

1. [QuickStart Guide](#) for Clover-for-Ant

## Installation

*A prototype has been tested using following software versions:*

- *Scala 2.8.3*
- *Ant 1.8.4*
- *JDK7*

Download and extract <http://www.atlassian.com/software/clover/downloads/binary/clover-ant-scala-3.1.8.zip>.

## Usage

Building Scala application with Clover

Running instrumented application

## Running unit tests

## Generating and viewing coverage results

## Current limitations

The prototype recognizes following language constructs: statement, method, class. It does not measure branch coverage as well as does not recognize closures (they are not shown in the report).

## Troubleshooting

In case of problems, you can search for more information in following places:

- [Atlassian Answers](#)

## Scala feature request

We are waiting for your feedback! Feel free to vote on implementing full Scala support in Clover:

[+ CLOV-1142](#) - Expose a Service Provider Interface for Clover [OPEN](#)

[+ CLOV-932](#) - Provide support for the Scala language [OPEN](#)

## Converting XML to database format

### DRAFT PAGE DO NOT PUBLISH

There is a problem with XmlConverter - it does not read <line> tags, as a consequence do not insert methods/statements/branches into the database. As a result the db model is very poor and thus an HTML report produced not very usable.

By the way - there is also a problem with PHPUnit which produces XML without a <package> tag leading to NPE in XmlConverter.

We should rather wait until DSL converter will be ready by Michael - see [CLOV-1383](#) - and next rewrite this page.

Normally, the XML report is produced **from** the Clover database.

But you can hack Clover and reverse the direction - create a database using data from XML report. Such conversion have some drawbacks of course - database will be incomplete, because the database normally holds much more data than is available in the XML file.

What are benefits of such conversion in such case? Well, you can create an HTML report out of the "reverse-engineered" database. It means that you could generate a Clover HTML report from **any** code coverage tool which can produce an XML report in Clover-compatible format. A good example of such tool is [PHPUnit](#) (`phpunit --coverage-clover clover.xml`).

How to convert:

```
// todo add SimpleXml2DbConverter source here
```

## Measuring per-test coverage for manual tests

Clover can measure code not only code coverage from whole application run but also a coverage generated by a single test case (JUnit / TestNG). It is possible to "hack" Clover and measure per-test coverage from manual test cases too. What has to be done is to "tell" Clover when manual test case starts and ends.

The easiest way to provide this information is to write a JUnit test case, with one test method for each manual test case. Clover will add "start test / end test" instrumentation to such methods. Next it's necessary run such JUnit test together with the application - thanks to the Clover's Distributed Coverage feature it's possible to run JUnit test and the application under test in separate JVMs.

### Steps

1) Write JUnit test case having following features:

- tests do not start until application under test is launched
- one test method per one manual test case
- single test method starts just before corresponding manual test case is started
- single test method ends just after corresponding manual test case is finished

Example:

```

import junit.framework.TestCase;

public class MyManualTest extends TestCase {

    public static void main(String args[]) {
        MyManualTest myTest = new MyManualTest();
        myTest.waitForApplicationStarts();
        int testNo = myTest.getTestNumber();
        switch (testNo) {
            case 1:
                myTest.testManualTest1();
                break;
            case 2:
                myTest.testManualTest2();
                break;
        }
    }

    private int getTestNumber() {
        // e.g. entered by user / read from commandline arg ...
        return 1;
    }

    private void waitForApplicationStarts() {
        // e.g. "Press any key when ready" / check for existence of some marker file
    }

    private void waitForTestEnds() {
        // e.g. "Press any key when test is finished" / check for some marker file
    }

    public void testManualTest1() {
        // Clover will add "test start" here
        waitForTestEnds();
        // Clover will add "test end" here
    }

    public void testManualTest2() {
        // Clover will add "test start" here
        waitForTestEnds();
        // Clover will add "test end" here
    }
}

```

2) Compile JUnit test case together with the application under test (produce two jars, for instance) with a Distributed Coverage feature enabled.

- flushpolicy = interval or threaded might be needed

Example:

```

<clover-setup initstring="/path/to/my/clover.db" flushpolicy="interval"
flushinterval="1000">
    <distributedCoverage/>
</clover-setup>

```

3) Run unit test and application:

- a) unit test shall be launched with -Dclover.server=true parameter, e.g.

```
java -cp ...;clover.jar -Dclover.server=true MyManualTest
```

b) application shall be launched as usual (note that distributed coverage configuration is already compiled into instrumented classes), e.g.

```
java -cp ...;clover.jar MyApplication
```

### Browsing per-test coverage

4) Generate coverage report. In the HTML report it's possible to browse coverage contributed by given test case(s) by clicking "Show tests" link on the class summary page.

### Per-test optimization for manual tests

Because of fact that all manual test cases were wrapped into JUnit tests it's possible use per-test optimization:

- create test optimization snapshot file after test execution (<clover-snapshot/> task)
- after code base changes, run optimized test set (using the <clover-optimized-testset/> selector for <junit/> task)
  - note that Clover-for-Ant optimization is based on test classes, not test methods (it's a limitation of <junit> <batchtest>) so it' might be worth to have one test method per test class

It is also possible to browse "source file - test case" and "class file - test case" mapping using the SnapshotPrinter tool:

```
java -cp clover.jar com.atlassian.clover.optimization.SnapshotPrinter  
clover.snapshot.file
```

SnapshotPrinter can print mapping in plain text or in JSON format (since 3.1.11).

## Updating optimization snapshot file

### Introduction

Imagine a following scenario:

- your project has a set of integration tests which follow a classic maven-failsafe-plugin approach:
  - tests are being executed in the 'integration-test' phase
  - but results are checked in the 'verify' phase
- test cases are being recognized by Clover
  - ... just because they are JUnit / TestNG test cases or you have defined custom test patterns for clover2:setup MOJO
  - ... you can see them on the "Tests"
- you would like to use test optimization for them
  - i.e. re-run only those which were failed or related sources were modified
- but Clover does not see test failures for them ⚠️ (in the clover.snapshot) and thus you cannot optimize your tests correctly
  - this is because test case is executed in 'integration-test' phase but it does not throw any exception (like `AssertionError` for JUnit) because verification is performed in later phase

How to solve this?

### Solution


You can update Clover's Optimization Snapshot file "manually". Exact integration would depend on the framework you are using and you need to write such integration, but in general it works this way:

- 1) Instrument code and run integration-tests (*mvn clean clover:setup integration-test*)
- 2) Save optimization snapshot file (*clover2:snapshot*)
- 3) Run test result verification (*mvn verify*) and store failed test results (somewhere)
- 4) Update optimization snapshot file and set status for failed tests (see code example below)

### Example

A simple application which sets test duration and test failure for certain test case:

<https://bitbucket.org/atlassian/maven-clover2-plugin> (src/it/optimized/snapshot-hacking)

 You must use Clover 3.1.11 or later.

## Using Clover for other programming languages

### General approach

Clover works with Java and Groovy languages. If you have code written in other programming languages, you could potentially generate XML/HTML/PDF/JSON reports for them as well. However, there are some prerequisites:

- you have another code coverage tool for that language (for example Cobertura or Emma for JVM languages)
- a structure of this programming language can be somehow mapped to the java/groovy-like structure (file-class-method-statement)
- you know the data format of the another code coverage tool

If all prerequisites are fulfilled, you could write a data converter from that coverage tool to Clover's database format.

- All you need to do is to load Clover database and call certain callback methods to fill the database with actual data. See the [Database Structure](#) page for more details.

As soon as you have the data converter ready, you could run it within your build to generate Clover database. You could also merge the resulting database with a database generated by Clover for Java/Groovy sources - thanks to this you could have a single, consolidated report. Next you can run Clover reporting tools to get reports you need.

And how the HTML report would look like for an unknown language? Well, you would see classes and methods as usual, but it Clover would use a plain text formatter for a source page (Clover has syntax highlighters for Java and Groovy only).

### Other solutions

- [PHP](#)
- [JSP](#)

## Instrumenting JSP files

Clover cannot instrument JSP files directly. However, as all JSP files are translated to Java and next compiled using standard javac compiler, it is possible to enhance generated Java sources with Clover instrumentation and compile them. Thanks to this it is possible to get code coverage and reports for them.

### JSP on Tomcat

By default, Tomcat compiles JSP file on the first access. Generated Java source code as well as compiled classes are stored in `<tomcat_home>/work/Catalina/localhost/<application_name>`. Luckily, Tomcat allows to perform pre-compilation of JSP files - classes should be bundled into WAR and proper servlet definitions added to web.xml. More details can be found on <http://tomcat.apache.org/tomcat-6.0-doc/jasper-howto.html> page.

### Examples

#### Tomcat

- 1) Download sample application from <http://tomcat.apache.org/tomcat-6.0-doc/appdev/sample/>
- 2) Take sample Ant build script from <http://tomcat.apache.org/tomcat-6.0-doc/jasper-howto.html>
- 3) Add Clover-related tasks and properties, for instance:

```
<property name="clover.jar" location="${user.home}/clover.jar"/>
<property name="clover.db" location="${java.io.tmpdir}/clover/clover.db"/>
<property name="clover.license.path" location="${user.home}/clover.license"/>
<taskdef resource="cloverlib.xml" classpath="${clover.jar}"/>

<target name="init">
  <clover-setup initstring="${clover.db}" flushpolicy="interval"
flushinterval="500"/>
</target>

<target name="report">
  <clover-html-report initstring="${clover.db}" outdir="report"/>
</target>

...

<target name="all" depends="init, jspc, compile"> <!-- add "init" -->
```

#### Comments:

- the "init" target has `<clover-setup>` task with `flushpolicy="interval"` set (so that coverage recording files will be written at specified time interval, instead of JVM shutdown - thanks to this it's required to shutdown Tomcat)
- the "init" target has "inistring" with an absolute path to database (in order to simplify deployment as there's no need to copy clover.db to Tomcat directory)
- the "all" target depends on "init, jspc, compile"

- 4) Build application using "ant" command:

- JSP files are precompiled into `WEB-INF/classes/org/apache/jsp`
- clover.db is created in `${java.io.tmpdir}/clover`
- the `WEB-INF\generated_web.xml` is created

- 5) Copy definitions of servlets from `WEB-INF/generated_web.xml` into `WEB-INF/web.xml`. Otherwise Tomcat would compile JSP files (without Clover instrumentation).

- 6) Copy also clover.jar into `<tomcat_home>/lib` directory.



- 7) Package sample.war and deploy to Tomcat (copy to <tomcat\_home>/webapps).
- 8) Open <http://localhost:8080/sample> page, click on the "To JSP page" link. Page should print "hello". Coverage files shall be generated in the location of clover.db database.
- 9) Run "ant report" in order to generate coverage report.

**Clover Coverage Report**

Dashboard  
 Coverage Reports   
 Coverage (Aggregate)  
 Test Code (Aggregate)  
 Test Results

**Application Packages**  
 org.apache.jsp (63,4%)

Class	Coverage
hello_jsp	(63,4%)

**Clover Coverage Report -**  
 Coverage timestamp: Pn lis 19  
 2012 10:18:53 CET

Overview Package **File**

FRAMES NO FRAMES SHOW  
 HELP

0% of files have more coverage

Expand All

hello_jsp	Line # 7	Total Statements 31	Complexity 10
-----------	----------	---------------------	---------------

No Tests

Collapse All

```

1 package org.apache.jsp;
2
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5 import javax.servlet.jsp.*;
6
7 public final class hello_jsp extends org.apache.jsp
8     implements org.apache.jasper.runtime.JspSource
9
10     private static final JspFactory _jspxFactory =
11
12     private static java.util.List _jspx_dependants;
13
14     private javax.el.ExpressionFactory _el_expressio
15     private org.apache.AnnotationProcessor _jsp_anno
16
17 0 public Object getDependants() {
18 0     return _jspx_dependants;
19
20
21 1 public void _jspInit() {
22 1     el expressionfactory = _jspxFactory.getJspApr
  
```

## Using Clover for PHP

Clover does not support PHP language. However, there is a PHPUnit framework which can measure code coverage for PHP application and to export coverage data to the Clover's XML report file format.

Just use `--coverage-clover` toggle, for example:

```
phpunit --log-junit 'reports/unitreport.xml' --coverage-clover
'reports/clover.xml' test/
```

Such XML file can be later used by Bamboo Clover Plugin to display code statistics on a Job Summary page and graphs on a Plan Summary page. In order to achieve this, you have to configure manual Clover integration

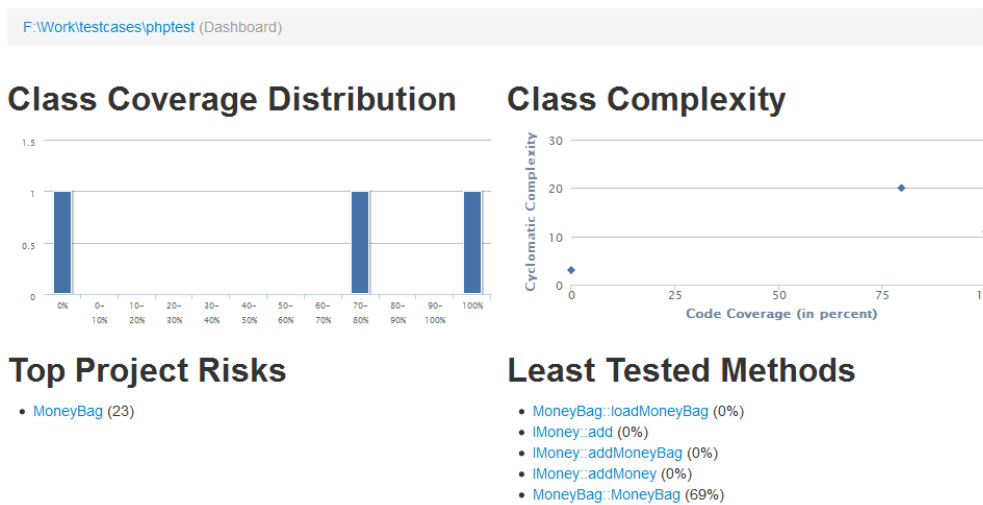
in Bamboo, pointing to the XML file generated by PHPUnit. See [Enabling the Clover add-on](#) for details.

There is also a similar plug-in available for Jenkins - see [Clover PHP Plugin](#) page.

