



Clover User Manual

Version 1.3.14

Includes:

Eclipse Plugin 1.2.12

IDEA 3.x Plugin 0.8

IDEA 4.x Plugin 1.0.7

IDEA 5.x Plugin 1.0.8

IDEA 6.x Plugin @IDEA6_RELEASE_NUM@

JDeveloper Plugin 1.0

JBuilder Plugin 1.0

NetBeans Module 0.6

1. Introduction

1.1. Starting Points

If you are new to Clover and want to get it going with your Ant project quickly, try the [Quickstart Guide](#). The [Introduction for Code Coverage](#) section provides a brief background on the theory and motivation behind Code Coverage.

If you are browsing and interested in seeing how Clover can be used on your project, see [Using Clover Interactively](#) and [Using Clover in Automated builds](#).

If you are using a Clover IDE Plugin, see the [Plugin Guides](#) section.

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>.

For Clover troubleshooting information, see the [FAQ](#) or [Online Forums](#).

1.1.1. System Requirements

JDK Version	JDK 1.2 or greater required to perform instrumented compilation and coverage measurement. JDK 1.3 or greater required to produce coverage reports.
Ant Version	Ant 1.4.1 or greater.
Operating System	Clover is a pure Java application and should run on any platform provided the above requirements are satisfied.

The Clover IDE Plugins document their own IDE version requirements. Please consult the [Plugins Section](#)

1.1.2. Installing your license file

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.cenqua.com>.

To install your Clover license file, you need to do one of the following:

- Place the license file next to the Clover jar file (or next to the Clover plugin jar file, if you are running a Clover IDE plugin).

- Place the license file on the Java Classpath that will be used to run Clover.
- 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.

1.1.3. Acknowledgements

Clover makes use of the following excellent 3rd party libraries.

Jakarta Velocity 1.2	Templating engine used for Html report generation.
Antlr 2.7.1	A public domain parser generator.
iText 0.96	Library for generating PDF documents.
Jakarta Ant	The Ant build system.

Note:

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 that may use different versions of these libraries in their projects.

2. Code Coverage

2.1. Code Coverage

2.1.1. What is Code Coverage?

Code coverage measurement simply determines those statements in a body of code have been executed through a test run and those which 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 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.

2.1.2. 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.

2.1.3. How Code Coverage Works

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

Source Code Instrumentation	This approach adds instrumentation statements to the source code and compiles the code with the normal compile tool chain to produce an instrumented assembly.
Intermediate code Instrumentation	Here the compiled class files are instrumented by adding new bytecodes and a new instrumented class generated.
Runtime Information collection	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.

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.

2.1.4. 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:

Statement	Statement coverage measures whether each statement is executed
Branch	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.
Method	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:

$$TPC = (BT + BF + SC + MC) / (2*B + S + M)$$

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

3. Clover with Ant

3.1. Quick Start Guide 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.

Follow these simple steps to integrate Clover with your build:

3.1.1. Install Clover

ensure you are using a recent version of Ant (v1.4.1 or greater)

copy `<CLOVER_HOME>/lib/clover.jar` into `<ANT_HOME>/lib`. (If you can't install into `ANT_HOME/lib`, see [Installation Options](#)).

3.1.2. Add Clover targets

Edit `build.xml` for your project:

1. add the Clover Ant tasks to your project:

```
<taskdef resource="clovertasks"/>
```

2. add a target to switch on Clover:

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

3. add one or more targets to run clover reports:
to launch the Swing viewer, use:

```
<target name="clover.swing" depends="with.clover">
  <clover-view/>
</target>
```

- OR - for html reporting, use (change the outfile to a directory path where Clover should put the generated html):

```
<target name="clover.html" depends="with.clover">
  <clover-report>
    <current outfile="clover_html">
      <format type="html"/>
    </current>
  </clover-report>
</target>
```

- OR - for xml reporting, use (change the outfile to a file where Clover should write the

xml file):

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

- OR - for pdf reporting, use (change the outfile to a file where Clover should write the pdf file):

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

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

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

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="${ant.home}/lib/clover.jar"/>
  </classpath>
</junit>
```

3.1.3. 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
```

3.1.4. Generate a Coverage Report

To generate a Clover coverage report:

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

3.2. Installation Options

In order to use Clover with Ant you must put `clover.jar` in Ant's classpath. Options for doing this depend on the version of Ant you are using.

3.2.1. Ant 1.4.1, 1.5.x

Prior to Ant 1.6, the easiest way to install Clover is to 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.

3.2.2. Ant 1.6.x

Ant 1.6 introduces several new ways to add jars to Ant's classpath. This allows more flexibility when installing Clover.

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 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).

3.2.3. Adding Clover to Ant's classpath from build.xml

In some cases it is not desirable to add `clover.jar` to Ant's classpath using the methods described above. This section outlines a method for adding `clover.jar` to Ant's classpath

by modifying only the project `build.xml` file, using a special utility Ant task called `<extendclasspath>` that is distributed with Clover.

The `<extendclasspath>` task is distributed in `CLOVER_HOME/etc/cenquataasks.jar`

1. copy `CLOVER_HOME/lib/clover.jar` and `CLOVER_HOME/etc/cenquataasks.jar` to a project-relative directory (the rest of these instructions assume both jars are installed at `PROJECT_HOME/lib`)
2. edit `build.xml` and add the following near the top of the file:

```
<taskdef resource="com/cenqua/ant/antlib.xml" classpath="lib/cenquataasks.jar"/>
<extendclasspath path="lib/clover.jar"/>
<taskdef resource="clovertasks" classpath="lib/clover.jar"/>
```

You can now use the standard Clover Ant tasks in your `build.xml` file.

3.2.4. Checking if Clover is available for the build

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.cenqua.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" />
</target>

<target name="with.clover" depends="guard.noclover">
  ...
</target>
```

3.3. Usage Scenarios

3.3.1. 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 initstring="clover-db/mycoverage.db">
  <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 initstring="clover-db/mycoverage.db">
  <files includes="${coverage.includes}"/>
</clover-setup>
```

Developers can then use a command line like:

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

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.

The [<clover-view>](#) task launches the Swing coverage viewer which allows interactive browsing of coverage results:

```
<clover-view/>
```

Note:

If you launch the viewer from a second window, it can be left running while you develop. At the end of every test run, you can hit the "refresh" button on the viewer to load the latest coverage results.

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"/>
```

3.3.2. 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 third party tools that support this type of build are [AntHill](#), [Centipede](#) 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

The [<clover-report>](#) task generates source-level html coverage reports that can be published for viewing by the whole team:

```
<target name="clover.report" depends="with.clover">
  <clover-report>
    <current outfile="clover_html">
      <format type="html"/>
    </current>
  </clover-report>
</target>
```

```
</clover-report>
</target>
```

Executive summary coverage reports

The [<clover-report>](#) task can generate summary reports in PDF suitable for email or audit purposes.

```
<target name="clover.summary" depends="with.clover">
  <clover-report>
    <current summary="yes" outfile="coverage.pdf">
      <format type="pdf"/>
    </current>
  </clover-report>
</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 colated 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"/>

  <clover-report>

    <!-- generate a current report -->
    <current outfile="clover_html">
      <format type="html"/>
    </current>

    <!-- generate a historical report -->
    <historical outfile="clover_html/historical.html"
               historyDir="clover_hist">
      <format type="html"/>
    </historical>
  </clover-report>
</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-report>
    <current summary="yes" outfile="coverage.pdf">
      <format type="pdf"/>
    </current>
  </clover-report>
  <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>

```

3.4. Ant Task Reference

3.4.1. Clover Ant Tasks

Installing the 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:

1. install `clover.jar` into `ANT_HOME/lib`
2. add the following lines to your build file:

```

<taskdef resource="clovertasks"/>
<typedef resource="clovertypes"/>

```

The tasks

<u><clover-setup></u>	Configures and initialises Clover. This task needs to be run before other Clover tasks.
<u><clover-report></u>	Produces coverage reports in different formats.
<u><clover-check></u>	Tests project/package code coverage against criteria, optionally failing the build if the criteria are not met.
<u><clover-log></u>	Reports coverage results to the console at various levels.
<u><clover-historypoint></u>	Records a coverage history point for use in historical coverage reports.

<u><clover-view></u>	Launches the Swing coverage viewer.
<u><clover-clean></u>	Delete the coverage database and/or associated coverage records.
<u><clover-merge></u>	Merges two or more Clover databases to allow multi-project reporting.

3.4.2. <clover-setup>

Description

The <clover-setup> task initialises Clover for use with your project. In Clover 1.0, Clover's operation was managed by setting various Ant properties. The <clover-setup> task simplifies this procedure.

Parameters

Attribute	Description	Required
initstring	The Clover initString describes the location of the clover coverage database. Typically this is a relative or absolute file reference. Note that this value is not resolved relative to the project's base directory.	Yes
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
clovercompiler	After instrumentation, Clover hands off compilation to the standard Ant compiler adapter (or the compiler specified by the build.compiler Ant property). This attribute specifies the adapter to use. It takes the same values as the standard Ant build.compiler property. If you wish to specify an alternative compiler, you can either set the	No

	build.compiler property or use this attribute.	
preserve	A boolean attribute which controls whether the instrumented source will be retained after compilation.	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
flushpolicy	This attribute controls how Clover flushes coverage data during a test run. Valid values are directed, interval, or threaded. directed Coverage data is flushed at JVM shutdown, and after an inline flush directive. interval Coverage data is flushed as for directed, as well as periodically at a <i>maximum</i> rate based on the value of flushinterval. This is a "passive" mode in that flushing potentially occurs as long as instrumented code is being executed. threaded Coverage data is flushed as for directed, as well as periodically at a rate based on the value of flushinterval. This is an "active" mode in that flushing occurs on a separate thread and is not dependent on the	No; defaults to directed

	execution of instrumented code. For more information, see Flush Policies .	
flushinterval	When the flushpolicy is set to interval or threaded this value is the minimum period between flush operations (in milliseconds)	No
relative	This controls whether the initstring parameter is treated as a relative path or not.	No; defaults to false

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 convenience method to set these properties. This means that builds that use the Clover 1.0 property set will continue to operate as expected.

Nested Elements of <clover-setup>

<files>

An Ant patternset element which controls which files are included or excluded from Clover instrumentation.

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.

<methodContext>

Specifies a method Context definition. See [Using 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 Using Contexts)	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

<statementContext>

Specifies a statement Context definition. See [Using 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 Using Contexts)	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

Examples

```
<clover-setup initstring="clover-db/coverage.db"/>
```

This example is the minimal setup to use clover. In this case the clover coverage database is

located in the clover-db relative directory.

```
<clover-setup initstring="clover-db/coverage.db"
  enabled="${enable}"
  <files>
    <exclude name="**/optional/**/*.*.java"/>
  </files>
</clover-setup>
```

This example shows the use of a property, "enable", to control whether Clover instrumentation is enabled. Also the instrumentation will exclude all java source files in trees named "optional". Note that the fileset can also be referenced using a refid attribute.

```
<clover-setup initstring="clover-db/coverage.db"
  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 initstring="clover-db/coverage.db"
  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 initstring="clover-db/coverage.db"
  clovercompiler="jikes"/>
```

This example will pass compilation to the "jikes" compiler once instrumentation is complete.

3.4.3. <clover-report>

Description

Generates current and historical reports in multiple formats. The basic nesting of elements within the <clover-report> task is as follows:

```
<clover-report>
  <current>
    <fileset/>
    <sourcepath/>
    <format/>
  </current>
  <historical>
    <format/>
    <overview/>
    <coverage/>
    <metrics/>
    <movers/>
  </historical>
</clover-report>
```

Parameters

Attribute	Description	Required
initstring	The initstring of the coverage database.	No; If not specified here, you must ensure <clover-setup> is called prior the execution of this task.
failOnError	If true, failure to generate a report causes a build failure.	No; defaults to "true".

Nested elements of <clover-report>

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.

<current>

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

Parameters

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"
alwaysReport	If set to true, a report will be generated even in the absence of coverage data.	No; defaults to "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, XML) or a directory (HTML).	Yes
summary	Specifies whether to generate a summary report or detailed report.	No; Defaults to "false".
package	Restricts the report to a particular package.	No
span	Specifies how far back in time to include coverage recordings from since the last Clover build. See Using Spans .	No; Defaults to "0s".

<historical>

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

Parameters

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"
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
historyDir	The directory containing Clover historical data as produced by the <clover-historypoint> task.	Yes
package	Restricts the report to a particular package.	No
from	Specifies the date before which data points will be ignored. The date must be specified either using the default java.text.SimpleDateFormat for your locale or using the pattern defined in the "dateFormat" attribute.	No
to	Specifies the date after which data points will be ignored. The date must be specified either using the default java.text.SimpleDateFormat for your locale or using the pattern defined in the "dateFormat" attribute.	No
dateFormat	Specifies a date format string for parsing the "from" and "to" fields. The string must contain a valid	No; default set to java.text.SimpleDateFormat using the default pattern and date format symbols for the

	java.text.SimpleDateFormat pattern.	default locale.
--	--	-----------------

Nested elements of <current>

<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's fileset selectors, more complicated selections are possible, such as the files which have recently changed, or files written by a particular author.

<sourcepath>

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

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. <historical> element.

<overview>

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

<coverage>

Specifies a chart showing percentage coverage over time.

Parameters

Attribute	Description	Required
include	A comma or space separated	No; the default is that

	list of coverage metrics to include in the chart. Valid values are: branches, statements, methods, total	everything is included
--	--	------------------------

<metrics>

Specifies a chart showing other metrics over time.

Parameters

Attribute	Description	Required
include	A comma or space separated list of metrics to include in the chart. Valid values are: loc, ncloc, statements, methods, classes, files, packages	No; defaults to loc, ncloc, methods, classes
logscale	Specifies that a log scale be used on the Range Axis. This can be useful if you are including, say LOC and packages in the same chart.	No; default is "true"

<movers>

Specifies a table that shows those classes that have a coverage delta higher than a specified threshold over a specified time period.

Parameters

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; The defaults to 5

interval	The time interval over which the delta should be calculated (from the last history point). Uses the Interval 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
----------	---	---

The <format> Element

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

Parameters

Attribute	Description	Required
type	The output format to render the report in. Valid values are pdf, xml, html. Note that not all reports support all formats.	Yes, unless refid is set
refid	the id of another format element that will be used for this report. See Sharing Report Formats .	No
id	the id of this format element.	No
bw	Specify that the report should be black and white. This will make HTML reports smaller (with no syntax highlighting) and make PDF reports suitable for printing on a non-colour printer.	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	No; defaults to PcCoveredAsc

	Total elements covered, ascending ElementsCoveredDesc Total elements covered, descending ElementsUncoveredAsc Total elements uncovered, ascending ElementsUncoveredDesc Total elements uncovered, descending	
noCache	(HTML only) 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 Using Contexts .	No
pageSize	(PDF only) 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"
tabWidth	(Source level reports only) 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

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">
    <format type="pdf"/>
  </current>
</clover-report>
```

Generates a PDF report of the current coverage.

```
<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, the "outfile" argument requires a directory instead of a filename.

```
<clover-report>
  <current outfile="clover_html" title="Util Coverage">
    <format type="html" orderBy="ElementsCoveredAsc"/>
  </current>
</clover-report>
```

Generates a detailed coverage report in HTML with output ordered by total number of covered elements, rather than percentage coverage.

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

Generates a sourcelevel 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.

Examples of Historical Report Configurations

```

    <clover-report>
      <historical outfile="historical.pdf"
                historyDir="clover_history">
      </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 history dir 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"/>
  </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.

3.4.4. <clover-historypoint>

Description

Records a coverage history point for use in historical coverage reports.

Parameters

Attribute	Description	Required
historyDir	The directory where historical data is stored.	Yes
initstring	The initstring of the coverage database.	No; If not specified here, you must ensure <clover-setup> is called prior the execution of this task.
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 java.text.SimpleDateFormat pattern.	No; default set to java.text.SimpleDateFormat using the default pattern and date format symbols for the default locale.
filter	comma or space separated list of contexts to exclude when	No

	generating the historypoint. See Using Contexts .	
span	Specifies how far back in time to include coverage recordings from since the last Clover build. See Using Spans .	No; Defaults to "0s".

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 historypoint. This allows historypoints 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.

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

3.4.5. <clover-check>

Description

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
-----------	-------------	----------

target	The target percentage total coverage for the project. e.g. "10%"	At least one of these, unless nested <package> elements are specified.
methodTarget	The target percentage method coverage for the project.	
statementTarget	The target percentage statement coverage for the project.	
conditionalTarget	The target percentage conditional coverage for the project.	
initstring	The initstring of the coverage database.	No; If not specified here, you must ensure <clover-setup> is called prior the execution of this task.
haltOnFailure	Specifies if the build should be halted if the target is not met.	No; default is "false"
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 Using Contexts .	No
span	Specifies how far back in time to include coverage recordings from since the last Clover build. See Using Spans .	No; Defaults to "0s".

Nested elements of <clover-check>

<package>

Specifies a target for a named package.

Parameters

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

name	The name of the package.	exactly one of these
regex	Regular expression to match package names.	
target	The target percentage total coverage for the package. e.g. "10%"	At least one of these.
methodTarget	The target percentage method coverage for the package.	
statementTarget	The target percentage statement coverage for the package.	
conditionalTarget	The target percentage conditional coverage for the package.	

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:

- 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 don'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 ensure 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 atleast 70%.

3.4.6. <clover-log>

Description

Reports coverage information to the console at different levels.

Parameters

Attribute	Description	Required
initstring	The initstring of the coverage database.	No; If not specified here, you must ensure <clover-setup> is called prior the execution of this task.
level	Controls the level of detail included in the report. Valid values are summary, class, method, statement	No; defaults to "summary"
filter	comma or space separated list of contexts to ignore when calculating coverage. See Using Contexts .	No

span	Specifies how far back in time to include coverage recordings from since the last Clover build. See Using Spans .	No; Defaults to "0s".
------	---	-----------------------

Nested elements

<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 a Ant path that Clover should use when looking for source files.

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.

3.4.7. <clover-view>

Description

Launches the Swing coverage viewer. The Ant build will pause until the viewer is closed.

Parameters

Attribute	Description	Required
initstring	The initstring of the coverage database.	No; If not specified here, you must ensure <clover-setup> is called prior the execution of this task.
span	Specifies how far back in time to include coverage recordings from since the last Clover build. See Using Spans .	No; Defaults to "0s".
tabwidth	Specifies tabwidth to use when rendering source files.	No;

Nested elements

<sourcepath>

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

Examples

```
<clover-view/>
```

Launches the viewer.

```
<clover-view>
  <sourcepath>
    <pathelement path="/some/other/location"/>
  </sourcepath>
</clover-view>
```

Launches the viewer. Clover will look for source files in the directory /some/other/location

3.4.8. <clover-clean>

Description

Delete the coverage database and associated coverage recording files.

Parameters

Attribute	Description	Required
initstring	The initstring of the database to clean.	No; If not specified here, you must ensure <clover-setup> is called prior the execution of this task.
keepdb	Keep the coverage database file. If "false", the coverage database will be deleted. ("true"/"false").	No; defaults to "true"
verbose	Show the name of each deleted file ("true"/"false").	No; defaults to "false"
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"

Examples

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

Deletes all of the coverage recordings.

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

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

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

Deletes the coverage database and all of the coverage recordings.

3.4.9. <clover-merge>

Description

Merges several Clover databases to allow for multi-project reports to be generated. To use with reporting tasks such as <clover-report>, <clover-historypoint> and <clover-view> you can use the optional "initstring" attribute on these tasks to specify the value of the merged database.

Parameters

Attribute	Description	Required
initString	The initString of the new coverage database. This has to be a writeable filepath.	Yes

Nested elements of <clover-merge>

<cloverDb>

Specifies a Clover database to merge.

Parameters

Attribute	Description	Required
initString	the initString 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.	No; defaults "0 seconds"

<cloverDbSet>

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

Parameters

Attribute	Description	Required
span	Specifies how far back in time to include coverage recordings from since the last Clover build for all databases matched.	No; defaults "0 seconds"

Examples

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

Produces a merged database containing the measured coverage of project A and project B.

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

Produces a merged database containing the measured coverage of all databases found under /home/projects.

3.5. 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 your reports.

Standalone format elements are created using the `<clover-format>` type. **Standalone formats elements are not compatible with Ant 1.4.1. You require at least Ant 1.5.1 to use this feature.** These standalone types support the same attributes and elements as the internal `<format>` elements of the `<clover-report>` task. You name the format using the standard ant "id" attribute.

In order to make the standalone format element available for use in your project, you need to add a typedef first:

```
<typedef resource="clovertypes"/>
```

