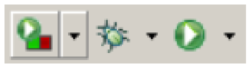
EclEmma Java Code Coverage Tool

Launching in Coverage Mode

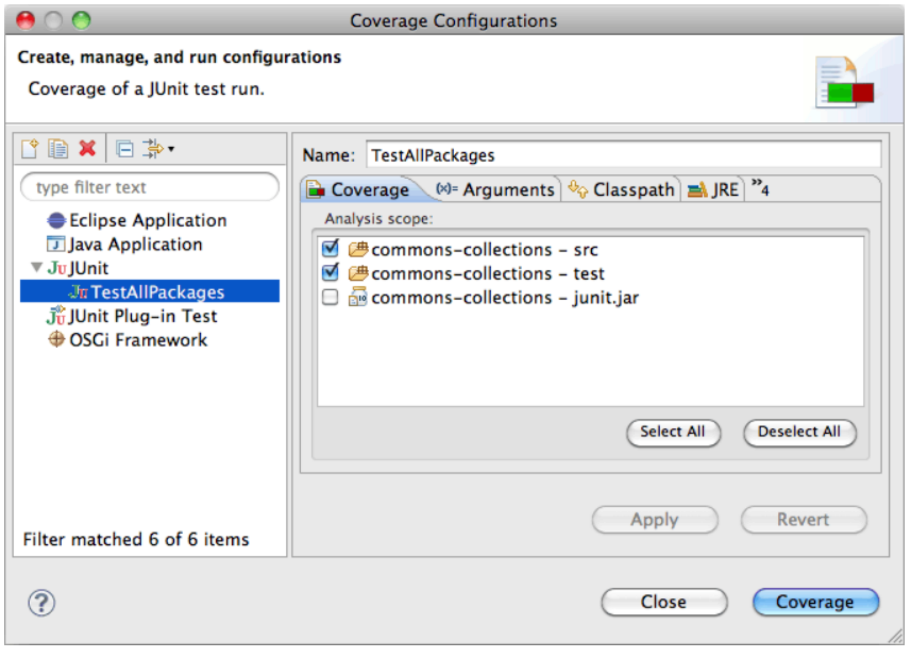
Eclipse allows running Java programs directly from the workbench. Programs can be launched in different so called launch modes. In a standard Eclipse installation you can launch your programs either in *Run* or in Debug mode. EclEmma adds a new launch mode Coverage which is available from the Run menu and the toolbar:



Note: If the Coverage drop-down toolbar button is not visible in your current workbench perspective, open the Customize Perspective... dialog and enable the Coverage command group on the Commands tab.
Currently the following launch types are supported:

- Local Java application
- Eclipse/RCP application
- Equinox OSGi framework
- JUnittest
- TestNG test
- JUnit plug-in test
- JUnit RAP test
- SWTBottest
- Scala application

Existing launch configurations can be launched directly in Coverage mode using default settings. As with the Run and Debug mode you might also select a Java element and launch it directly from the Coverage As context menu. If required some settings can be modified in the coverage launch dialog:



Coverage Configurations

Create, manage, and run configurations

Coverage of a JUnit test run.

Name: TestAllPackages

Analysis scope:

- [✓] commons-collections - src
- [✓] commons-collections - test
- [ ] commons-collections - junit.jar