Note: PHPUnit has also an option to produce the HTML report, but this is **not** a report in Clover's HTML format.

```
phpunit --log-junit 'reports/unitreport.xml' --coverage-html
'reports/clover_html' test/
```

A report produced by PHPUnit looks like this:



Generated by PHP\_CodeCoverage 1.2.13 using PHP 5.5.5 and PHPUnit 3.7.28 at Thu Oct 31 14:33:33 UTC 2013.

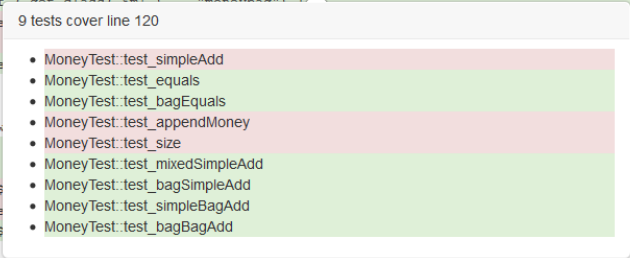
F:\Work\testcases\phptest / money.php5

	Code Coverage									
	Classes and Traits			Functions and Methods				Lines		
Total	<div style="width: 33.33%;"></div>	33.33%	1 / 3	<div style="width: 73.68%;"></div>	73.68%	14 / 19	CRAP	<div style="width: 82.09%;"></div>	82.09%	55 / 67
IMoney	<div style="width: 0.00%;"></div>	0.00%	0 / 1	<div style="width: 0.00%;"></div>	0.00%	0 / 3	12	<div style="width: 0.00%;"></div>	0.00%	0 / 3
add( \$money )	<div style="width: 0.00%;"></div>	0.00%	0 / 1	<div style="width: 0.00%;"></div>	0.00%	0 / 1	2	<div style="width: 0.00%;"></div>	0.00%	0 / 1
addMoney( \$money )	<div style="width: 0.00%;"></div>	0.00%	0 / 1	<div style="width: 0.00%;"></div>	0.00%	0 / 1	2	<div style="width: 0.00%;"></div>	0.00%	0 / 1
addMoneyBag( \$moneyBag )	<div style="width: 0.00%;"></div>	0.00%	0 / 1	<div style="width: 0.00%;"></div>	0.00%	0 / 1	2	<div style="width: 0.00%;"></div>	0.00%	0 / 1
Money	<div style="width: 100.00%;"></div>	100.00%	1 / 1	<div style="width: 100.00%;"></div>	100.00%	7 / 7	11	<div style="width: 100.00%;"></div>	100.00%	20 / 20
Money( \$amt, \$curr )	<div style="width: 100.00%;"></div>	100.00%	1 / 1	<div style="width: 100.00%;"></div>	100.00%	1 / 1	1	<div style="width: 100.00%;"></div>	100.00%	3 / 3
add( \$money )	<div style="width: 100.00%;"></div>	100.00%	1 / 1	<div style="width: 100.00%;"></div>	100.00%	1 / 1	1	<div style="width: 100.00%;"></div>	100.00%	1 / 1
addMoney( \$money )	<div style="width: 100.00%;"></div>	100.00%	1 / 1	<div style="width: 100.00%;"></div>	100.00%	1 / 1	2	<div style="width: 100.00%;"></div>	100.00%	5 / 5
addMoneyBag( \$moneyBag )	<div style="width: 100.00%;"></div>	100.00%	1 / 1	<div style="width: 100.00%;"></div>	100.00%	1 / 1	1	<div style="width: 100.00%;"></div>	100.00%	1 / 1
equals( \$money )	<div style="width: 100.00%;"></div>	100.00%	1 / 1	<div style="width: 100.00%;"></div>	100.00%	1 / 1	4	<div style="width: 100.00%;"></div>	100.00%	8 / 8
amount()	<div style="width: 100.00%;"></div>	100.00%	1 / 1	<div style="width: 100.00%;"></div>	100.00%	1 / 1	1	<div style="width: 100.00%;"></div>	100.00%	1 / 1
currency()	<div style="width: 100.00%;"></div>	100.00%	1 / 1	<div style="width: 100.00%;"></div>	100.00%	1 / 1	1	<div style="width: 100.00%;"></div>	100.00%	1 / 1
MoneyBag	<div style="width: 0.00%;"></div>	0.00%	0 / 1	<div style="width: 77.78%;"></div>	77.78%	7 / 9	23.42	<div style="width: 79.55%;"></div>	79.55%	35 / 44
MoneyBag( \$m1, \$m2=FALSE )	<div style="width: 0.00%;"></div>	0.00%	0 / 1	<div style="width: 0.00%;"></div>	0.00%	0 / 1	5.73	<div style="width: 69.23%;"></div>	69.23%	9 / 13
loadMoneyArray( \$moneyArray )	<div style="width: 100.00%;"></div>	100.00%	1 / 1	<div style="width: 100.00%;"></div>	100.00%	1 / 1	2	<div style="width: 100.00%;"></div>	100.00%	5 / 5

```

111
112 // Due to current version of PHP (4.3) not supporting 'Overriding' of methods...
113 // this has been somewhat 'fudged' to work in a similar fashion...
114
115 // the first parameter can be either an array of monies, a money object, or a money bag object...
116 // the second parameter can be either a money object, or a money bag object, and is optional...
117
118 // deal with the first parameter...
119 if ( is_array( $m1 ) ){
120     $this->loadMoneyArray( $m1 );
121 } else if ( is_object( $m1 ) && $m1 instanceof Money ){
122     $this->loadMoneyObject( $m1 );
123 } else {
124     $this->loadMoneyBag( $m1 );
125 }
126
127 // deal with the second parameter...
128 if ( $m2 ){
129     if ( is_array( $m2 ) ){
130         $this->loadMoneyArray( $m2 );
131     } else if ( is_object( $m2 ) && $m2 instanceof Money ){
132         $this->loadMoneyObject( $m2 );
133     } else {
134         $this->loadMoneyBag( $m2 );
135     }

```



## References

- <http://phpunit.de/manual/3.8/en/index.html>
- <https://github.com/sebastianbergmann/phpunit/>
- BAMBOO/Enabling the Clover add-on
- <https://wiki.jenkins-ci.org/display/JENKINS/Clover+PHP+Plugin>

# Glossary

[branch coverage](#)

[code coverage](#)

[coverage](#)

[coverage clouds](#)

[decision coverage](#)

[history point](#)

[interval](#)

[method coverage](#)

[span](#)

[statement coverage](#)

[test coverage](#)

## branch coverage

Branch coverage (or 'decision coverage') is a [code coverage](#) metric that measures which possible branches in flow control structures are followed. Clover does this by recording if the boolean expression in the control structure evaluated to both true and false during execution.

## code coverage

Code coverage (or 'test coverage', or just 'coverage') is a measurement, usually expressed as a percentage, of how much of your source-code is being executed by your test suite.

In general, a code coverage system collects information about the running program, then combines that with

source information to generate a report on the test suite's code coverage.

This information can then be used to improve the quality of the test suite, either by adding tests or modifying existing tests to increase coverage.

## coverage

Code coverage (or 'test coverage', or just 'coverage') is a measurement, usually expressed as a percentage, of how much of your source-code is being executed by your test suite.

In general, a code coverage system collects information about the running program, then combines that with source information to generate a report on the test suite's code coverage.

This information can then be used to improve the quality of the test suite, either by adding tests or modifying existing tests to increase coverage.

## coverage clouds

A [Tag Cloud](#) or 'weighted list' is a way of visually representing information.

In Clover, '**Coverage Clouds**' provide an instant overview of your entire project and individual packages, enabling you to identify areas of your code that pose the highest risks or shortcomings.

For details please see [Tag Clouds](#).

## decision coverage

Branch coverage (or 'decision coverage') is a [code coverage](#) metric that measures which possible branches in flow control structures are followed. Clover does this by recording if the boolean expression in the control structure evaluated to both true and false during execution.

## history point

A history point is a point-in-time which you define. It is used to generate [historical coverage reports](#).

Also see the `<clover-historypoint>` Ant task.

## interval

An interval specifies a period of time for use by the `<span>` attribute (see [Using Spans](#) and [Specifying an Interval](#)).

## method coverage

Method coverage is a [code coverage](#) metric that measures whether a method was entered at all during execution.

## span

The `span` attribute allows you to control which coverage recordings are merged to form a current coverage report. By default, Clover includes all coverage data found. You can configure it to include a different span of coverage recordings. The span attribute lets you do this.

See also [Using Spans](#).

## statement coverage

Statement coverage is a [code coverage](#) metric that measures which statements in a body of code have been executed through a test run, and which statements have not.

## test coverage

Code coverage (or 'test coverage', or just 'coverage') is a measurement, usually expressed as a percentage, of how much of your source-code is being executed by your test suite.

In general, a code coverage system collects information about the running program, then combines that with source information to generate a report on the test suite's code coverage.

This information can then be used to improve the quality of the test suite, either by adding tests or modifying existing tests to increase coverage.

# Clover FAQ

## Clover FAQ

Answers to frequently asked questions about configuring and using Clover:

- [Top Questions For Evaluators](#)
  - [Can Clover Optimise My Tests?](#)
  - [Can I Exclude Files From Clover Coverage Reports?](#)
  - [Can The Clover Reports Be Configured?](#)
  - [Does Clover Depend On JUnit?](#)
  - [How Does Clover Benefit Developers?](#)
  - [What Are The Limitations Of Code Coverage?](#)
  - [What Build Tools Does Clover Integrate With?](#)
  - [What Is Code Coverage Analysis?](#)
  - [Why Does Clover Use Source Code Instrumentation?](#)
  - [Will Clover Integrate With My IDE?](#)
- [Concepts & Usage FAQ](#)
  - [Can I create a Clover Report on Server A if I have the clover.db which I generated on Server B?](#)
  - [Does Clover depend on JUnit?](#)
  - [Does Clover integrate with Maven?](#)
  - [Does Clover support the new language features in JDK1.5?](#)
  - [Does Clover work with JUnit4 and TestNG?](#)
  - [How are the Clover coverage percentages calculated?](#)
  - [How do I compare the code coverage between two releases of my code?](#)
  - [How do I get started with Clover?](#)
  - [How do I use Clover with NetBeans?](#)
  - [What are the limitations of Code Coverage?](#)
  - [What does the name "Clover" mean?](#)
  - [What is Code Coverage Analysis?](#)
  - [What is the coverage.db file and why am I seeing files like coverage.dbxxxxxxxx\\_xxxx\\_xxxx?](#)
  - [What third-party libraries does Clover utilise?](#)
  - [Where did Clover originally come from?](#)
  - [Why does Clover instrument classes I have excluded using the 'exclude' element of the 'clover-setup' task?](#)
  - [Why does Clover use source code instrumentation?](#)
  - [Will Clover integrate with my IDE?](#)
- [Eclipse Plugin FAQ](#)
  - [I only need instrumented classes for unit testing and I don't want to risk publishing them to my production environment. How can I do this with Clover?](#)
  - [Is Clover supported on IBM's RAD?](#)
  - [I store my plugins and features in an Eclipse extension area. Does Clover support this?](#)
  - [Why can I only see coverage data for the last test case I executed?](#)
- [IDEA Plugin FAQ](#)
  - [I've run my tests, but coverage information does not show in IDEA](#)
  - [What does enabling Instrument Test Source Folders do?](#)
  - [Where does IDEA write its log file?](#)
- [Maven 2 and 3 Plugin FAQ](#)
  - [Deploying Instrumented Jars](#)
  - [How to keep Clover reports between builds?](#)
  - [How to remove -clover suffix from artifact name?](#)
  - [Is there an alternative to using the Maven Central repository?](#)
  - [Preparing multi-module projects for remote deployment with Clover-for-Maven 2](#)
  - [Troubleshooting License problems](#)
  - [Troubleshooting problems with displaying characters](#)

- Support Policies
  - Bug Fixing Policy
  - How to Report a Security Issue
  - New Features Policy
  - Security Advisory Publishing Policy
  - Security Update Policy
  - Severity Levels for Security Issues
  - Update Policy
- Troubleshooting
  - Compiling my instrumented sources fails with a 'code too large' error.
  - For some statements in my code Clover reports "No Coverage information gathered for this expression". What does that mean?
  - Hit count for multi-threaded test is incorrect in Clover's report.
  - I'm trying to get a coverage report mailed, but I keep getting "mail Failed to send email". How do I fix this?
  - I'm using the maven-clover-plugin version 2.4 with a license downloaded from Atlassian and get the message 'Invalid or missing License'
  - Tools for Troubleshooting Clover-for-Ant
  - Two questions to ask yourself first when troubleshooting Clover
  - When generating some report types on my UNIX server with no XServer, I get an exception "Can't connect to X11 server" or similar.
  - When using Clover, why do I get a java.lang.NoClassDefFoundError when I run my code?
  - When using Clover from Ant, why do I get "Compiler Adapter 'org.apache.tools.ant.taskdefs.CloverCompilerAdapter' can't be found." or similar?
  - Why does the 'Test Results' summary page report show that I have unique coverage, when the source page shows no unique coverage?
  - Why do I get 0% coverage when I run my tests and then a reporter from the same instance of Ant?
  - Why do I get a 'java.lang.OutOfMemoryError - PermGen space' error?
  - Why do I get an java.lang.OutOfMemoryError when compiling with Clover turned on?



Do you have a question, or need help with Clover? Please create a support request or post a question on a forum.