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 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>
```

Here, we are generating 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 you would use:

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

4. IDE Plugin Guides

4.1. Clover IDE Plugins

Clover provides fully integrated plugins for many popular Integrated Development Environments. The plugins allow you to measure and view code coverage without leaving the IDE. They are also compatible Clover for Ant.

4.1.1. Plugin Guides

- [Eclipse](#)
- [IntelliJ 3.x](#)
- [IntelliJ 4.x](#)
- [IntelliJ 5.x](#)
- [JBuilder](#)
- [Netbeans](#)
- [JDeveloper](#)

4.2. Eclipse Plugin Guide

Plugin Version 1.2.12

Note:

This plugin has been tested with Eclipse 2.1, 2.1.1, 2.1.2, 3.0, 3.1, and 3.2.x ; using JDKs 1.3.x, JDKs 1.4.x, and JDKs 1.5.x. The plugin has also been tested on WSAD 5.1 (which is based on Eclipse 2.1.1). This plugin will **not** work with WebSphere Studio Application Developer v5.0 (WSAD v5.0 is based upon Eclipse 2.0.2).

4.2.1. 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.

4.2.2. Caveats / Known problems

Please be aware of the following when using this version of the plugin.

- This plugin may not work correctly if you have configured your Eclipse project so that the Java source and output directories are the same.
- When compiling your Java project with the Clover plugin, you must add and use a Java Development Kit (JDK) to your list of Installed JRE locations, or give the Clover plugin a JDK_HOME override value. (see below).

- This plugin will **not** work with WebSphere Studio Application Developer v5.0 (WSAD v5.0 is based upon Eclipse 2.0.2).
- There have been some reported problems when using the plugin with WSAD J2EE projects, where the build-path contains .JARs that are embedded in an .EAR or .WAR. This issue is being investigated and a fix is slated for the next release.

4.2.3. Installation

1 Locating your Eclipse plugin directory

You will need to locate where you installed Eclipse on your system. The rest of this document will refer to this location as `ECLIPSE_HOME`.

2 Removing previous versions of the plugin

It is important to remove previous version of the Clover Eclipse plugin.

- Go to to the `ECLIPSE_HOME/plugins` directory.
- Remove any directory named `com.cenqua.clover*`

3 Installing the plugin

- Once you have downloaded the Clover Eclipse plugin .zip file, extract it to a temporary location on your drive. This will create a directory named `com.cenqua.clover_x.x`
- Copy this directory to the `ECLIPSE_HOME/plugins` directory. That is, you should end up the directory `ECLIPSE_HOME/plugins/com.cenqua.clover_x.x`

4 Installing the license

- If you don't have one already, you will need to download a `clover.license` file to activate the plugin. A free evaluation license is available from [here](#).
- Copy the `clover.license` file into the `ECLIPSE_HOME/plugins/com.cenqua.clover_x.x` directory.

5 Start Eclipse

Next time you start Eclipse, the Clover plugin will be available.

4.2.4. Using the plugin

Setting up a JDK

Note:

When compiling your Java project with the Clover plugin, you **must** specify a JDK for Clover to use.

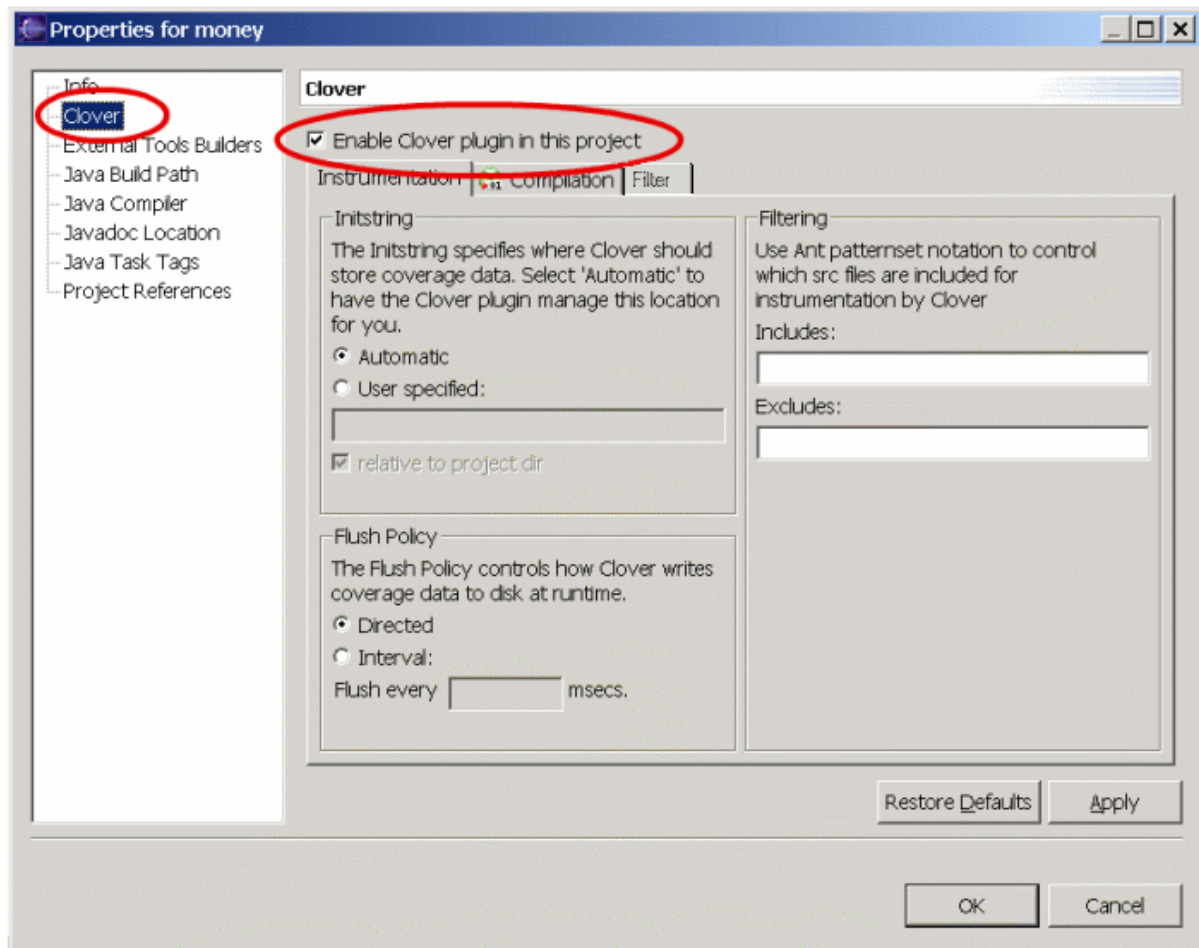
There are two ways to instruct the Clover plugin which JDK it should use.

1. Set a JDK_HOME override setting on a per-project basis. In the Compilation tab of the Clover section of your Project properties, enter a value such as "C:\j2sdk1.4.2" into the JDK_HOME override field.
2. Globally choose a JDK instead of a JRE as your Default JRE.
 - In Eclipse, choose "Windows | Preferences" then select "Java / Installed JREs".
 - Click "Add..." and enter the path to your JDK in the "JRE home directory" field. For example, enter "C:\j2sdk1.4.2".
 - Choose a name (such as "JDK1.4.2") and click "OK".
 - Ensure you have this JDK checked as the default build JRE.

Activating the Clover Eclipse plugin

The Clover Eclipse plugin can be activated in any Eclipse project when using the Java (JDT) perspective.

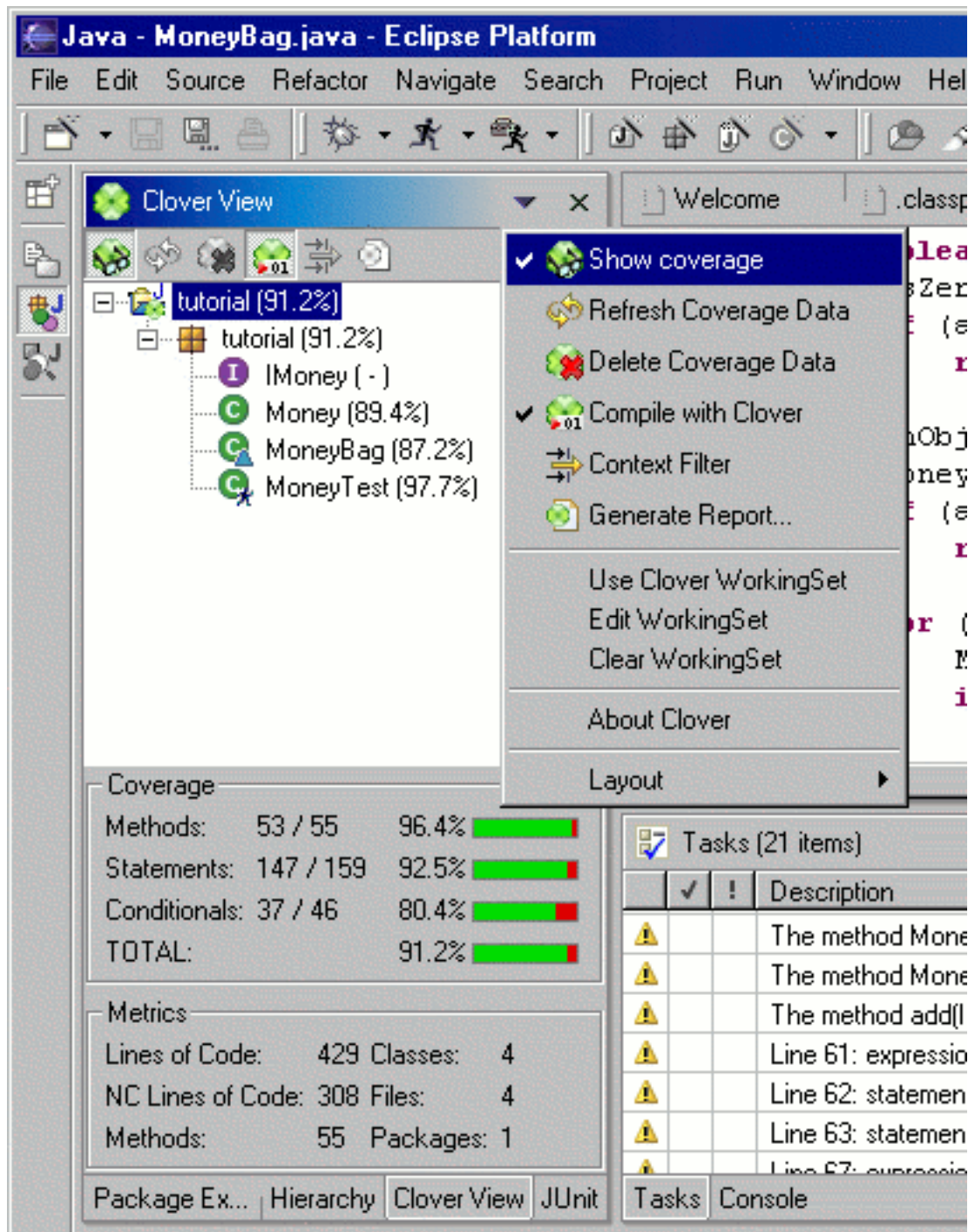
- Open up the project's properties, by using "Project | Properties" or right-clicking on the project in the Package Explorer.
- Select the "Clover" page.
- Toggle the "Enable Clover plugin in the project" checkbox. You can leave the Clover options at the defaults for now.
- When you hit OK, the **Clover Viewer** tool should appear on your workbench.



Clover Properties

The Clover Viewer tool

The Clover Viewer tool allows you to control Clover's instrumentation of your Java projects, and shows you the coverage statistics for each project. The tree shows the package and class coverage information for each project. Summary statistics are displayed below the tree for the selected project/package/class.



Clover Viewer

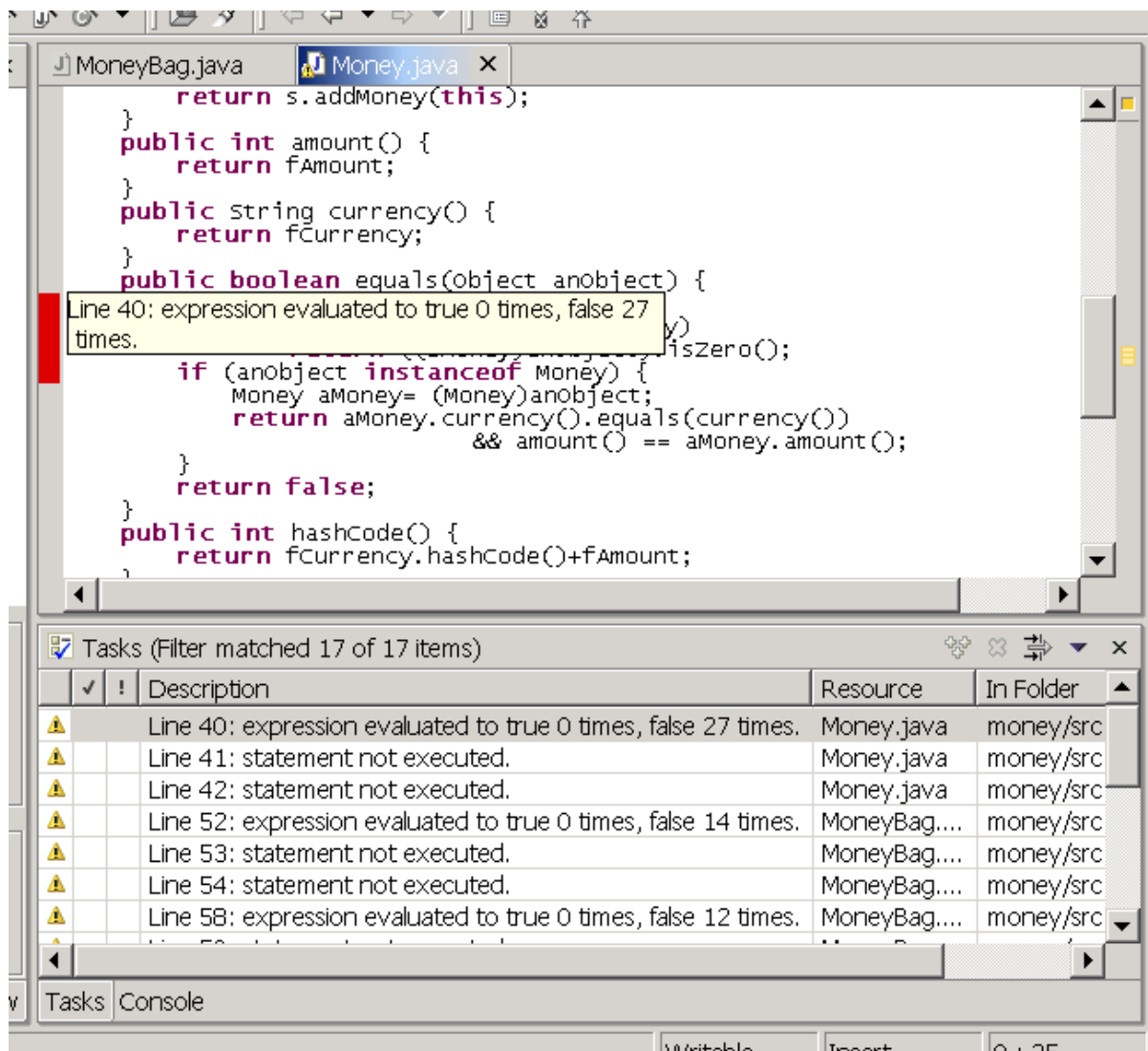
The Clover Viewer is automatically added to the workbench when you enable Clover for your project. If the viewer is closed, you can open it again using "Window | Show View | Other..." and selecting "Clover | Clover View".

The viewer allows the following actions:

- **Show coverage.** Toggles the display of coverage information in the Java editors and in the Tasks list.
- **Refresh Coverage Data.** Re-loads from disk the Clover coverage data for the selected project.
- **Delete Coverage Data.** Deletes the recorded coverage data for the selected project.
- **Compile with Clover.** Toggles the use of Clover instrumentation when Eclipse compiles the selected Java project.
- **Context Filter...** Allows you to specify what coverage contexts are shown in the Java Editor.
- **Generate Report...** Launches the report generation wizard that will take you through the steps required to generate a Pdf, Html or Xml. report.
- **Use Clover WorkingSet.** Toggles the use of the Clover WorkingSet. This limits the files Clover will consider when instrumenting and when showing coverage data. This is particularly useful for large projects.
- **Edit WorkingSet.** Brings up a dialog for editing the Clover WorkingSet.
- **Clear WorkingSet.** Empties the Clover WorkingSet. This means Clover will not consider any files while "Use Clover WorkingSet" is enabled.

Viewing Coverage Results

The Clover Eclipse plugin allows you to view Clover coverage data within the Eclipse IDE. This may include coverage data created using Clover external to the Eclipse IDE, or coverage data generated by the Clover Plugin internal to Eclipse.

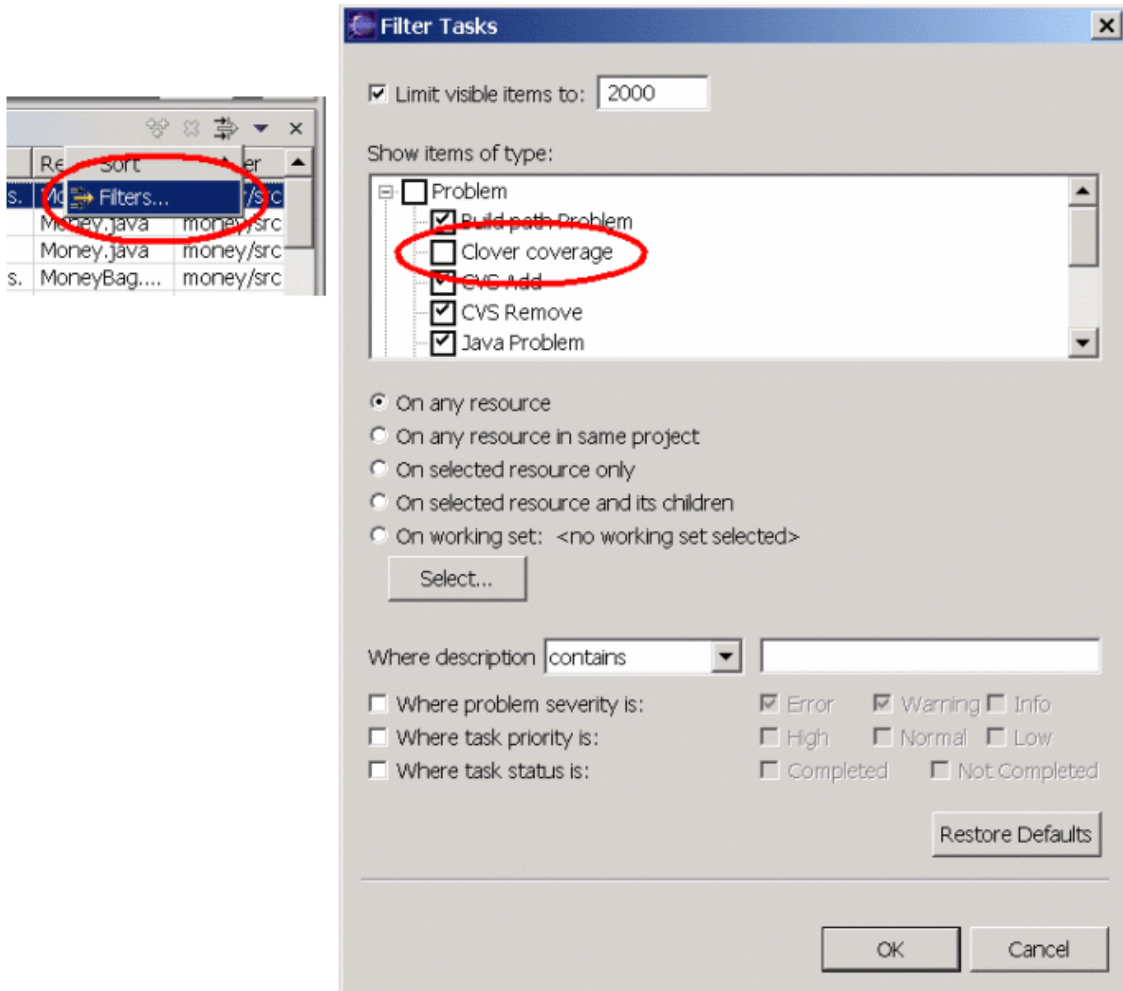


Markers

The Clover plugin shows coverage data in three ways:

- As a marker in the overview bar (right-hand side). This marker has a tooltip indicating the coverage problem.
- As a marker in the vertical ruler (left-hand side). This marker has a tooltip indicating the coverage problem.
- As a warning item in Eclipse Tasks list. If you do not want coverage warnings to appear in the task list, you can filter them out using the Tasks list filter preferences. Note that

warnings associated with a file will appear in the Tasks list only for those files that are currently opened by an editor.



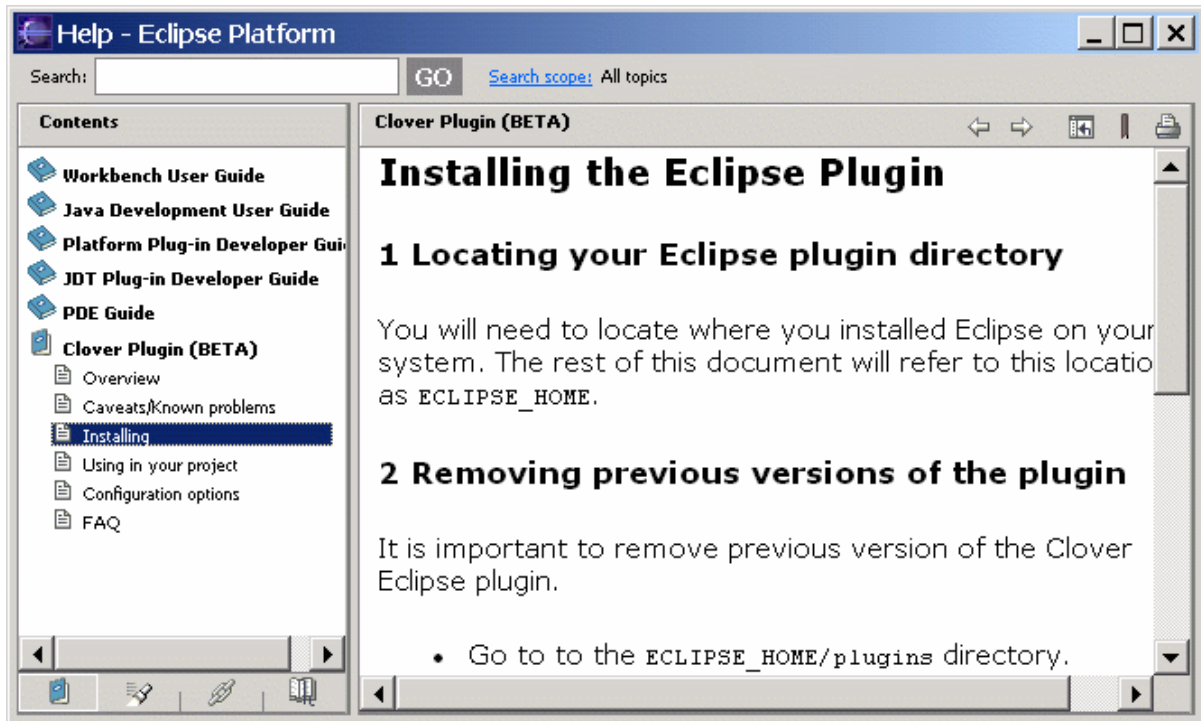
Marker Filter

Instrumenting your code

You can use the Clover Eclipse plugin to instrument the Java source in your project each time it is built. This option is activated on a per-project basis by toggling the "Compile with Clover" button in the Clover Viewer.

Online help

The Clover Eclipse plugin includes help documentation integrated into the Eclipse help system.



Help

Deactivating the Clover Eclipse plugin

You can disable Clover instrumentation using the "Compile with Clover" button, and you can prevent coverage information from being displayed in the Java editors by toggling the "Show Coverage" button.

But if you want to completely de-activate Clover support in a project, then un-check "Enable Clover plugin in the project" on the Clover page of the project's properties dialog.

Disabling Clover in this way will require a full rebuild of that project. If this is undesirable then you can simply toggle the "Compile with Clover" and "Show Coverage" options.

4.2.5. Configuration options

The Clover Eclipse plugin's configuration can be accessed in two places; a) from the "Clover" page of a project's properties dialog (Project | Properties), and b) from the "Clover"

page of the workspace preferences (Window | Preferences).

Project Properties - Instrumentation Options

These options control how Clover instrumentation works when "Compile with Clover" is selected.

Initstring

This controls where the Clover plugin stores (and looks for) the coverage database. You may want to specify a "User specified" value if you want to view Clover coverage data generated external to the Eclipse IDE.

Flush Policy

The Flush Policy controls how Clover writes coverage data to disk at runtime. "Directed" is the default and means coverage data is written to disk when the JVM exists. "Interval" allows you to specify that coverage data should be written out at regular intervals. See [Flush Policies](#).

Filtering Includes/Excludes

If you do not want all of your source instrumented, then you can control which this using these two Ant patternsets. For example, you may prevent instrumentation of files in the "remote" package using an "Excludes" value of `**/remote/*.java`.

Project Properties - Compilation Options

These options allow you to specify how Clover will compile your instrumented files.

Fork compiler into separate JVM

If enabled, Clover will launch a separate JVM to compile your instrumented files.

Heap size of compiler JVM

The heap size of the forked JVM (in MB). Leave blank to use the default.

Project Properties - Filter Options

These options allow you to define custom coverage filters.

Name

The name for this context. Must be unique, a valid java identifier and not be one of the reserved context names

Type

The type for this context. A method context type matches against method signatures, and a statement context type against statement signatures.

Regexp

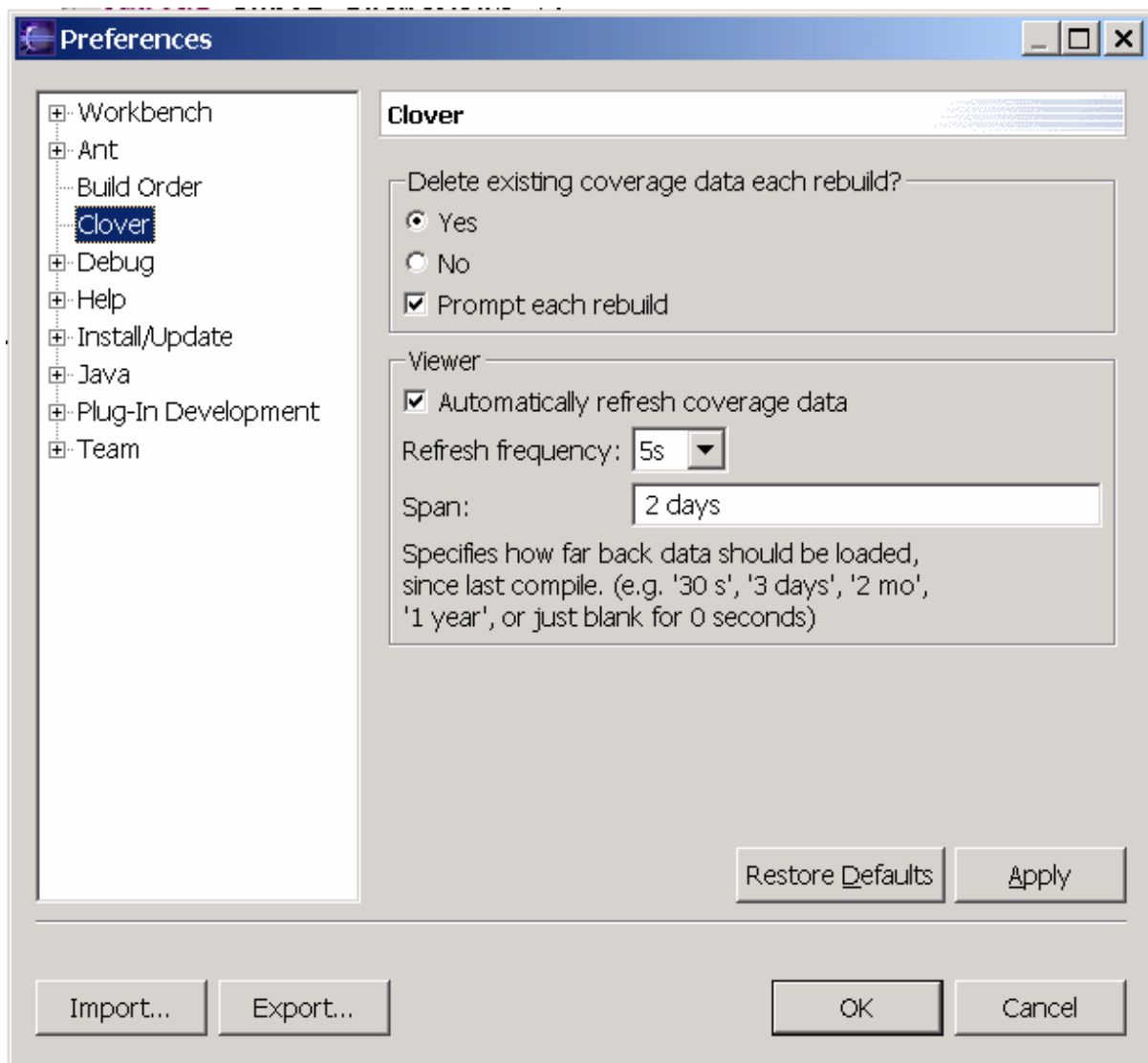
A Perl 5 Regexp that defines the context. This regexp should match the

signatures of the method/statement you wish to include in this context. Note that when signatures are tested against this regexp, whitespace is normalised and comments are ignored.

Note:

See [Coverage Contexts](#) for more information.

Clover Preferences



Clover Preferences

Deleting existing coverage data

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.

Automatically refresh coverage data

If enabled, the plugin will check for updated coverage data at the frequency given below. If it is not enabled, then you will need to use the "Refresh Coverage Data" button to see newer coverage data.

Span

The span attribute allows you to control which coverage recordings are merged to form a current coverage report. For more information, see [Using spans](#)

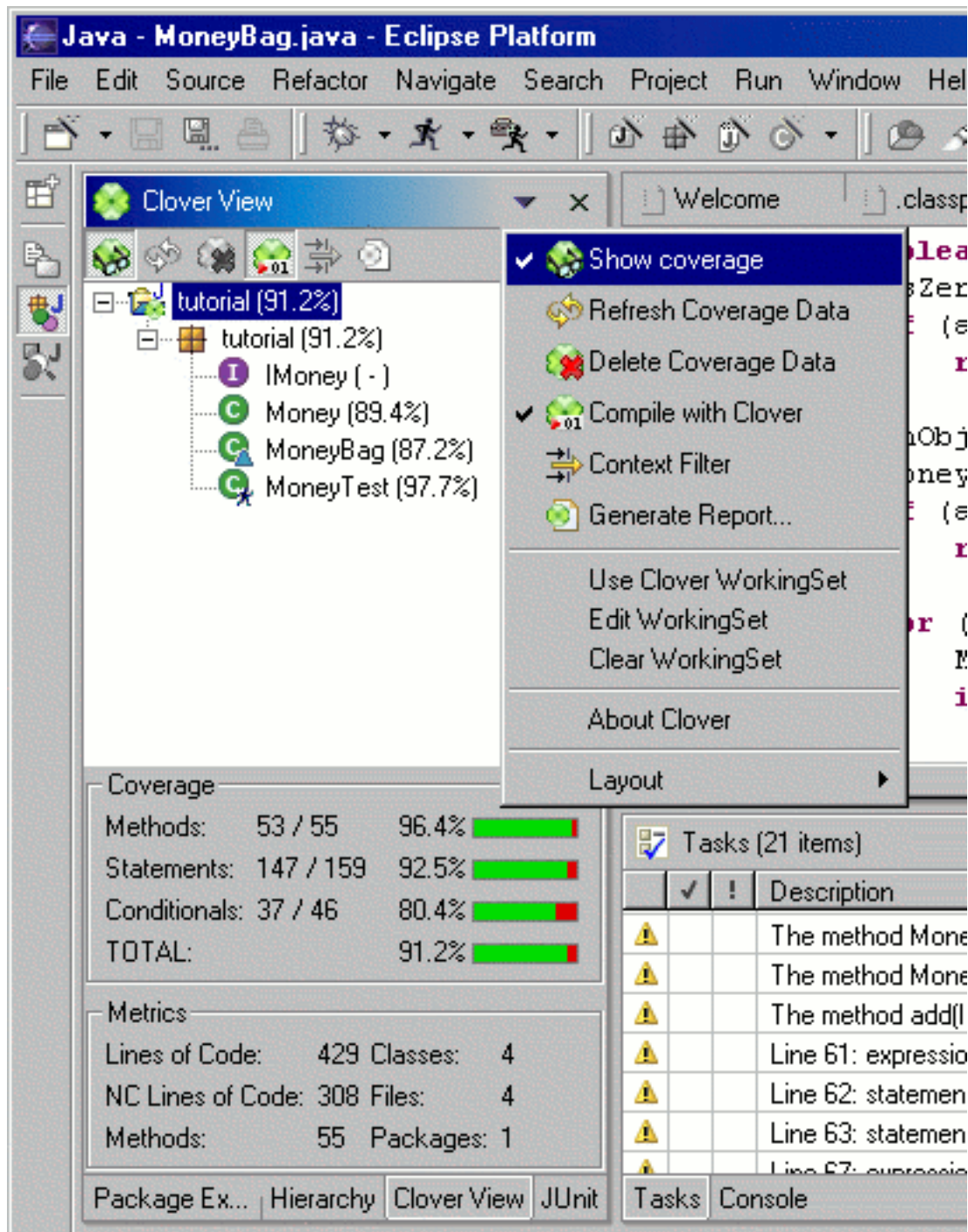
4.2.6. Large Projects