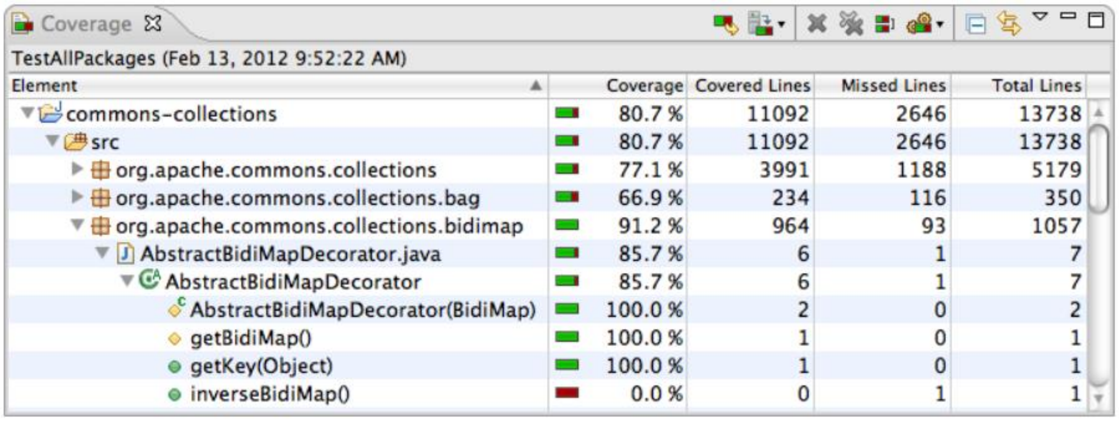
In the *Coverage* tab the Java class path entries for code coverage analysis can be selected. At least one entry must be selected to run an application in Coverage mode. The rules which class path entries are selected by default can be adjusted in the code coverage preferences.

Coverage Data Collection

Coverage data is collected and presented automatically when the application terminates on its own. If the Java VM is killed externally, e.g. with the Eclipse *Terminate* action, coverage results can not be shown. In addition intermediate coverage data dumps can also be requested from any process running in Coverage mode. Such dumps can be triggered from the toolbar of the Coverage view.

Using the Coverage View

The Coverage view automatically appears when a new coverage session is added or can manually opened from the *Window* —> *Show View* menu in the Java category. It shows coverage summaries for the active session.



Coverage

TestAllPackages (Feb 13, 2012 9:52:22 AM)

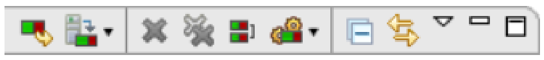
Element	Coverage	Covered Lines	Missed Lines	Total Lines
commons-collections	80.7 %	11092	2646	13738
src	80.7 %	11092	2646	13738
org.apache.commons.collections	77.1 %	3991	1188	5179
org.apache.commons.collections.bag	66.9 %	234	116	350
org.apache.commons.collections.bidimap	91.2 %	964	93	1057
AbstractBidiMapDecorator.java	85.7 %	6	1	7
AbstractBidiMapDecorator	85.7 %	6	1	7
AbstractBidiMapDecorator(BidiMap)	100.0 %	2	0	2
getBidiMap()	100.0 %	1	0	1
getKey(Object)	100.0 %	1	0	1
inverseBidiMap()	0.0 %	0	1	1

The *Coverage* view shows all analyzed Java elements within the common Java hierarchy. Individual columns contain the following numbers for the active session: always summarizing the child elements of the respective Java element:

- Coverage ratio
- Items covered
- Items not covered
- Total items

The elements may be sorted in ascending or descending order by clicking the respective column header. Double-clicking an element opens its declaration in an editor with highlighted source code. You can select between different metrics, see last section for details.

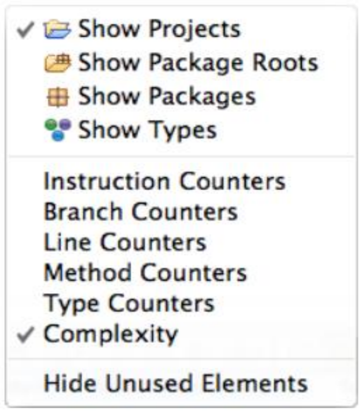
Toolbar and Drop-Down Menu



The coverage view's toolbar offers the following actions:

Coverage Last Launched: Re-run the currently selected coverage session.
Dump Execution Data: Dump execution data from a running process and create a new session from the data. Only active when at least one process is running in Coverage mode.
Remove Active Session: Remove the currently selected coverage session.
Remove All Sessions: Remove all coverage sessions.
Merge Sessions: Merges multiple sessions into a single one.
Select Session: Select session from the drop down-menu and make it the active session.
Collapse All: Collapse all expanded tree nodes.
Link with Current Selection: If this toggle is checked the coverage view automatically reveals the Java element currently selected in other views or editors.