## Concepts & Usage FAQ

## Clover Concepts & Usage FAQ

- [Can I create a Clover Report on Server A if I have the clover.db which I generated on Server B?](#)
- [Does Clover depend on JUnit?](#) — Clover has no dependence on JUnit. We mention it frequently in our documentation only because of JUnit's widespread use in the Java development community.
- [Does Clover integrate with Maven?](#) — There is a Clover Plugin for Maven and Maven2 — both are independent open source developments supported by Cenqua/Atlassian.
- [Does Clover support the new language features in JDK1.5?](#) — Clover fully supports all JDK1.5 language features.
- [Does Clover work with JUnit4 and TestNG?](#) — Clover is fully compatible with JUnit4 and TestNG.
- [How are the Clover coverage percentages calculated?](#) — The "total" coverage percentage of a class (or file, package, project) is provided as a quick guide to how well the class is covered — and to allow ranking of classes.
- [How do I compare the code coverage between two releases of my code?](#)
- [How do I get started with Clover?](#) — See the Clover-for-Ant QuickStart Guide
- [How do I use Clover with NetBeans?](#)
- [What are the limitations of Code Coverage?](#) — Code Coverage is not a "silver bullet" of software quality, and 100% coverage is no guarantee of a bug-free application. You can infer a certain level of quality in your tests based on their coverage, but you still need to be writing meaningful tests.
- [What does the name "Clover" mean?](#) — Clover is actually a shortened version of the tool's original name, "Cover Lover", from the nick name that the tool's author gained while writing Clover ("Mr Cover Lover").
- [What is Code Coverage Analysis?](#) — Code Coverage Analysis is the process of discovering code within a program that is not being exercised by test cases.
- [What is the coverage.db file and why am I seeing files like coverage.dbxxxxxxxx\\_xxxx\\_xxxx?](#) — The coverage.db file is the instrumentation database telling Clover the structure of your project and files during the last instrumentation event. coverage.dbxxxxxxxx\_xxxx\_xxxx hold the code coverage from your unit test or application run.
- [What third-party libraries does Clover utilise?](#)
- [Where did Clover originally come from?](#) — Clover was originally developed at Cenqua (now part of Atlassian) as an internal tool to support development of large J2EE applications.
- [Why does Clover instrument classes I have excluded using the 'exclude' element of the 'clover-setup' task?](#) — There are two possible causes:
- [Why does Clover use source code instrumentation?](#) — Source code instrumentation is the most powerful, flexible and accurate way to provide code coverage analysis.
- [Will Clover integrate with my IDE?](#) — Clover 2 provides an integrated plugin for Eclipse, with more plugins soon to follow.

### Can I create a Clover Report on Server A if I have the clover.db which I generated on Server B?

Yes you can if you use the command line options. You need to copy over your coverage.db and all the coverage.db\* files that were generated as well as copying over the clover.jar and license that you used.

However if you do not have the source files on your Server A, that you did have on your Server B you are going to get a number of the following errors

```
ERROR: C:/Applications/Confluence_STD/Source
Code/confluence-2.9-source/confluence-project/confluence/src/test/java/c
om/atlassian/integrationtest/confluence/core/TestConfluenceClasspath.ja
va (No such file or directory)
```

The report you generate will have all the coverage statistics but when you try and drill down to the class, you will get a error stating it cannot find the source file.



If you do have the source on Server A you can specify it using the `-p` option from [Console reporter](#).

## Does Clover depend on JUnit?

**Q: Does Clover depend on JUnit?** Clover has no dependence on JUnit. We mention it frequently in our documentation only because of JUnit's widespread use in the Java development community. You can certainly instrument your code and run it however you like; Clover will still record coverage which can then be used to generate reports.

## Does Clover integrate with Maven?

### Q: Does Clover 1 integrate with Maven?

There is a Clover Plugin for Maven and Maven2 — both are independent open source developments supported by Cenqua/Atlassian. See the [Maven](#) website for details.

### Q: Does Clover 2 integrate with Maven?

Atlassian has brought Maven plugin development in-house. The Maven plugin remains open source. See the [instructions for using Clover with Maven 2](#)

## Does Clover support the new language features in JDK1.5?

### Q: Does Clover support the new language features in JDK1.5?

Clover fully supports all JDK1.5 language features.

## Does Clover work with JUnit4 and TestNG?

**Q: Does Clover work with JUnit4 and TestNG?** Clover is fully compatible with JUnit4 and TestNG.

## How are the Clover coverage percentages calculated?

### Q: How are the Clover coverage percentages calculated?

The "total" coverage percentage of a class (or file, package, project) is provided as a quick guide to how well the class is covered — and to allow ranking of classes.

The Total Percentage Coverage (TPC) is calculated using the formula:

$$\text{TPC} = (\text{CT} + \text{CF} + \text{SC} + \text{MC}) / (2 * \text{C} + \text{S} + \text{M}) \{\text{excerpt}\}$$

where

```
CT - conditionals that evaluated to "true" at least once
CF - conditionals that evaluated to "false" at least once
SC - statements covered
MC - methods entered

C - total number of conditionals
S - total number of statements
M - total number of methods
```

## How do I compare the code coverage between two releases of my code?

A third party developer has created a Perl script that carries out a comparison of the coverage in two different releases of a codebase.

See the [Atlassian Forum Page](#) where it was posted for more information.

**i** This functionality is not part of Clover and as such is not supported.

## How do I get started with Clover?

See the [Clover-for-Ant QuickStart Guide](#)



## How do I use Clover with NetBeans?

Clover can be used with NetBeans 6.1 by integrating Clover for Ant into your NetBeans project build, which is Ant-based. This integration will allow seamless instrumentation, test execution and hard-copy coverage report generation from within NetBeans.

To start, download Clover for Ant at <http://www.atlassian.com/software/clover/CloverDownloadCenter.jspa>. Once you've downloaded Clover for Ant, expand it to a separate folder (referred to as `CLOVER_HOME`). You'll also need a valid Clover license file, which you can obtain at <http://www.atlassian.com/software/clover/>.

### 1. Add Clover to the NetBeans Ant

1.1 Go to Preferences->Miscellaneous->Ant and use Add JAR/ZIP to add `CLOVER_HOME/lib/clover.jar` to the classpath, you can also add `clover.license` (or you can specify this in project's build.xml)

### 2. Create a new Clover Library

2.1 Open Tools/Libraries

2.2 Click "New Library..." and name it "Clover"

2.3 Add `CLOVER_HOME/lib/clover.jar` to the new library.

### 3. Use Add JAR/Folder to add CLOVER\_HOME/lib/clover.jar to the project classpaths

3.1 Open Project/Properties...

3.2 In Libraries add the Clover library to the Compile, Run, Compile tests, Run tests classpaths

### 4. Add Clover targets to the build

4.1 Add the following to the project build.xml (go to Files view and edit this file)

```
<target name="-pre-init" depends="with.clover"/>
<target name="-post-clean" depends="clover.clean"/>

<property name="clover.enable" value="on"/>
<property name="clover.reportdir" value="clover_html"/>
<!-- You can also specify license here
<property name="clover.license.path" value="path/to/clover.license"/>
-->

<taskdef resource="cloverlib.xml"/>

<target name="with.clover" if="clover.enable">
  <clover-setup/>
</target>

<target name="clover.report" depends="-pre-init">
  <clover-report>
    <current outfile="${clover.reportdir}">
      <format type="html"/>
    </current>
  </clover-report>
</target>

<target name="clover.clean">
  <clover-clean/>
</target>
```

## 5. Using Clover from within NetBeans

- 5.1 Perform a complete clean and rebuild of the project by selecting Build->Clean and Build Main Project...
- 5.2 Select the project `build.xml` and run the `test` target using the Ant Targets window
- 5.3 Run the `clover.report` target to generate a Clover HTML report
- 5.4 The `clover.enable` can be used to disable Clover integration
- 5.5 The `clover.reportdir` can be used to control where the HTML report is generated

## 6. Extending the Clover integration

Because NetBeans uses a standard Ant-based build, you can use all of Clover's Ant tasks from your project build file. This allows you to control includes and excludes, set up source-level filters, change report formats and more. For an overview of the Clover Ant tasks, see <http://confluence.atlassian.com/display/CLOVER/6.+Ant+Tasks+Reference>

## What are the limitations of Code Coverage?

**Q: What are the limitations of Code Coverage?** Code Coverage is not a "silver bullet" of software quality, and 100% coverage is no guarantee of a bug-free application. You can infer a certain level of quality in your tests based on their coverage, but you still need to be writing meaningful tests.

As with any metric, developers and project management should be careful not to over-emphasize coverage, because this can drive developers to write unit tests that just increase coverage, at the cost of actually testing the application meaningfully.

## What does the name "Clover" mean?

**Q: What does the name "Clover" mean?** Clover is actually a shortened version of the tool's original name, "Cover Lover", from the nick name that the tool's author gained while writing Clover ("Mr Cover Lover").

## What is Code Coverage Analysis?

**Q: What is Code Coverage Analysis?**

Code Coverage Analysis is the process of discovering code within a program that is not being exercised by test cases. This information can then be used to improve the test suite, either by adding tests or modifying existing tests to increase coverage.

Code Coverage Analysis shines a light on the quality of your unit testing. It enables developers to quickly and easily improve the quality of their unit tests which ultimately leads to improved quality of the software under development.

For more information, see [About Code Coverage](#).

## What is the coverage.db file and why am I seeing files like coverage.dbxxxxxxxx\_xxxx\_xxxx?

**Q: What is the coverage.db file and why am I seeing files like coverage.dbxxxxxxxx\_xxxx\_xxxx?** The `coverage.db` file is the instrumentation database telling Clover the structure of your project and files during the last instrumentation event. `coverage.dbxxxxxxxx_xxxx_xxxx` hold the code coverage from your unit test or application run.

You will generally only have one instrumentation database per directory and you should expect to have many (sometimes 100s or even 1000s) of coverage recording files per directory.

## What third-party libraries does Clover utilise?

**Q: What third-party libraries does Clover utilise?**

Clover makes use of the following excellent third-party libraries:

<a href="#">Apache Ant</a>	The Ant build system.
<a href="#">ANTLR</a>	A public domain parser generator.
<a href="#">Apache Commons</a>	A set of reusable Java components.

Apache Velocity	Template engine used for HTML report generation.
Cajo	A lightweight library for multi-machine communication.
FastUtil	A library for high-performance operations on primitive types.
Groovy	An agile and dynamic language for the Java Virtual Machine.
GSON	A library converting Java objects into their JSON representation.
Guava	Google's core libraries for collections, caching, primitives support, string processing, I/O etc.
iText (2.0.1)	A library for generating PDF documents.
JCommon / JFreeChart	An open source library for generating charts.
JDOM	A library for accessing, manipulating, and outputting XML data from Java code.
overLIB	A JavaScript library for pop-ups and tool tips.
TheJIT	An open source toolkit for creating interactive data visualisations.
Utils.js	A JavaScript library.

**i** To prevent library version mismatches, all of these libraries have been obfuscated and/or repackaged and included in the Clover JAR. We do this to prevent pain for users who may use different versions of these libraries in their projects.

## Where did Clover originally come from?

### Q: Where did Clover originally come from?

Clover was originally developed at Cenqua (now part of Atlassian) as an internal tool to support development of large J2EE applications. Existing tools were found to be too cumbersome to integrate with complex build systems and often required specialised development and/or runtime environments that were not compatible with target J2EE Containers. Another feature that we found lacking in other tools was simple, source-level coverage reporting — the kind that is most useful to developers.

## Why does Clover instrument classes I have excluded using the 'exclude' element of the 'clover-setup' task?

**Q: Why does Clover instrument classes I have excluded using the <exclude> element of the <clover-setup> task?** There are two possible causes:

#### 1. Cascading build files:

Clover uses Ant patterns to manage the includes and excludes specified in the <clover-setup> task. By default Ant does not pass these patterns to the sub-builds. If you are using a master-build/sub-build arrangement, with compilation occurring in the sub-builds and <clover-setup> done in the master-build, you will need to explicitly pass these patterns as references:

```
<ant ...>
<reference refid="clover.files"/>
<reference refid="clover.useclass.files"/>
</ant>
```

#### 2. Excluded files are still registered in the Clover database:

Clover's database is built incrementally, and this can mean that files that are now excluded but were previously included are still reported on. The simple workaround is to delete the Clover database whenever you change the Clover includes or excludes. This is fixed in Clover 1.2.

## Why does Clover use source code instrumentation?

**Q: Why does Clover use source code instrumentation?** Source code instrumentation is the most powerful, flexible and accurate way to provide code coverage analysis. The following table compares different methods of obtaining code coverage and their relative benefits:

Possible feature	JVM/PI	Bytecode instrumentation	Source code instrumentation
Gathers method coverage	yes	yes	yes
Gathers statement coverage	line only	indirectly	yes
Gathers branch coverage	indirectly	indirectly	yes
Can work without source	yes	yes	no
Requires separate build	no	no	yes
Requires specialised runtime	yes	yes	no
Gathers source metrics	no	no	yes
View coverage data inline with source	not accurate	not accurate	yes
Source level directives to control coverage gathering	no	no	yes
Control which entities are reported on	limited	limited	yes
Compilation time	no impact	variable	variable
Runtime performance	high impact	variable	variable
Container friendly	no	no	yes

## Will Clover integrate with my IDE?

**Q: Will Clover integrate with my IDE?**

Clover 2 provides an integrated plugin for Eclipse, with [more plugins](#) soon to follow. Clover should also work happily with any integrated development environment (IDE) that provides integration with the Ant build tool.

## Eclipse Plugin FAQ

**Clover Eclipse Plugin FAQ**

- **I only need instrumented classes for unit testing and I don't want to risk publishing them to my production environment. How can I do this with Clover?** — Clover supports writing both instrumented and uninstrumented class files to different directories during a build.
- **Is Clover supported on IBM's RAD?** — Yes, IBM RAD is supported. See Supported Platforms page for more details.
- **I store my plugins and features in an Eclipse extension area. Does Clover support this?** — The "Clover 4" and "Clover 4 Ant Support" features can be placed in any extension location.
- **Why can I only see coverage data for the last test case I executed?** — Clover can be set to only display the coverage information gathered since your last compile — full build or auto build. The default behaviour is to include all coverage data found.

## I only need instrumented classes for unit testing and I don't want to risk

## publishing them to my production environment. How can I do this with Clover?

**Q: I only need instrumented classes for unit testing and I don't want to risk publishing them to my production environment. How can I do this with Clover?** Clover supports writing both instrumented and uninstrumented class files to different directories during a build.

To enable the feature, right click on your project and select properties, select Clover, select Compilation tab, select "User specified folder" and then select a project directory where you wish instrumented classes.

All your other Eclipse plugins will then pick up uninstrumented class files from your normal output folder(s) but instrumented classes will also be available for unit testing. The trick, though, is to ensure your JUnit or TestNG launch configuration can see the instrumented classes before your uninstrumented ones. To do this, you will need to modify the classpath of your JUnit launch configuration so that under User Entries the folder containing instrumented classes is *\*before\** the regular output folder for your project.

## Is Clover supported on IBM's RAD?

**Q: Is Clover supported on IBM's RAD?** Yes, IBM RAD is supported. See [Supported Platforms](#) page for more details.

## I store my plugins and features in an Eclipse extension area. Does Clover support this?

**Q: I store my plugins and features in an Eclipse extension area. Does Clover 2 support this?**

The "Clover 4" and "Clover 4 Ant Support" features can be placed in any extension location.