It is common for developers to work in different ways when working on an extremely large project compared to small/medium sized projects. For example, doing a complete rebuild then running all the unit tests can take hours for some projects. For this reason, some developers may want to focus only on a few files or packages at a time. The Clover Eclipse plugin has a *Working-Set* mode to assist in this style of development.

The Clover Working-Set

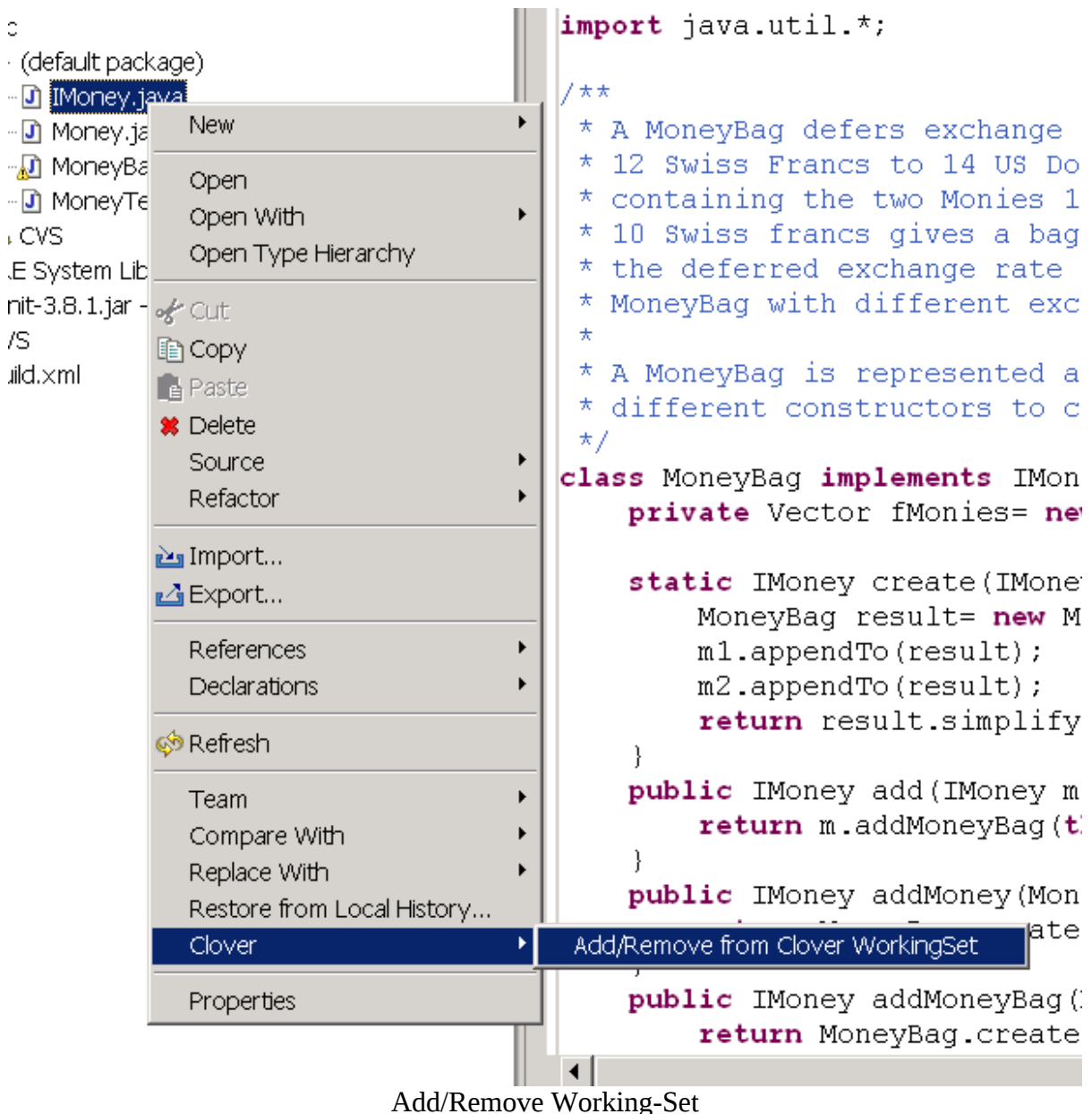
Eclipse has an inbuilt concept of a Working-Set, which allows you to specify a subset of the Workspace that you want to consider. Clover can use one of these Working-Sets to:

- Limit the files that will be instrumented by the Clover plugin.
- Filter the files/packages/directories for which Clover will display coverage information. This includes filtering the coverage statistics. For example, Clover will report 100% coverage if just all the files *in the Working-Set* are covered.



Clover Viewer

The Clover Working-Set can be manipulated via the tool menu in the Clover Viewer and by the context menu on files, packages and projects.



Enabling the Clover Working-Set is a Workspace-wide setting; it affects all projects in Eclipse. If you have an "Excludes" setting on a project (in the Clover section of the Project Properties), then those files are excluded *in addition* to those excluded by the Working Set. Similarly, if you have an "Includes" setting, then only those files that are included in both this setting *and* the Working-Set are Instrumented by Clover.

4.2.7. Working with custom filters.

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 "Clover | Filters".
- Select the **Add** button 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.
- When a filter has been newly created or edited, a (*) will be displayed next to its name. This indicates that the filter is currently 'unavailable' for use. To make this new filter available, you will need to run a clean build of your project. Once available, 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.

4.2.8. FAQ

Q: Why, when doing a build, do I get an error dialog with the message "*Clover build error. Error running javac.exe compiler*"? Why do I need to configure a JDK instead of a JRE for my project?

In order to compile your instrumented source code, the Clover Plugin needs to find the "javac" command that comes with the JDK. The plugin does not use Eclipse's inbuild Java Builder.

Q: I've run my tests, but coverage information does not show in Eclipse.

You may need to press the Refresh Button in the Clover tool window.

Q: Why can I only see coverage data for the last test case I executed? Why does my coverage information disappear each time I compile a file?

By default, Clover will display the coverage information gathered since your last compile. 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).

4.3. Clover IDEA 3 Plugin UserGuide

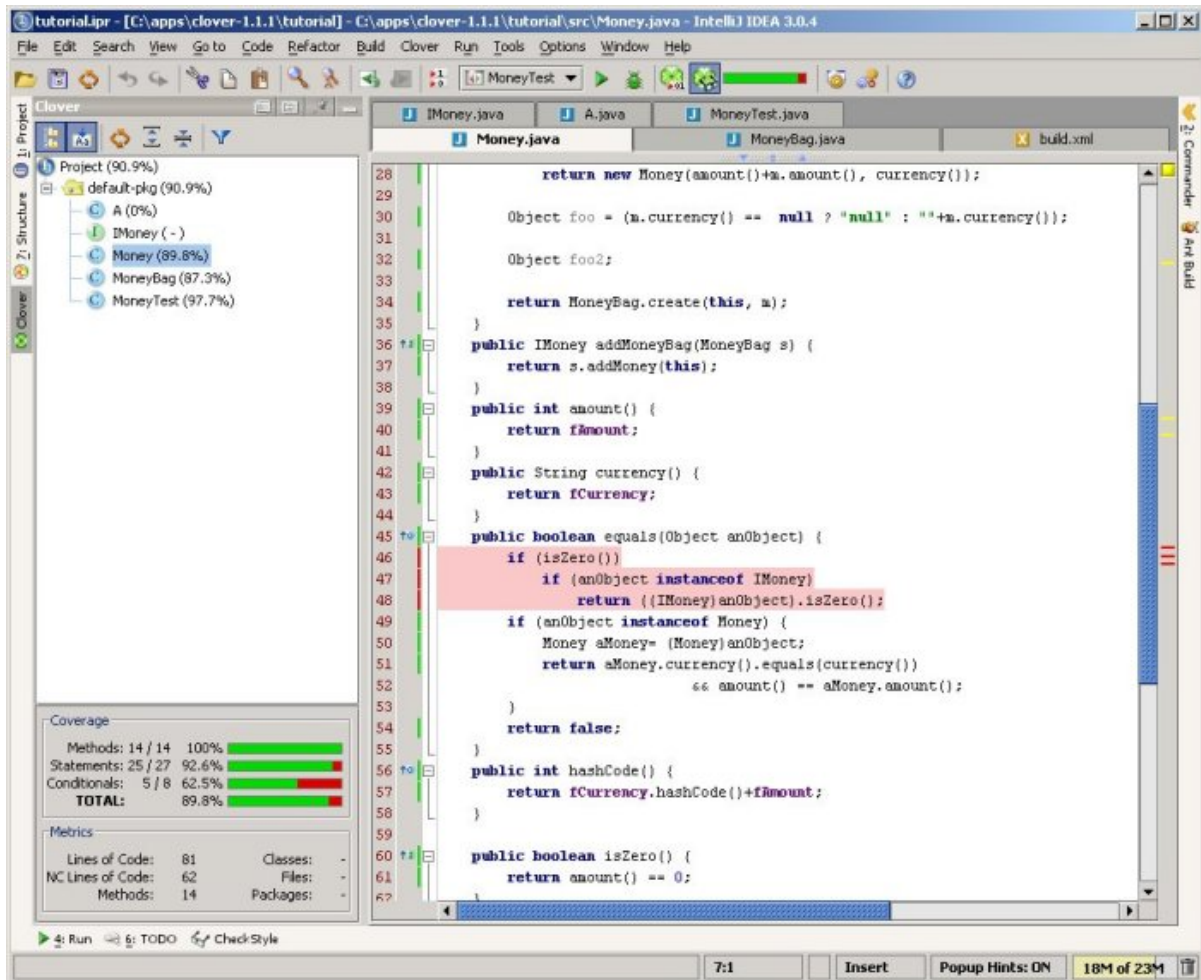
Plugin Version 0.8 Beta

Note:

The Clover IDEA 3 Plugin is currently a **beta version**. The plugin has been tested with IntelliJ IDEA versions 3.0.x.

4.3.1. Overview

The Clover IDEA Plugin allows you to instrument your Java code easily from within the IntelliJ IDEA 3.x Java IDE, and then view your coverage results inside IDEA.



Clover IDEA plugin

4.3.2. Installing the plugin

To install the plugin:

1. shutdown any running instances of IDEA
2. remove any previous versions of the the clover plugin jar from `IDEA_HOME/plugins`.
3. copy `CLOVER_HOME/lib/cloverIdeaPlugin.jar` into the `IDEA_HOME/plugins` directory, and restart IDEA.

4.3.3. Using the plugin

Enabling the Clover Plugin for your project

Add `cloverIdeaPlugin.jar` to your project classpath:

- Open the project properties "File | Project Properties".
- In the "Paths" section, select the "Classpath" tab. Remove any old clover jars and add a reference to `cloverIdeaPlugin.jar` (you must reference the `cloverIdeaPlugin.jar` that you installed in `IDEA_HOME/plugins`).

(The `cloverIdeaPlugin.jar` needs to be in the classpath because it is needed at runtime when you are running your unit tests. It is also needed when you are compiling with Clover)

Building your Project with Clover

Clover works by pre-processing your Java files before they are compiled. This means that when you want to measure coverage with Clover, you cannot use the standard IDEA "Rebuild Project" or "Make Project" functionality. Instead, you need to use either "Clover | Rebuild Project with Clover" (for a full rebuild), or "Clover | Make Project with Clover" (builds only modified files). The "Make Project with Clover" action can also be launched with the toolbar button

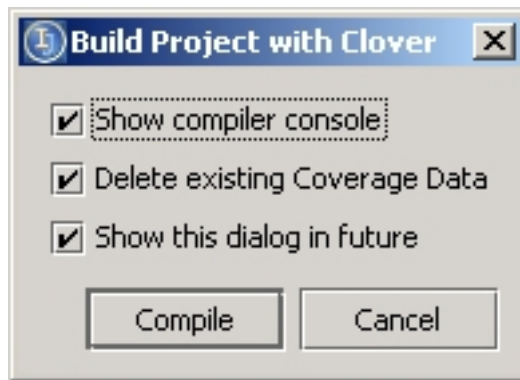


Clover Compile Button

Note:

Clover collects code coverage by instrumenting a copy of your Java files before they are compiled. If you "Build with Clover", then re-compile some of your files normally with IDEA, those files will no longer be instrumented; and coverage will not be collected for them until you do another "Build with Clover".

Build Options



build dialog

Show Compiler Console

Check this box to see output from the Clover build process.

Delete existing Coverage Data

(option only appears on a full rebuild) This option allows you to delete existing coverage data and registry information before the build.

Show this dialog in future

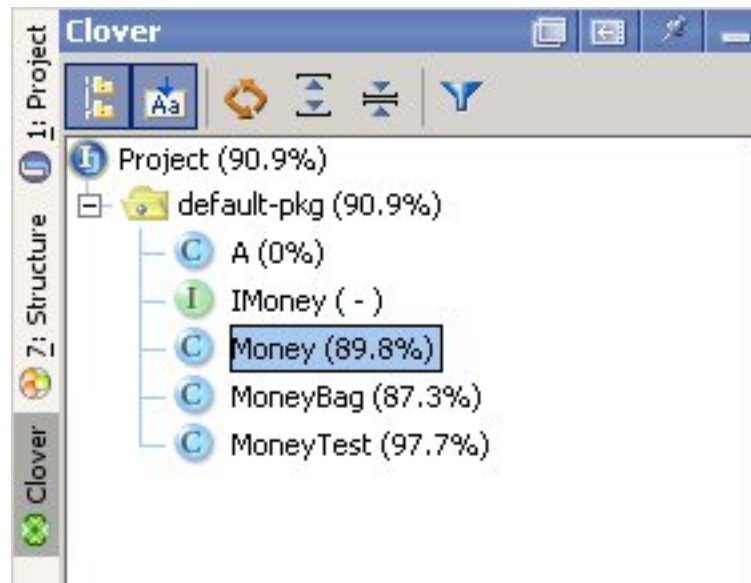
Uncheck this box if you don't want this dialog appearing for future builds. You can enable the dialog again via the Clover Project Properties screen.

Viewing Coverage Results

Once you have instrumented your code (see [Building your Project with Clover](#)), each time you run your application or a unit-test Clover will record the code coverage. Once the application or unit-test exits, the coverage information is available for viewing using IDEA.

The coverage information can be browsed using the Clover Tool Window. This presents the data in a similar way to the existing Clover GUI Viewer.

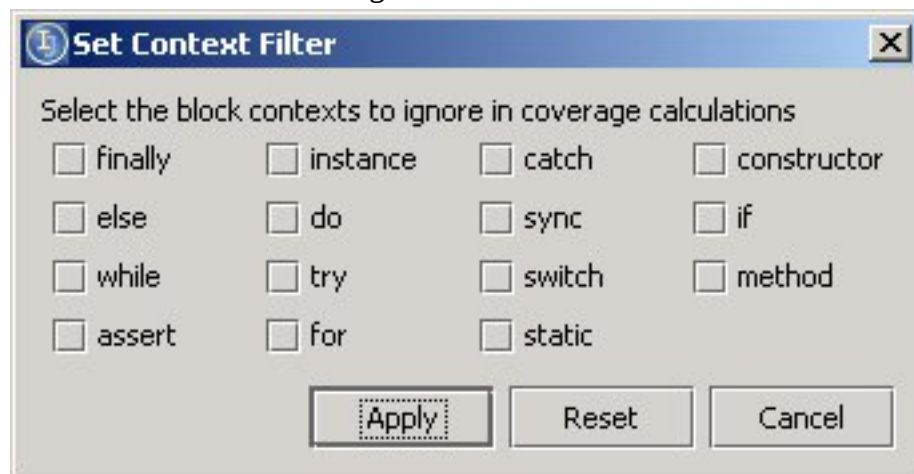
The top pane of the Tool Window contains a class browser with inline coverage information:



Clover class browser

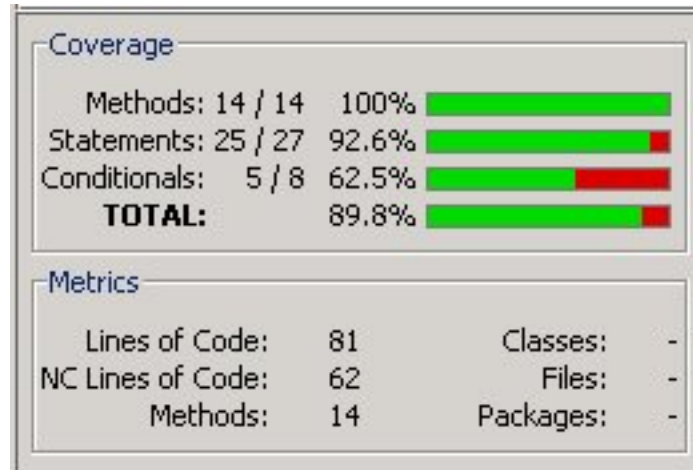
The tool bar at the top of the browser contains the following buttons

- **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.
- **Refresh.** Reloads coverage data.
- **Expand All.** Expand all nodes in the browser.
- **Collapse All.** Collapse all nodes in the browser.
- **Set Context Filter.** Launches a dialog to set the context filter:



Context Filter Dialog

The bottom pane of the Tool Window contains Coverage and other Metrics information for the currently selected node in the browser:



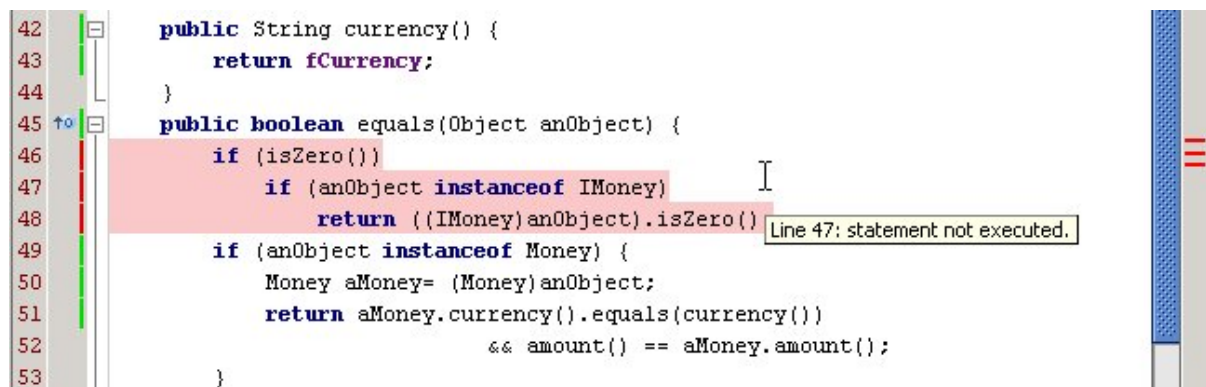
Coverage info view

In addition, the plugin can annotate the Java code with the coverage information. This can be turned on using the "Clover | Show Coverage" menu option, or by pressing the Show Coverage



view coverage button

toolbar button.



editor pane with overlaid coverage information

Note:

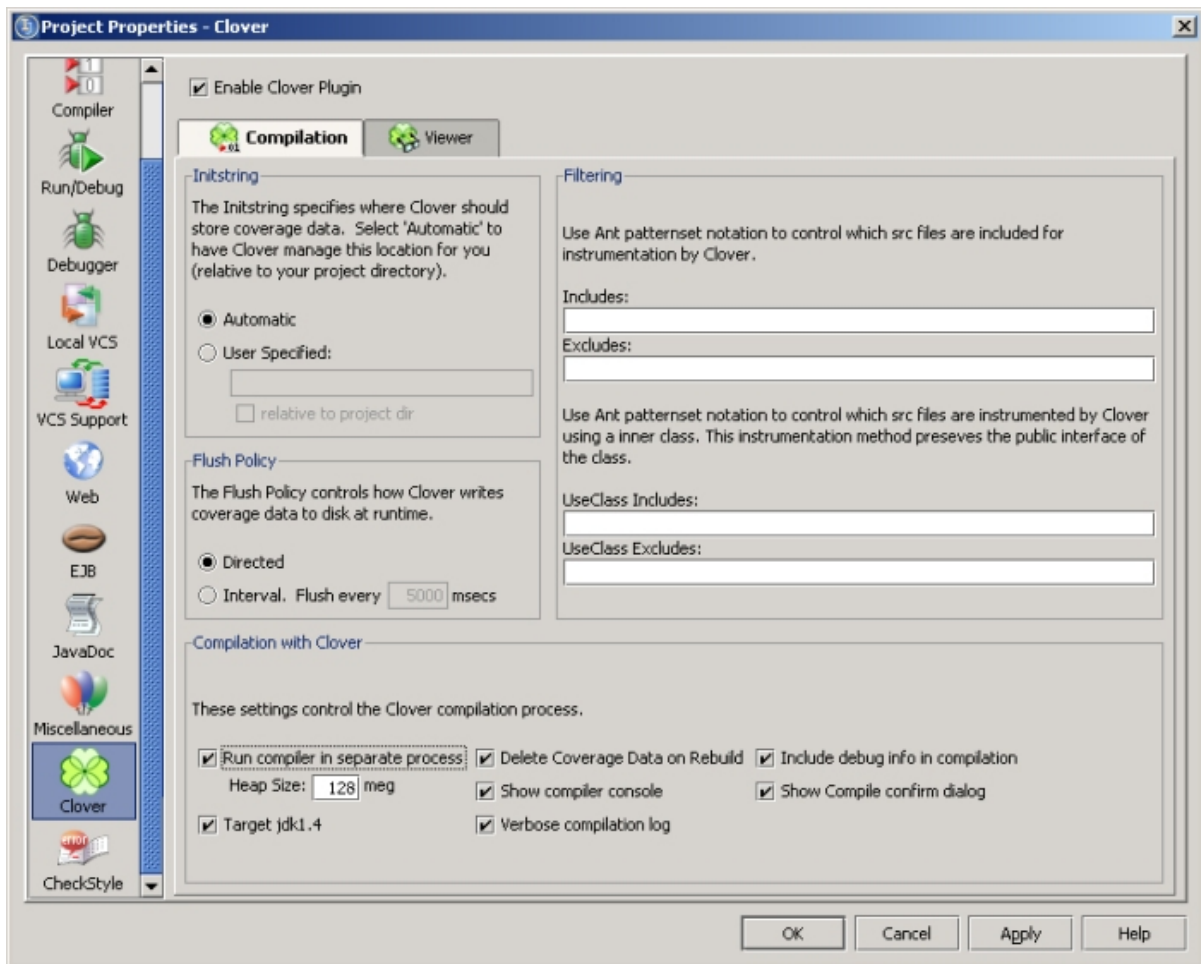
If you do not have "Auto Coverage Refresh" enabled, you will need to press the Refresh Button in the Clover Tool Window to see the updated coverage information.

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.

4.3.4. Configuration Options

Configuration options for Clover are accessible on the Clover panel of the Project Properties dialog.

Compilation options



Compiler configuration screen

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](#).

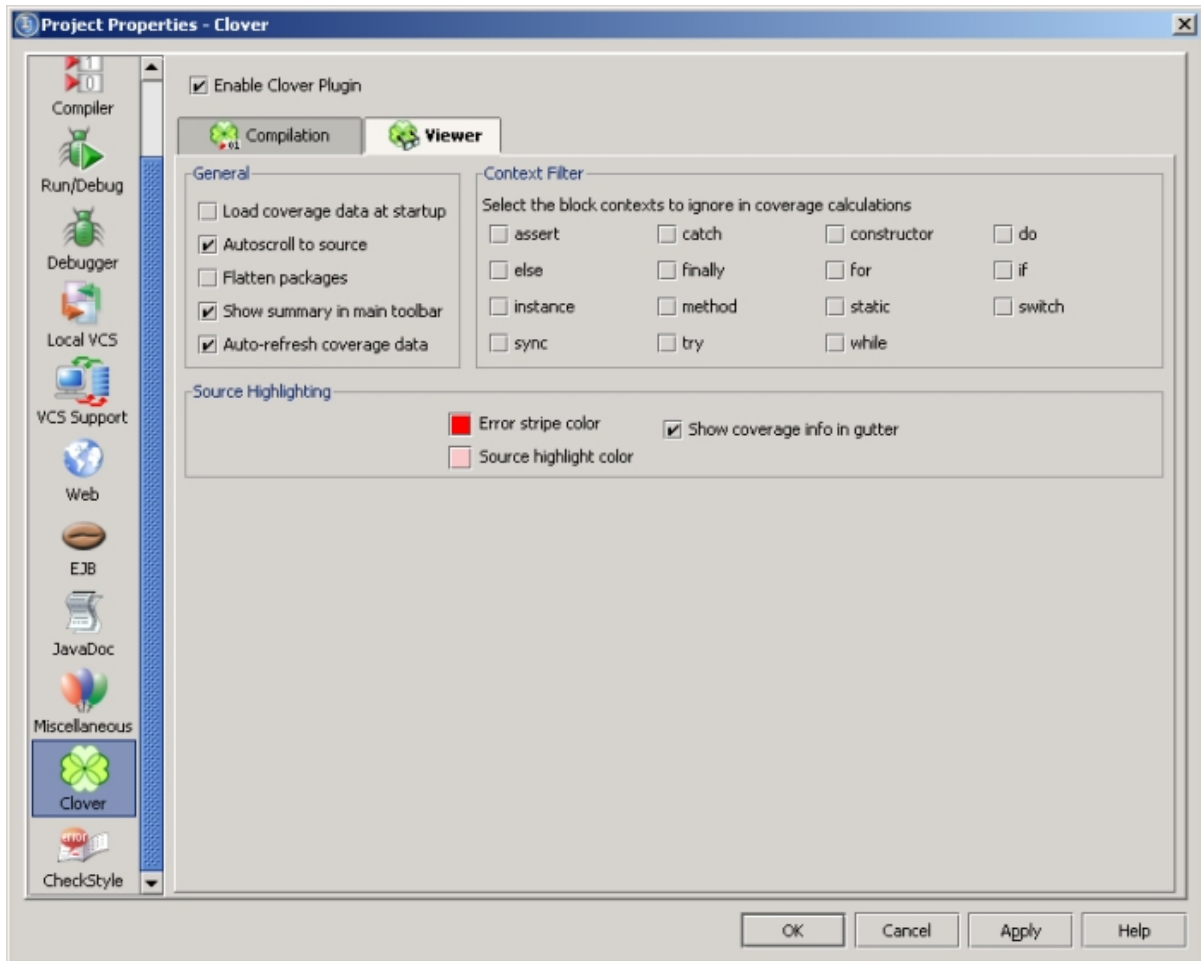
Compilation with Clover

These settings control how the Java compiler operates when building your project with Clover.

Filtering

Allows you to specify a comma separated list of set of Ant Patternsets that describe which files to include and exclude in instrumentation. These options are the same as those described in the [<clover-setup>](#) task.

Viewer options



Viewer configuration screen

General

Controls the operation of the Clover tool window on the left-hand side of the IDE.

Context Filters

Allows you to specify contexts to *ignore* when viewing coverage information.

Source Highlighting

Allows you to specify colors used when displaying source level coverage information.

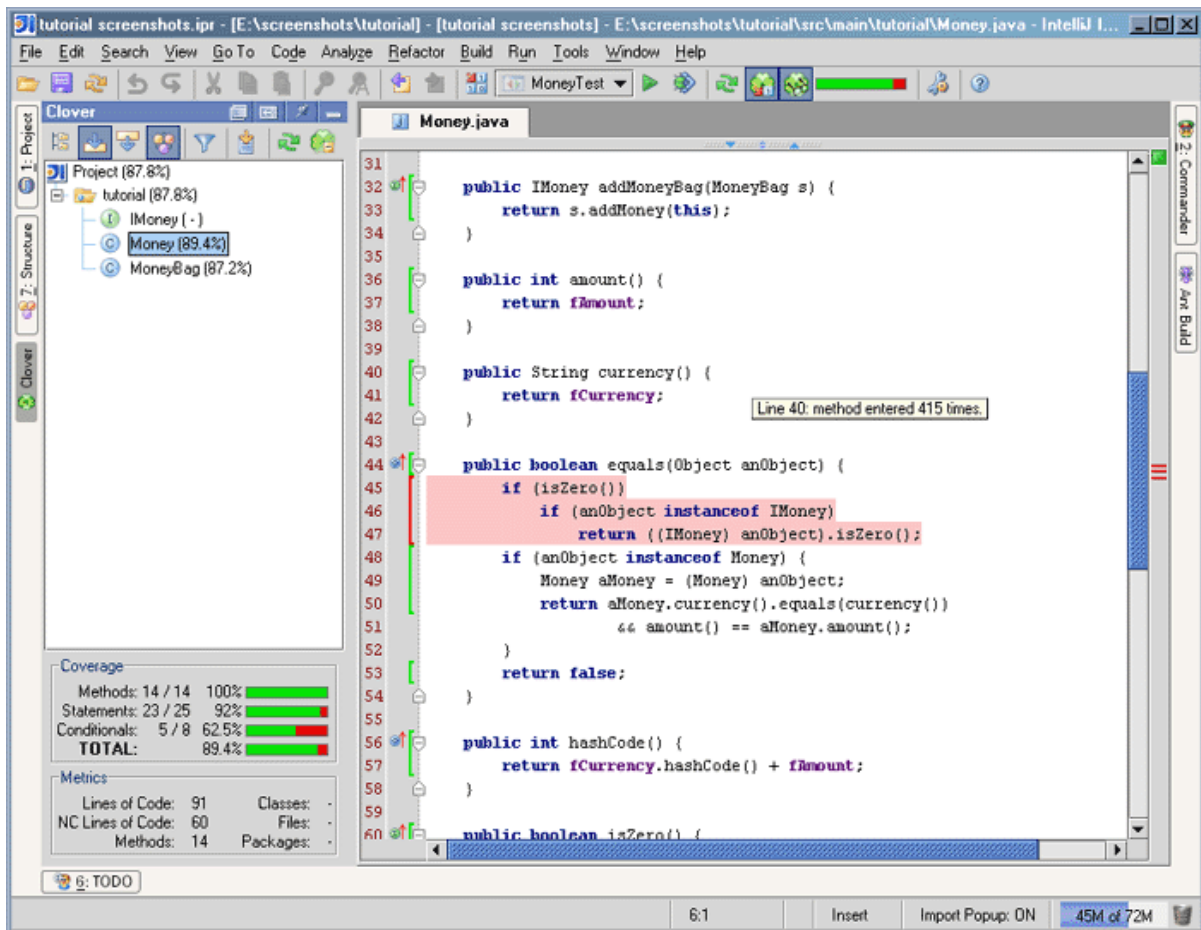
4.4. Clover IDEA 4 Plugin UserGuide

Plugin Version 1.0.7

System Requirements: IntelliJ IDEA 4.5.4

4.4.1. 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.



Clover IDEA plugin

4.4.2. Installing

If you have downloaded the Clover IDEA Plugin package from <http://www.cenqua.com/>, you can install the plugin manually as follows:

1. shutdown any running instances of IDEA
2. remove any previous versions of the the clover plugin jar from `IDEA_HOME/config/plugins` OR `IDEA_HOME/plugins`.
3. copy `CLOVER_HOME/lib/clover-idea4.jar` into the `IDEA_HOME/config/plugins` directory, and restart IDEA.

Alternatively, if you have downloaded the plugin via the "File | Settings | IDE | Plugins" interface, the plugin will be available after a restart.

Note:

The plugin installation directory has changed. For IDEA 4.0.x it was `IDEA_HOME/plugins`. For 4.5.x it is now `IDEA_HOME/config/plugins`.

You will need a license to activate your plugin.

- Download your clover.license file from <http://www.cenqua.com/licenses.jspa>. Evaluation licenses are available free of charge.
- Place the clover.license file next to the clover-idea4.jar file in either the `IDEA_HOME/config/plugins` or `IDEA_HOME/plugins` directory.

4.4.3. Uninstalling

To uninstall the Clover IDEA Plugin:

1. shutdown any running instances of IDEA
2. delete the clover-idea4.jar file from its installation directory, either `IDEA_HOME/config/plugins` OR `IDEA_HOME/plugins`.
3. restart IDEA

Alternatively, you can uninstall the Clover IDEA Plugin via the "File | Settings | IDE | Plugins" interface. Just select the Clover IDEA Plugin from the list and click 'Uninstall Plugin'. The uninstall will take affect after you restart IDEA.

4.4.4. Configuring your project

Add the clover jar to your 'project' classpath.

- Open the project properties "File | Settings | Project".
- In the "Paths" section, select the "Libraries (Classpath)" tab. Remove any old clover jars and add a reference to `clover-idea4.jar` you must reference the `clover-idea4.jar` that you installed in `IDEA_HOME/config/plugins`).

4.4.5. Getting Started

This getting started guide will take you through the steps required to generate Clover

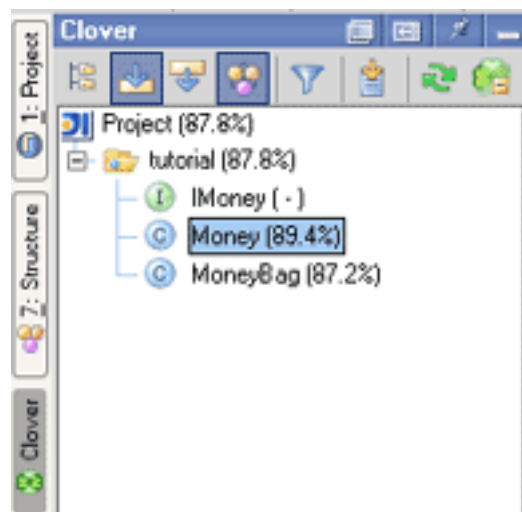
coverage for your project.

1. Ensure that you have added the clover plugin jar 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. View the project coverage data by selecting the toolbar item.

4.4.6. 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 Clover Tool Window. This presents the data in a similar way to the existing Clover GUI Viewer. The upper portion of the Tool Window contains a class browser with inline coverage information:

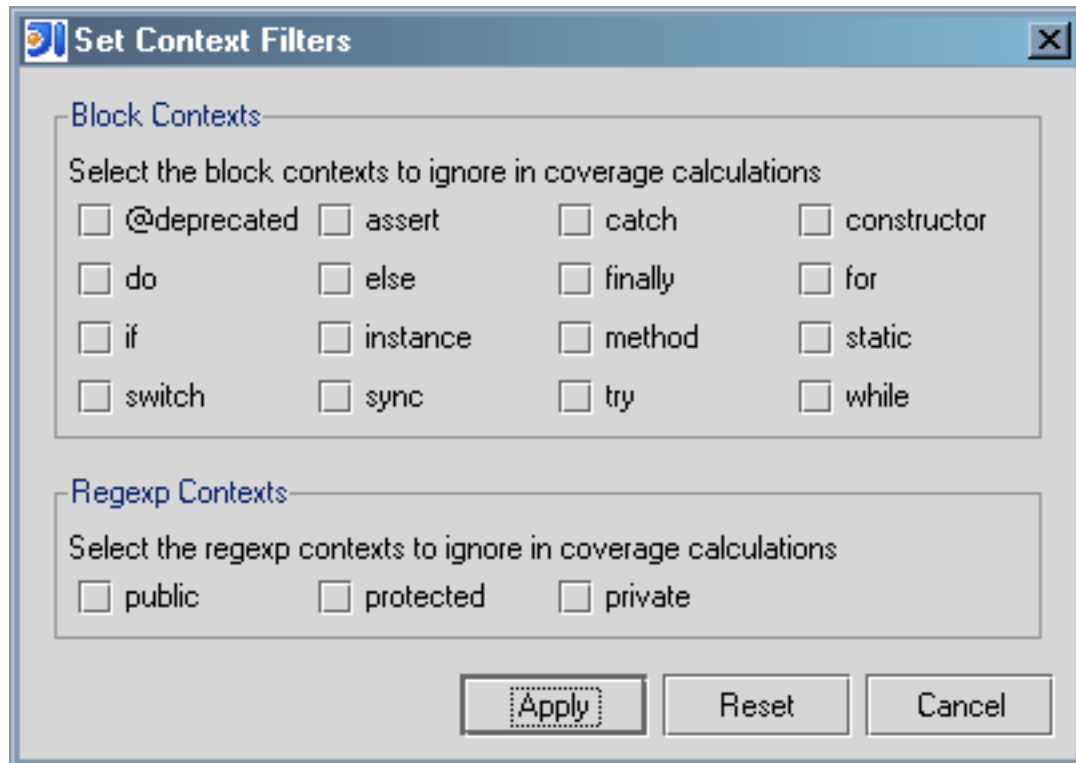


Clover class browser