Some of the actions are deactivated if there is no session or only a single session. More settings are available form the coverage view's drop-down menu:



- Show Elements: Select Java elements shown as root entries in the coverage tree: Projects, package fragment roots (source folders or libraries), package fragments or types.
- Counter Mode: Different counter modes can be selected from the view's drop-down menu: bytecode instructions, branches, lines, methods, types and cyclomatic complexity. Please see JaCoCo documentation for detailed counter definitions.
- Hide Unused Elements: Filter all elements from the coverage view that have not been executed at all during the coverage session.
-

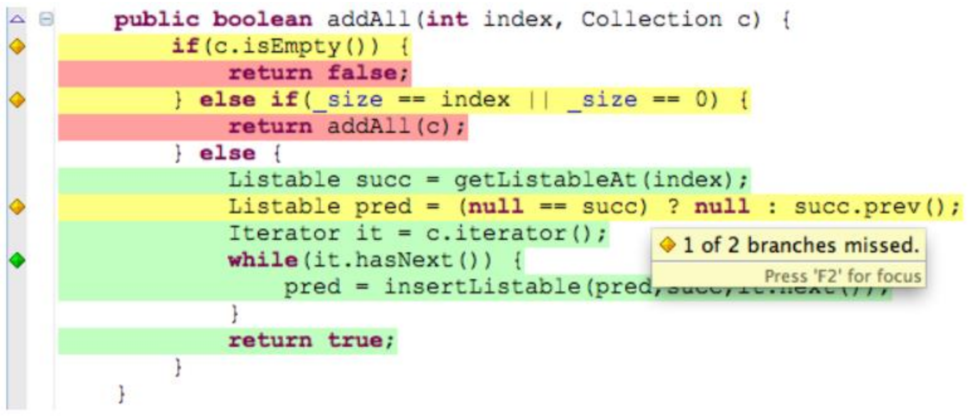
Filtering

If you’re working on a particular unit you can filter out all classes which have not been loaded during the test run. This filtering can be enabled with the Hide Unused Types menu entry in the Coverage view's drop-down menu.
Tip: Combine the Hide Unused Types option with types as root entries. This will give you a plain list of all classes loaded for your test case.

Source Code Annotation

Line coverage and branch coverage of the active coverage session is also directly displayed in the

Java source editors. This works for Java source files contained in the project as well as source code attached to binary libraries.



```
public boolean addAll(int index, Collection c) {
    if(c.isEmpty()) {
        return false;
    } else if(_size == index || _size == 0) {
        return addAll(c);
    } else {
        Listable succ = getListableAt(index);
        Listable pred = (null == succ) ? null : succ.prev();
        Iterator it = c.iterator();
        while(it.hasNext()) {
            pred = insertListable(pred, succ, it.next());
        }
        return true;
    }
}
```

1 of 2 branches missed.

Source lines containing executable code get the following color code:
- green for fully covered lines,
- yellow for partly covered lines (some instructions or branches missed) and
- red for lines that have not been executed at all.

In addition colored diamonds are shown at the left for lines containing decision branches. The colors for the diamonds have a similar semantic than the line highlighting colors:
- green for fully covered branches,
- yellow for partly covered branches and
- red when no branches in the particular line have been executed.

These default colors can be modified in the Preferences dialog (see next section). The source annotations automatically disappear when you start editing a source file or delete the coverage session.