## Why can I only see coverage data for the last test case I executed?

**Q: Why can I only see coverage data for the last test case I executed?**

Clover can be set to only display the coverage information gathered since your last compile — full build or auto build. The default behaviour is to include all coverage data found. You can change how far back in time Clover will look for coverage data by setting the Span parameter in the Clover page in the Workspace preferences (Window | Preferences).

## IDEA Plugin FAQ

**Clover IDEA Plugin FAQ**

- [I've run my tests, but coverage information does not show in IDEA](#)
- [What does enabling Instrument Test Source Folders do?](#)
- [Where does IDEA write its log file?](#)

## I've run my tests, but coverage information does not show in IDEA

**Q: I've run my tests, but coverage information does not show in IDEA**

**A:** If you do not have "Auto Coverage Refresh" enabled, you will need to press the Refresh Button in the Clover Tool Window.

## What does enabling Instrument Test Source Folders do?

**Q: What does enabling Instrument Test Source Folders do?**

**A:** To view per-test coverage, it is required that Clover instrument all your test sources.

## Where does IDEA write its log file?

**Q: Where does IDEA write its log file?**

**A:** On Mac OS X IDEA will write its log file to

```
~/Library/Caches/IntelliJIDEAnn/log/idea.log
```

(where nn=version number) by default. This value is configured however in:

```
/Applications/IntelliJ IDEA X.Y.Z.app/Contents/Info.plist
```

On Windows, IDEA will write its log file to:

```
~\.IntelliJIdea\system\log\idea.log
```

(where ~ stands for user's home directory, e.g. c:\Users\Alice). This value can be configured in <IDEA\_installation\_dir>\bin\idea.properties

(it's a file common for both 32-bit and 64-bit executables).

## Maven 2 and 3 Plugin FAQ

**Clover Maven 2 and 3 Plugin FAQ**

- [Deploying Instrumented Jars](#)
- [How to keep Clover reports between builds?](#)
- [How to remove -clover suffix from artifact name?](#)
- [Is there an alternative to using the Maven Central repository?](#)
- [Preparing multi-module projects for remote deployment with Clover-for-Maven 2](#)
- [Troubleshooting License problems](#)
- [Troubleshooting problems with displaying characters](#)

### Deploying Instrumented Jars

When the `deploy` target is run, the Clover lifecycle doesn't deploy its artifacts. There is a [JIRA issue CLMVN-9](#) open for this limitation.

As a work around, you can use the `build-helper-maven-plugin` as follows:

- The general idea is to attach the instrumented jar (the primary artifact of the clover-plugin forked lifecycle) as a secondary artifact to the original lifecycle by means of the `build-helper-maven-plugin`. A normal 'mvn deploy' (which targets the original lifecycle) will then lead to the desired deployment of the instrumented jar.
- The complicated thing in the attachment of the forked lifecycle's primary artifact (the instrumented jar, that is) to the original lifecycle is, that the forked lifecycle will inherit the whole original lifecycle's configuration, including the introduced attachment. Thus, the forked lifecycle will have the same artifact (its primary artifact) both as primary and as a secondary artifact. Maven will enforce distinct names for the two, leading to necessary classifier substitution in the `build-helper` configuration:

```

<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>build-helper-maven-plugin</artifactId>
  <executions>
    <execution>
      <id>attach-instrumented-jar</id>
      <phase>verify</phase>
      <goals>
        <goal>attach-artifact</goal>
      </goals>
      <configuration>
        <artifacts>
          <artifact>
            <file>

${basedir}/target/clover/${project.artifactId}-${project.version}-clover.jar
            </file>
            <type>jar</type>
            <classifier>clovered</classifier>
          </artifact>
        </artifacts>
      </configuration>
    </execution>
  </executions>
</plugin>

```

A subsequent 'mvn deploy' will lead to a deployment of the instrumented jar, the 'Clovered' version as a secondary artifact along with the non-instrumented (original lifecycle's) primary artifact.

## How to keep Clover reports between builds?

If you want to keep Clover reports between builds, outside of source directory you can use `<outputDirectory/>` element in Clover configuration.

In your configuration put something like this:

```

<build>
  <plugins>
    <plugin>
      <artifactId>maven-clover2-plugin</artifactId>
      <groupId>com.atlassian.maven.plugins</groupId>
      <configuration>
        <!-- Other configuration options -->

        <!-- Set output directory outside maven build -->
        <outputDirectory>c:\dev\cloverReport\${pom.artifactId}</outputDirectory>
      </configuration>
      <!-- Other elements -->
    </plugin>
  </plugins>
</build>

```

Use `${pom.artifactId}` if you have multi module directory - reports for each module will be placed in a separate directory.

## How to remove -clover suffix from artifact name?

**Q: How to remove -clover suffix from artifact name?**

If I use 'clover2:instrument' it creates two artifacts: my-artifact.jar (normal code) and my-artifact-clover.jar (instrumented). How to get rid of the suffix?

**A:** The clover2:instrument' goals performs instrumentation in a parallel lifecycle. Thanks to this you can be sure that "normal" classes will not be mixed with the instrumented versions in final JAR file.

You can use a 'clover2:setup' goal, which does the same, but does not run a parallel build and does not add "-clover" suffix to generated artifacts.

## Is there an alternative to using the Maven Central repository?

### Configuring Clover for Maven to use the Atlassian repository

The Atlassian repository is updated immediately when a new version of Clover is released.

1. Set up your `.m2/settings.xml` by adding:

#### `.m2/settings.xml`

```
...
<pluginGroups>
  <pluginGroup>com.atlassian.maven.plugins</pluginGroup>
</pluginGroups>
...
<profiles>
  <profile>
    <id>myprofile</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    ...
    <pluginRepositories>
      <pluginRepository>
        <releases><enabled>true</enabled></releases>
        <id>atlassian-m2-repository</id>
        <name>Atlassian Maven 2.x Repository</name>
        <url>http://repository.atlassian.com/maven2</url>
      </pluginRepository>
    </pluginRepositories>
    ...
  </profile>
</profiles>
```

to tell Maven where to look for the plugin, and



**.m2/settings.xml**

```

<profiles>
  ...
  <profile>
    <id>myprofile</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    ...
    <properties>
      <maven.clover.licenseLocation>...path to your Clover license
file...</maven.clover.licenseLocation>
    </properties>
    ...
  </profile>
  ...
</profiles>

```

to set a license location property which you can refer to from all your poms.

## Preparing multi-module projects for remote deployment with Clover-for-Maven 2

This page contains instructions on preparing a multi-module project for remote deployment with Clover-for-Maven 2.

### To prepare a multi-module project with Clover-for-Maven 2,

1. Instrument the source with Clover and generate EAR/WAR file, then include the `clover.jar` file in the `lib` directory.
2. Deploy to application server and ensure Clover can find the registry at runtime.
3. Copy the database, the `clover.db` file to a directory in the test machine and specify the location in the Clover initstring. For details, see [Documentation on setting Clover initstring](#).
4. If the project contains sub-modules, copy each database with its directory. For example:

```
Sub-Module1\target\clover\clover.db
```

Copy the text above into a location (as specified in the clover initstring) in the test machine. Alternatively, create databases with different names.

5. After the tests, copy all the databases to the build machine, run an aggregate goal (merge databases) and generate the reports from there.

## Troubleshooting License problems

This page lists the various ways in which you can specify your Clover for Maven 2 license. You can try one of the following processes if your Clover license is not being recognised correctly.

**⚠ You need a valid Clover license file to run Clover.** You can obtain a free 30 day evaluation license or purchase a commercial license at <http://my.atlassian.com>.

- [Specifying your license location in the pom.xml file](#)
- [Embedding your license in the pom.xml file](#)
- [Specifying your license location in your ~/.m2/settings.xml file](#)
- [Specifying your license on the command line](#)

### Specifying your license location in the pom.xml file

You can set this property to point to your clover license in the pom.xml file. In the example below,

replace '/path/to/clover.license' with the path to your Clover license file:

```
...
  <plugin>
    <groupId>com.atlassian.maven.plugins</groupId>
    <artifactId>maven-clover2-plugin</artifactId>
    <configuration>
      <licenseLocation>/path/to/clover.license</licenseLocation>
    </configuration>
  </plugin>
```

### Embedding your license in the pom.xml file

You can configure the Maven 2 plugin to include the license data in your pom.xml file. Simply add a `<license>` element inside `<configuration>` and make its contents contain the four line Atlassian license string.

**⚠** Some Atlassian licenses may contain XML characters, so you will need to ensure you wrap your license in CDATA tags:

```
<configuration>
  <license><![CDATA[ YOURLICENSESTRINGHERE ]]></license>
</configuration>
```

This will make the Maven build less reliant on local file system layout, or the availability of remote servers.

### Specifying your license location in your ~/.m2/settings.xml file

You can set this property to point to your clover license in your settings.xml file. In the example below, replace '/path/to/clover.license' with the path to your Clover license file:

```
<properties>

<maven.clover.licenseLocation>/path/to/clover.license</maven.clover.licenseLocation
>

</properties>
```

### Specifying your license on the command line

To specify your license at the command line, specify a property as follows (replacing '/path/to/clover.license' with the path to your Clover license file):

```
-Dmaven.clover.licenseLocation=/path/to/clover.license
```

## Troubleshooting problems with displaying characters

As of version 2.3.0, the plugin now supports a `-Dmaven.clover.encoding` system property, and an `<encoding>` element in the pom.xml.

This allows you to specify an alternate encoding for your Java source files, such as [UTF-8](#), or ['Big5'](#).

## Support Policies

Welcome to the support policies index page. Here, you'll find information about how Atlassian Support can help

you and how to get in touch with our helpful support engineers. Please choose the relevant page below to find out more.

- [Bug Fixing Policy](#)
- [How to Report a Security Issue](#)
- [New Features Policy](#)
- [Security Advisory Publishing Policy](#)
- [Security Update Policy](#)
- [Severity Levels for Security Issues](#)
- [Update Policy](#)

To request support from Atlassian, please raise a support issue in our online support system. To do this, visit [support.atlassian.com](https://support.atlassian.com), log in (creating an account if need be) and create an issue under Clover. Our friendly support engineers will get right back to you with an answer.

## Bug Fixing Policy

### Summary

- Atlassian Support will help with workarounds and bug reporting.
- Critical bugs will generally be fixed in the next maintenance release.
- Non critical bugs will be scheduled according to a variety of considerations.



### Raising a Bug Report

Atlassian Support is eager and happy to help verify bugs — we take pride in it! Please open a support request in our [support system](#) providing as much information as possible about how to replicate the problem you are experiencing. We will replicate the bug to verify, then lodge the report for you. We'll also try to construct workarounds if they're possible.

Customers and plugin developers are also welcome to open bug reports on our issue tracking systems directly. Use the appropriate project on <http://jira.atlassian.com> to report bugs for Atlassian products.

When raising a new bug, you should rate the priority of a bug according to our [JIRA usage guidelines](#). Customers **should watch** a filed bug in order to receive e-mail notification when a "Fix Version" is scheduled for release.

### How Atlassian Approaches Bug Fixing

Maintenance (bug fix) releases come out more frequently than major releases and attempt to target the most critical bugs affecting our customers. The notation for a maintenance release is the final number in the version (ie the 1 in 3.0.1).

If a bug is critical (production application down or major malfunction causing business revenue loss or high numbers of staff unable to perform their normal functions) then it will be fixed in the next maintenance release provided that:

- The fix is technically feasible (i.e. it doesn't require a major architectural change).
- It does not impact the quality or integrity of a product.

For non-critical bugs, the developer assigned to fixing bugs prioritises the non-critical bug according to these factors:

- How many of our supported configurations are affected by the problem.
- Whether there is an effective workaround or patch.
- How difficult the issue is to fix.
- Whether many bugs in one area can be fixed at one time.

The developers responsible for bug fixing also monitor comments on existing bugs and new bugs submitted in JIRA, so you can provide feedback in this way. We give high priority consideration to [security issues](#).

When considering the priority of a non-critical bug we try to determine a 'value' score for a bug which takes into account the severity of the bug from the customer's perspective, how prevalent the bug is and whether roadmap

features may render the bug obsolete. We combine this with a complexity score (i.e. how difficult the bug is). These two dimensions are used when developers self serve from the bug pile.

#### Further reading

See [Atlassian Support Offerings](#) for more support-related information.

## How to Report a Security Issue

### Finding and Reporting a Security Issue

If you find a security issue in the product, open an issue on <https://jira.atlassian.com> in the relevant project.

- Set the **security level** of the bug to **'Reporters and Developers'**.
- Set the priority of the bug to **'Blocker'**.
- Provide as much information on reproducing the bug as possible.

All communication about the security issue should be performed through JIRA, so that Atlassian can keep track of the issue and get a patch out as soon as possible.

If you cannot find the right project to file your issue in, email the details to [security@atlassian.com](mailto:security@atlassian.com).

**i** When reporting a security vulnerability, please keep in mind the following:

We need a technical description that allows us to assess exploitability and impact of the issue.

- Provide steps to reproduce the issue, including any URLs or code involved.
- If you are reporting a cross-site scripting (XSS), your exploit should at least pop up an alert in the browser. It is much better if the XSS exploit shows user's authentication cookie.
- For a cross-site request forgery (CSRF), use a proper CSRF case when a third party causes the logged in victim to perform an action.
- For a SQL injection, we want to see the exploit extracting database data, not just producing an error message.
- HTTP request / response captures or simply packet captures are also very useful to us.

Please refrain from sending us links to non-Atlassian web sites, or reports in PDF / DOC / EXE files. Image files are ok. Make sure the bug is exploitable by someone other than the user himself (e.g. "self-XSS").

Without this information it is not possible to assess your report and it is unlikely to be addressed.

We are not looking for the reports listing generic "best practice" issues such as:

- Specific cookies being not marked as Secure or HTTPOnly
- Presence or absence of HTTP headers (X-Frame-Options, HSTS, CSP, nosniff and so on)
- Clickjacking
- Stack traces
- Mixed HTTP and HTTPS content
- Auto-complete enabled or disabled
- SSL-related issues

We are also not looking for reports on the following bug classes:

- Username enumeration using login or password reset features. While username enumeration can be a vulnerability in web applications, most of Atlassian products and web sites include a number of social features. As a result, usernames can be discovered by design in a number of ways.