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.

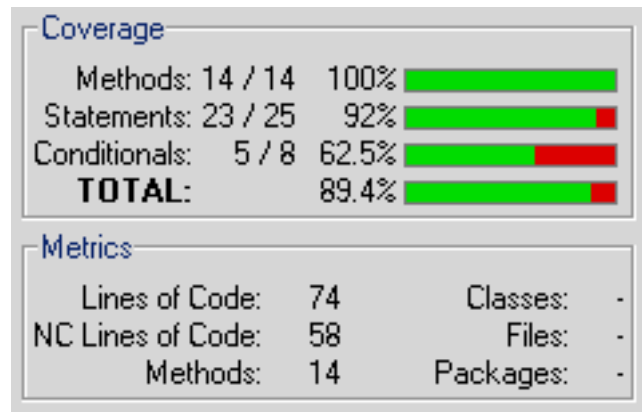
- **Show Coverage Summary.** With this selected, the Coverage metrics (see below) will be visible.
- **Set Context Filter.** Launches a dialog to set the context filter:



Context Filter Dialog

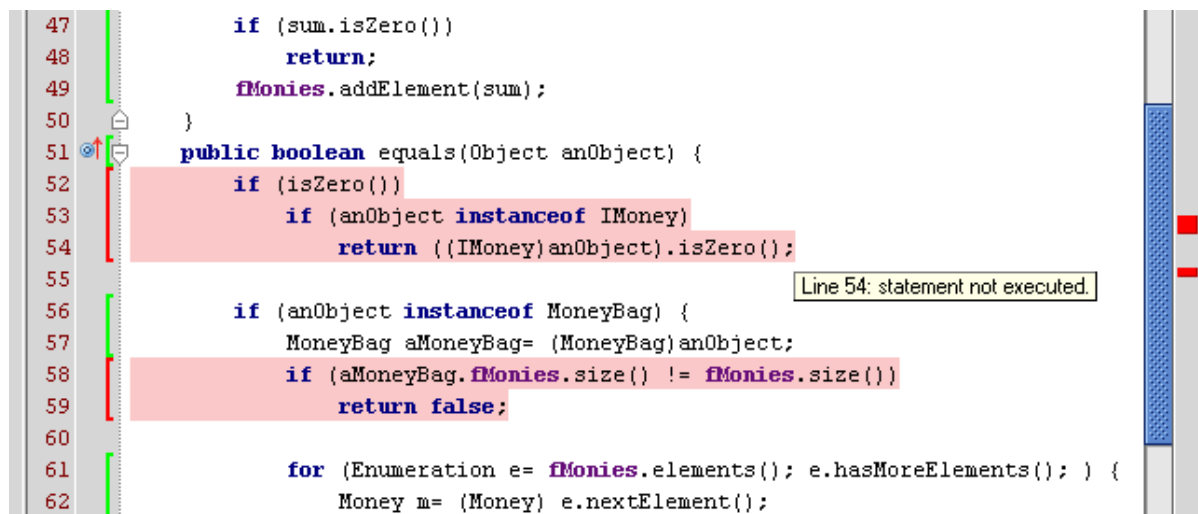
- **Generate Clover Report.** Launches the report generation wizard that will take you through the steps required to generate a Pdf, Html or XML report.
- **Refresh.** Reloads coverage data.
- **Delete.** Delete the current coverage database.

The lower portion of the Tool Window contains various Metrics for the currently selected node in the browser:



Coverage info view

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.



editor pane with overlaid coverage information

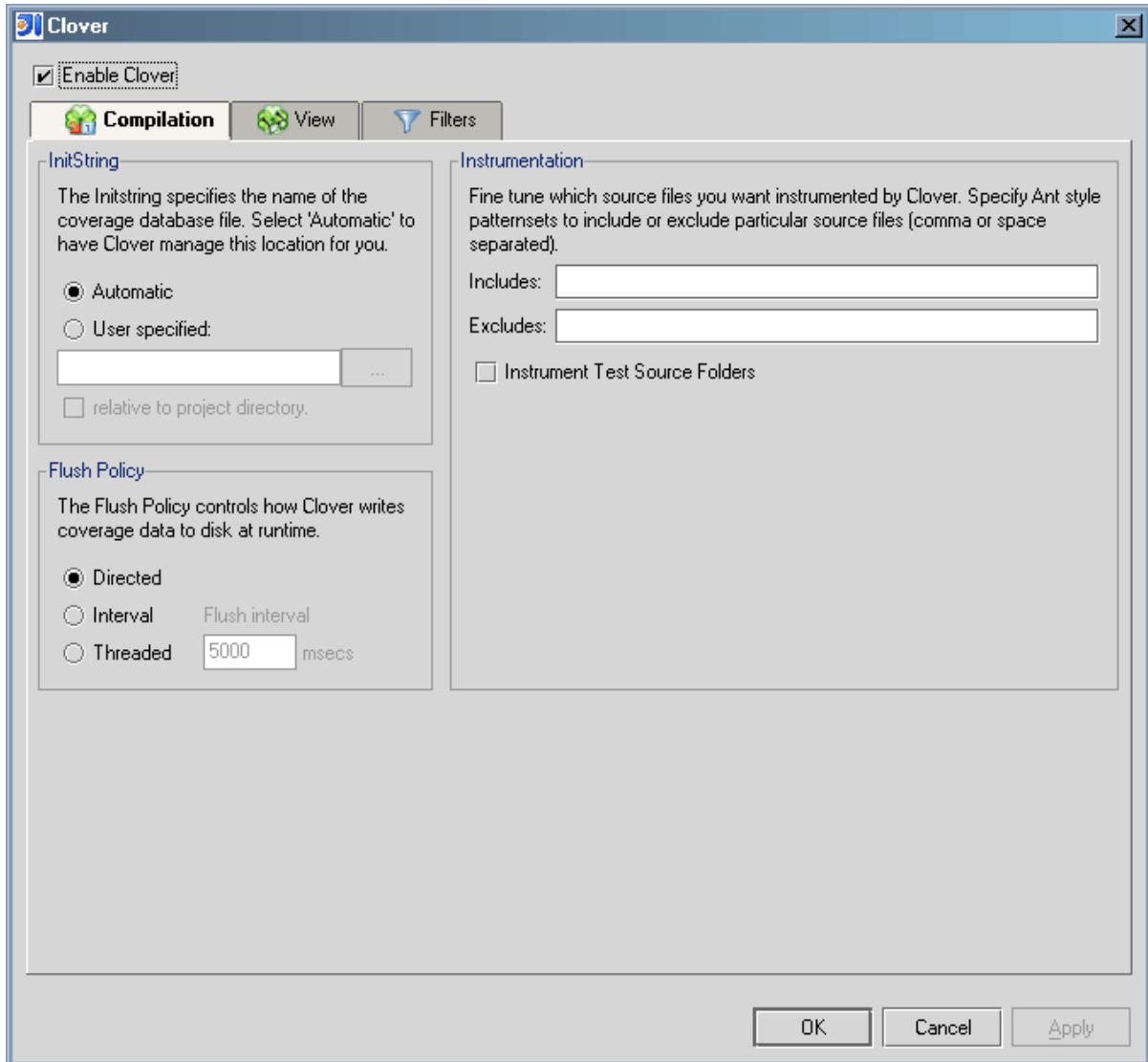
Note:

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.

4.4.7. 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:



Compilation Configuration Screen

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

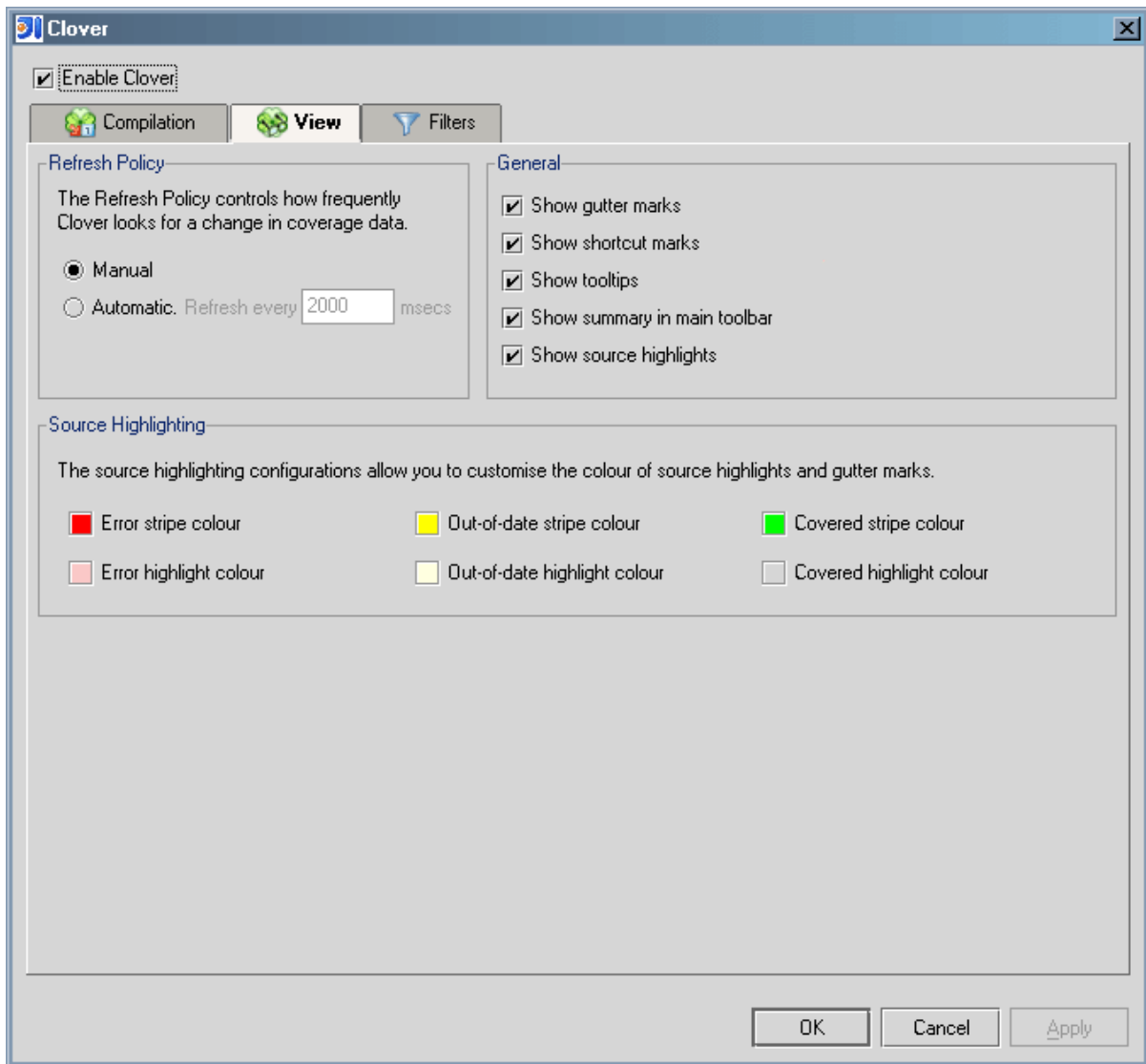
The Flush Policy controls how Clover writes coverage data to disk at runtime. See [Flush Policies](#).

Instrumentation

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.

Viewer options

The second Tab on the configuration panel provides viewing options;



Viewer Configuration Screen

Refresh Policy

The Refresh Policy controls how the Clover Plugin monitors the Coverage Database for new data. "Manual" is the default and means that you have to click button to refresh the coverage data. "Automatic" means that the Clover Plugin will periodically check for new coverage data for you.

General

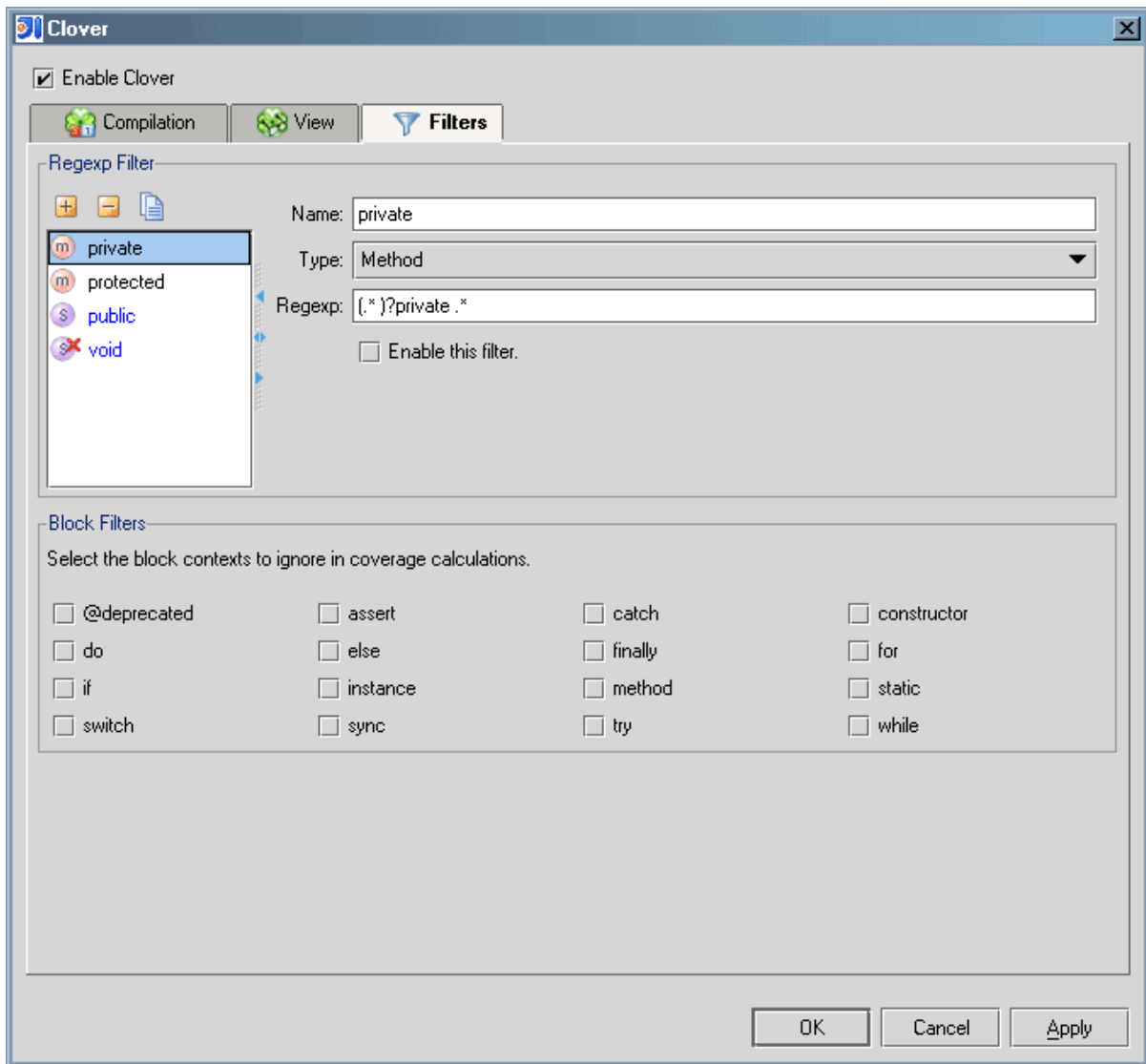
Allows you to customize where coverage data is displayed within the IntelliJ IDE. Gutter marks appear in the left hand gutter of the Java Source Editor. Source highlights appear directly over your source code. Shortcut marks appear in the right hand gutter and allow you to navigate directly to uncovered regions of code.

Source Highlighting

Allows you to fine tune the colours used Clover in its coverage reporting. The 'xxx highlight colour' is used for Source Highlights and the 'xxx stripe colour' is used for Gutter and Shortcut marks.

Filter Options

The third Tab on the configuration panel provides filter options;



Filter Configuration Screen

Regexp Filters

The regexp filters 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 available, you need to delete the existing coverage database using the button and rebuild your project/module.

Note:

See [Coverage Contexts](#) for more information.

Block Filters

Allows you to specify contexts to *ignore* when viewing coverage information. For example, selecting the *if* context will remove *if* body (not the conditional) from the coverage reports.

4.4.8. 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.

4.4.9. FAQ**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.

Q: When I compile with Clover instrumentation enabled, I get the following error: Error: line (31) package com_cenqua_clover does not exist

A: You need to add the clover-idea4.jar file 'each' of your modules classpaths.

Q: I have the Clover plugin enabled, but my files are not being instrumented.

A: As of v0.9.1, there is a toggle button on the main IDEA Toolbar for enabling and disabling instrumentation. You will need to ensure that this toggle button is enabled .

Q: When I compile my program I get the stack trace `java.lang.IllegalArgumentException: Prefix string too short`. Whats going on?

A: There is an known issue within IDEA that is triggered by the Clover integration. This issue relates to the size of the project name. If its less then 3 characters, then you see the exception that you are seeing. The only known 'workaround' for this issue is increasing the length of your project name.

Q: I have an enabled Regexp Filter that does not seem to be filtering.

A: Have you checked your regexp? It may be that your regexp is not matching the methods/statements that you expect. See [Coverage Contexts](#) for more information about Regexp Contexts.

A: Try reseting the current context filter and then re-enabling them. There is a known scenario where regexp filters are not being applied to the coverage data when they are enabled at activation time.

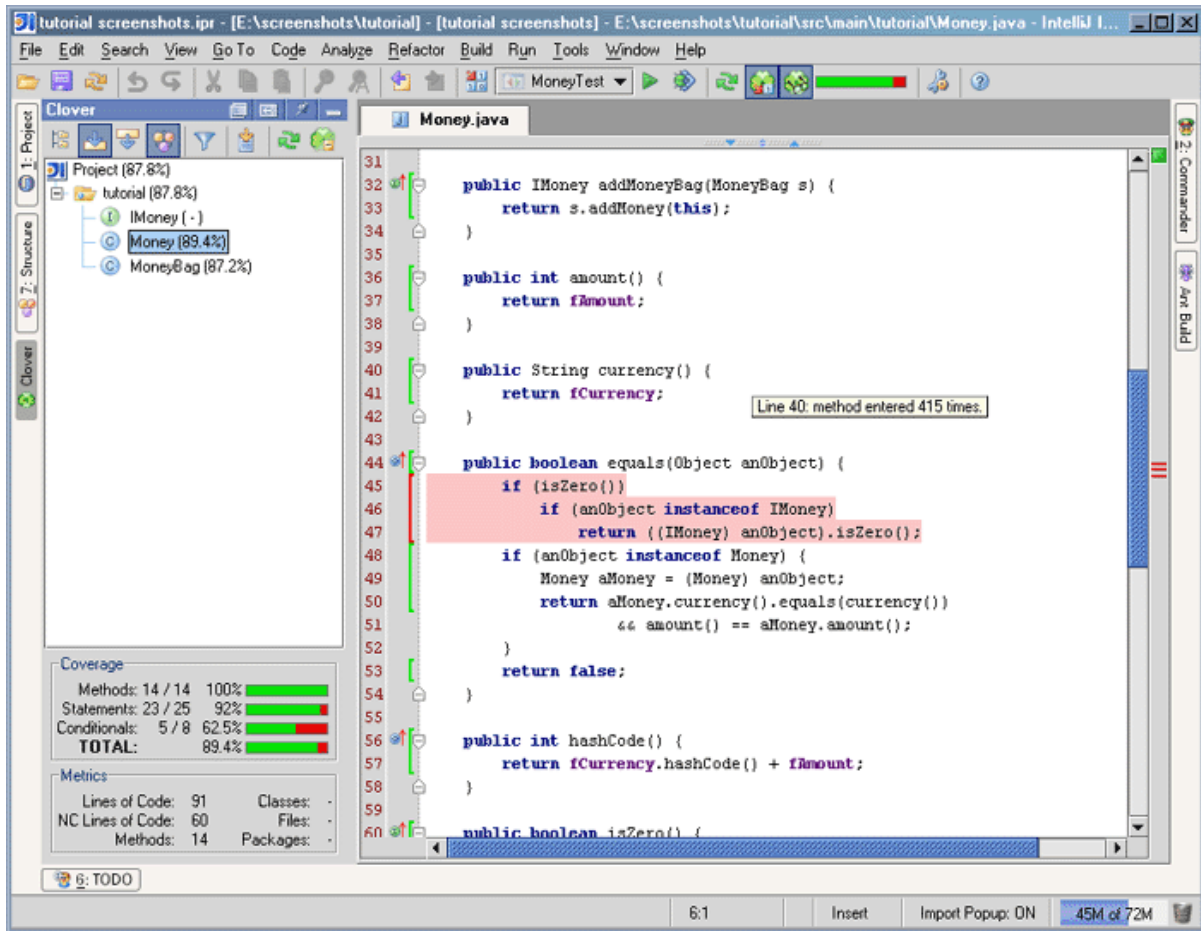
4.5. Clover IDEA5/6 Plugin UserGuide

Plugin versions: 5-1.0.8, 6-1.0.2

System Requirements: IntelliJ IDEA 5.x, 6.x

4.5.1. 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.



Clover IDEA plugin

4.5.2. Installing

If you have downloaded the Clover IDEA Plugin package from <http://www.cenqua.com/>, you can install the plugin manually as follows:

1. shutdown any running instances of IDEA
2. remove any previous versions of the the clover plugin jar from `IDEA_HOME/config/plugins` OR `IDEA_HOME/plugins`.
3. copy `CLOVER_HOME/lib/clover-idea5.jar` into the `IDEA_HOME/config/plugins` directory, and restart IDEA.

Alternatively, if you have downloaded the plugin via the "File | Settings | IDE | Plugins" interface, the plugin will be available after a restart.

You will need a license to activate your plugin.

- Download your clover.license file from <http://www.cenqua.com/licenses.jspa>. Evaluation licenses are available free of charge.
- Place the clover.license file next to the clover-idea5.jar file in either the IDEA_HOME/config/plugins or IDEA_HOME/plugins directory.

4.5.3. Uninstalling

To uninstall the Clover IDEA Plugin:

1. shutdown any running instances of IDEA
2. delete the clover-idea5.jar file from its installation directory, either IDEA_HOME/config/plugins OR IDEA_HOME/plugins.
3. restart IDEA

Alternatively, you can uninstall the Clover IDEA Plugin via the "File | Settings | IDE | Plugins" interface. Just select the Clover IDEA Plugin from the list and click 'Uninstall Plugin'. The uninstall will take affect after you restart IDEA.

4.5.4. Configuring your project

Add the clover jar to your 'project' classpath.

- Open the project properties "File | Settings | Project".
- In the "Paths" section, select the "Libraries (Classpath)" tab. Remove any old clover jars and add a reference to clover-idea5.jar you must reference the clover-idea5.jar that you installed in IDEA_HOME/config/plugins).

4.5.5. Getting Started

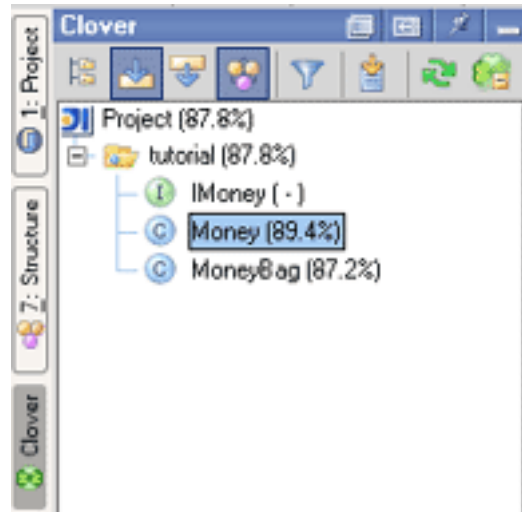
This getting started guide will take you through the steps required to generate Clover coverage for your project.

1. Ensure that you have added the clover plugin jar 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. View the project coverage data by selecting the toolbar item.

4.5.6. 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.

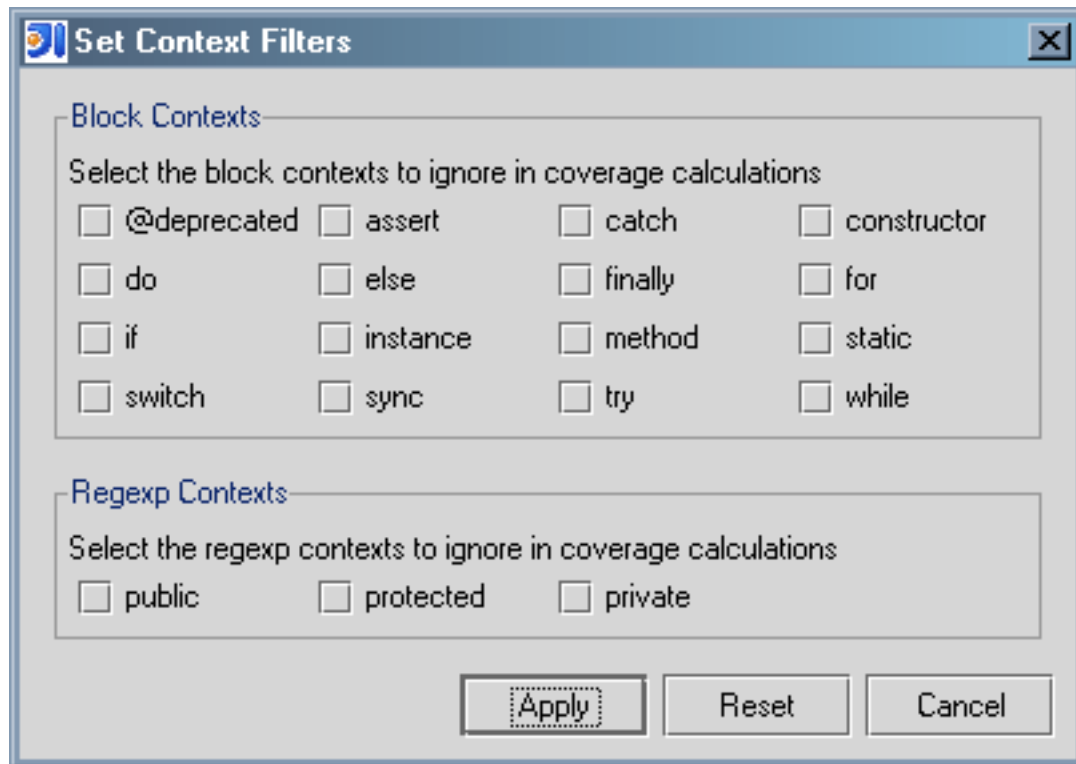
The coverage information can be browsed using the Clover Tool Window. This presents the data in a similar way to the existing Clover GUI Viewer. The upper portion of the Tool Window contains a class browser with inline coverage information:



Clover class browser

The tool bar at the top of the browser contains the following buttons:

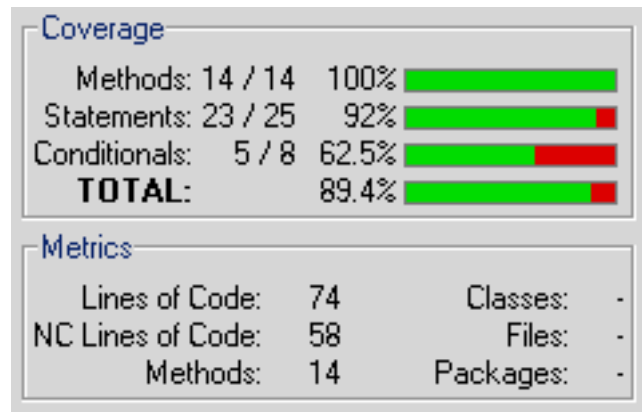
- **Flatten Packages.** With this selected, only concrete packages are shown in the browser.
- **Autoscroll to Source.** With this selected, a single click on a class in the browser will load the corresponding source file in an editor pane, with coverage info overlaid.
- **Autoscroll from Source.** With this selected, the coverage browser will track the currently active source file in the editor pane.
- **Show Coverage Summary.** With this selected, the Coverage metrics (see below) will be visible.
- **Set Context Filter.** Launches a dialog to set the context filter:



Context Filter Dialog

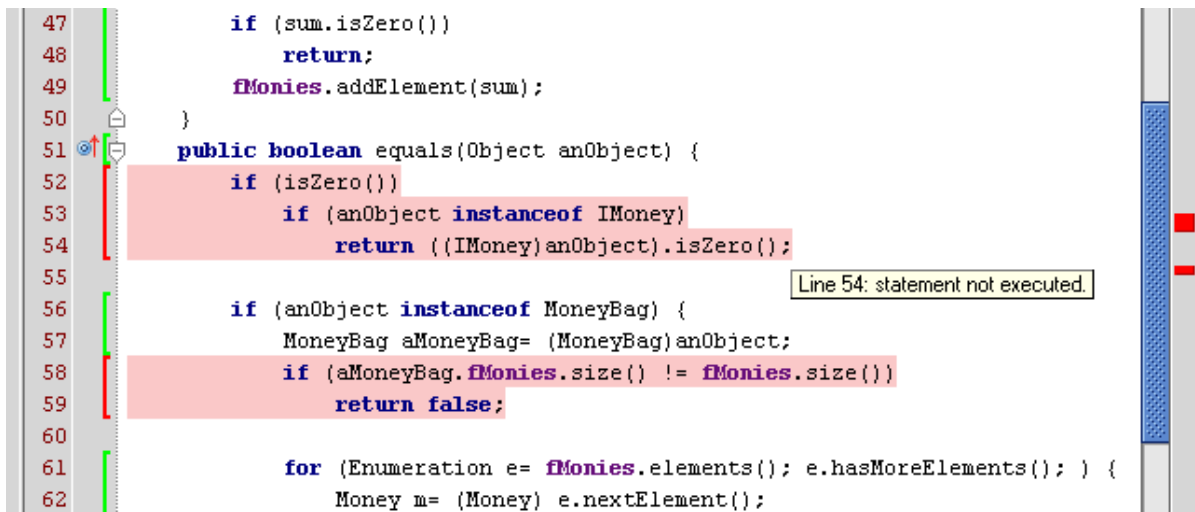
- **Generate Clover Report.** Launches the report generation wizard that will take you through the steps required to generate a Pdf, Html or XML report.
- **Refresh.** Reloads coverage data.
- **Delete.** Delete the current coverage database.

The lower portion of the Tool Window contains various Metrics for the currently selected node in the browser:



Coverage info view

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.



editor pane with overlaid coverage information

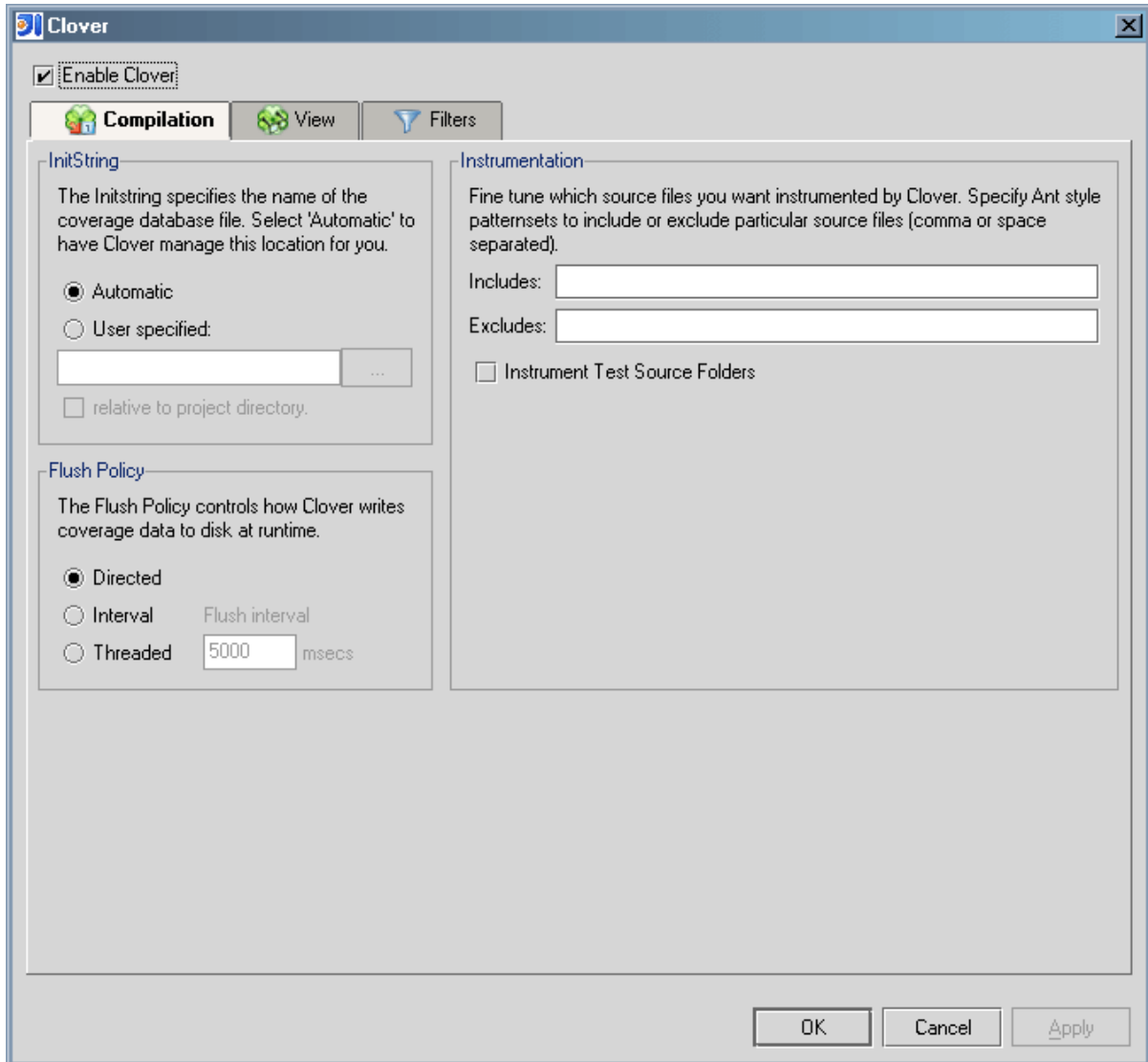
Note:

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.

4.5.7. 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:



Compilation Configuration Screen

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

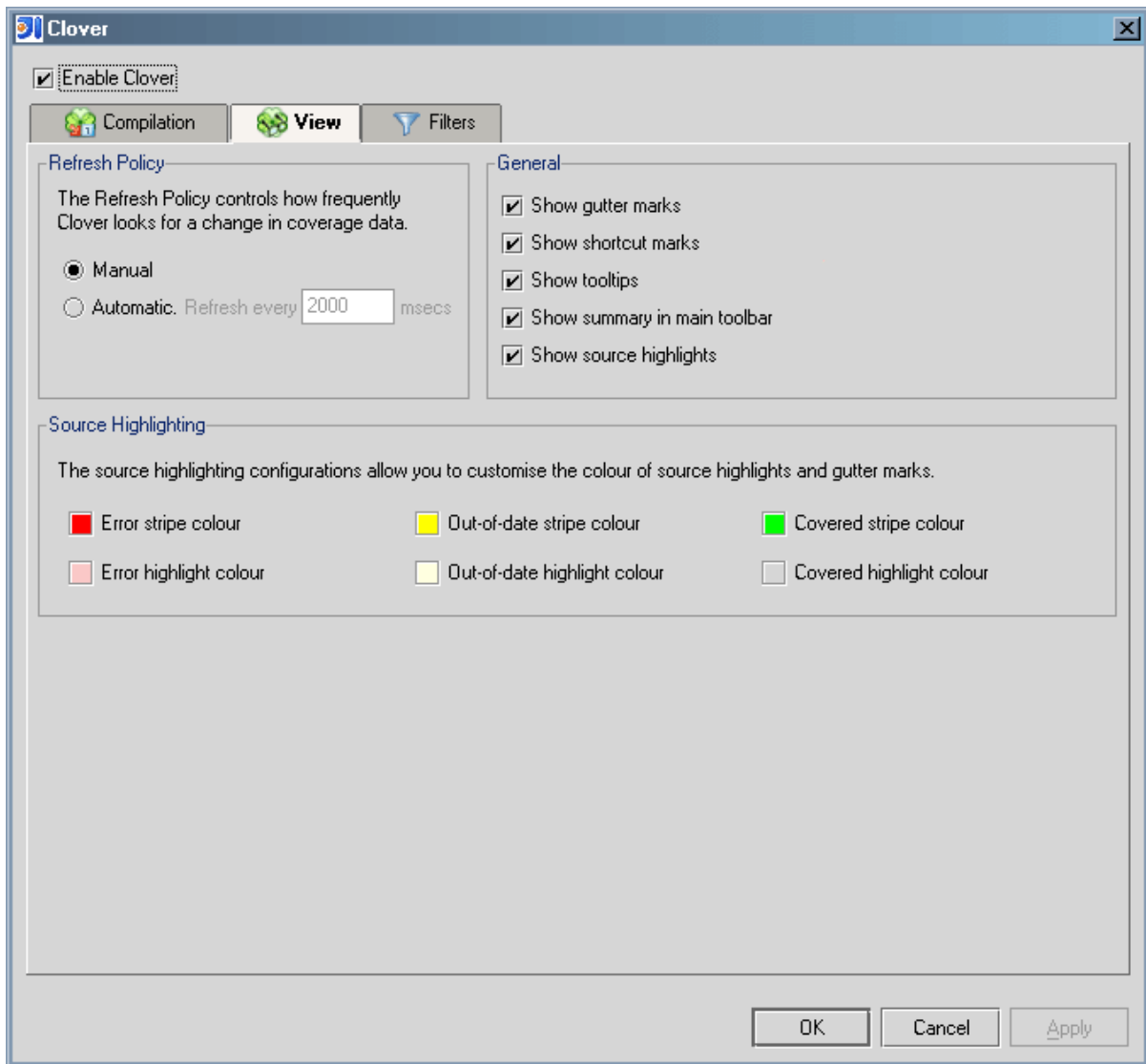
The Flush Policy controls how Clover writes coverage data to disk at runtime. See [Flush Policies](#).

Instrumentation

Allows you to specify a comma separated list of set of Ant Patternsets that describe which files to include and exclude in instrumentation. These options are the same as those described in the [<clover-setup>](#) task.

Viewer options

The second Tab on the configuration panel provides viewing options;



Viewer Configuration Screen

Refresh Policy

The Refresh Policy controls how the Clover Plugin monitors the Coverage Database for new data. "Manual" is the default and means that you have to click button to refresh the coverage data. "Automatic" means that the Clover Plugin will periodically check for new coverage data for you.

General

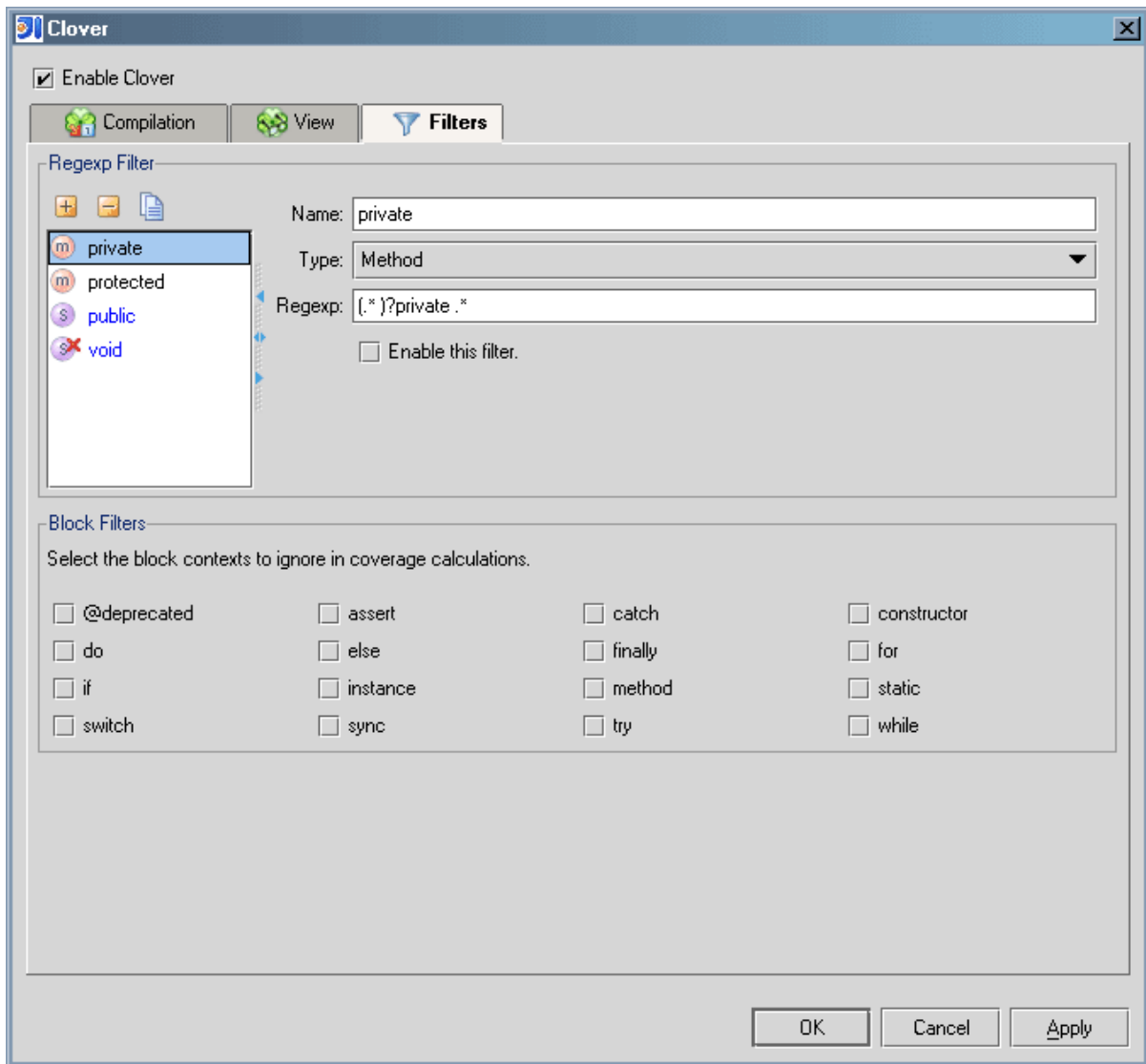
Allows you to customize where coverage data is displayed within the IntelliJ IDE. Gutter marks appear in the left hand gutter of the Java Source Editor. Source highlights appear directly over your source code. Shortcut marks appear in the right hand gutter and allow you to navigate directly to uncovered regions of code.

Source Highlighting

Allows you to fine tune the colours used Clover in its coverage reporting. The 'xxx highlight colour' is used for Source Highlights and the 'xxx stripe colour' is used for Gutter and Shortcut marks.

Filter Options

The third Tab on the configuration panel provides filter options;



Filter Configuration Screen

Regexp Filters

The regexp filters 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 available, you need to delete the existing coverage database using the button and rebuild your project/module.

Note:

See [Coverage Contexts](#) for more information.

Block Filters

Allows you to specify contexts to *ignore* when viewing coverage information. For example, selecting the *if* context will remove *if* body (not the conditional) from the coverage reports.

4.5.8. 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.

4.5.9. FAQ**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.

Q: When I compile with Clover instrumentation enabled, I get the following error: Error: line (31) package com_cenqua_clover does not exist

A: You need to add the clover-idea5.jar file 'each' of your modules classpaths.

Q: I have an enabled Regexp Filter that does not seem to be filtering.

A: Have you checked your regexp? It may be that your regexp is not matching the methods/statements that you expect. See [Coverage Contexts](#) for more information about Regexp Contexts.