Highlighting Preferences

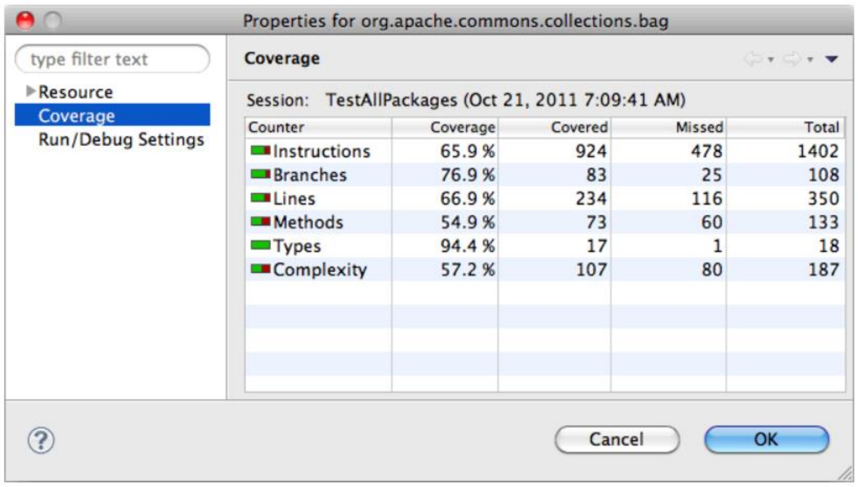
The Eclipse preferences section *General* —> *Appearance* —> *Editors* —> *Text Editors* —> *Annotations* allows to modify the visual representation of coverage highlighting. The corresponding entries are:
- Full Coverage
- Partial Coverage
- No Coverage

Remarks about Line Coverage

In some situations it is not obvious, why particular lines do have highlighting or have a particular color. The reason is that the underlying code coverage library JaCoCo works on Java class files only. In some cases the Java compiler creates extra byte code for a particular line of source code. Such situations might be filtered by future versions of JaCoCo/EclEmma.

Coverage Properties

For each Java element (Java project, source folder package, type or method) EclEmma provides a Coverage property page summarizing all coverage counters:



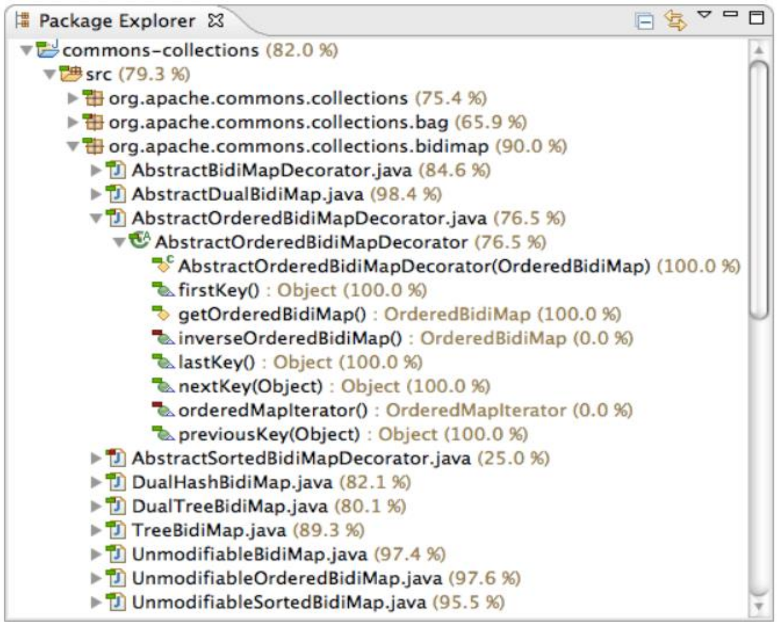
Session: TestAllPackages (Oct 21, 2011 7:09:41 AM)

Counter	Coverage	Covered	Missed	Total
Instructions	65.9 %	924	478	1402
Branches	76.9 %	83	25	108
Lines	66.9 %	234	116	350
Methods	54.9 %	73	60	133
Types	94.4 %	17	1	18
Complexity	57.2 %	107	80	187

The *Properties* dialog can be activated from the context menu of for example the Package Explorer or Coverage view. The figures are only available if there is a active coverage session, e.g. after a coverage launch.