#### Further reading

See [Atlassian Support Offerings](#) for more support-related information.

## New Features Policy

### Summary

- We encourage and display customer comments and votes openly in our issue tracking system, <http://jira.atlassian.com>.
- We do not publish roadmaps.
- Product Managers review our most popular voted issues on a regular basis.
- We schedule features based on a variety of factors.
- Our [Atlassian Bug Fixing Policy](#) is distinct from this process.
- Atlassian provides consistent updates on the top 20 issues.

### How to Track what Features are Being Implemented

When a new feature or improvement is scheduled, the 'fix-for' version will be indicated in the JIRA issue. This happens for the upcoming release only. We maintain roadmaps for more distant releases internally, but because these roadmaps are often pre-empted by changing customer demands, we do not publish them.

### How Atlassian Chooses What to Implement

In every [major release](#) we *aim* to implement highly requested features, but it is not the only determining factor. Other factors include:

- **Customer contact:** We get the chance to meet customers and hear their successes and challenges at Atlassian Summit, Atlassian Unite, developer conferences, and road shows.
- **Customer interviews:** All product managers at Atlassian do customer interviews. Our interviews are not simply to capture a list of features, but to understand our customers' goals and plans.
- **Community forums:** There are large volumes of posts on [answers](#), of votes and comments on [jira.atlassian.com](http://jira.atlassian.com), and of conversations on community forums like groups on LinkedIn.
- **Customer Support:** Our support team provides clear insights into the issues that are challenging for customers, and which are generating the most calls to support
- **Atlassian Experts:** Our [Experts](#) provide insights into real-world customer deployments, especially for customers at scale.
- **Evaluator Feedback:** When someone new tries our products, we want to know what they liked and disliked and often reach out to them for more detail.
- **In product feedback:** The [JIRA Issue Collectors](#) that we embed our products for evaluators and our Early Access Program give us a constant pulse on how users are experiencing our product.
- **Usage data:** Are customers using the features we have developed?
- **Product strategy:** Our long-term strategic vision for the product.
- Please read our [post on Atlassian Answers](#) for a more detailed explanation.

### How to Contribute to Feature Development

#### Influencing Atlassian's release cycle

We encourage our customers to vote on issues that have been raised in our public JIRA instance, <http://jira.atlassian.com>. Please find out if your request [already exists](#) - if it does, vote for it. If you do not find it you may wish to create a new one.

#### Extending Atlassian Products

Atlassian products have powerful and flexible extension APIs. If you would like to see a particular feature implemented, it may be possible to develop the feature as a plugin. Documentation regarding the [plugin APIs](#) is available. Advice on extending either product may be available on the user mailing-lists, or at [Atlassian Answers](#).

If you require significant customisations, you may wish to get in touch with our [partners](#). They specialise in extending Atlassian products and can do this work for you. If you are interested, please [contact us](#).

#### Further reading

See [Atlassian Support Offerings](#) for more support-related information.

## Security Advisory Publishing Policy

### Publication of Security Advisories

When a [critical severity](#) security vulnerability in an Atlassian product is discovered and resolved, Atlassian will inform customers through the following mechanisms:

- We will post a security advisory in the latest documentation of the affected product at the same time as releasing a fix for the vulnerability.
- We will send a copy of all posted security advisories to the **'Technical Alerts' mailing list** for the product concerned.

*Note:* To manage your email subscriptions and ensure you are on this list, please go to [my.atlassian.com](http://my.atlassian.com) and click 'Communications Centre' near the top right of the page.

- If the person who reported the vulnerability wants to publish an advisory through some other agency, such as [CERT](http://CERT), we will assist in the production of that advisory and link to it from our own.

If you want to track non-critical severity security vulnerabilities, you need to monitor the issue trackers for the relevant products on <http://jira.atlassian.com>. For example, <https://jira.atlassian.com/browse/JRA> for JIRA and <https://jira.atlassian.com/browse/CONF> for Confluence. Security issues in trackers will be marked with a "security" label. All security issues will be listed in the release notes of the release where they have been fixed, similar to other bugs.

One of the ways to monitor updates to security issues is subscribing to the results of a [sample search](#) via email or RSS.

#### Further reading

See [Atlassian Support Offerings](#) for more support-related information.

## Security Update Policy

As Clover is a plugin, patches do not apply. Instead, a new version of the plugin is released.

You can follow the progress of Clover development on our [issue tracking system](#).

You can follow Clover releases on the [Release Notes](#).

For information about the timeliness and prioritisation of Clover releases, see the [Atlassian Bug Fixing Policy](#).

#### Further reading

See [Atlassian Support Offerings](#) for more support-related information.

## Severity Levels for Security Issues

### Severity Levels

Atlassian security advisories include a severity level. This severity level is based on our self-calculated CVSS score for each specific vulnerability. CVSS is an industry standard vulnerability metric. You can learn more about CVSS at [FIRST.org](http://FIRST.org) web site.

CVSS scores are mapped into the following severity ratings:

- Critical
- High
- Medium
- Low

An approximate mapping guideline is as follows:

CVSS score range	Severity in advisory
0 – 2.9	Low
3 – 5.9	Medium
6.0 – 7.9	High
8.0 – 10.0	Critical

Below is a summary of the factors which illustrate types of vulnerabilities usually resulting in a specific severity level. Please keep in mind that this rating does not take into account details of your installation.

**Severity Level: Critical**

Vulnerabilities that score in the critical range usually have most of the following characteristics:

- Exploitation of the vulnerability results in root-level compromise of servers or infrastructure devices.
- The information required in order to exploit the vulnerability, such as example code, is widely available to attackers.
- Exploitation is usually straightforward, in the sense that the attacker does not need any special authentication credentials or knowledge about individual victims, and does not need to persuade a target user, for example via social engineering, into performing any special functions.

For critical vulnerabilities, is advised that you patch or upgrade as soon as possible, unless you have other mitigating measures in place. For example, if your installation is not accessible from the Internet, this may be a mitigating factor.

**Severity Level: High**

Vulnerabilities that score in the high range usually have some of the following characteristics:

- The vulnerability is difficult to exploit.
- Exploitation does not result in elevated privileges.
- Exploitation does not result in a significant data loss.

**Severity Level: Medium**

Vulnerabilities that score in the medium range usually have some of the following characteristics:

- Denial of service vulnerabilities that are difficult to set up.
- Exploits that require an attacker to reside on the same local network as the victim.
- Vulnerabilities that affect only nonstandard configurations or obscure applications.
- Vulnerabilities that require the attacker to manipulate individual victims via social engineering tactics.
- Vulnerabilities where exploitation provides only very limited access.

**Severity Level: Low**

Vulnerabilities in the low range typically have very little impact on an organisation's business. Exploitation of such vulnerabilities usually requires local or physical system access.

**Further reading**

See [Atlassian Support Offerings](#) for more support-related information.

## Update Policy

As Clover is a plugin, patches do not apply. Instead, a new version of the plugin is released.

You can follow the progress of Clover development on our [issue tracking system](#).

You can follow Clover releases on the [Release Notes](#).

For information about the timeliness and prioritisation of Clover releases, see the [Atlassian Bug Fixing Policy](#).

**Further reading**

See [Atlassian Support Offerings](#) for more support-related information.

## Troubleshooting



## Clover Troubleshooting

- [Compiling my instrumented sources fails with a 'code too large' error.](#)
- [For some statements in my code Clover reports "No Coverage information gathered for this expression". What does that mean?](#) — Clover will not measure coverage of a conditional expression if it contains an assignment operator.
- [Hit count for multi-threaded test is incorrect in Clover's report.](#)
- [I'm trying to get a coverage report mailed, but I keep getting "mail Failed to send email". How do I fix this?](#) — The Ant task depends on external libraries that are not included in the Ant distribution. You need to install the following jars in ANT\_HOME/lib, both freely available from Sun:
- [I'm using the maven-clover-plugin version 2.4 with a license downloaded from Atlassian and get the message 'Invalid or missing License'](#)
- [Tools for Troubleshooting Clover-for-Ant](#)
- [Two questions to ask yourself first when troubleshooting Clover](#)
- [When generating some report types on my UNIX server with no XServer, I get an exception "Can't connect to X11 server" or similar.](#) — This is a limitation of the Java implementation on Unix.
- [When using Clover, why do I get a java.lang.NoClassDefFoundError when I run my code?](#) — This probably indicates that you do not have clover.jar in your runtime classpath.
- [When using Clover from Ant, why do I get "Compiler Adapter 'org.apache.tools.ant.taskdefs.CloverCompilerAdapter' can't be found." or similar?](#) — You need to install Clover in Ant's classpath.
- [Why does the 'Test Results' summary page report show that I have unique coverage, when the source page shows no unique coverage?](#) — The source view only shows unique coverage aggregated at the line level, not per statement or branch. The unique coverage indicates that either:
- [Why do I get 0% coverage when I run my tests and then a reporter from the same instance of Ant?](#) — This occurs because Clover hasn't had a chance to flush coverage data out to disk.
- [Why do I get a 'java.lang.OutOfMemoryError - PermGen space' error?](#)
- [Why do I get an java.lang.OutOfMemoryError when compiling with Clover turned on?](#) — A: Instrumenting with Clover increases the amount of memory that the compiler requires in order to compile.

### Compiling my instrumented sources fails with a 'code too large' error.

A single Java method cannot compile to more than 64KB of byte code. As Clover adds statements to record code coverage to every statement in your source file, a method which is close to this limit may exceed it when instrumented. The solutions at present are:

1. As a work-around, split your method into two smaller ones. Or;
2. Exclude the entire file using the `<files>` element of `<clover-setup>`.

### For some statements in my code Clover reports "No Coverage information gathered for this expression". What does that mean?

**Q: For some statements in my code Clover reports "No Coverage information gathered for this expression". What does that mean?**

Clover will not measure coverage of a conditional expression if it contains an assignment operator. In practice we have found this only a minor limitation. To understand why Clover has this limitation, consider the following (very contrived) code fragment:



```

1 public int foo(int i) {
2     int j;
3     if ((j = i) == 1) {
4         return j;
5     }
6     return 0;
7 }
\\
at (2) the variable "j" is declared but not initialised.
at (3) "j" is assigned to inside the expression
at (4) "j" is referenced.

```

During compilation, most compilers can inspect the logic of the conditional at (3) to determine that "j" will be initialised by the time it is referenced (4), since evaluating the expression (3) will always result in "j" being given a value. So the code will compile. But Clover has to rewrite the conditional at (3) so that it can measure coverage, and the rewritten version makes it harder for compilers to infer the state of "j" when it is referenced at (4). This means that the instrumented version may not compile. For this reason, Clover scans conditionals for assignment. If one is detected, the conditional is not instrumented.

### Hit count for multi-threaded test is incorrect in Clover's report.

This is limitation of Clover's per-test coverage whereby it does not support parallel test execution.

There is an outstanding feature request for this issue - [CLOV-418](#).

Please refer to [Implementation of New Features and Improvements](#) for further details on how Atlassian choose features for inclusion into our products.

### I'm trying to get a coverage report mailed, but I keep getting "mail Failed to send email". How do I fix this?

**Q: I'm trying to get a coverage report mailed to the team as shown in your example, but I keep getting "[mail] Failed to send email". How do I fix this?**

The Ant <mail> task depends on external libraries that are not included in the Ant distribution. You need to install the following jars in ANT\_HOME/lib, both freely available from Sun:

1. mail.jar — from the JavaMail API (<http://java.sun.com/products/javamail/>)
2. activation.jar — from the JavaBeans Activation Framework (<http://java.sun.com/products/javabeans/jaf/index.jsp>)

You should also check the details of your local SMTP server with your system administrator. It may help to specify these details directly to the <mail> task:

```

<mail mailhost="smtp.myisp.com" mailport="25" from="build@example.com"
tolist="team@example.com" subject="coverage criteria not met"
message="$
{coverageFailed}
" files="coverage_summary.pdf"/>

```

### I'm using the maven-clover-plugin version 2.4 with a license downloaded from Atlassian and get the message 'Invalid or missing License'

Version 2.4 of the maven-clover-plugin uses Clover 1.3.13, which doesn't recognise new Atlassian-issued Clover licences.

You need to use version 2.4.1 of the plugin, which is hosted at <http://repository.atlassian.com/maven2>.