A: Try resetting the current context filter and then re-enabling them. There is a known scenario where regexp filters are not being applied to the coverage data when they are enabled at activation time.

4.6. Clover Netbeans Module

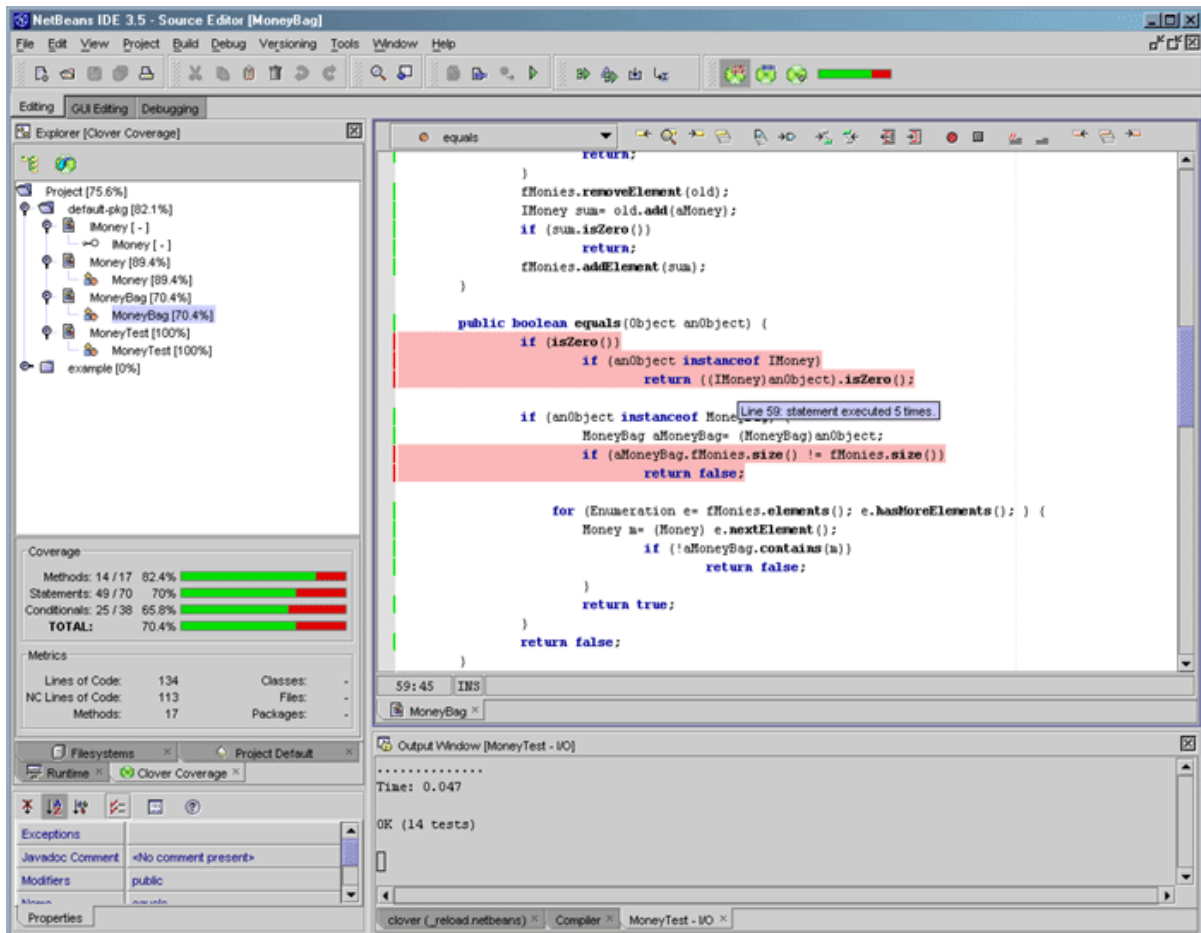
Plugin Version 0.5.1.02

Note:

The Clover Netbeans Module is currently a **beta version**. The module has been tested against **Netbeans 3.5**

4.6.1. Overview

The Clover Netbeans Module allows you to instrument your Java code easily from within the [Netbeans](#) Java IDE, and immediately view your coverage results within the IDE.



Clover Netbeans Plugin

4.6.2. Installing the Module

Install the Clover Netbeans Module using the Update Center.

- Open the Update Center Wizard by selecting "Tools | Update Center".
- Select the "Install manually" option and follow the onscreen instructions (The Clover Netbeans Module is located in the lib/clover-netbeans.nbm file).

4.6.3. Configuring the Module

Add clover-netbeans.jar to your project classpath by mounting the clover-netbeans.jar.

- Open the mount filesystem wizard "File | Mount Filesystem".

- Select "Archive Files" and "Next"
- Select the clover.jar located in either the modules subdirectory of your user directory, OR if the module is installed as 'global', then the modules subdirectory of the NB Installation directory

(The clover-netbeans.jar needs to be in the classpath because it is needed at runtime when you are running your unit tests. It is also needed when you are compiling with Clover)

4.6.4. Using the Module

Clover instruments your code by pre-processing your Java files prior to compilation. This means that you will need to select "Clover Instrumentation" as your Default Compiler for Java Sources. You can do this by the following steps.

- Select the "Build with Clover" toggle button



Compile with clover button

in the Clover Toolbar

Alternatively, the following option is equivalent.

- You can modify the Default Compiler through the "Tools | Options" interface. Select "Options | Editing | Java Sources" and select "Clover Instrumentation" as the Default Compiler property.

Now, whenever you Compile or Build your source, it will be instrumented by Clover. This includes the "Project | Build Project" menu item, and the build/compile options available through the context sensitive right-click popup menus.

Build Options



Build with clover button

Rebuild Coverage

This option allows you to delete existing coverage data, and rebuild the project using the latest configuration and available source. Use this when the coverage database has become 'out-of-sync' with the project as a result of java sources being deleted, or excludes configurations changing.

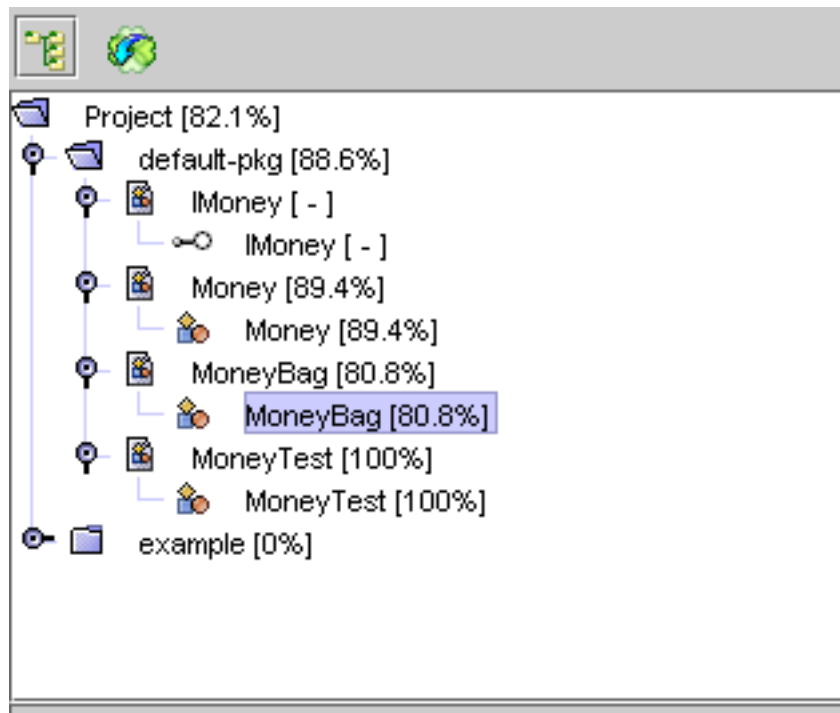
4.6.5. Viewing Coverage Results

Once you are instrumenting your code, Clover will gather code coverage information

whenever you execute your code. This information can be viewed in the form of a clover coverage browser and annotations of your source file.

Coverage Browser

The top pane of the Clover Coverage Tab contains a class browser with inline coverage information:

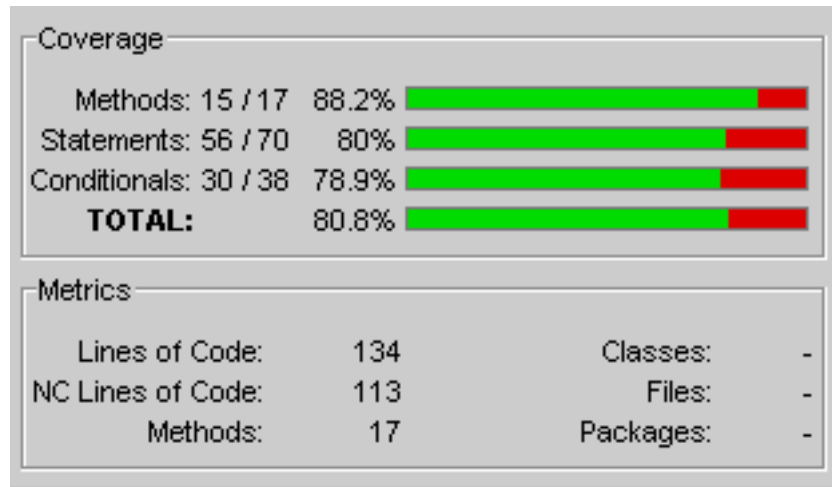


Clover class browser

The tool bar at the top of this browser contains the following options:

-  tool bar icon
Flatten Packages. With this selected, only concrete packages are shown in the browser.
-  tool bar icon
Refresh. Reloads coverage data. This button is disabled when ["Auto Refresh"](#) is active.

The bottom pane of the Clover Coverage Tab contains Coverage and other Metrics information for the currently selected node in the browser.



Coverage info view

Inline source annotation

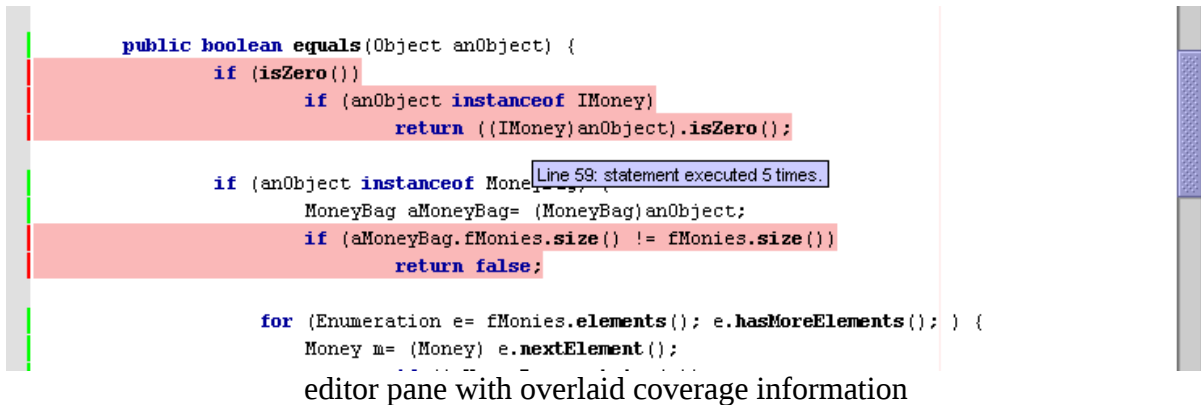
In addition, the plugin can annotate the Java source with the coverage information in the editor pane. This is available whenever the open source file has associated coverage data. The annotations are controlled via a toolbar in the editor pane



editor tool bar

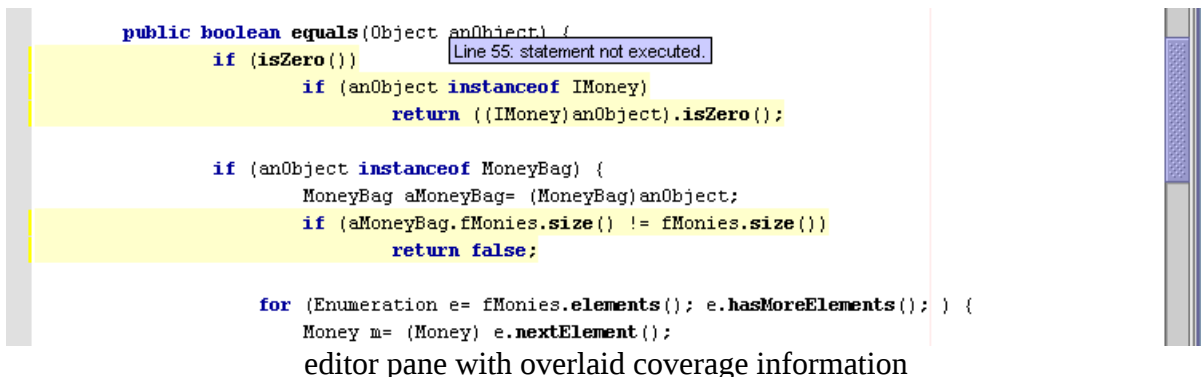
. These toolbar buttons allow you to:

-  editor tool bar icon
Move to the previous uncovered line
-  editor tool bar icon
Toggle display of annotations
-  editor tool bar icon
Move to the next uncovered line



To ensure that the presented coverage information is up-to-date, either tell Clover to refresh by selecting the "Refresh" button in the Clover Coverage Tab OR configure clover to [periodically check](#) for updated coverage information for you.

When the coverage information becomes out of date, the inline source annotations change to yellow.



4.6.6. Configuration

You will find Clover Configuration Options within the Netbeans Options Viewer.

- "Building | Compiler Types | Clover Instrumentation"
- "IDE Configuration | Server And External Tool Settings | Clover Settings"

Clover Instrumentation

Initstring

This section controls where the Clover coverage database will be stored. If left blank, Clover will manage this location for you (relative to your project directory). Otherwise, you may 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](#).

View Settings

Auto Refresh

The Auto Refresh control allows you to enable/disable Clovers automatic coverage update monitoring. When set to true, Clover will automatically refresh coverage data. The refresh interval can be managed via the "Refresh interval" setting.

Refresh interval

The Refresh interval value, specifies, in milliseconds, the time interval Clover will use when automatically checking for coverage updates. See also [Auto Refresh](#).

Show Summary

The Show Summary control allows you to show/hide the Coverage Summary bar in the Clover Toolbar. This summary bar provides you a graphical indication of your coverage for the entire project.

4.6.7. FAQ

Q: Ive run my tests, but coverage information is not being displayed in the IDE:

A: You will need to either select "Auto Refresh" to true, or select reload



reload icon

Q: I only see coverage data for the last test case I executed.:

A: Clover will display the coverage information gathered from your last test run. This means,

that if you run each of your tests individually, then only the coverage from the last test executed will be shown. Support for aggregating multiple test runs is supported via spans, to be included in a future release.

Q: I have an existing Ant build script with clover integration. Can I view the coverage information within Netbeans:

A: By setting the "InitString" property on "Clover Instrumentation" to an existing coverage database (one maintained by an Ant build script for example), you can access all of the viewing features supported by the plugin. Just make sure that you have the source files for this coverage data mounted. You can then safely make changes to your source within Netbeans, build and run your tests with Ant, and the view the coverage results with Clover. If you are going to take this approach, it is best to ensure that the "Build with Clover" toggle button



Compile with Clover

is **NOT** selected.

Q: My source files will not compile:

A: Ensure that you have the clover-netbeans.jar in your project classpath.

4.6.8. Known Issues

- Use Class Include/Exclude does not work.
- It is not expected that future releases will be backward compatible with this release.
- A problem exists with "Auto Refresh" where it will not refresh coverage data after an IDE restart. The coverage information will need to change before the coverage data will be updated.

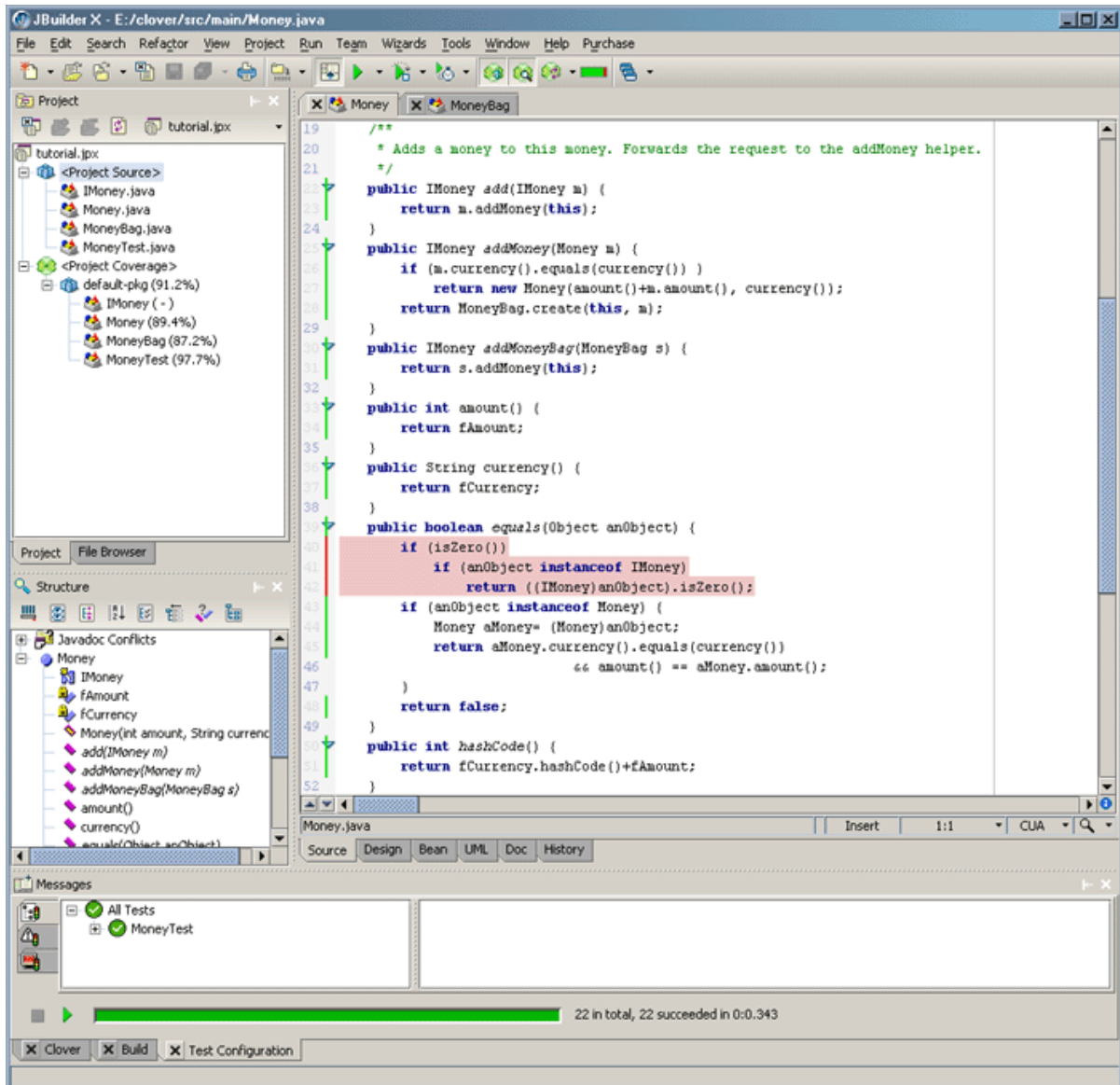
4.7. JBuilder Plugin Guide

Plugin version 1.0

System Requirements: JBuilder 9 (Enterprise Edition), JBuilder X (Enterprise Edition), JBuilder 2005 (Enterprise Edition)

4.7.1. Overview

The Clover JBuilder Plugin allows you access the functionality of Clover from within your IDE. Clover will instrument your java source and show you your test coverage, highlighting areas of code that have not been executed.



The Clover plugin for JBuilder

4.7.2. Installing the JBuilder Plugin

To install the Clover JBuilder Plugin:

1. Locate your JBuilder installation directory. For the rest of this document, this directory will be referred to as JBUILDER_HOME.

2. Download the Clover JBuilder zip file, and extract it into a temporary directory.
3. Copy the lib\clover-jbuilder.jar file into the JBUILDER_HOME/lib/ext directory.
4. The next time you start JBuilder, Clover will be available.

You will need a license to activate your plugin.

- Download your clover.license file from <http://www.cenqua.com/licenses.jspa>. Evaluation licenses are available free of charge.
- Place the clover.license file next to the clover-jbuilder.jar file in the JBUILDER_HOME/lib/ext directory.

4.7.3. Uninstalling the JBuilder Plugin

To uninstall the Clover JBuilder Plugin:

1. Shutdown any running instances of JBuilder.
2. Delete the clover-jbuilder.jar file from the JBUILDER_HOME/lib/ext directory.
3. Restart your JBuilder instance.

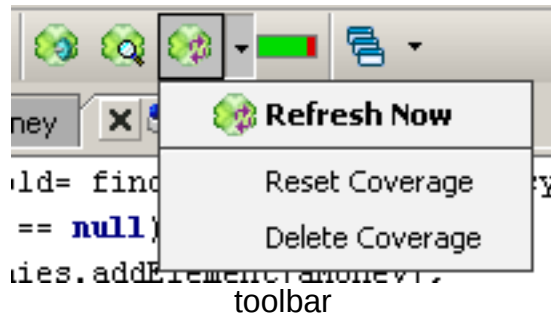
4.7.4. Quick Start Guide

This quick start guide will take you through the steps required to generate a clover coverage report for your project.

1. Enable Clover, by selecting the 'Enable Clover' check box in the 'Project | Project Properties | Clover' properties page.
2. **Add the clover plugin jar to your project library path.**
3. Turn on clover instrumentation by clicking the toolbar item
4. Rebuild your project using any of the build mechanisms provided by JBuilder.
5. View the project coverage data by clicking the toolbar item.
6. Run your application or test cases. This will generate your Coverage data.
7. Refresh/load your coverage data by clicking the to see which parts of your application were covered.

4.7.5. Working with Clover

Your most frequent interaction with Clover will be via the Clover Toolbar (shown below). All the functions available through the toolbar are also available in the 'Project' menu.



The Clover toolbar contains the following functions:

- **Build with Clover.** Toggles the use of the Clover Compiler when JBuilder compiles the current project.
- **Show Coverage.** Toggles the display of coverage information in the Source editors.
- **Refresh coverage button.** Coverage data for the current project is reloaded from disk.
- **Coverage summary bar.** Displays the coverage level of the current project.
- **Reset Coverage.** Deletes the recorded coverage data for the current project.
- **Delete Coverage.** Deletes the recorded coverage data AND the coverage database for the current project.

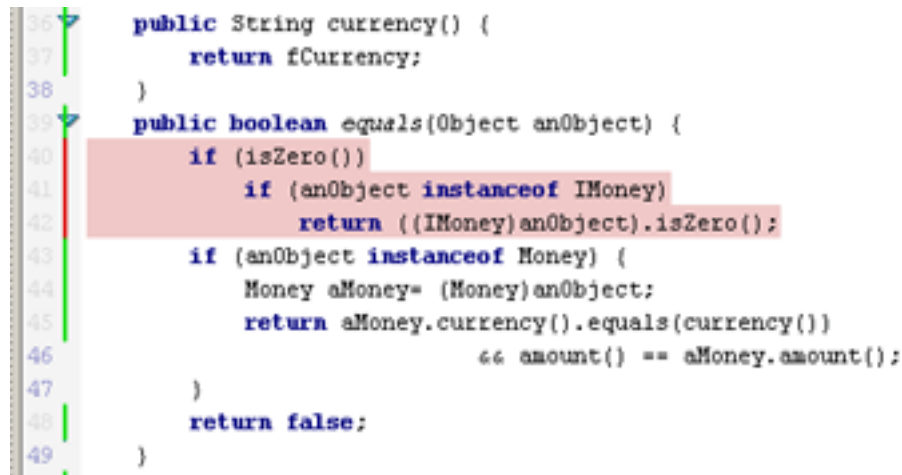
In addition, the 'Project' menu contains additional functions:

- **Generate Report....** Launches the report generation wizard that will take you through the steps required to generate a Pdf, Html or XML report.
- **Filter Coverage....** Launches a dialog to set the context filter.

4.7.6. Viewing Coverage Results

Java Source Editor

The Clover JBuilder plugin allows you to view the clover coverage data directly within the Java Source Editor (as seen below).