Coverage Decorators

Note: This is an optional feature not enabled by default.
The Eclipse workbench has the concept of so called decorators which add graphical and textual information to elements shown in the workbench views. EclEmma provides coverage decorators for the currently active coverage session: A little green/red bar on the elements' icons and a percentage value next to the names.



```
commons-collections (82.0 %)
  src (79.3 %)
    org.apache.commons.collections (75.4 %)
    org.apache.commons.collections.bag (65.9 %)
    org.apache.commons.collections.bidimap (90.0 %)
      AbstractBidiMapDecorator.java (84.6 %)
      AbstractDualBidiMap.java (98.4 %)
      AbstractOrderedBidiMapDecorator.java (76.5 %)
        AbstractOrderedBidiMapDecorator (76.5 %)
          AbstractOrderedBidiMapDecorator(OrderedBidiMap) (100.0 %)
          firstKey() : Object (100.0 %)
          getOrderedBidiMap() : OrderedBidiMap (100.0 %)
          inverseOrderedBidiMap() : OrderedBidiMap (0.0 %)
          lastKey() : Object (100.0 %)
          nextKey(Object) : Object (100.0 %)
          orderedMapIterator() : OrderedMapIterator (0.0 %)
          previousKey(Object) : Object (100.0 %)
      AbstractSortedBidiMapDecorator.java (25.0 %)
      DualHashBidiMap.java (82.1 %)
      DualTreeBidiMap.java (80.1 %)
      TreeBidiMap.java (89.3 %)
      UnmodifiableBidiMap.java (97.4 %)
      UnmodifiableOrderedBidiMap.java (97.6 %)
      UnmodifiableSortedBidiMap.java (95.5 %)
```

The percentage shown is calculated based upon the instruction counters. Coverage decorators are only visible if there is an active coverage session and only shown for elements containing executable code, therefore e.g. not for abstract methods.
This optional feature can be enabled in the Eclipse preferences dialog:

- Open the preferences dialog from *Window*—> *Preferences*...
- Navigate to page *General* —> *Appearance* —> Label *Decorators*
- Select *Java Code Coverage* and press OK

Managing Coverage Sessions

A *coverage session* is the code coverage information of particular program run. It contains the list of considered Java classes along with the recorded coverage details.

Session Lifecycle

A coverage session is automatically created at the end of each coverage launch or whenever an intermediate execution data has been trigger by the user. Alternatively, sessions can be imported from external launches. The coverage view allows removing sessions.
All coverage sessions are deleted when the workbench is closed.

The Active Session

Even if there can be multiple coverage sessions, only one session can be the active coverage session. The active session can be selected from a drop-down list in the coverage view and defines the input of this view as well as the Java source highlighting.

Merging Sessions

If the overall test set consists of multiple test launches, they will result in multiple different coverage session. For analysis it may make sense to combine these sessions to a single session. If there is more than one session the coverage view provides the Merge Sessions command. This command allows selecting a subset from the existing sessions and combining it to a single coverage session.

Session Import and Export

While EclEmma is primarily designed for test runs and analysis within the Eclipse workbench, it provides import and export functionality.
Session Import
If your program is launched outside the Eclipse debugging environment, you might import JaCoCo execution data from these launches. This allows to study the coverage results directly in your source code. The Coverage Session import wizard can be activated form the File —> Import... menu or from the Coverage view's context menu.
The wizard dialog requires you to specify the source of the execution data on its first page. There are three possible sources:

- Local * .exec file location.
- Remote *. exec file URL.
- IP address and port of a JaCoCo agent attached to a running process. You can also specify

whether the execution data should be reset in the target VM when dumping the data.

If you keep a reference to the original execution data source (check the corresponding import option) you can simply re-import the data as often as you want. For this use the Refresh option from the context menu of the Coverage view or simply hit the fs key.
On the second page you can specify session name and the scope, which are the source folders and libraries that should be considered.

Warning: Imported execution data must be based on the exact same class files that are also used within the Eclipse IDE. If the external launch was based on different class files (e g. created with different compiler) no coverage will be shown.

Session Export

The session export wizard allows to export coverage sessions in one of these formats:

- **HTML**: A detailed and browseable report as a set of HTML files.
- **Zipped HTML**: Same as above but zipped into a single file.
- **XML**: Coverage data as a single, structured XML file.
- **CSV**: Coverage data on class level granularity as comma-separated values.
- **Execution data file**: Native JaCoCo execution data format.

The Coverage Session export wizard can be activated form the *File —> Export...* menu or from the Coverage view's context menu. There must be at least one coverage session available to use the export wizard.

Select one of the existing sessions and the export format.