You'll need to update your pom.xml with the new version:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-clover-plugin</artifactId>
  <version>2.4.1</version>
  <configuration>
    <licenseLocation>...your licence file path...</licenseLocation>
    ...
  </configuration>
</plugin>
```

and add the Atlassian public repository as a plugin repository in your `pom.xml` or `~/.m2/settings.xml` file:

```
<pluginRepositories>
  <pluginRepository>
    <id>atlassian-m2-repository</id>
    <url>http://repository.atlassian.com/maven2</url>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
  </pluginRepository>
</pluginRepositories>
```

## Tools for Troubleshooting Clover-for-Ant

### Troubleshooting

- To enable logging of the Clover installation, set the environment variable `ANT_OPTS` to `'-Dclover.debug=true'`
- Run `ant` with the `-debug` and `-verbose` options
- Certain environments may require the `clover.jar` to be placed directly on Ant's Classpath. Details are outlined [here](#).
- To enable logging of Clover at runtime set the environment variable `-Dclover.logging.level=debug` on the JVM that is running your Clover instrumented code. e.g. the JUnit JVM, the Tomcat JVM.

## Two questions to ask yourself first when troubleshooting Clover

### Two questions to ask yourself first when troubleshooting Clover:

1. **Does my code compile and run as expected without Clover?**  
You need to ensure that your project compiles and runs as expected before attempting to use Clover.
2. **Am I using the latest version of Clover?**  
The latest version of Clover incorporates many bugfixes and improvements.

If the answers in this section don't fix the problem you are encountering, please don't hesitate to [contact us](#).

## When generating some report types on my UNIX server with no XServer, I get an exception "Can't connect to X11 server" or similar.

### Q: When generating some report types on my UNIX server with no XServer, I get an exception "Can't connect to X11 server" or similar.

This is a limitation of the Java implementation on Unix. Prior to JDK 1.4, the Java graphics toolkit (AWT) requires the presence of an XServer, even in the case where no "on-screen" graphics are rendered.

With JDK 1.4, you can set the System property `java.awt.headless=true` to avoid this problem. When running Ant, this is most easily achieved by using the `ANT_OPTS` environment variable:

```
export ANT_OPTS=-Djava.awt.headless=true
```

When running your code outside Ant, you may also need to set this system property.

With earlier JDKs, you need to use a virtual X Server. See <http://java.sun.com/products/java-media/2D/forDevelopers/java2dfaq.html#xvfb>.

## When using Clover, why do I get a `java.lang.NoClassDefFoundError` when I run my code?

**Q: When using Clover, why do I get a `java.lang.NoClassDefFoundError` when I run my code?** This probably indicates that you do not have `clover.jar` in your runtime classpath. See '[Classpath Issues](#)' in [Working with Distributed Applications](#).

## When using Clover from Ant, why do I get "Compiler Adapter 'org.apache.tools.ant.taskdefs.CloverCompilerAdapter' can't be found." or similar?

**Q: When using Clover from Ant, why do I get "Compiler Adapter 'org.apache.tools.ant.taskdefs.CloverCompilerAdapter' can't be found." or similar?** You need to install Clover in Ant's classpath.

Depending on what version of Ant you are using, there are several options to do this. See [Installation Options](#).

## Why does the 'Test Results' summary page report show that I have unique coverage, when the source page shows no unique coverage?

**Q: Why does the 'Test Results' summary page report show that I have unique coverage, when the source page shows no unique coverage?**

The source view only shows unique coverage aggregated at the line level, not per statement or branch. The unique coverage indicates that either:

- the test was the only one to follow a particular path through a branch; or
- the test uniquely covered a statement on a line containing more than one statement.

 Unique coverage is indicated by [colour-coding](#).

## Why do I get 0% coverage when I run my tests and then a reporter from the same instance of Ant?

**Q: Why do I get 0% coverage when I run my tests and then a reporter from the same instance of Ant?**

This occurs because Clover hasn't had a chance to flush coverage data out to disk. By default Clover flushes coverage data only at JVM shutdown or when explicitly directed to (using an [inline directive](#)). The simplest thing to do is to use the `{{fork="true"}}` attribute when running your tests. The tests will then be run in their own JVM, and the coverage data will be flushed when that JVM exits. Alternatively, you can use interval-based flushing by changing the [Flush Policy](#).

## Why do I get a 'java.lang.OutOfMemoryError - PermGen space' error?

This page contains instructions relating to this error:

```
java.lang.OutOfMemoryError: PermGen space
```

If you see this error when running Clover, you may need to increase the PermGen settings on your server JVM.

This error may sometimes come about when implementing Clover on large projects, due to Clover's additional requirements.

The required memory can be increased by setting the `-XX:MaxPermSize` setting on the JVM.

See the [OutOfMemoryError: PermGen](#) KB article for more details.

## Why do I get an java.lang.OutOfMemoryError when compiling with Clover turned on?

### Q: Why do I get an java.lang.OutOfMemoryError when compiling with Clover turned on?

**A:** Instrumenting with Clover increases the amount of memory that the compiler requires in order to compile. To solve this problem, you need to give the compiler more memory. Increasing the memory available to the compiler depends on how you are launching the compiler:

- If you are using the "in-process" compiler (the `<javac>` task with the "fork" attribute set to `false`), you will need to give Ant itself more memory to play with. To do this, use the `ANT_OPTS` environment variable to set the heap size of the JVM used to run Ant:

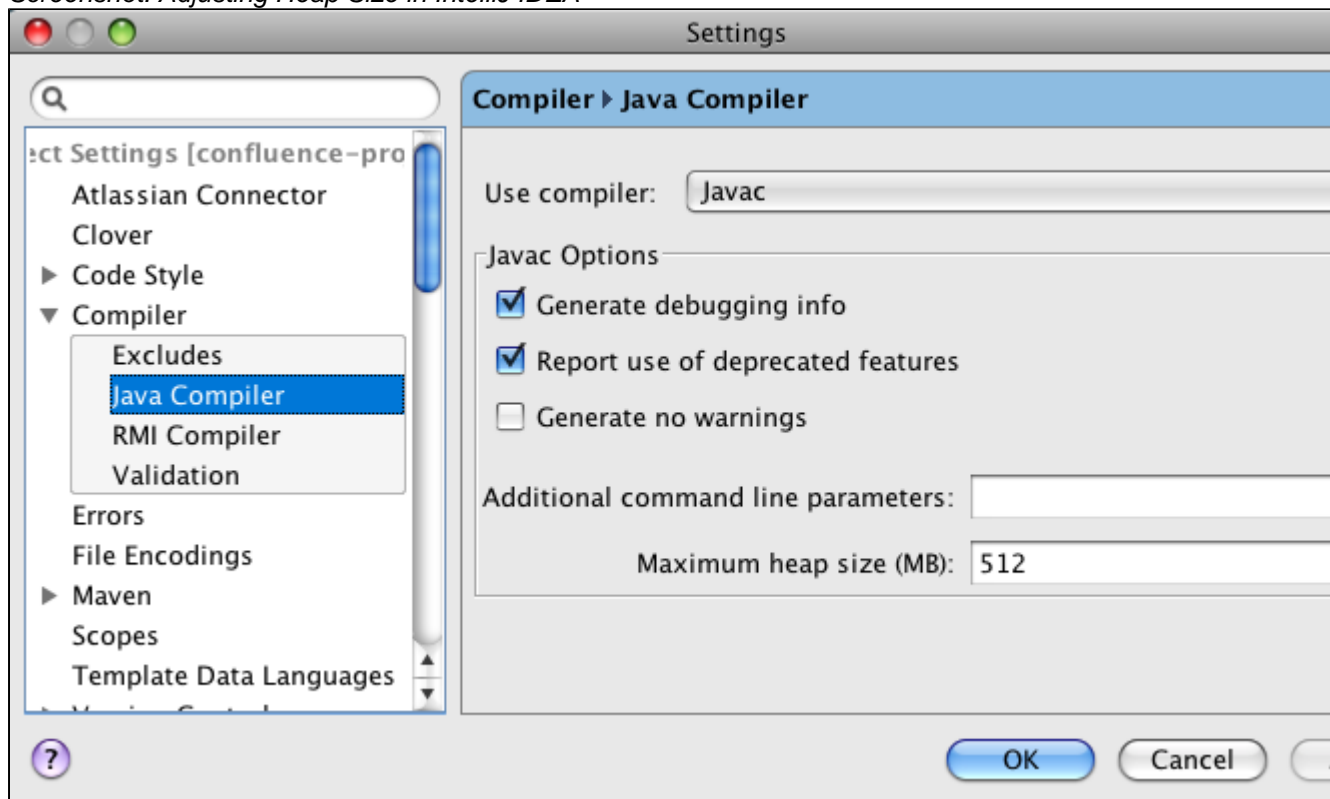
```
export ANT_OPTS=-Xmx256m
```

- If you are using an external compiler (the `<javac>` task with the "fork" attribute set to `true`), you can set the `memoryInitialSize` and `memoryMaximumSize` attributes of the `javac` task:

```
<javac srcdir="${src}"
  destdir="${build}"
  fork="true"
  memoryInitialSize="128m"
  memoryMaximumSize="256m" />
```

- If you are using IntelliJ, increase the Maximum heap size on the Java Compiler.

Screenshot: *Adjusting Heap Size in IntelliJ IDEA*



## Clover Resources

Resources for Evaluators

- [Free Trial](#)
- [Feature Tour](#)

### Resources for Administrators

- [Clover Knowledge Base](#)
- [Clover FAQ](#)
- [Guide to Installing an Atlassian Integrated Suite](#)
- [The big list of Atlassian gadgets](#)

### Forum

- [Atlassian Answers](#) - a quick way to find answers for common problems

### Twitter

- [@cloverallover](#) - an unofficial developer tweet about Clover. You will get notifications about new Clover releases, features, tips & tricks etc.
- [@atlassiandev](#) - the Atlassian Developer Relations Team tweet.
- [@AtlDevTools](#) - a Dev Tools Guru - follow to receive updates on Atlassian Developer Tools - FishEye, Crucible & Bamboo - including news, tips, and answers to your questions.

### Support

- [Atlassian Support](#) - raising a support ticket
- [Support Policies](#)

### Feature Requests

- [Issue Tracker and Feature Requests for Clover](#) - bugs, features, development road map, release notes

### Downloadable Documentation

- [Clover documentation in PDF, HTML or XML formats](#) - generated for every feature release

### Plug-ins


- [Clover Developer Documentation](#) - guides how to develop Clover integrations with other tools
- [Atlassian Marketplace](#) - search for Clover-related plug-ins for Atlassian tool suite

### Mailing Lists

- Visit <http://my.atlassian.com> to sign up for mailing lists relating to Atlassian products, such as technical alerts, product announcements and developer updates.

## Clover Development Hub

If you're doing custom development with Clover, you've come to the right place.

 **Bamboo and Hudson users:** Clover already has working plugins that integrate Clover into these products (see the '**Clover Plugins**' section below). No additional programming is required for Bamboo or Hudson users to take advantage of Clover.

The Clover API is aimed at CI server vendors wishing to add support for Clover to their products, or users wishing to program new solutions for meshing Clover's Test Optimization with your test framework (see the '**Reference Documentation**' section below).

### Clover Plugins

These plugins allow customers using continuous integration servers to easily make use of Clover's advanced code coverage analysis, in a turnkey solution.

### Continuous Integration (CI) Plugins

- **Hudson Clover Plugin**  
*Integrates Hudson with Clover code coverage analysis.*
- **Jenkins Clover Plugin**  
*Integrates Jenkins with Clover code coverage analysis.*
- **Bamboo Clover Plugin**  
*Integrates Bamboo with Clover, provided by Atlassian.*
- **Coverage Plugin for Bamboo**  
*Plugin developed by community which integrates Atlassian Bamboo with code coverage analysis tools like Clover, Emma, Cobertura.*

## Reference Documentation

### Clover Development Documentation

- **Clover API Javadocs**  
The Clover API allows developers to develop new hooks for Clover, to connect it into Continuous Integration servers such as [AnthillPro](#), [TeamCity](#), [Cruise Control](#) and similar products.

The Clover API also provides classes to optimise tests programmatically. This may be necessary if you are using a custom testing framework or your tests are defined in JUnit TestSuites.

- **JSON Reference**  
The JSON format is supported as an output type in Clover specifically to create integration opportunities with other applications. The JSON data from Clover is easy to manipulate programmatically, allowing innovative developers to use it for displaying or processing their coverage data in novel ways.

## Plugin Hosting on Bitbucket or ecosystem.atlassian.net

Atlassian can host your plug-in development project. We'll provide a Mercurial or Git repository, Confluence space and a JIRA project.

## The Atlassian Developer Blog

For up-to-date news and opinions from the Clover, FishEye and other Atlassian development teams:

- <http://blogs.atlassian.com/blog-cat/developer>

## Clover for Grails Developer Guide

### Preconditions

- You have a **2.0.3** version of Grails installed and `$GRAILS_HOME` is set to this.
  - Note that the minimum Grails version required to run the Clover-for-Grails plugin is currently **1.3.0** - it's declared in `CloverGrailsPlugin.groovy` file, but
  - the compilation and deployment of the plugin itself can be done using higher Grails version - it's declared in `application.properties` file.
- Set `JAVA_HOME` to **JDK 1.6** (otherwise you'll end up with `java.lang.NoClassDefFoundError: org.codehaus.gant.GantBuilder` error)

## Getting the Source Code

The Clover-for-Grails plug-in source is stored in Mercurial repository on Bitbucket.org. To get a local copy of the source code, a Mercurial client is required.

The following command will checkout the source code of the [atlassian/grails-clover-plugin](#):

```
hg clone ssh://hg@bitbucket.org/atlassian/grails-clover-plugin
```

## Installing the Plugin

The plugin can then be built, tested and installed via:

```
grails package-plugin --plain-output --verbose
grails test maven-install --plain-output --verbose # it puts zip into local maven
cache ~/.m2
```

## Running Integration Tests

Test it against the enclosed Grails test applications located in testcases directory: daily-groove, petclinic, petclinic203, petclinic210, weceem.

1. Update plugin clover number in `<grails>/testcases/<project>/grails-app/conf/BuildConfig.groovy` like

```
plugins {
    build ":clover:X.X.X"
    ...
}
```

2. Change directory to `<grails>/testcases/<project_name>` and run
  - a. **grails test-app -clover.on -clover.view**  
⚠ you have to set `GRAILS_HOME` and `PATH` variables according to test project settings.

## Submitting a Patch

To submit a patch:

1. Make and test the change in your local work area
2. Create a JIRA issue in project "Clover" (**CLOV**), set "Grails Plugin" component.
3. Ensure any new features/configuration options have been documented in the issue description.
4. Commit changes and:
  - clone a repository on bitbucket.org and push changes to it
  - create a pull request for [atlassian/grails-clover-plugin](#) project

or

- generate a patch by running the following command in your local work area (where XXXX is the id of the JIRA issue created above)

```
hg diff > CLOV-XXXX.patch
```

- Upload the patch to the JIRA issue you created.

## Binaries

Stable releases can be downloaded from [Grails Plugins](#) site.

## Creating Grails plugins using Clover

### Important notice

In case you're developing a Grails plugin and this plugin is referencing the Clover plugin, it's necessary to define that the Clover plugin is not exported. Otherwise, the Clover plugin will be transitively picked up by an application using your own plug-in. As a consequence, it can lead to conflicts in case when the application is also referencing a Clover plugin, but in a different version.

Example how to define a Clover dependency in a Grails plugin:

#### BuildConfig.groovy

```
grails.project.dependency.resolution = {
  plugins {
    compile(":clover:3.2.0") {
      export = false
    }
  }
}
```

## Clover for Hudson Developer Guide

**i** Hudson was migrated to the Eclipse Foundation. It's being hosted on the <http://eclipse.org/hudson/> site (mailing list [HUDSON-DEV@ECLIPSE.ORG](mailto:HUDSON-DEV@ECLIPSE.ORG)). The old site <http://HUDSON-CI.ORG/> is still available and hosts plugins not under by EPL license (including Clover-for-Hudson).

Sources were also split into two copies (disaster, don't ask me why):

- Hudson 2 - <https://github.com/hudson2-plugins>
- Hudson 3 - <https://github.com/hudson3-plugins>

## Common

### Preconditions

- you have proper credentials in `~/.m2/settings.xml`