```

36 public String currency() {
37     return fCurrency;
38 }
39 public boolean equals(Object anObject) {
40     if (isZero())
41         if (anObject instanceof IMoney)
42             return ((IMoney)anObject).isZero();
43     if (anObject instanceof Money) {
44         Money aMoney= (Money)anObject;
45         return aMoney.currency().equals(currency())
46             && amount() == aMoney.amount();
47     }
48     return false;
49 }

```

Source coverage view

The coverage data is displayed in two ways:

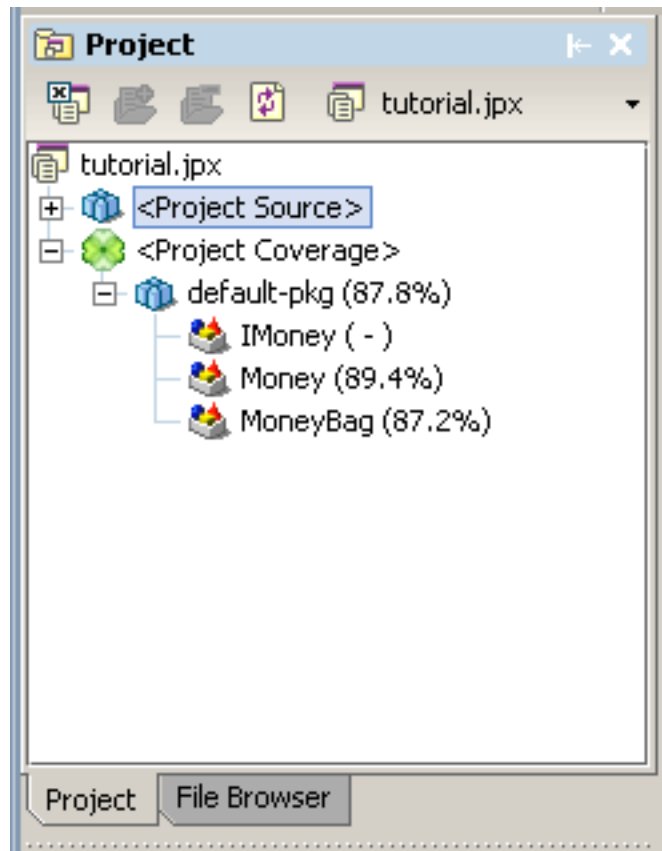
- As a marker in the left hand gutter.
- As highlights within the Java Source Editor.

By default, coverage data is represented by three colours.

- **Red** indicates that the line of source is not fully covered.
- **Green** indicates that the line of source is fully covered.
- **Yellow** indicates that the coverage data being displayed is out of date. The source file has changed since the coverage data was generated, and will need to be re-instrumented.

Project Coverage Tree

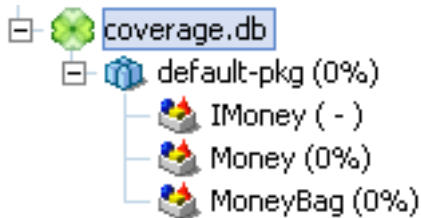
The Clover Plugin allows you to view a coverage tree (see below) for the current project. This coverage tree is located in the Project Pane, on the left hand side of the IDE.



Project view

Clover Coverage Database File Type support

Clover provides support for viewing arbitrary coverage databases. Just add a coverage database file to your project and explore the coverage recordings. The default coverage database file extension is db. This can be modified as necessary via 'Tools | Preferences' and then 'Browser | File types' properties page.

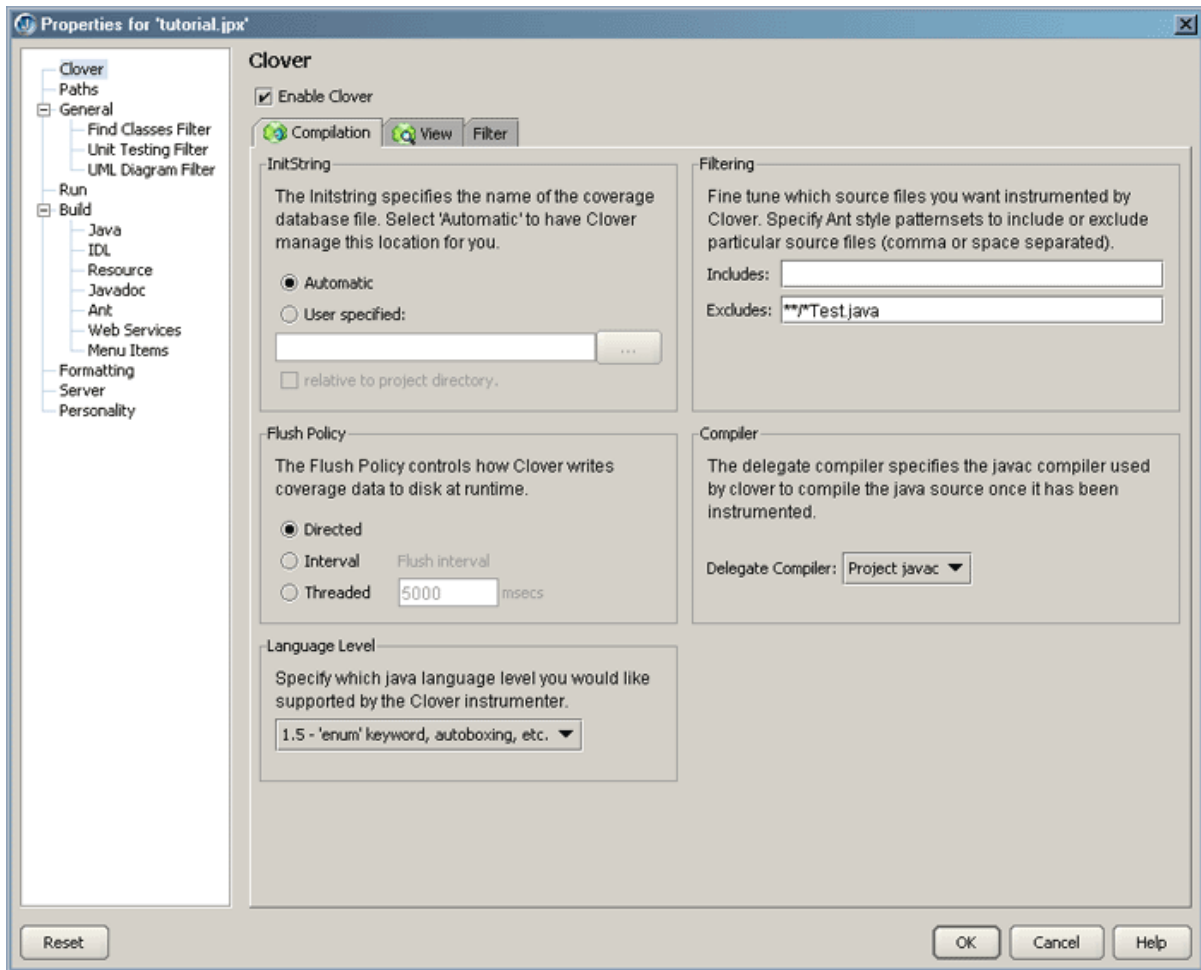


Coverage database

4.7.7. Configuration Options

The Clover Plugin configuration options are available through the 'Project | Project Properties' menu, or the project node (right click) context menu. Configuration is split into Compilation configuration and View configuration.

Compilation Options



Compilation properties

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. "Directed" is the default and means coverage data is written to disk when the JVM exists (or when your test cases finish). "Interval" allows you to specify that coverage data should be written out at regular intervals. "Threaded" will actively flush coverage data to disk at regular intervals. See [Flush Policies](#).

Filtering

If you do not want all of your source instrumented, then you can control which this using these two Ant pattern sets. For example, you may prevent tests from being instrumented by using an "Excludes" value of `**/*Test.java` as shown.

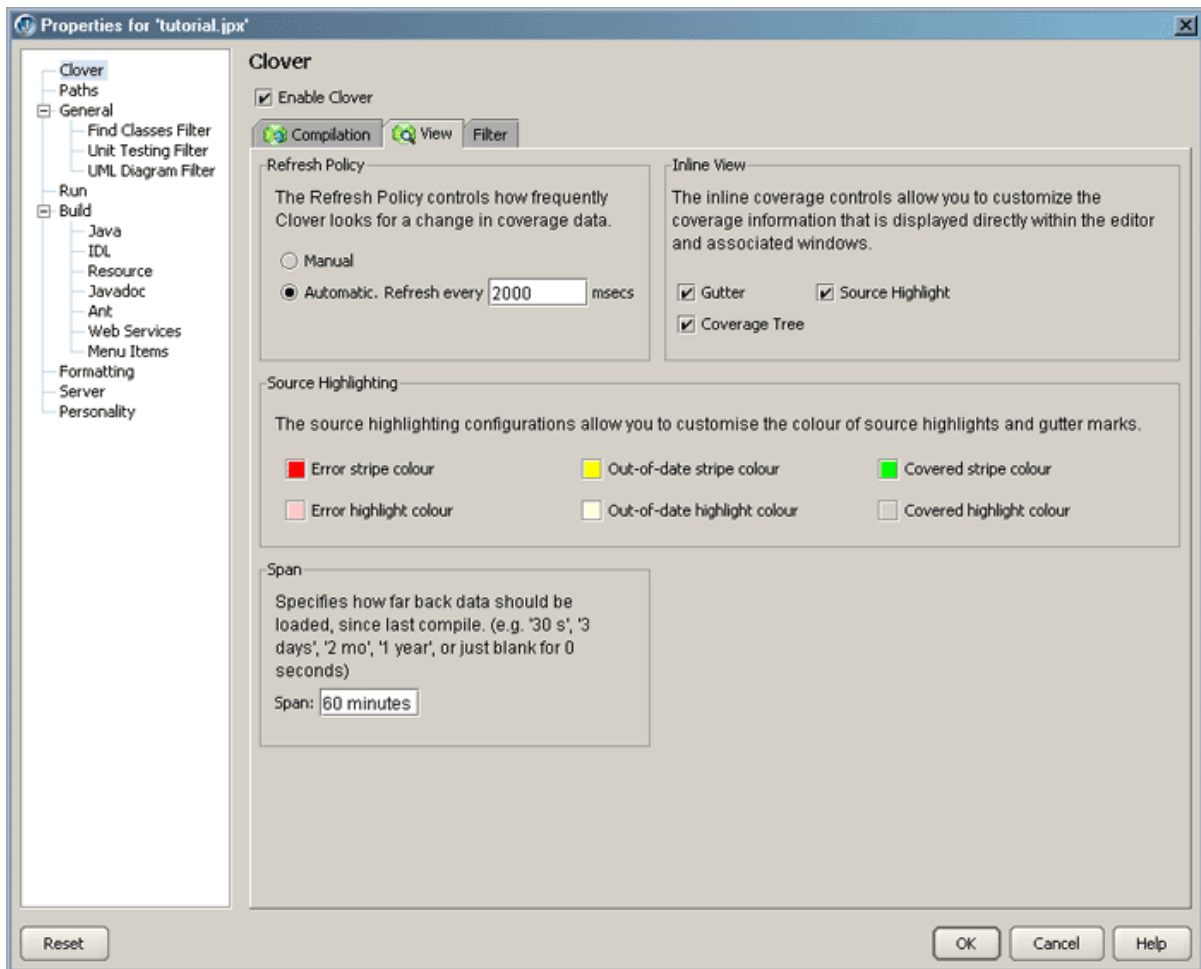
Compiler

This allows you to select the java compiler used by clover to compile your java source once it has been instrumented.

Language Level

Allows you to specify which language features you would like Clover to support. If you use asserts within your code, you would need to select '1.4' or higher, if you use enums, then you need to select '1.5'.

View Options



Viewer properties

Refresh Policy

The Refresh Policy controls how the Clover Plugin monitors the Coverage Database for new data. "Manual" is the default and means that you have to click to refresh the coverage data. "Automatic" means that the Clover Plugin will periodically check for new coverage data for you.

Inline View

Allow you to customize where coverage data is displayed within the JBuilder IDE. Gutter marks appear in the left hand gutter of the Java Source Editor. Inline highlights appear

directly over your source code. The Coverage Tree is located within the IDEs project view, and provides a per file view of your project coverage.

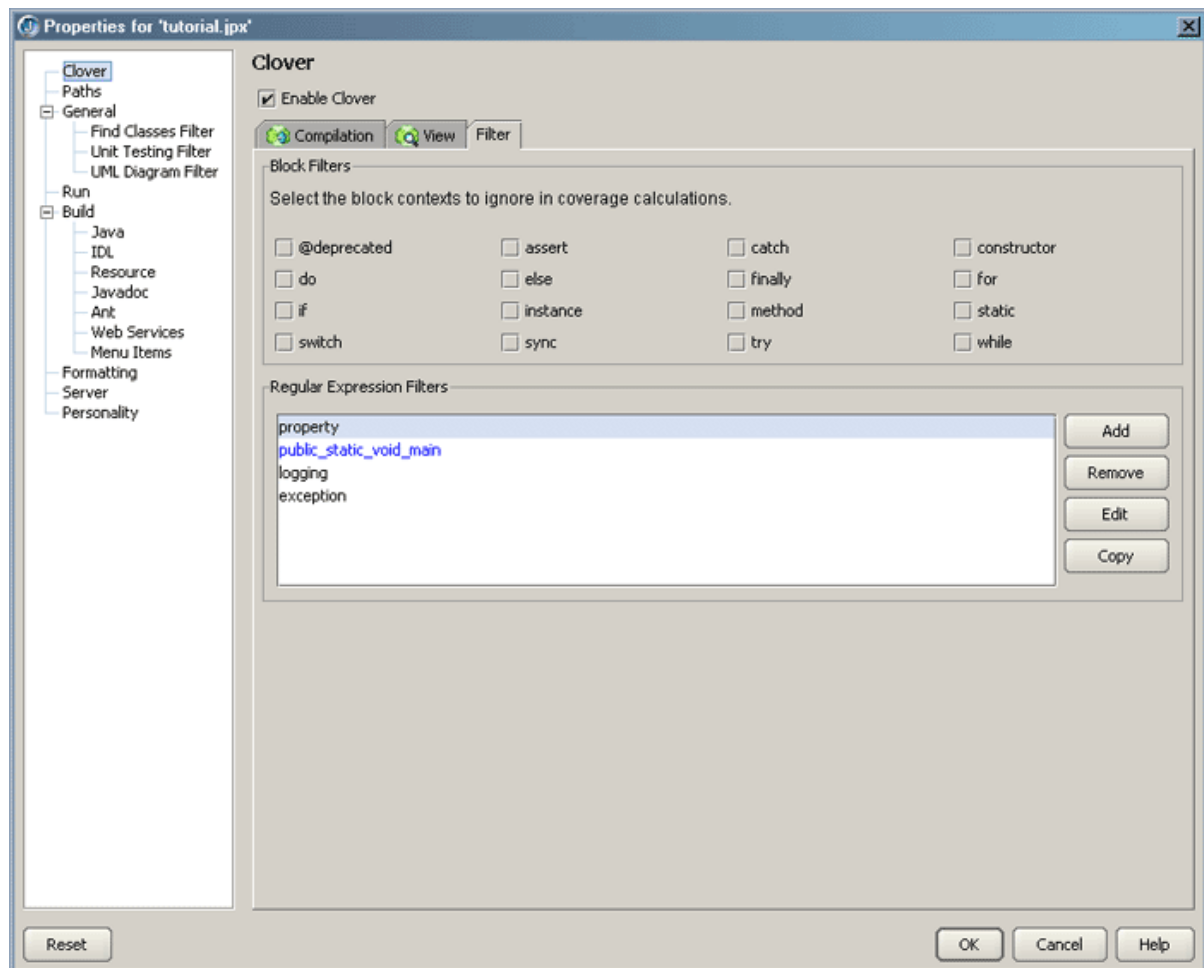
Source Highlighting

Allows you to fine tune the colours used Clover in its coverage reporting. The 'xxx highlight colour' is used for Source Highlights and the 'xxx stripe colour' is used for Gutter marks.

Span

Allows you to configure the span used by Clover. See [Spans](#) for more information.

Filter Options



Filters properties

Block Filters

Allows you to specify contexts to *ignore* when viewing coverage information.

Regex Filters

The regexp filters allow you to define custom contexts to *ignore* when viewing coverage information.

Working with regexp filters.

- Use group of button on the right hand side to Create, Delete, Edit or Copy 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 available, you need to delete the existing coverage database using the **Delete Coverage** menu item and rebuild your project.

Note:

See [Coverage Contexts](#) for more information.

4.7.8. 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 "Tools | Project Properties | Clover | Filters".
- Select **Add** to create a new Regexp Context Filter.
- Select **Edit** to open up the Regexp Edit dialog.
- 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 **Delete Coverage** from the Clover menu and recompile your project. Once active, you will notice the `private` filter appears in the Context Filter Dialog. You will now be able to filter private methods out of your Clover coverage calculations and reports.

4.7.9. FAQ

Q: Why does JBuilder prompt me to save a clover enabled project on exit when I have not changed any settings?

A: During the project build process, clover needs to modify the projects sourcepath to allow for various compiler optimizations. Although it only exist for the duration of the build, this configuration change is what triggers the project to be considered 'dirty', and hence the save upon exit dialog.

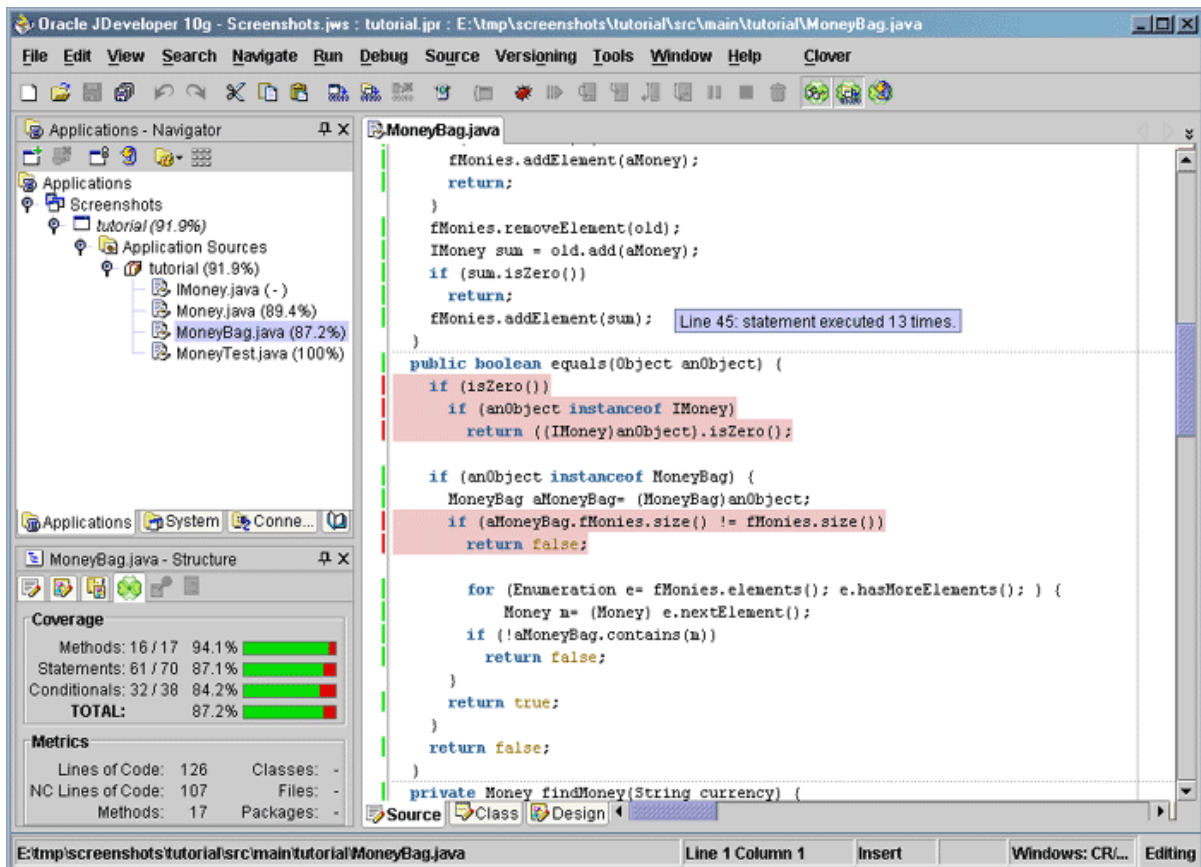
4.8. Clover JDeveloper 10g Plugin UserGuide

Plugin Version 1.0

System Requirements: Oracle JDeveloper 9.0.5.1

4.8.1. Overview

The Clover JDeveloper Plugin allows you to instrument your Java code and view your coverage results easily from within the Oracle JDeveloper Java IDE.



Clover JDeveloper plugin

4.8.2. Installing

Once you have downloaded the Clover JDeveloper Plugin package from <http://www.cenqua.com>, you can install the plugin as follows:

1. shutdown any running instances of JDeveloper
2. remove any previous versions of the the clover plugin jar from `JDEVELOPER_HOME/jdev/lib/ext`
3. copy `CLOVER_HOME/lib/clover-jdeveloper.jar` into the `JDEVELOPER_HOME/jdev/lib/ext` directory, and restart JDeveloper.

You will need a license to activate your plugin.

- Download your `clover.license` file from <http://www.cenqua.com/licenses.jspa>. Evaluation licenses are available free of charge.
- Place the `clover.license` file next to the `clover-jdeveloper.jar` file in the

JDEVELOPER_HOME/jdev/lib/ext directory.

If you are upgrading from a previous version of the Clover JDeveloper Plugin, you will also need to do the following.

- Edit the JDEVELOPER_HOME/jdev/system.../ide.properties file, removing all of the MainWindow.Toolbar.item property references to Clover.

4.8.3. Uninstalling

To uninstall the Clover JDeveloper Plugin:

1. shutdown any running instances of JDeveloper
2. delete the clover-jdeveloper.jar file from the JDEVELOPER_HOME/jdev/lib/ext directory.
3. restart JDeveloper

4.8.4. Configuring your Project

Add clover jar to your project classpath.

- Open the project properties "Tools | Project Properties..." or by double clicking on the project within the Navigator window.
- Go to "Profiles | Active Profile | Libraries". Create a new library and add the JDEVELOPER_HOME/jdev/lib/ext/clover-jdeveloper.jar to the libraries classpath. Add this library to your projects 'Selected Libraries'.

(The clover jar needs to be in the classpath because it is needed at runtime when you are running your unit tests and at compile time when you are compiling instrumented source files.)

4.8.5. Getting Started

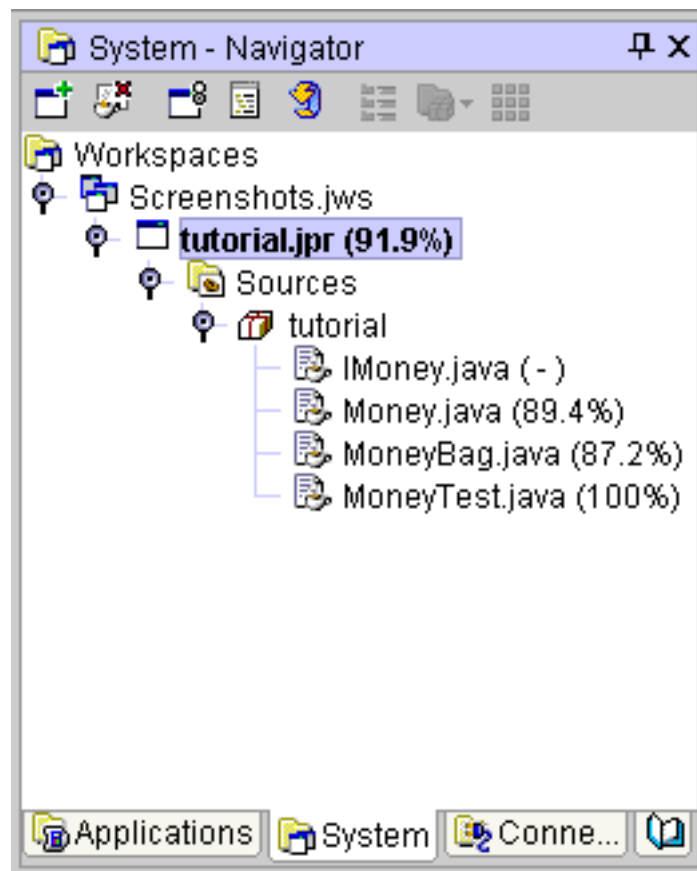
This getting started guide will take you through the steps required to generate Clover coverage for your project.

1. Ensure that you have [configured your project](#) to use Clover.
2. Enable Clover, by selecting the 'Enable Clover' check box in the "Tools | Project Properties... | Clover" interface.
3. Turn on clover instrumentation by selecting the toolbar item
4. Rebuild your project using any of the build mechanisms provided by JDeveloper.
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. View the project coverage data by selecting the toolbar item.

4.8.6. Viewing Coverage Results

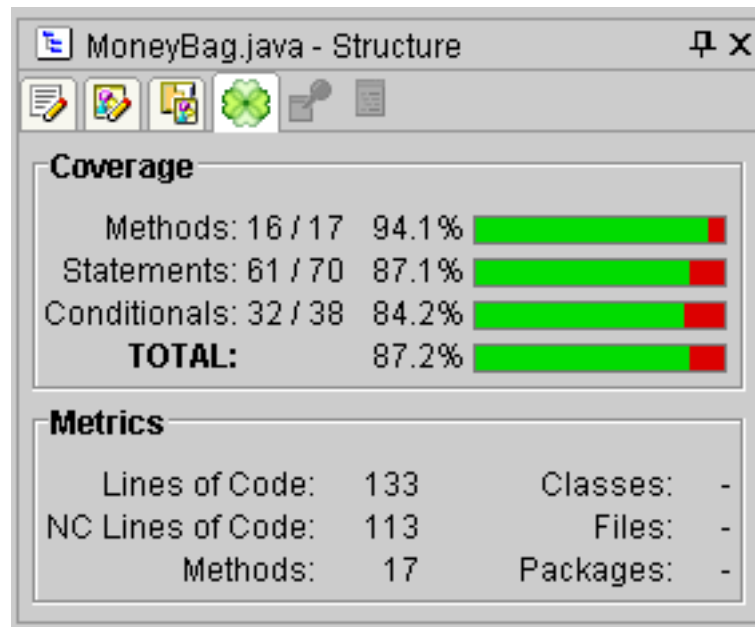
Code Coverage information will be available for viewing within JDeveloper after you have built and run your application. The display of coverage information within the IDE can be controlled via the toggle button in the IDE toolbar, or the "Show Coverage" menu item in the "Clover" menu.

Within the Application Navigator, you will see coverage percentages displayed next to projects, packages and source files that have been clovered. The coverage displayed at each level of the Navigator is the sum of coverage of the packages or source files below it. That is, the coverage of a package is the sum of the coverage for the files contained within the package and all sub-packages.



Application Navigator coverage overlay

Within the Structure window, you can view a summary of the coverage details for the currently 'active node'. This summary information includes the coverage of methods, conditional and statements, as well as the number of lines of code, files, classes and packages associated with this summary.



Structure window coverage summary panel

In addition, the plugin can annotate the Java code with the coverage information. Green indicates that the line of source has been 'covered', red indicates it has not been 'covered', and yellow indicates that the coverage information is out of date. The tooltips indicate exactly how many times a line has been executed, or an expression has evaluated to true etc.

```

IMoney sum= old.add(aMoney);
if (sum.isZero())
    return;
fMonies.addElement(sum);
}

public boolean equals(Object anObject) {
    if (isZero())
        if (anObject instanceof IMoney)
            return ((IMoney)anObject).isZero();

    if (anObject instanceof MoneyBag) {
        MoneyBag aMoneyBag= (MoneyBag)anObject;
        if (aMoneyBag.fMonies.size() != fMonies.size())
            return false;

        for (Enumeration e= fMonies.elements(); e.hasMoreElements(); ) {
            Money m= (Money) e.nextElement();
            if (!aMoneyBag.contains(m))

```

Line 54: statement not executed.

editor pane with overlaid coverage information

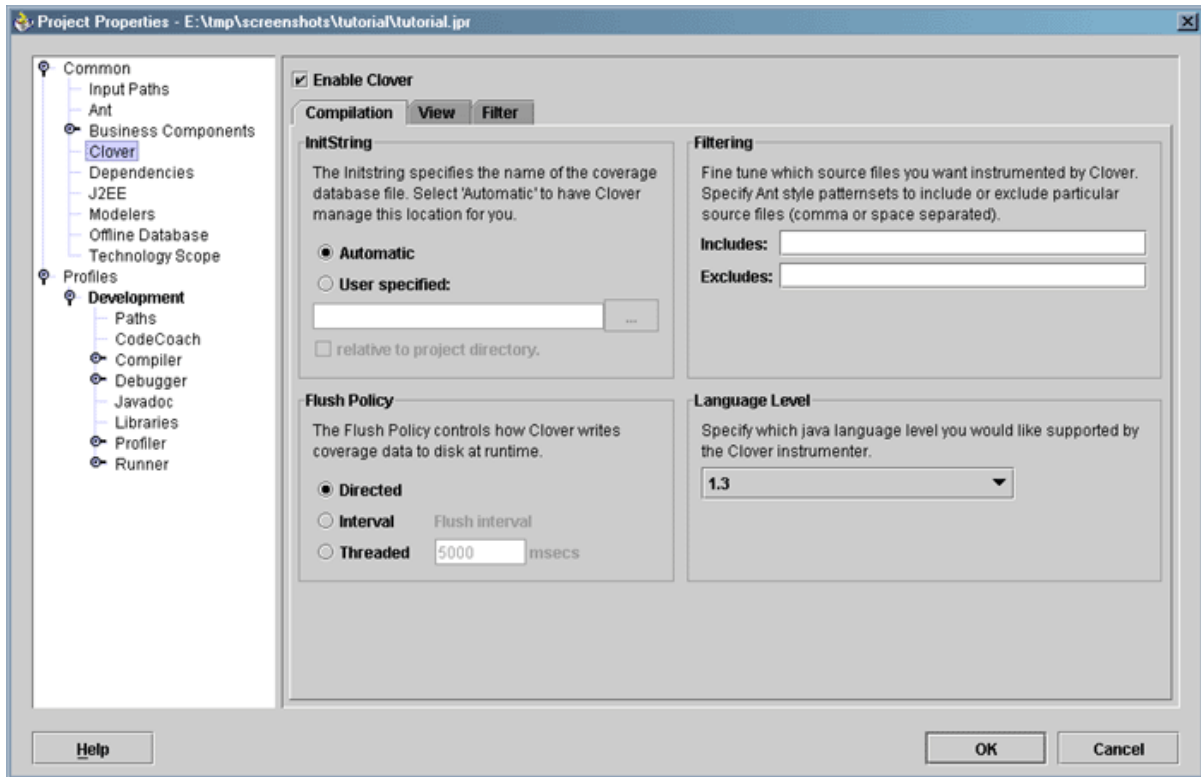
4.8.7. Working with Clover

There are a number of menu items and toolbar actions that allow you to interact with Clover. They are as follows:

- **Show Coverage** When selected, coverage information will be displayed within the IDE, as decided in the [previous section](#).
- **Build with Clover.** When selected, Clover will instrument your source files during the JDeveloper build cycle.
- **Refresh Coverage** Will force the plugin to load the latest coverage information. You will need to refresh after building or running your application.
- **Delete Coverage** Delete the current coverage database.
- **Generate Report...** Launches the report generation wizard that will take you through the steps required to generate a Pdf, Html or XML report.
- **Filter Coverage...** Launches a dialog to set the context filter.

4.8.8. 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:



Compiler configuration screen

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 defines 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](#).

Filtering

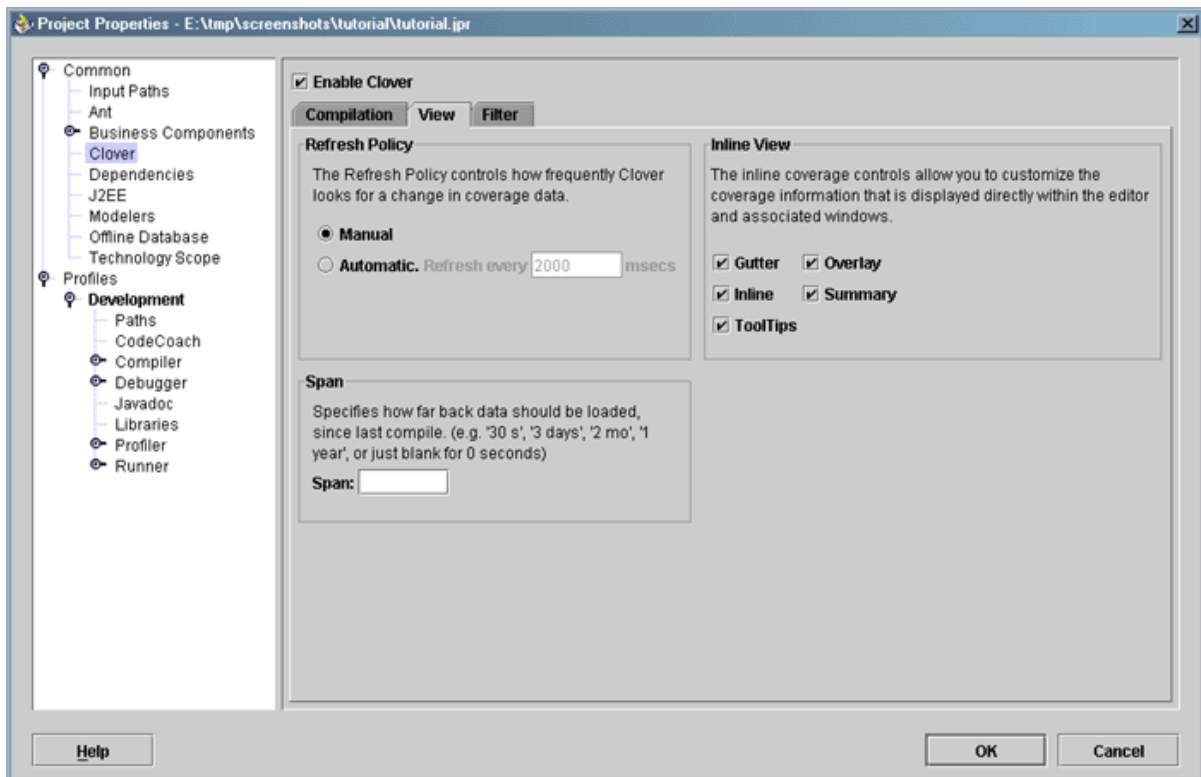
Allows you to specify a comma separated list of set of Ant Patterns that describe which files to include and exclude in instrumentation. These options are the same as those described in the [<clover-setup>](#) task.

Language Level

Allows you to specify which language features you would like Clover to support. If you use asserts within your code, you would need to select '1.4' or higher, if you use enums, then you need to select '1.5'.

4.8.9. Viewing options

The second Tab on the configuration panel provides viewing options;



Viewer configuration screen

Refresh Policy

The Refresh Policy controls how the Clover Plugin monitors the Coverage Database for new data. "Manual" is the default and means that you have to click button to refresh the coverage data. "Automatic" means that the Clover Plugin will periodically check for new coverage data for you.

Inline View

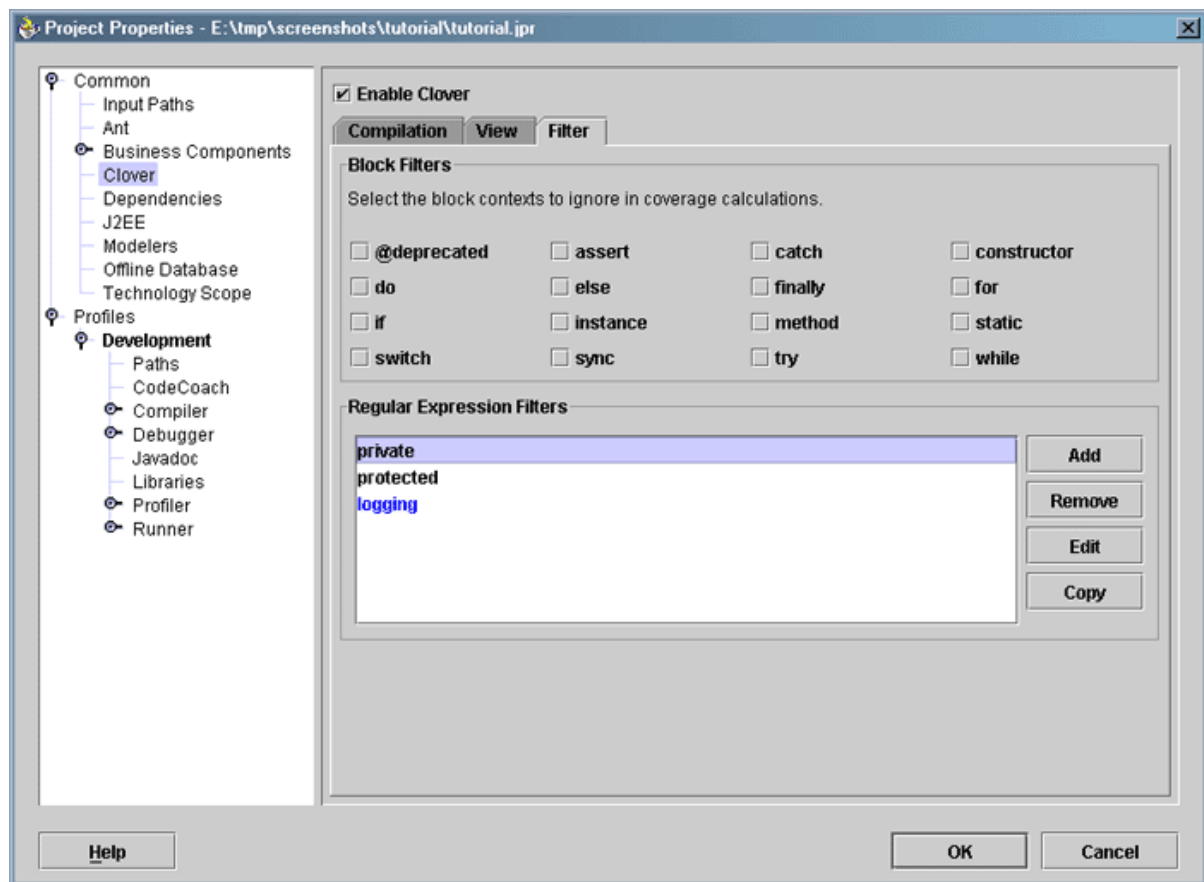
Allows you to customize where coverage data is displayed within the JDeveloper IDE. Gutter marks appear in the left hand gutter of the Java Source Editor. Inline refers to the annotations that appear directly over your source code. Overlay refers to the coverage information displayed within the Application Navigator window, and Summary refers to the coverage summary panel available within the Structure Window.

Span

Allows you to configure the span used by Clover. See [Spans](#) for more information.

4.8.10. Filter Options

The third Tab on the configuration panel provides viewing options;



Filter configuration screen

Regex Filters

The regexp filters allow you to define custom contexts to *ignore* when viewing coverage information.

Working with regexp filters.

- Use group of button on the right hand side to Create, Delete, Edit or Copy 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 available, you need to delete the existing coverage database using the menu item and rebuild your project.

Note:

See [Coverage Contexts](#) for more information.

Block Filters

Allows you to specify contexts to *ignore* when viewing coverage information. For example, selecting the *if* context will remove *if* body (not the conditional) from the coverage reports.

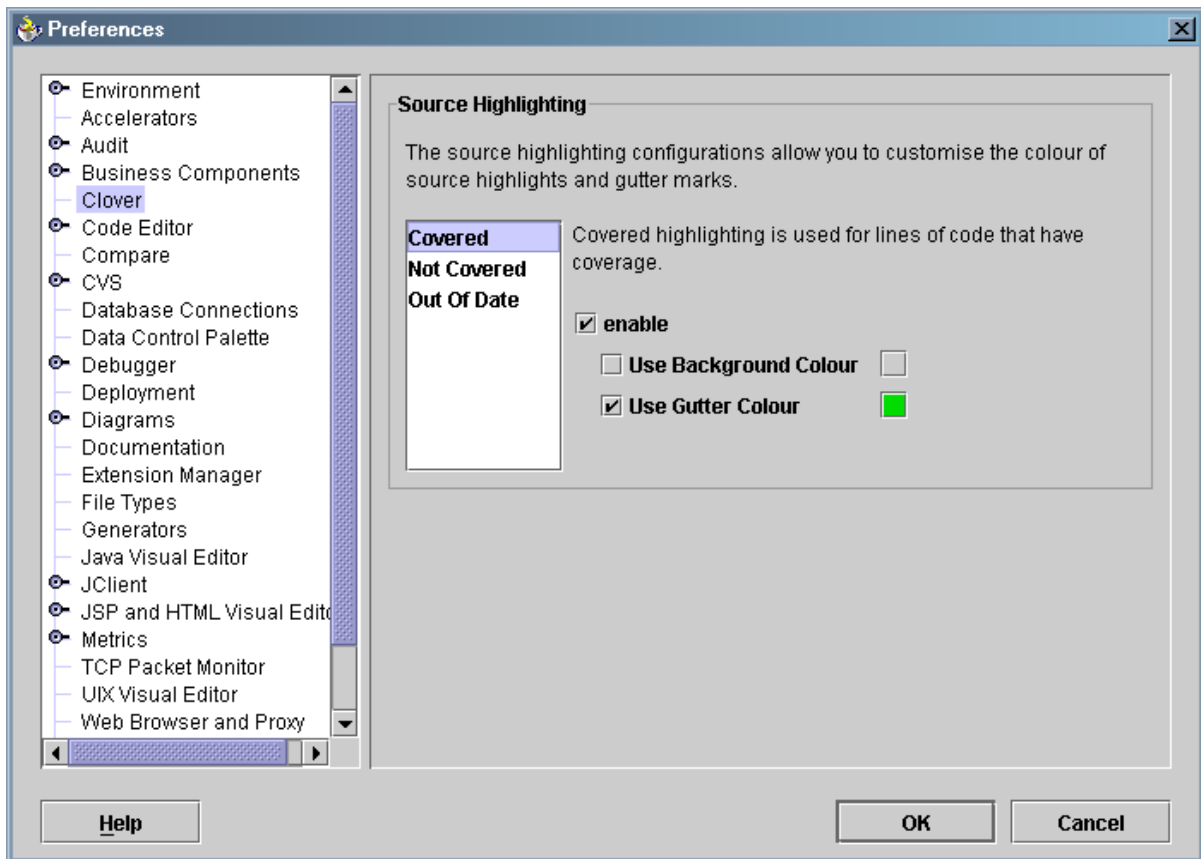
4.8.11. Example: Creating a regexp context filter

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 "Tools | Project Properties | Clover | Filters".
- Select **Add** to create a new Regexp Context Filter.
- Select **Edit** to open up the Regexp Edit dialog.
- 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 Clover menu and recompile your project. Once active, you will notice the `private` filter appears in the Context Filter Dialog. You will now be able to filter private methods out of your Clover coverage calculations and reports.

4.8.12. Source Highlight Options

The configuration panel for the source highlighting options is available in the JDeveloper preferences located at "Tools | Preferences | Clover".



Compiler configuration screen

Source Highlighting

The source highlighting configuration panel allows you to specify the colours used by clover when it renders coverage information in the JDeveloper editor panel. The Background colour represents the colour used to highlight each line of source code, the Gutter colour is the colour of the mark located in the editors left side gutter.

4.8.13. FAQ

Q: Clover has caused blank actions to appear in my Main Toolbar.

A: This happens when you are upgrading to version 1.0RC1 of the plugin. To fix this, you will need to edit the `JDEVELOPER_HOME/jdev/system.../ide.properties` file, removing all of the `MainWindow.Toolbar.item` property references to Clover.

5. Command Line Tools

5.1. 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](#) provide 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](#), [ConsoleReporter](#) or [SwingViewer](#) to view the measured coverage results.

5.1.1. Command line tools:

CloverInstr	Copies and instruments individual java source files, or a directory of source files.
CloverMerge	Merges existing Clover databases to allow for combined reports to be generated.
XmlReporter	Produces coverage reports in XML
HtmlReporter	Produces coverage reports in HTML
PDFReporter	Produces coverage reports in PDF format
ConsoleReporter	Reports coverage results to the console
SwingViewer	Launches the Swing coverage viewer

5.2. 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.

5.2.1. Usage

```
java com.cenqua.clover.CloverInstr [OPTIONS] PARAMS [FILES...]
```

5.2.2. Params

-i, --initstring <file>	Clover initstring. This is the full path to the dbfile that will be used to construct/update to store coverage data.
-s, --srcdir <dir>	Directory containing source files to be instrumented. If omitted individual source files should be specified on the command line.
-d, --destdir <dir>	Directory where Clover should place the instrumented sources. Note that files will be overwritten in the destination directory.

5.2.3. Options

-p, --flushpolicy <policy>	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".
-f, --flushinterval <int>	Tell Clover how often to flush coverage data when using either "interval" or "threaded" flushpolicy. Value in milliseconds.
--instrumentation <policy>	Set the instrumentation strategy. Valid values are "field" and "class". Default is "class".
-e, --encoding <encoding>	Specify the file encoding for source files. If not specified, the platform default encoding is used.
-jdk14	Direct Clover to parse sources using the JDK1.4 grammar.
-jdk15	Direct Clover to parse sources using the JDK1.5 grammar.
-v, --verbose	Enable verbose logging.

5.2.4. 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.cenqua.clover.CloverInstr;

...

String [] cliArgs = { "-jdk14", "-i", "clover.db", "-d", "build/instr", "Money.java"
int result = CloverInstr.mainImpl(cliArgs);
if (result != 0) {
    // problem during instrumentation
}
```

5.2.5. Examples

```
java com.cenqua.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.cenqua.clover.CloverInstr -jdk14 -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 (ie. support the 'assert' keyword).

5.3. CloverMerge

This tool merges existing Clover databases to allow for combined reports to be generated.

5.3.1. Usage

```
java com.cenqua.clover.CloverMerge [OPTIONS] PARAMS [DBFILES...]
```

5.3.2. Params

-i, --initstring <file>	Clover initstring. Clover initstring. This is the path where the new merged database will be written.
-------------------------	---

5.3.3. Options

-s, --span <interval>	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
-v, --verbose	Enable verbose logging.

-d, --debug	Enable debug logging.
-------------	-----------------------

5.3.4. 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.cenqua.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
}
```

5.3.5. Examples

```
java com.cenqua.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.cenqua.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.

5.4. XmlReporter

Produces an XML report of Code Coverage for the given coverage database.

5.4.1. Usage

```
java com.cenqua.clover.reporters.xml.XMLReporter [OPTIONS] PARAMS
```

5.4.2. Params

-i, --initstring <file>	The initstring of the coverage database.
-o, --outfile <file>	The file to write XML output to.

5.4.3. Options

-t, --title <string>	Report title
-l, --lineinfo	Include source-level coverage info
-s, --span <interval>	Specifies how far back in time to include coverage recordings from since the last Clover build. See Using Spans . Defaults to 0 seconds.
-d, --debug	Switch logging level to debug
-v, --verbose	Switch logging level to verbose

5.4.4. 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.cenqua.clover.reporters.xml.XMLReporter;

...

String [] cliArgs = { "-i", "clover.db", "-o", "coverage.xml" };
int result = XMLReporter.mainImpl(cliArgs);
if (result != 0) {
    // problem during report generation
}
```

5.4.5. Examples

```
java com.cenqua.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.cenqua.clover.reporters.xml.XMLReporter -l -t "My Coverage" -i clover.db -o c
```

Produce the same report as above, but include source-level coverage information, and a report title.

5.5. HtmlReporter

Produces an HTML report of Code Coverage for the given coverage database.

5.5.1. Usage

```
java com.cenqua.clover.reporters.html.HtmlReporter [OPTIONS] PARAMS
```

5.5.2. Params

-i, --initstring <file>	The initstring of the coverage database.
-o, --outputdir <dir>	The directory to write the report to. Will be created if it doesn't exist.

5.5.3. Options

-t, --title <string>	Report title
-bw	Don't colour syntax-highlight source - smaller html output.
-h, --hidesrc	Don't render source level coverage.
-p, --sourcepath <path>	The source path to search when looking for source files.
-b, --hidebars	Don't render coverage bars.
-tw, --tabwidth <int>	The number of spaces to substitute TAB characters with. Defaults to 4.
-c, --orderby <compname>	comparator to use when listing packages and classes. Default is PcCoveredAsc. 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
-l, --ignore <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 Using Contexts
-s, --span <interval>	Specifies how far back in time to include coverage recordings from since the last Clover build. See Using Spans . Defaults to 0 seconds.
-d, --debug	Switch logging level to debug
-v, --verbose	Switch logging level to verbose

5.5.4. 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.cenqua.clover.reporters.html.HtmlReporter;

...

String [] cliArgs = { "-i", "clover.db", "-o", "clover_html" };
int result = HtmlReporter.mainImpl(cliArgs);
if (result != 0) {
    // problem during report generation
}
```

5.5.5. Examples

```
java com.cenqua.clover.reporters.html.HtmlReporter -i clover.db -o clover_html
```

Read coverage for the Clover database "clover.db", and produce a report in the directory "clover_html"

```
java com.cenqua.clover.reporters.html.HtmlReporter -c ElementsCoveredAsc
-t "My Coverage" -i clover.db -o clover_html
```

Produce the same report as above, but include a report title, and order lists by total elements covered rather than percentage covered.

5.6. PDFReporter

Produces a PDF summary report of Code Coverage for the given coverage database.

5.6.1. Usage

```
java com.cenqua.clover.reporters.pdf.PDFReporter [OPTIONS] PARAMS
```

5.6.2. Params

-i, --initstring <file>	The initstring of the coverage database.
-o, --outputfile <file>	The file to write the report to.

5.6.3. Options

-t, --title <string>	Report title
-b, --hidebars	Don't render coverage bars.
-p, --pagesize <size>	Specify the page size to render. Valid values are "Letter" and "A4". Default is "A4".
-c, --orderby <compname>	comparator to use when listing packages and classes. Default is PcCoveredAsc. 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
-l, --ignore <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 Using Contexts
-s, --span <interval>	Specifies how far back in time to include coverage recordings from since the last Clover build. See Using Spans . Defaults to 0 seconds.
-d, --debug	Switch logging level to debug
-v, --verbose	Switch logging level to verbose

5.6.4. 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.cenqua.clover.reporters.pdf.PDFReporter;

...

String [] cliArgs = { "-i", "clover.db", "-o", "coverage.pdf" };
int result = PDFReporter.mainImpl(cliArgs);
if (result != 0) {
    // problem during report generation
}
```

5.6.5. Examples

```
java com.cenqua.clover.reporters.pdf.PDFReporter -i clover.db -o coverage.pdf
```

Read coverage for the Clover database "clover.db", and produce a pdf report in the file "coverage.pdf"

```
java com.cenqua.clover.reporters.pdf.PDFReporter -c ElementsCoveredAsc
-t "My Coverage" -i clover.db -o coverage.pdf
```

Produce the same report as above, but include a report title, and order lists by total elements covered rather than percentage covered.

5.7. ConsoleReporter

Reports Code Coverage for the given coverage database to the console.

5.7.1. Usage

```
java com.cenqua.clover.reporters.console.ConsoleReporter [OPTIONS] PARAMS
```

5.7.2. Params

-i, --initstring <file>	The initstring of the coverage database.
-------------------------	--

5.7.3. 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.
-s, --span <interval>	Specifies how far back in time to include coverage recordings from since the last Clover build. See Using Spans . Defaults to 0 seconds.
-d, --debug	Switch logging level to debug
-v, --verbose	Switch logging level to verbose

5.7.4. 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.cenqua.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
}
```

5.7.5. Examples

```
java com.cenqua.clover.reporters.console.ConsoleReporter -i clover.db
```

Read coverage for the Clover database "clover.db", and produce a summary report to the console.

```
java com.cenqua.clover.reporters.xml.XMLReporter -l "method" -t "Method Coverage" -i c
```

Produce the same report as above, but include method-level coverage information, and a report title.

5.8. SwingViewer

Launches the Swing Viewer to allow interactive browsing of Code Coverage.

5.8.1. Usage

```
java com.cenqua.clover.reporters.jfc.Viewer [OPTIONS] PARAMS
```

5.8.2. Params

-i, --initstring <file>	The initstring of the coverage database.
-------------------------	--

5.8.3. Options

-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 since the last Clover build. See Using Spans . Defaults to 0 seconds.
-tw, --tabwidth <number>	Width to use when rendering tabs in source code.

5.8.4. API Usage

SwingViewer 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.cenqua.clover.reporters.jfc.Viewer;

...

String [] viewerArgs = { "-i", "clover.db" };
int result = Viewer.mainImpl(viewerArgs);
if (result != 0) {
    // problem
}
```

5.8.5. Examples

```
java com.cenqua.clover.reporters.jfc.Viewer -i clover.db
```

Launch the Swing Viewer reading the Clover database "clover.db".

For more information about using the Swing Viewer, see the [Using The Swing Viewer](#).

6. Advanced Usage

6.1. Background: The Clover Coverage Database

This section provides background information on the structure, lifecycle and management of the Clover database.

6.1.1. 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 table shows the various files 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 1.3.7 introduced a new 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.

6.1.2. 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.

6.2. Using Clover with Distributed Applications

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 setup is required.

When deploying you application in container environments, you should also check to ensure that Clover has sufficient permissions to function.

6.2.1. Background: 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 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.

6.2.2. Telling Clover how to find it's 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 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.
- 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.
- 3. Specify an Initstring at runtime via System properties**
You can override the Clover initstring at runtime via System Properties. Two 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 initstring 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 initstring 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.

Note:

For methods 2 and 3 above, as part of the test deployment process, 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. This needs to occur *after the Clover build is complete, and before you run your tests*. Once test execution is complete, you'll need to copy the coverage recording files from each remote machine to the initstring path on build machine to generate coverage reports.

6.2.3. Classpath Issues

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 particular environment being deployed to.

6.2.4. 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 2 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";
}
```

6.3. Flush Policies

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 that instrumented code is executing in. In the most common unit testing scenarios the default flushpolicy will suffice. In situations where instrumented code is executing in a hosted environment (like 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>///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). Note that no coverage recordings will be written if the hosting JVM is not shut down, or if the hosting JVM terminates abnormally.
interval	The interval 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 <clover-setup> , or IDE plugin guides). The interval mode is a "passive" mode in that flushing potentially occurs only while instrumented code is still being executed. 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. Any coverage not flushed in this manner will be flushed if/when the hosting JVM shuts down. The interval policy should be used in environments where shutdown of the hosting JVM is not practical and thread creation by Clover is not desired . If you don't mind Clover

	creating a thread, use the threaded policy. Runtime performance overhead is determined by the flush interval.
threaded	The threaded 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 <clover-setup> , or IDE plugin guides). The threaded mode starts a separate thread to perform flushes. The threaded policy should be used in environments where shutdown of the hosting JVM is not practical. Runtime performance overhead is determined by the flush interval.

6.4. Source Directives

Clover supports a number of 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.

6.4.1. Switching Clover on and off

```
///CLOVER:ON
///CLOVER:OFF
```

Switch Clover instrumentation on/off. This might be useful if you don't want Clover to instrument a section of code for whatever reason. Note that the scope of this directive is the current file only.

6.4.2. 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 [Flush Policies](#).

6.4.3. Change instrumentation strategy

Note:

This source directive has been deprecated and has no effect in Clover 1.3 and above.

```
///CLOVER:USECLASS
```

Clover will use a static holder class rather than a static member variable to support instrumentation for the current file. The directive must occur before the first top level class declaration in the file. This directive is useful when you don't want Clover to change the public interface of your class (in EJB compilation for example).

6.5. 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.

Note:

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

6.5.1. 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 are shown below.

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

6.5.2. 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

You can define your own method contexts via the `<methodContext>` subelement of [<clover-setup>](#), or via the configuration panel of your Clover IDE Plugin.

Note:

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

6.5.3. Statement Contexts

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.*`.

6.5.4. 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 logging statements

To remove logging statements for coverage reports, you'll 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.*">
</clover-setup>
```

This defines two statement contexts. 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
}
```

Once 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.

6.6. Using Spans

The span attribute allows you to control which coverage recordings are merged to form a current coverage report. By default, Clover only considers coverage recording files that were written after the last Clover compilation. In some situations you may want to include earlier 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.

6.7. Extracting coverage data programmatically

6.7.1. 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, via the [Swing Viewer](#), or using one of the [Clover IDE plugins](#). The following example XPath expressions show how to extract data from a Clover XML coverage report:

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

Extracts the total number of statements in the project.

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

Extracts the total number of uncovered statements in the project.

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

Extracts the total number of statements in the package `com.foo.bar`

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

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);

...
```

7. Tutorials

7.1. Using Clover with Ant and JUnit

7.1.1. Using Clover with Ant and JUnit

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. It is split into three parts covering Current Reports, Historical Reports and Advanced Features.

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 Clover, [Ant](#) and [JUnit](#) installed on your system, preferably the latest versions.

Instructions for installing Ant can be found in the [Apache Ant User Manual](#).

Instructions for installing Clover can be found in the [Installation Options](#) section.

For instructions on installing JUnit consult the [JUnit website](#). To allow JUnit to work with Ant, you must also copy `<JUNIT_HOME>/junit.jar` into `<ANT_HOME>/lib`.

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 source files for this tutorial are located in the standard Clover distribution, under the 'tutorial' directory. 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 and 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.

7.1.2. Part 1 - Measuring coverage with Clover

Introduction

Part 1 of the Clover Tutorial focuses on the creation and interpretation of 'Current' Clover 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 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 be compiling and unit-testing the Money library provided in the `tutorial/src` directory, then using 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.

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`.

Output should be similar to the following:

```
$ ant code
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`.

Output should be similar to the following:

```
$ ant test
Buildfile: build.xml

test:
  [java] .....
  [java] Time: 0.041

  [java] OK (22 tests)

BUILD SUCCESSFUL
Total time: 3 seconds
```

This shows that all the tests have been run and have passed.

Note:

To keep things simple we are not using the optional `<junit>` task that ships with Ant to run the JUnit tests. The `<junit>` task provides many advanced features for controlling the execution of unit tests and generating unit test reports. Modifying the `build.xml` file to use the `<junit>` task is left as an exercise for the reader.

We have now compiled the Money library, and run tests against it. In the next step, we'll add Clover targets and properties to the build file to enable measurement of code coverage.

Adding Clover targets

Now that we've 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

Load the `build.xml` file into your favourite text editor and add the Clover Ant task and type definitions:

```
<taskdef resource="clovertasks"/>
<typedef resource="clovertypes"/>
```

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 initString="demo_coverage.db"/>
</target>
```

The `initString` value defines the location of the Clover coverage database. During compilation, Clover stores information about all the artifacts in your sourcebase to this file. **If the database exists already, Clover updates it. If it doesn't exist, Clover will create a fresh database file.** When instrumented code is run, Clover uses this database to initialise itself and then writes coverage recording files alongside the database file.

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 bold below to the `build.classpath` Ant path:

```
<path id="build.classpath">
  <pathelement path="${ant.home}/lib/clover.jar"/>
  <pathelement path="${ant.home}/lib/junit.jar"/>
  <pathelement path="${build}"/>
</path>
```

Note:

This assumes that you have installed `clover.jar` in `ANT_HOME/lib`. If you've installed it elsewhere, adjust the path accordingly.

Once you've made these changes, you can save the `build.xml` file. We will add some more Clover targets later to perform coverage reporting, but first we'll re-compile the Money library with Clover and re-run the tests to obtain coverage data.

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`.

You will get output similar to the following:

```
$ ant with.clover code
Buildfile: build.xml
with.clover:

compile:
  [mkdir] Created dir: C:\clover\tutorial\build
  [javac] Compiling 4 source files to C:\tutorial\build
  [clover] Clover Version 1.x, built on ...
  [clover] No coverage database 'C:\clover\tutorial\demo_coverage.db'
found. Creating a fresh one.
  [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.

Running the tests

We now need to run the tests again (with the command `ant test`). This will run the tests, this time measuring coverage. Output from Ant will be the same as a normal test run:

```
$ ant test
Buildfile: build.xml
run:
  [java] .....
  [java] Time: 0.08
  [java] OK (22 tests)
BUILD SUCCESSFUL
Total time: 4 seconds
```

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.

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 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="report.html" depends="with.clover">
```

```
<clover-report>
  <current outfile="clover_html" title="Clover demo">
    <format type="html"/>
  </current>
</clover-report>
</target>
```

The `<current>` element specifies that the type of report to be produced is a snapshot report of 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 [Tutorial Part 2](#)). 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 report.html`. You will get output similar to the following:

```
$ ant report.html
report.html:
  [java] Clover Version 1.x, built on ...
  [java] Reading data for database at
  'c:\clover\tutorial\demo_coverage.db'
  [java] Writing Html report to 'c:\clover\tutorial\clover_html'
  [java] Done. Processed 1 packages.

BUILD SUCCESSFUL
Total time: 3 seconds
```

You can now view the report by opening the file `tutorial\clover_html\index.html` in a web browser. The next few sections of the tutorial will show you how to interpret the report and use it to improve the Money library unit tests.

Interpreting the report

We will now look at how to interpret the HTML report that you generated in the previous step.

The screenshot below shows the generated HTML report in a browser. 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), Number of Methods, Number of Classes, Number of Files and Number of Packages.

The screenshot 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](#) [All Classes](#)

All Packages
[default-pkg](#) (91.9%)

All Classes
[Money](#) (89.4%)
[MoneyBag](#) (87.2%)
[MoneyTest](#) (100%)

Clover coverage report - clover demo
Coverage timestamp: Fri Dec 20 2002 17:54:20 EST

[Overview](#) [Package](#) [File](#)

file stats: LOC: 73 Methods: 14
NCLOC: 58 Classes: 1

Your 30 day evaluation period has expired. Please visit <http://www.thecortex.net/clover> to obtain a licensed version of Clover.

Source file	Conditionals	Statements	Methods	TOTAL
Money.java	62.5%	92%	100%	89.4%

```

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

```

coverage measured for the Money class

The left-most column shows line numbers and those that contain executable content are highlighted in blue. 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	12	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		}	

code not executed

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.

In the next step, we will enhance the JUnit tests to improve code coverage of the Money library.

Improving coverage

After having a look at the coverage report generated in the last step, 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.

32		}
33	341	<code>public int amount() {</code>
34	341	<code> return fAmount;</code>
35		<code>}</code>
36	415	<code>public String currency() {</code>
37	415	<code> return fCurrency;</code>
38		<code>}</code>
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> && amount() == aMoney.amount();</code>
47		<code> }</code>
48	3	<code> return false;</code>
49		<code>}</code>

lines not covered in money class

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 `MoneyTest.java` file.

Declare the variable `f0USD`:

```
public class MoneyTest 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 (by running `ant with.clover` code) and run the tests again (by running `ant test`) and then re-generate the HTML report (by running `ant report.html`). You will see that the `Money` class now has 100% coverage.

7.1.3. Part 2 - Historical Reporting

Introduction

Part 2 of the Clover Tutorial focuses on the creation and interpretation of 'Historical' Clover reports. 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'll edit the Ant build file to generate a History Point.

Creating history points

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. The generation of historical reports is discussed in later sections. In the meantime, this section demonstrates how to set up the relevant Ant target and run the command so that a history point can be created.

Adding a history point target

Add the following target to your `build.xml` file:

```
<target name="record.point" depends="with.clover">
  <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 `historyDir` is the directory where the history points will be stored. You should create this directory before executing this target.

Note:

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 with the commands `ant with.clover` and `ant test` respectively.

Run the command `ant record.point`. Output should be similar to the following:

```
$ ant record.point
Buildfile: build.xml

with.clover:

record.point:
[clover-historypoint] Clover Version 1.x, built on ...

[clover-historypoint] Merged results from 2 coverage recordings.
[clover-historypoint] Writing report to
'C:\tutorial\clover_history\clover-20030307111326.xml'
[clover-historypoint] Done.

BUILD SUCCESSFUL
Total time: 2 seconds
```

In the next step we'll add more tests to improve coverage of the Money Library, recording Clover history points along the way.