```
<server>
  <id>sonatype-nexus-snapshots</id>
  <username>xxxxxx</username>
  <password>xxxxxx</password>
</server>
<server>
  <id>sonatype-nexus-staging</id>
  <username>xxxxxx</username>
  <password>xxxxxx</password>
</server>
```

- JAVA\_HOME points to **JDK1.6+**
- M2\_HOME points to **Maven 3.0.4+** (otherwise checksum validation will fail on OSS)



- You have GPG signature configured to sign artefacts

## Hudson 2

### Preconditions

- You are a member of the Hudson Plug-in Development team, including
  - permissions to publish release artefacts on <https://oss.sonatype.org/content/repositories/releases/org/jvnet/hudson/plugins/clover>
  - write permissions in master git repository on <https://github.com/hudson2-plugins/clover-plugin>
  - account on Hudson Wiki in order to edit the <http://wiki.hudson-ci.org/display/HUDSON/Clover+Plugin> page

### Build and Test

1. Clone the [git@github.com/hudson2-plugins/clover-plugin.git](https://github.com/hudson2-plugins/clover-plugin) locally and later push directly.
2. Create a bug on <https://bugs.eclipse.org> (Hudson project, Plugins component) for the issue you are fixing.
3. Make the necessary changes
4. Bump plug-in version number (in pom.xml).  
Bump the dependency version on Clover Core if necessary (in pom.xml).  
Bump *org.jvnet.hudson.plugins:hudson-plugin-parent* version if necessary (in pom.xml).
5. Test by running Hudson with the Clover plug-in installed and setting up a Clovered project for CI. This can be achieved via:
 

```
mvn clean hpi:run
```

  - a. it starts Hudson on localhost:8080 by default, open it in a web browser
  - b. open "Manage Hudson" > "Manage plugins" > "Installed" and check if new "Hudson Clover Plugin" is listed
  - c. configure new project (you can use MoneyBags as a test case) and a build job (e.g. "freestyle project" with "ant task")
  - d. configure "Post-build actions" > "Publish Clover Coverage Report" in the build job
  - e. run "Build now" and check if Clover summary report is available
6. Commit changes and push to GitHub. Include your Bugzilla bug ID from above in the commit line.

### Release and Publish

Detailed instruction is [here](#). If this is your first time publishing a Hudson release, allow at least a day or two for Sonatype to process your JIRA requests, set you up and eventually enable Central Sync.

#### Steps in short:

1. Go to your local clover-plugin workspace and type
 

```
mvn release:clean
mvn release:prepare -DpushChanges=false
mvn release:perform -DlocalCheckout=true
```
2. Login to Nexus OSS, open Staging Repositories, click "Close" button.
3. Download clover-X.X.X.hpi from OSS Nexus staging area and install and test it in your Hudson instance.
4. If works OK, push changes to *hudson2-plugins/clover-plugin* and click "Release" button on Nexus OSS.
5. Update the content of [HUDSON/Clover+Plugin](http://wiki.hudson-ci.org/display/HUDSON/Clover+Plugin) wiki page.

## Hudson 3

### Preconditions

- You are a member of the Hudson Plug-in Development team, including
  - permissions to publish release artefacts on <https://oss.sonatype.org/content/repositories/releases/org/hudsonci/plugins/clover>
  - write permissions in master git repository on <https://github.com/hudson3-plugins/clover-plugin>
  - account on Hudson Wiki in order to edit the <http://wiki.hudson-ci.org/display/HUDSON/Clover+Plugin> page

## Build and Test

1. Clone the [git@github.com/hudson3-plugins/clover-plugin.git](https://github.com/hudson3-plugins/clover-plugin.git)
2. Create a bug on <https://bugs.eclipse.org> (Hudson project, Plugins component) for the issue you are fixing.
3. Make the necessary changes
4. Bump the plug-in version number (in pom.xml).  
Bump the `com.cenqua.clover:clover` version number if necessary (in pom.xml).  
Bump `org.eclipse.hudson.plugins:hudson-plugin-parent` version if necessary (in pom.xml).
5. Test by running Hudson with the Clover plugin installed and setting up a Clovered project for CI. This can be achieved via:
  - mvn clean hpi:run**
    - a. it starts Hudson on localhost:8080 by default, open it in web browser
    - b. open "Manage Hudson" > "Manage plugins" > "Installed" and check if new "Hudson Clover Plugin" is listed
    - c. configure new project (you can use MoneyBags as a test case) and a build job (e.g. "freestyle project" with "ant task")
    - d. configure "Post-build actions" > "Publish Clover Coverage Report" in the build job
    - e. run "Build now" and check if Clover summary report is available
6. Commit changes and push to GitHub. Include your Bugzilla bug ID from above in the commit line.

## Release and Publish

Detailed instruction [here](#). If this is your first time publishing a Hudson release, allow at least a day or two for Sonatype to process your JIRA requests, set you up and eventually enable Central Sync.

### Steps in short:

1. Go to your local clover-plugin workspace and type
 

```
mvn release:clean
mvn release:prepare -DpushChanges=false
mvn release:perform -DlocalCheckout=true
```
2. Login to Nexus OSS, open Staging Repositories, click "Close" button.
3. Download clover-X.X.X.hpi from OSS Nexus staging area and install and test it in your Hudson instance.
4. If works OK, *git push* changes from your local workspace to `hudson3-plugins/clover-plugin` and click "Release" button on Nexus OSS.
5. Update the content of [HUDSON/Clover+Plugin](#) wiki page.

## Tips

### Don't release Hudson 2 and Hudson 3 plugin in parallel


Why? Because a staging repository created on the [oss.sonatype.org](https://oss.sonatype.org) will contain mixed artifacts from both plugins.

### How to see the latest version in Hudson Plugin Manager? Where's the magic?

Once your plugin appears in the Maven Central repository (wait for couple of hours after pressing the release button), it will be included in the Hudson Update Center:

- <http://hudson-ci.org/update-center3/update-center.json> (Hudson 3)
- <http://hudson-ci.org/update-center.json> (Hudson 2)

An update center generator tools runs periodically to generate the Hudson Update Center JSON file from

- <http://repo1.maven.org/maven2/org/hudsonci/plugins> and
- <http://repo1.maven.org/maven2/org/jvnet/hudson/plugins>  *not sure if it still runs*

After this, the latest plugin version will be seen in the Plugin Manager in the Hudson administration panel.

## Clover for Jenkins Developer Guide

## Preconditions

- You have GitHub account with SSH keys configured and the local SSH key is not protected by password (otherwise release:prepare will hang)
- Configure [maven.jenkins-ci.org](http://maven.jenkins-ci.org) in settings.xml:

```

settings.xml
<!-- user/password of your Jenkins account! http://jenkins-ci.org/account -->
<server>
  <id>maven.jenkins-ci.org</id>
  <username>xxx</username>
  <password>yyy</password>
</server>

```

- Clone the repository from GitHub using SSH:

```

git clone git@github.com:jenkinsci/clover-plugin.git
jenkins-clover-plugin

```

- **JDK1.6+**
- **Maven 3.0+**

## Build and Test

1. If you have already cloned jenkinsci/clover-plugin then ensure you *git pull* upstream
2. Create a JIRA on <https://issues.jenkins-ci.org/> for the issue you are fixing.
3. Make changes necessary.
4. Bump the dependency version on Clover Core if necessary (in pom.xml)
5. Test by running Jenkins with the Clover plugin installed and setting up a Clovered project for CI. This can be achieved via:

### mvn clean hpi:run

- a. it will start Jenkins on localhost:8080 by default, open it in web browser
  - b. open "Manage Jenkins" > "Manage plugins" > "Installed" and check if new "Jenkins Clover Plugin" is listed
  - c. configure new project (you can use MoneyBags as a test case) and a build job (e.g. "freestyle project" with "ant task")
  - d. configure "Post-build actions" > "Publish Clover Coverage Report" in the build job
  - e. run "Build now" and check if Clover summary report is available
6. Commit changes and push to GitHub. Include your JIRA issue from above.

## Release and Publish

Run

```

mvn release:prepare
mvn release:perform

```

Verify that the plugin has been deployed:

- visit <http://maven.jenkins-ci.org/content/repositories/releases/org/jenkins-ci/plugins/clover/>
- visit <http://repo.jenkins-ci.org/releases/org/jenkins-ci/plugins/>
- check <http://updates.jenkins-ci.org/update-center.json> (file is updated every 6 hours)
- run your local Jenkins and open Administration page, check if you see the latest version number

## Documentation

Edit the <http://wiki.jenkins-ci.org/display/JENKINS/Clover+Plugin> page.

## Clover for Maven 2 and 3 Developer Guide

### Getting the Source Code

The Clover-for-Maven2&3 plug-in source is stored in Mercurial repository on Bitbucket.org. To get a local copy of the source code, a Mercurial client is required.

The following command will checkout the source code of the [atlassian/maven-clover2-plugin](#):

```
hg clone ssh://hg@bitbucket.org/atlassian/maven-clover2-plugin
```

**⚠ TIP:** do not mislead with the [atlassian/maven-clover-plugin](#) repository which is a Clover-for-Maven1.

### Installing the Plugin

The plugin can then be built, tested and installed via:

```
mvn clean install
```

### Running Integration Tests

To run the integration tests, use:

```
mvn clean integration-test -Pintegration-tests
```

### Submitting a Patch

To submit a patch:

1. Make and test the change in your local subversion work area
2. Create a JIRA issue in project "Clover" (**CLOV**), set "Maven Plugin" component.
3. Ensure any new features/configuration options have been documented in the issue description.
4. Commit changes and:
  - clone a repository on bitbucket.org and push changes to it
  - create a pull request for [atlassian/maven-clover2-plugin](#) project

or

- generate a patch by running the following command in your local work area (where XXXX is the id of the JIRA issue created above)

```
hg diff > CLOV-XXXX.patch
```

- Upload the patch to the JIRA issue you created.

### Binaries

Stable releases can be downloaded from [Maven Central](#) or from <https://maven.atlassian.com/content/repositories/atlassian-public/com/atlassian/maven/plugins/maven-clover2-plugin>

## Miscellaneous

⚠ The JIRA issue tracker <https://studio.plugins.atlassian.com/browse/CLMVN> is deprecated, please raise issues on <https://jira.atlassian.com/browse/CLOV>

⚠ The SVN repository <https://studio.plugins.atlassian.com/svn/CLMVN/trunk> is deprecated, use the HG repository from <https://bitbucket.org/atlassian/maven-clover2-plugin>

## Clover-for-Maven1 Developer Guide

### Getting the Source Code

The Clover-for-Maven1-Plugin source is stored in Mercurial repository on bitbucket.org. To get a local copy of the source code, a Mercurial client is required.

The following command will checkout the source code of the `atlassian/maven-clover-plugin`:

```
hg clone ssh://hg@bitbucket.org/atlassian/maven-clover-plugin
```

### Testing the Plugin

1) Running test cases:

```
cd maven-clover-plugin
maven plugin:install plugin:repository-install
cd src/plugin-test
maven testPlugin
-Dmaven.repo.remote=https://maven.atlassian.com/maven1,http://repo1.maven.org/maven
-Dmaven.clover.license.path=/path/to/clover.license
```

2) Generating site documentation:

```
maven site
-Dmaven.repo.remote=http://mirrors.ibiblio.org/maven,http://repo1.maven.org/maven
```

⚠ There might be a problem with finding `qdox-current.jar` - in such case, download it from <http://mirrors.ibiblio.org/pub/mirrors/maven2/vdoclet/qdox/current/qdox-current.jar> and install manually in `~/.maven/repository/vdoclet/jars/qdox-current.jar`

### Submitting a Patch

To submit a patch:

1. Make and test the change in your local work area.
2. Create a JIRA issue in project "Clover" (**CLOV**), set "Maven Plugin" component.
3. Ensure any new features/configuration options have been documented in the issue description.
4. Commit changes and:

- clone a repository on bitbucket.org and push changes to it
  - create a pull request for [atlassian/maven-clover-plugin](#) project
5. or
- generate a patch by running the following command in your local work area

```
hg diff > CLOV-XXXX.patch
```


where XXXX is the id of the JIRA created above.


- Upload the patch to the JIRA issue you created.

## Binaries

Stable releases can be downloaded from [Maven Central](#) or from <https://maven.atlassian.com/maven1/maven/plugins/>

## Miscellaneous

 The JIRA issue tracker <https://studio.plugins.atlassian.com/browse/CLMVNONE> is deprecated, please raise issues on <https://jira.atlassian.com/browse/CLOV>

 The SVN repository <https://studio.plugins.atlassian.com/svn/CLMVNONE/trunk> is deprecated, use the HG repository from <https://bitbucket.org/atlassian/maven-clover-plugin>

## Clover Road Map




### Disclaimer

All bugs and feature requests are managed and scheduled according to the [Atlassian Bug Fixing Policy](#) and the [Implementation of New Features Policy](#).

In particular, it means that issues can be moved between releases, priority of bugs can change, releases can be split or merged etc. Changes to the existing road map are usually triggered by events like: critical bug found, customer feedback, tickets raised on Atlassian Support. Therefore, please do not treat the following road map as an official commitment, but rather as a vision in which direction Clover will develop. The most accurate planning is for the incoming release.

### List of upcoming Clover releases

Year	2014					2015				
Version	<b>4.0.0</b> 	4.0.1	4.0.2	4.0.3	4.0.4	<b>4.1.0</b>	4.1.1	4.1.2	4.1.3	<b>4.2.0</b>
Content	New HTML report	ADG, Grails, Groovy fixes	Bamboo Clover Plugin	REST API for reports	API / SPI for new languages	Scala language support	Bug fixes (Ant, Maven)	Bug fixes (Eclipse, IDEA)	Bug fixes (Test optimization)	Groovy in IDE

More details on the [CLOV project road map on jira.atlassian.com](#)

### End of technical support for Clover versions

- [Atlassian Support End of Life Policy](#)

### Ideas for new Clover features

If you have an idea for a new feature in Clover, you can report it on <https://jira.atlassian.com/browse/CLOV> (just

make sure that it's not reported already). We're also very interested in your feedback about feature requests raised already - feel free to vote on them.

Top 10 most voted new features are:

Key	Summary	Votes	Fix Version/s
CLOV-932	Provide support for the Scala language	33	someday
CLOV-250	Clover support for AspectJ	22	won't fix
CLOV-939	Add Groovy support in Clover for IntelliJ	14	4.2.0
CLOV-1009	Easy integration with Gradle	5	someday
CLOV-286	Expose a Java API to the Clover database	3	4.0.4
CLOV-739	New option to "Run Optimized Tests" on every Make (CMD-F9)	2	someday
CLOV-738	Allow users to enable/disable coverage collection at runtime.	2	someday
CLOV-570	Android for Clover-for-Maven	2	someday
CLOV-333	Allow for sub expression Analysis	2	4.3.x
CLOV-1142	Expose a Service Provider Interface for Clover	2	4.0.4

Showing 10 out of 87 issues

## Contributing to the Clover Documentation

Would you like to share your Clover hints, tips and techniques with us and with other Clover users? We welcome your contributions. Have you found a mistake in the documentation, or do you have a small addition that would be so easy to add yourself rather than asking us to do it? You can update the documentation page directly.

### Getting Permission to Update the Documentation

Our documentation wiki contains developer-focused documentation (such as API guides, plugin and gadget development guides and guides to other frameworks) as well as product documentation (user's guides, administrator's guides and installation guides).

If you want to update the [Clover product documentation](#), we ask you to sign the Atlassian Contributor License Agreement (ACLA) before we grant you wiki permissions to update the documentation space. Please read the [ACLA](#) to see the terms of the agreement and the documentation it covers. Then sign and submit the agreement as described on the form attached to that page.

### Following our Style Guide

Please read our short [guidelines for authors](#).

### How we Manage Community Updates

Here is a quick guide to how we manage community contributions to our documentation and the copyright that applies to the documentation:

- **Monitoring by technical writers.** The Atlassian technical writers monitor the updates to the documentation spaces, using RSS feeds and watching the spaces. If someone makes an update that needs some attention from us, we will make the necessary changes.
- **Wiki permissions.** We use wiki permissions to determine who can edit the documentation spaces. We ask people to sign the [Atlassian Contributor License Agreement \(ACLA\)](#) and submit it to us. That allows us to verify that the applicant is a real person. Then we give them permission to update the documentation.
- **Copyright.** The Atlassian documentation is published under a Creative Commons CC BY license. Specifically, we use a [Creative Commons Attribution 2.5 Australia License](#). This means that anyone can copy, distribute and adapt our documentation provided they acknowledge the source of the documentation. The CC BY license is shown in the footer of every page, so that anyone who contributes to our documentation knows that their contribution falls under the same copyright.

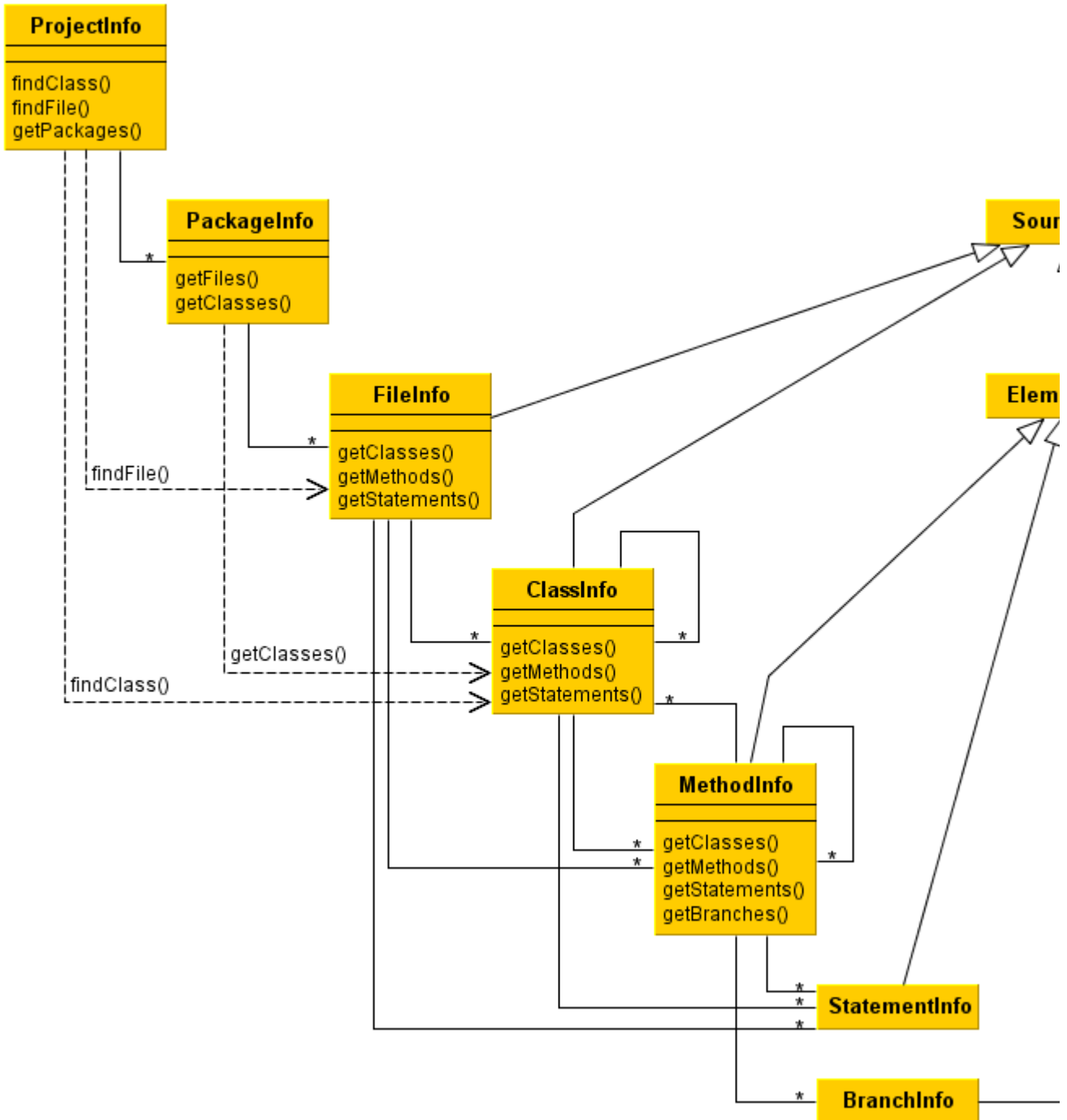
### RELATED TOPICS

Author Guidelines  
 Atlassian Contributor License Agreement

## Database Structure

### Model

#### Class overview



#### Possible entity nesting

entity below can be nested in entity	branch	statement	method	class	file	package
branch	✗	✗	✓	✗	✗	✗
statement	✗	✗	✓	✓ N	✓ N	✗



method			N		N	
class			N	N		1
file						
package						2

### Comments

N - new in Clover 3.2.0

1 - there are helper methods `PackageInfo.getClasses() / getAllClasses()` which returns classes from a package

2 - there are helper methods `PackageInfo.getClassesIncludingSubPackages() / getAllClassesIncludingSubPackages()` searching for classes in nested packages

From a logical perspective a branch should be nested inside a statement, e.g. "if (a > 5) .." has one statement with true and false branches in it. However, due to performance reasons, branches are kept aside statements, directly under a method. It's planned to add branches also under a class and a file in one of future Clover releases.

Since Clover 3.2 it's possible to nest classes inside classes. This can be used to model an inner class such as:

```
class A {
    class B { }
}
```

Clover does not keep inner classes this way, however. All inner classes are kept directly under a file. One of the reasons for such approach is a separation of code metrics, i.e. a complexity of an inner class B does not count to the complexity of a parent class A.

Clover does not keep anonymous inline classes as a class entity in the model. Instead of this, methods of an anonymous class are being added to the parent class. This is a legacy issue.

Note that Clover 3.2 keeps lambda functions as classes declared under a method. Due to fact that lambda functions can be converted to a functional interface and vice versa, we plan to fix it and make it consistent in a future Clover release. Therefore, anonymous inline classes will have their own entity in a database model and will be kept under an enclosing method.

## Java API

Interfaces describing the database structure are located in the **com.atlassian.clover.api.registry** package ([JavaDoc](#)).

They can be grouped into few categories:

- basic entities stored in a database are represented by *ProjectInfo*, *PackageInfo*, *FileInfo*, *ClassInfo*, *MethodInfo*, *StatementInfo* and *BranchInfo*
- these entities implement *HasPackages*, *HasFiles*, *HasClasses*, *HasMethods*, *HasStatements* or *HasBranches* interfaces which allow to navigate to their children
- *HasParent*, *EntityContainer* and *EntityVisitor* allows to get to the parent entity (note that some entities might have different parent types)
- *HasMetrics*, *HasAggregatedMetrics* returns information about code metrics
- helper interfaces describing data structures such as *MethodSignatureInfo*, *Annotation*, *AnnotationValue* etc

## Reading from a Clover database

An example how to read a content of a database.

If you'd like to read a database without coverage, then replace "CloverDatabase.loadWithCoverage(..)" by "new CloverDatabase(initstring)"

```
import com.atlassian.clover.CloverDatabase;
import com.atlassian.clover.CoverageDataSpec;
import com.atlassian.clover.api.registry.ClassInfo;
import com.atlassian.clover.api.registry.FileInfo;
import com.atlassian.clover.api.registry.MethodInfo;
import com.atlassian.clover.api.registry.PackageInfo;
import com.atlassian.clover.api.registry.ProjectInfo;

import java.io.PrintStream;

public class SimpleRegistryDumper {
    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("Usage:");
            System.err.println("java " + SimpleRegistryDumper.class.getName() + " "
database");
        } else {
            // read clover database together with coverage recording files, use
time span=0 (latest build)
            CloverDatabase db = CloverDatabase.loadWithCoverage(args[0], new
CoverageDataSpec());
            ProjectInfo projectInfo = db.getRegistry().getProject();
            // print some project details
            printProject(projectInfo, System.out);
        }
    }
    private static void printProject(ProjectInfo db, PrintStream out) {
        for (PackageInfo packageInfo : db.getAllPackages()) {
            out.println("package: " + packageInfo.getName());
            for (FileInfo fileInfo : packageInfo.GetFiles()) {
                out.println("\tfile: " + fileInfo.getName());
                for (ClassInfo classInfo : fileInfo.getClasses()) {
                    out.println("\t\tclass: " + classInfo.getName());
                    for (MethodInfo methodInfo : classInfo.getMethods()) {
                        out.println("\t\t\tmethod: " + methodInfo.getName());
                    }
                }
            }
        }
    }
}
```

## Writing to a Clover database

```
import com.atlassian.clover.api.CloverException;
import com.atlassian.clover.api.instrumentation.InstrumentationSession;
import com.atlassian.clover.api.registry.FileInfo;
import com.atlassian.clover.context.ContextSet;
import com.atlassian.clover.registry.Clover2Registry;
import com.atlassian.clover.registry.FixedSourceRegion;
import com.atlassian.clover.registry.entities.MethodSignature;
import com.atlassian.clover.registry.entities.Modifier;
```

```

import com.atlassian.clover.registry.entities.Modifiers;
import com.atlassian.clover.registry.entities.Parameter;
import com.atlassian.clover.spi.lang.LanguageConstruct;

import java.io.File;
import java.io.IOException;

public class SimpleCodeInstrumenter {
    private Clover2Registry registry;
    private InstrumentationSession session;

    public SimpleCodeInstrumenter(String initString, String projectName) throws
CloverException {
        try {
            final File dbFile = new File(initString);
            registry = Clover2Registry.createOrLoad(dbFile, projectName);
            if (registry == null) {
                throw new CloverException("Unable to create or load clover registry
located at: " + dbFile);
            }
        } catch (IOException e) {
            throw new CloverException(e);
        }
    }

    public void startInstrumentation(String encoding) throws CloverException {
        session = registry.startInstr(encoding);
    }

    public Clover2Registry endInstrumentation(boolean append) throws
CloverException {
        try {
            session.close();
            if (append) {
                registry.saveAndAppendToFile();
            } else {
                registry.saveAndOverwriteFile();
            }
            return registry;
        } catch (IOException e) {
            throw new CloverException(e);
        }
    }

    /**
     * This method should perform the actual instrumentation. On every code
construct you find in your
     * source file(s) being instrumented (such as file, class, method, statement,
branch) you shall call
     * proper handler from InstrumentationSession class in order to record data for
a given code entity
     * in the Clover database.
     */
    public void instrument() {
        // note: there is no need to call session.enterPackage(packageName), it
will be called from
        // session.enterFile(); the same applies to session.exitPackage()

        // example: register a file with attributes such as enclosing package,
number of lines, time stamp, checksum
        String packageName = "com.acme.my.package";
        File sourceFile = new File("com/acme/my/package/Foo.java");
        FileInfo fileInfo = session.enterFile(packageName, sourceFile,

```

```
        200, 100, sourceFile.lastModified(), sourceFile.length(), 3423452);

    // example: register a class (in current file)
    session.enterClass("Foo", new FixedSourceRegion(10, 1), false, false,
false);

    // example: add a method to the Foo class
    MethodSignature methodSignature = new MethodSignature("helloWorld", null,
// method name and generic type
        "void",
// return type
        new Parameter[] { new Parameter("String", "name") },
// formal parameters
        null,
// throws
        Modifiers.createFrom(Modifier.PROTECTED | Modifier.STATIC, null));
// modifiers
    session.enterMethod(new ContextSet(), new FixedSourceRegion(12, 1),
// start row:column
        methodSignature, false, false, 5,
LanguageConstruct.Builtin.METHOD); // other attributes

    // example: add a statement in the helloWorld method
    session.addStatement(new ContextSet(), new FixedSourceRegion(13, 1, 13,
44),
        3, LanguageConstruct.Builtin.STATEMENT);

    // end method, class and a file
    session.exitMethod(14, 1); // end row:column
    session.exitClass(30, 2); // end row:column
    session.exitFile();
}

public static void main(String[] args) throws CloverException {
    if (args.length != 1) {
        System.err.println("Usage:");
        System.err.println("java " + SimpleCodeInstrumenter.class.getName() + "
database");
    } else {
        SimpleCodeInstrumenter instrumenter = new
SimpleCodeInstrumenter(args[0], "MyProject");
        instrumenter.startInstrumentation("UTF-8");
        instrumenter.instrument();
        instrumenter.endInstrumentation(true);
    }
}
```