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, however there are several sections of code that remain untested in the MoneyBag.java file. These uncovered lines of code are shown below in red. This section will focus on bringing the coverage of this class to 100% as well as creating historical data in the form of history points.

```

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 12              for (Enumeration e= fMonies.elements(); e.hasMoreElements(); ) {
62 22                  Money m= (Money) e.nextElement();
63 22                  if (!aMoneyBag.contains(m))
64 2          return false;

```

money not covered 1

```

117 4      public IMoney subtract(IMoney m) {
118 4          return add(m.negate());
119
120 0      public String toString() {
121 0          StringBuffer buffer = new StringBuffer();
122 0          buffer.append("(");
123 0          for (Enumeration e= fMonies.elements(); e.hasMoreElements(); )
124 0              buffer.append(e.nextElement());
125 0          buffer.append(")");
126 0          return buffer.toString();
127
128 12      public void appendTo(MoneyBag m) {
129 12          m.appendBag(this);
130
131

```

money not covered 2

Open the source file MoneyTest.java in your favourite text editor and make the following additions shown in bold:

Declare the variables f0CHF and fMB3:

```

public class MoneyTest 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 with.clover` code and `ant test` 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, reinstrument 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.

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" depends="with.clover">
  <clover-report>
    <historical outfile="historical.pdf"
               historyDir="clover_history"/>
  </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. 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
```

```
with.clover:
hist.report:
[clover-report] Clover Version 1.x, built on ...
[clover-report] Writing report to 'C:\tutorial\historical.pdf'
[clover-report] Merged results from 2 coverage recordings.
[clover-report] Done. Processed 1 packages.
[clover-report] Writing historical report to 'C:\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](#). We'll look at how to interpret this report in the next section.

Interpreting historical reports

We will now look at interpreting the report that you generated in the previous step by enabling the report in an appropriate PDF viewer. When you view the report you should see a picture similar to the screenshot below, although it is likely that the the graphs that you produce will contain different values.

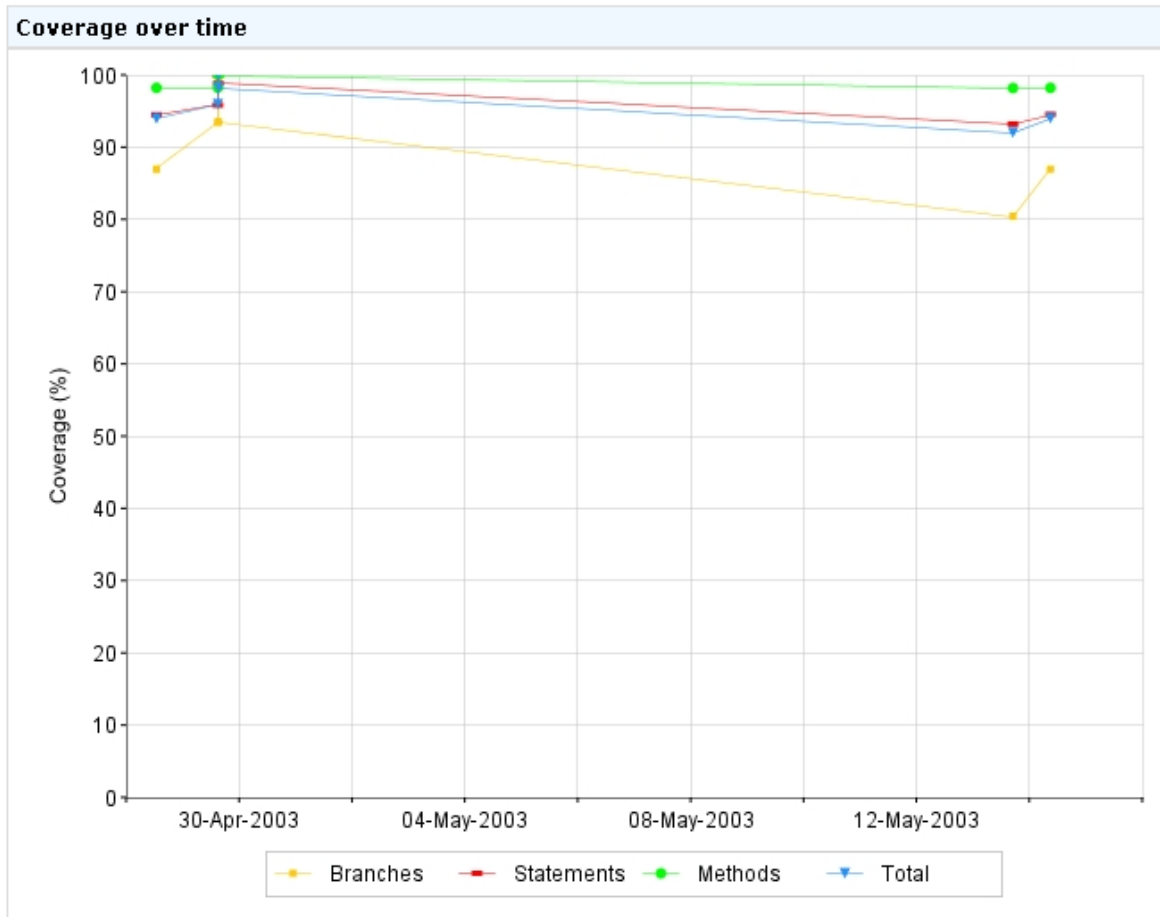
Like the 'current' report, the historical report begins with a header containing relevant project information. This includes the report title, the project metrics and the period for which history points are included in the report. Below this header is the Project Overview Chart which shows the branch, 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.

Clover 1.3.14 User Manual

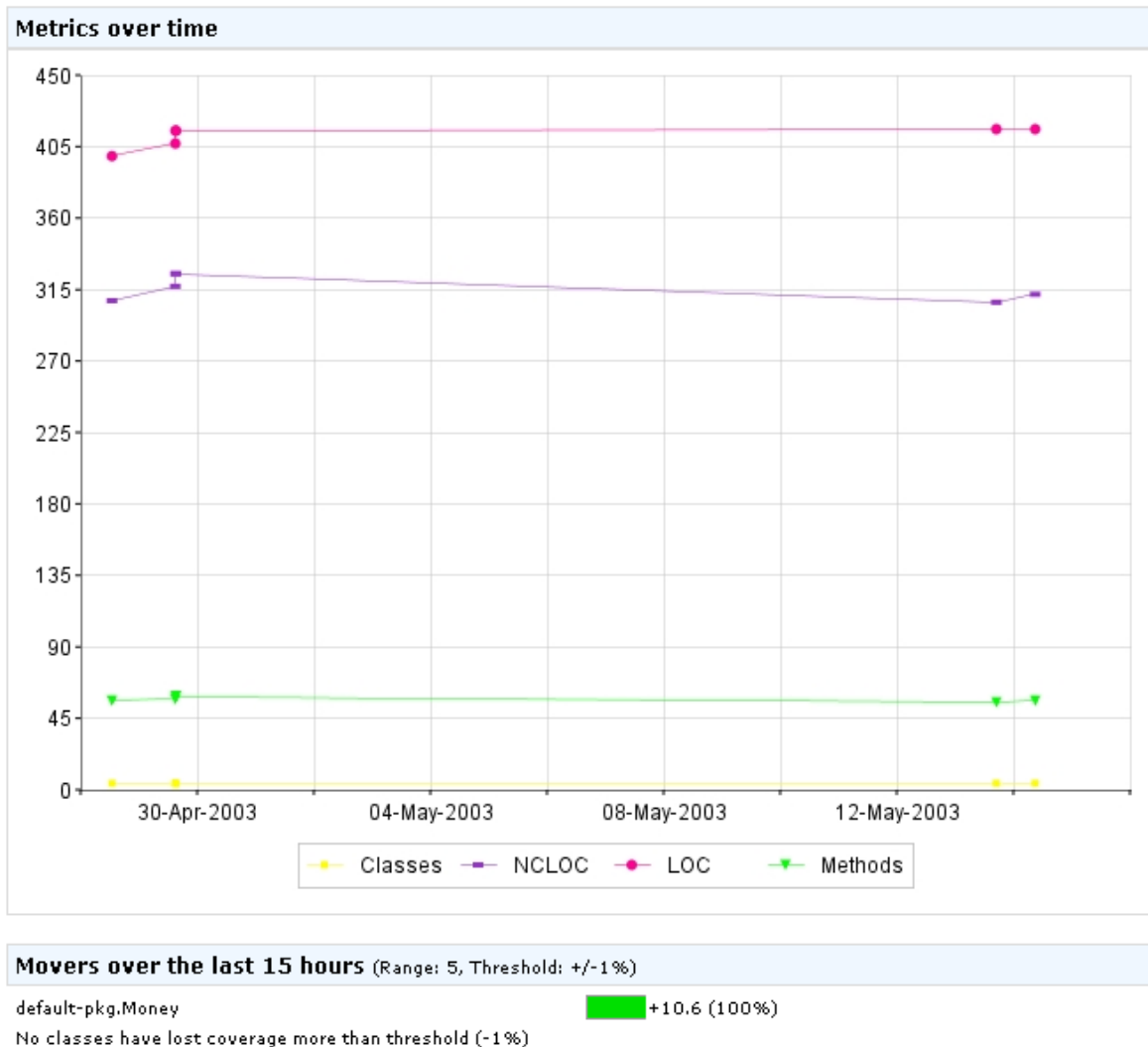
Historical coverage report - Money Demo Coverage		project stats:	LOC: 416	Methods: 56
From: Mon Apr 28 2003 12:54:03 EST			NCLOC: 312	Classes: 4
To: Wed May 14 2003 09:00:28 EST			Files: 4	

	Conditionals	Statements	Methods	TOTAL
Project	87%	94.5%	98.2%	94%



historical chart overview and coverage

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 below, the number of methods can be seen shown in green.



metrics and movers

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 1 percentage point over the two latest history points. In this case there have not been any classes which have lost more than 1 percentage point coverage, hence the only item displayed here is the Money package which has gained 10.6 percentage points coverage over the two latest history points.

The next section of this tutorial will discuss how you can customise many aspects of the

historical report.

Customising historical reports

In the previous sections of this tutorial we've 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" depends="with.clover">
  <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" depends="with.clover">
  <clover-report>
    <historical outfile="histCoverage.pdf"
      title="My Project"
      historyDir="clover_history">
      <overview/>
      <coverage/>
    </historical>
  </clover-report>
</target>
```

```

        </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

The 'Coverage over time' and 'Metrics over time' charts also allow you to choose which metrics information should be included. The default elements for the coverage chart are branches, statements, methods and total, while the default elements for the metrics chart are loc, ncloc, methods and classes. By using the `include` attribute you can specify the required configuration:

```

<target name="hist.report.select" depends="with.clover">
    <clover-report>
        <historical outfile="histSelect.pdf"
            title="My Project"
            historyDir="clover_history">
            <coverage include="total"/>
            <metrics include="methods, packages"/>
        </historical>
    </clover-report>
</target>

```

This will produce a PDF file with the filename 'histSelect.pdf' with two sections: the 'Coverage over time' chart with total coverage information; and the 'Metrics over time' chart with method and package information. You can also specify whether or not a chart uses a log scale by adding the `logscale` attribute:

```

    <metrics include="methods, packages" 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" depends="with.clover">
    <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.

7.1.4. Part 3 - Advanced Features

Introduction

This section looks at some advanced features of Clover.

- [Automating coverage checking](#)

Automating coverage checking

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.

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.haltOnfail" 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 comes in 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 from 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 clover coverage percentage check as normal but will calculate coverage with omission of all 'catch' blocks.

8. Miscellaneous

8.1. Swing Viewer

8.1.1. Overview

The Swing Viewer is a standalone coverage viewer that allows you to browse coverage results and generate coverage reports.

Launching the viewer from Ant

Add the following target to your build file:

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

The viewer can then be launched with `ant clover.view`.

Note:

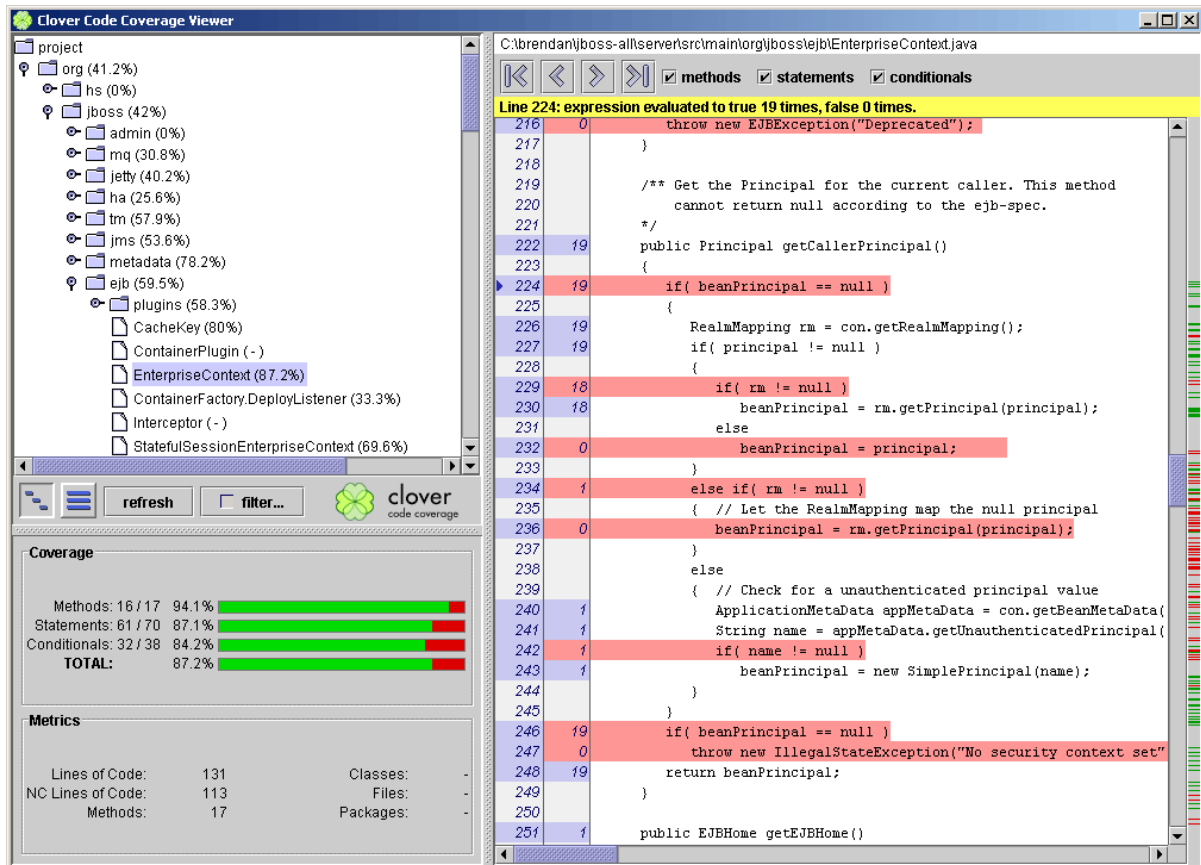
This assumes you have added a `with.clover` target to your build that initialises Clover. See the for more details.

Launching the viewer from the Command Line

To launch the Swing viewer from the command line:

```
java com.cenqua.clover.reporters.jfc.Viewer <initstring>
```

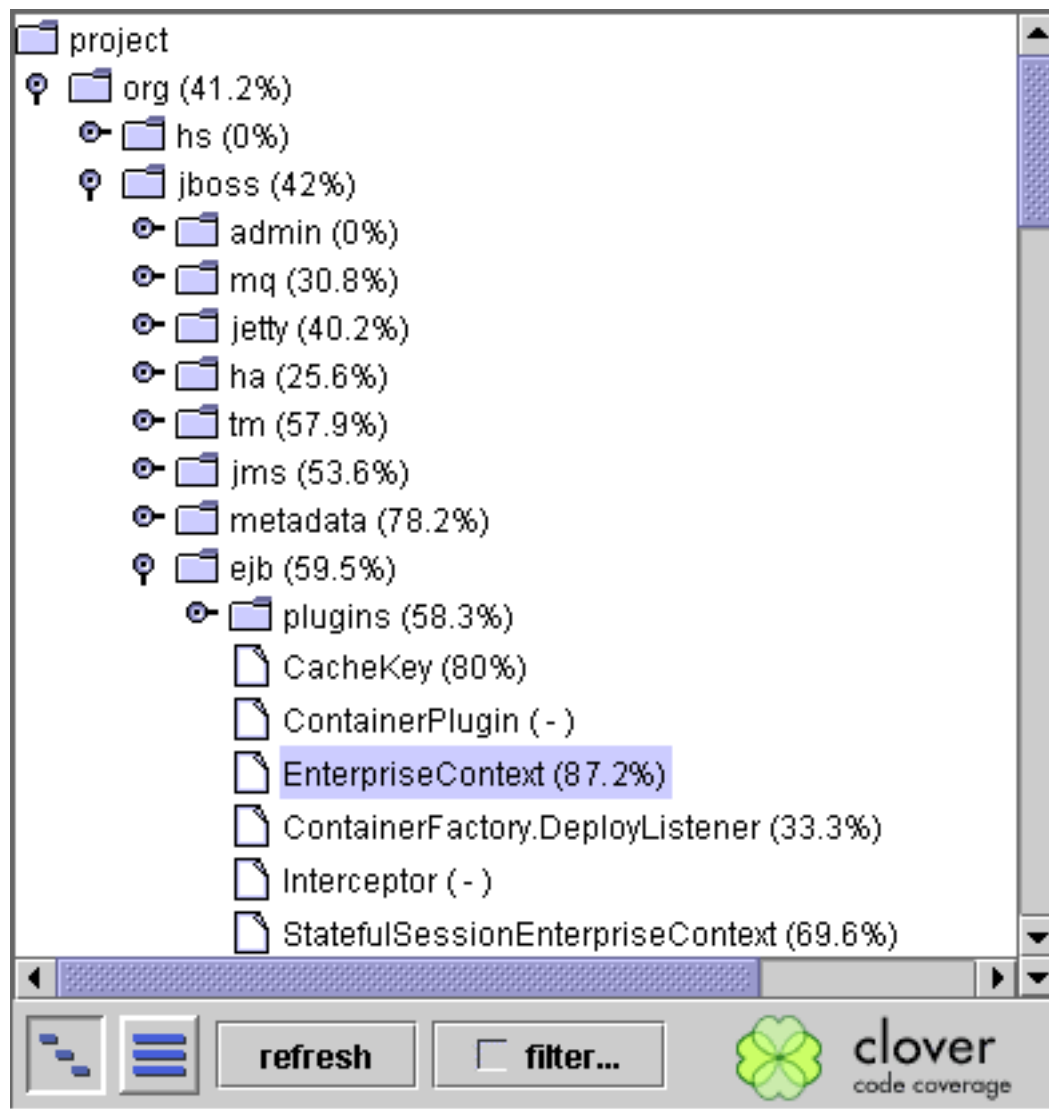
The swing viewer will appear in a new frame which will look like the following (sections of this image are displayed in greater detail below):



swing viewer screenshot

Package View

In the top left hand corner you will see the Project Package View which gives you a quick snapshot of the coverage percentage of each package. The two buttons on the left below the Package View allow you to choose between a nested view of the packages or a flat view. The button on the left will display the packages in a hierachical structure whereas the button to the right will display each package separately.



project package view

The 'refresh' button will reread coverage data and will update the display accordingly. This allows you to change, re-compile and test code while the swing viewer is running and then instantly see the new code coverage results.

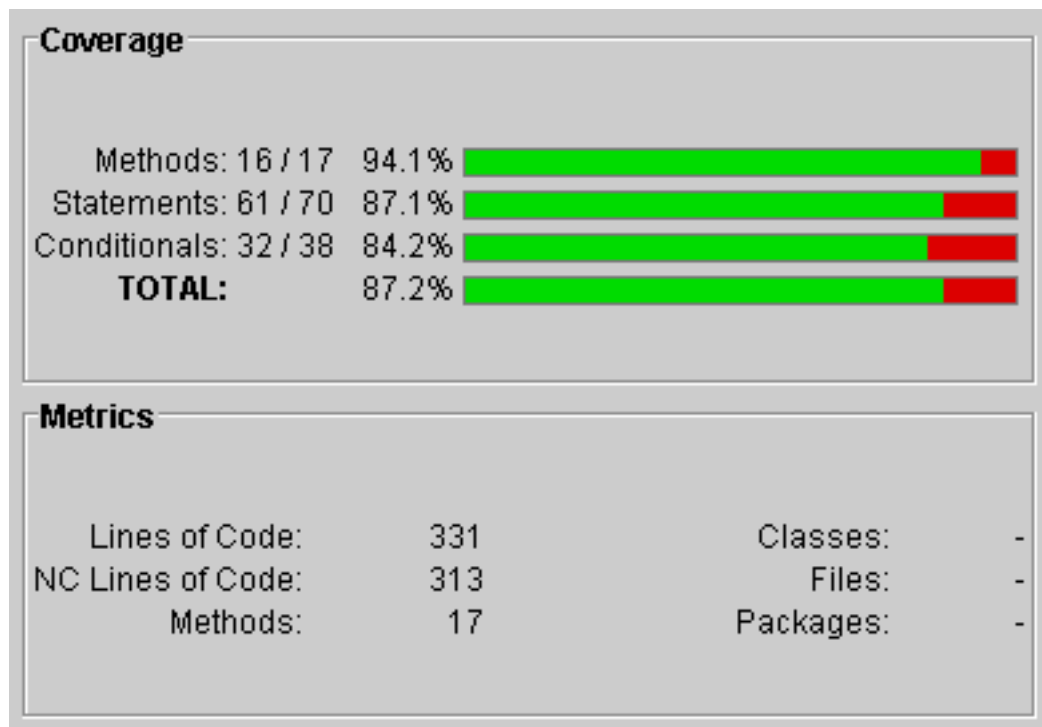
Clicking on the filter button opens up a new frame that allows you to filter the displayed coverage. The context filter allows you to select certain blocks to ignore when calculating coverage, and the coverage filter allows you to specify a level of coverage that needs to be achieved before the class is displayed. Once the filtering has been selected, click 'Apply' to

see the results. 'Reset' will return to the default settings, and 'Cancel' will leave the settings as they were.

Double-clicking on a package or selecting the icon to the left of the package name will display all the files that exist within that package. These files can then be selected and the file will appear in the window to the right for closer examination (see Code View section below).

Coverage and Metrics

Depending on the current selection in the Package View, the relevant coverage details and statistics will be displayed in the two sections below the project packages.



coverage and metrics details

The coverage details show the method, statement and conditional coverage. The statistics in the bottom left give the metrics of the selection in the project package section and provide details such as the number of lines of code, number of classes, etc.

Code View

The window on the right of the Swing Viewer (shown below), which displays your selected

file, allows you to see exactly which sections of your code remain uncovered, much like the HTML Reporter. The name and location of the file are shown at the top of this window, and clicking on the left and right arrows below this allow you to cycle through the coverage of one file. The check boxes beside this can be used to omit or include method, statement or conditional coverage.

C:\brendan\jboss-all\server\src\main\org\jboss\ejb\EnterpriseContext.java

☐ methods
 ☒ statements
 ☒ conditionals

Line 224: expression evaluated to true 19 times, false 0 times.

Line	Count	Code
216	0	throw new EJBException("Deprecated");
217		}
218		
219		/** Get the Principal for the current caller. This method
220		cannot return null according to the ejb-spec.
221		*/
222	19	public Principal getCallerPrincipal()
223		{
224	19	if(beanPrincipal == null)
225		{
226	19	RealmMapping rm = con.getRealmMapping();
227	19	if(principal != null)
228		{
229	18	if(rm != null)
230	18	beanPrincipal = rm.getPrincipal(principal);
231		else
232	0	beanPrincipal = principal;
233		}
234	1	else if(rm != null)
235		{ // Let the RealmMapping map the null principal
236	0	beanPrincipal = rm.getPrincipal(principal);
237		}
238		else
239		{ // Check for a unauthenticated principal value
240	1	ApplicationMetaData appMetaData = con.getBeanMetaData(
241	1	String name = appMetaData.getUnauthenticatedPrincipal(
242	1	if(name != null)
243	1	beanPrincipal = new SimplePrincipal(name);
244		}
245		}
246	19	if(beanPrincipal == null)
247	0	throw new IllegalStateException("No security context set"
248	19	return beanPrincipal;
249		}
250		
251	1	public EJBHome getEJBHome()

class code view

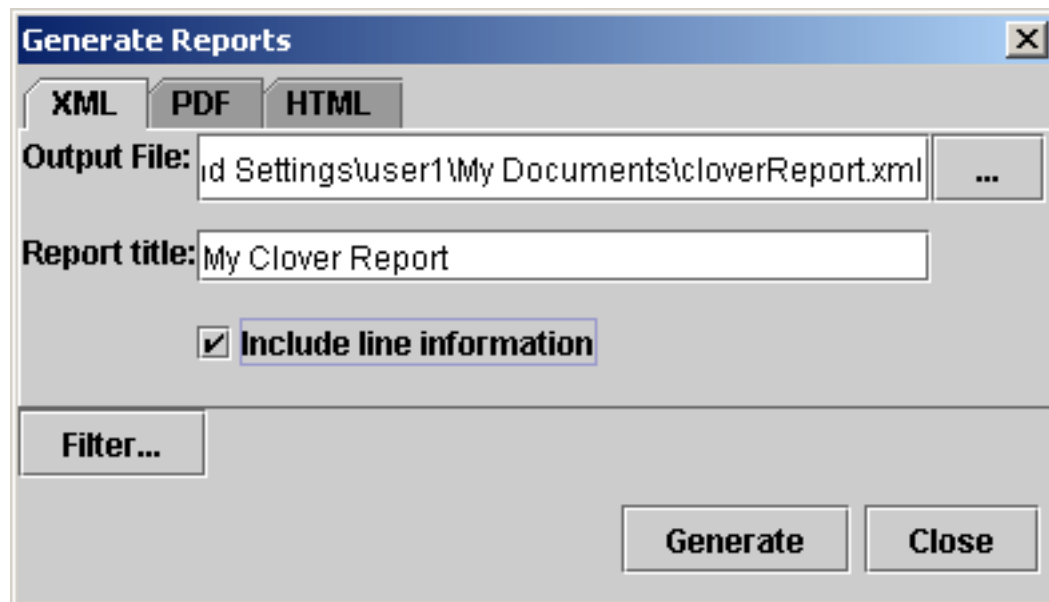
The non-comment lines of code have their numbers highlighted in blue and beside this is the number of times a line has been executed. This second number is highlighted in red if the line has never been executed, and blue otherwise.

The 'quick jump' bar on the furthest right of this window highlights lines that have not been covered by the testing. Clicking on the dashes beside these lines allows you to instantly skip to the uncovered sections of code.

8.1.2. Generating Reports

The Swing Viewer can be used for generating other types of reports.

To generate reports, click on 'File' and select 'Generate other reports'. This will bring up a new frame allowing you to choose which sort of report you want to generate and also configure other options relevant to specific reports.



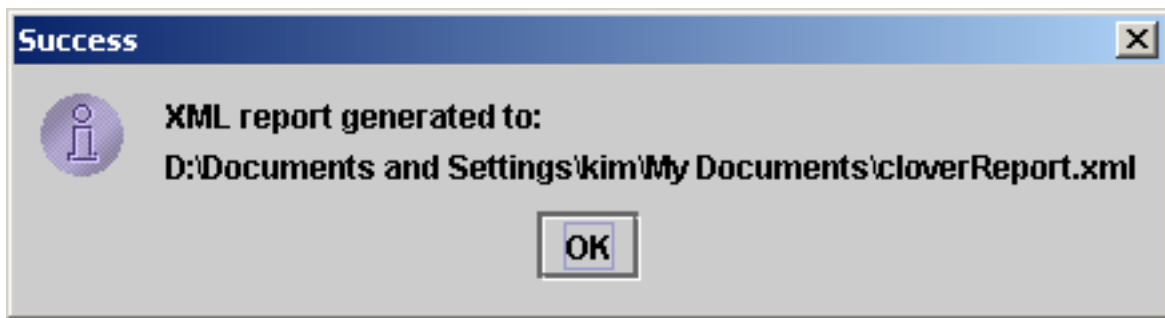
report generation options

For all three report types that you can generate, you must specify an output path. For XML/PDF, this is a file, and for html, this is a directory. You can also add a report title if you wish. By clicking the 'Filter' button you can again select specific blocks to exclude when generating the report.

The XML report gives you the option of including line information which details the line number, the line type (method/statement/conditional) and the execution count (in the case of a conditional, the true count and the false count).

When generating the HTML report you can select whether or not you want the source files to be shown, including coverage information, by checking the 'Show source' check box. You can also choose to sort the classes alphabetically, by ascending coverage, or descending coverage.

To generate the report simply click 'Generate' and a pop-up will be displayed saying the type of report generated and the path of that report. If, for instance, you select an invalid path, a relevant error message will be displayed detailing the problem.



successful generation

You can now view the generated report by opening it in a relevant application.

8.2. Interval Format

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	Abbrev.	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 (eg. 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.

8.3. Frequently Asked Questions

8.3.1. Questions

1. General

- [Can't find an answer here?](#)
- [What is Code Coverage Analysis?](#)
- [What are the limitations of Code Coverage?](#)
- [Where did Clover originally come from?](#)
- [Why the name "Clover"?](#)

2. Technical Background

- [Does Clover depend on JUnit?](#)
- [Does Clover work with JUnit4 and TestNG?](#)
- [Why does Clover use Source Code Instrumentation?](#)
- [Will Clover integrate with my IDE?](#)
- [Does Clover integrate with Maven?](#)
- [What 3rd Party libraries does Clover utilise?](#)
- [How are the Clover coverage percentages calculated?](#)
- [Does Clover support the new language features in JDK1.5?](#)

3. Troubleshooting

- [Two questions to ask yourself first when troubleshooting Clover:](#)
- [When using Clover from Ant, why do I get "Compiler Adapter 'org.apache.tools.ant.taskdefs.CloverCompilerAdapter' can't be found." or similar?](#)
- [When using Clover, why do I get a java.lang.NoClassDefFoundError when I run my code?](#)
- [When generating some report types on my unix server with no XServer, I get an exception "Can't connect to X11 server" or similar.](#)
- [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 an java.lang.OutOfMemoryError when compiling with Clover turned on?](#)
- [For some statements in my code Clover reports "No Coverage information gathered for this expression". What does that mean?](#)
- [Why does Clover instrument classes I have excluded using the <exclude> element of the <clover-setup> task?](#)
- [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?](#)

8.3.2. Answers

1. General

1.1. Can't find an answer here?

Try our [Online Forums](#), or [contact us directly](#).

1.2. 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.

A good introduction to the various types of Code Coverage Analysis can be found [here](#).

1.3. 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.

1.4. Where did Clover originally come from?

Clover was originally developed at Cenqua 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 specialized 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.

1.5. Why the name "Clover"?

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").

2. Technical Background

2.1. 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 dev 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.

2.2. Does Clover work with JUnit4 and TestNG?

Clover is fully compatible with JUnit4 and TestNG.

2.3. 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	JVMDI/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 specialized 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

2.4. Will Clover integrate with my IDE?

Clover provides integrated plugins for IntelliJ IDEA 4.x and 5.x, NetBeans, and Eclipse, JBuilder and JDeveloper. Clover should also work happily with any IDE that provides integration with the Ant build tool.

2.5. Does Clover integrate with Maven?

There is a Clover Plugin for Maven and Maven2 - both are independent open source developments supported by Cenqua. See the Maven and Maven2 websites for details.

2.6. What 3rd Party libraries does Clover utilise?

Clover makes use of the following excellent 3rd party libraries:

Jakarta Velocity 1.2	Templating engine used for Html report generation.
Antlr 2.7.1	A public domain parser generator.
iText 0.96	Library for generating PDF documents.
Jakarta Ant	The Ant build system.

Note:

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 that may use different versions of these libraries in their projects.

2.7. 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:

$$\text{TPC} = (\text{CT} + \text{CF} + \text{SC} + \text{MC}) / (2 * \text{C} + \text{S} + \text{M})$$

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

2.8. Does Clover support the new language features in JDK1.5?

Clover fully supports all JDK1.5 language features.

3. Troubleshooting

3.1. 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](#).

3.2. 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](#)

3.3. 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](#)

3.4. 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 JDK1.4, 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>.

3.5. 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 a [inline directive](#)). The simplest thing to do is to use the fork="true" attribute when running your tests. The tests will be then run in their own JVM, and the coverage data will be flushed when the that JVM exits. Alternatively, you can use interval-based flushing by changing the [Flush Policy](#).

3.6. Why do I get an java.lang.OutOfMemoryError when compiling with Clover turned on?

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"/>
```

3.7. 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.

3.8. 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 patternsets to manage the includes and excludes specified in the `clover-setup` task. By default Ant does not pass these patternsets 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 patternsets 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.

3.9. 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 SysAdmin. 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"/>
```