



BEA WebLogic JRocket™ SDK

Using Code Coverage with WebLogic JRocket

Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Contents

Using Code Coverage with WebLogic JRockit

What is Code Coverage?	1
How Code Coverage Works	1
When to Use Code Coverage	2
Enabling Code Coverage	2
Command Line Options for Code Coverage	2
Enabling Code Coverage	3
Identifying the Classes to be Covered	3
Naming the Filter File	3
Identifying the File Location	3
Identifying the Initial Test	3
Modifying the Testid During Runtime	4
Displaying Code Coverage in the Verbose Mode	4
Appending an Output File	4
Code Coverage Output	5
Events Described	5
Filter File Rules	5
File Format	6

Using Code Coverage with WebLogic JRockit

This document describes how to use code coverage on applications you plan to run on WebLogic JRockit. It includes information on the following subjects:

- [What is Code Coverage?](#)
- [Enabling Code Coverage](#)
- [Command Line Options for Code Coverage](#)
- [Code Coverage Output](#)

What is Code Coverage?

Code coverage ensures that the tests you run are actually testing your code. When you run a tests, the possibility exists that the parameters of that test might not expose all of the code in your applicaiton; thus it will not be tested. Code coverage anticipates this situation and will tell you how much of your code was actually tested. Consider that even though your tests might all pass successfully, if you've only exercised half of your code, those tests could be woefully inaccurate.

How Code Coverage Works

Code coverage is an iterative tool; that is, it is most useful when you can compare the results of one application coverage with an earlier version of coverage for the same application. Code coverage is a simple process that requires you to define the coverage parameters at the command line and then let the application run. During the run, the coverage tool determines what code is actually being tested an which code is not. As it makes this determination, it writes to a filter file

all code not covered. You can then compare the information in the latest filter file to the same information in an earlier iteration to determine if the test suite you are using is covering more or less code.

When to Use Code Coverage

Code coverage requires knowledge of and access to the code itself rather than simply using the interface provided and is therefore considered a “white box” text. You will find code coverage most effective during the module testing and to some extent during integration testing. Depending upon what you are testing, you might find other occasions where code coverage will be helpful. Regression tests are usually black box tests and as such may be unsuitable for use with code coverage.

Enabling Code Coverage

Code coverage is run from the command line upon program startup. Simply include the option `-XXcodecoverage` along with the rest of your command string; for example:

```
java -XXcodecoverage -Djrockit.codecoverage.filter=java.util.Hashtable;com.bea.*;  
-com.bea.blabla.* -Xgc:parallel -Xms:64 -Xmx:64 myClass
```

(where `myClass` is the name of the class that contains the `main()` method.)

Command Line Options for Code Coverage

Along with `-XXcodecoverage`, WebLogic JRockit provides a number of additional options that you can use to define the classes to cover, specify an alternate name for the filter file, identify the initial test, and so on. These options are also specified at startup; for example:

```
java -XXcodecoverage -Djrockit.codecoverage.filter=java.util.Hashtable;  
com.bea.*;-com.bea.blabla.* -Xgc:parallel -Xms:64 -Xmx:64 myClass
```

This section describes the how to use these command line options to enable these functions:

- [Enabling Code Coverage](#)
- [Identifying the Classes to be Covered](#)
- [Naming the Filter File](#)
- [Identifying the File Location](#)
- [Identifying the Initial Test](#)

- [Displaying Code Coverage in the Verbose Mode](#)
- [Appending an Output File](#)

Enabling Code Coverage

`-XXcodecoverage`

As mentioned in [Enabling Code Coverage](#), `-XXcodecoverage` enables code coverage. This option cannot be used together with `-Xdebug`.

Identifying the Classes to be Covered

`-Djrockit.codecoverage.filter=<filterspec>`

The `filter=` option identifies which classes should be covered. Filterstrings starting with “-” will be considered as classes that should not be covered. Separate filters with “;” on windows and “.” on linux

Example:

`-Djrockit.codecoverage.filter=java.util.Hashtable;com.bea.*;-com.bea.blabla.*`

Naming the Filter File

`-Djrockit.codecoverage.filterfile=<filename>`

The `filterfile=` option sets the name of a file that will include the filter (specified or default). The file format is one filterstring per line. If no filter or filterfile is specified JRockit will default to `filter.txt` in the current directory.

Identifying the File Location

`-Djrockit.codecoverage.outputfile=<filename>`

The `outputfile=` option to set the file where output is written. If the output file cannot be opened for writing it will sequence through `<filename>_0`, `<filename>_1` until a usable name can be found. This can be useful if several JVMs share a common commandline. If no outputfile is specified JRockit will default to `coverage.txt` in the current directory.

Identifying the Initial Test

`-Djrockit.codecoverage.testid=<id-string>`

The `testid` option sets the initial test identifier. If no test identifier is specified, WebLogic JRockit will default to an empty string.

Modifying the Testid During Runtime

The application you are covering can modify the `testid` during runtime by calling one of the following methods shown in [Listing 1](#).

Listing 1 Modifying the Test ID During Runtime

```
package COM.jrockit.internal;
public final class CodeCoverage
{
    public static native void setTestID(String str);
    public static native void setTestIDAndReset(String str);
}
```

`setTestID` changes the test id so that all code that we has not covered been before will be reported with the new id.

`setTestIDAndReset` reports all code covered by the new test.

You will need to invoke these methods through reflection, for example:

```
Class.forName("COM.jrockit.internal.CodeCoverage").getMethod("setTestIDAndReset", new Class[]{String.class}).invoke(null, new Object[] { myNewTestIdString })
```

Displaying Code Coverage in the Verbose Mode

```
-Djrockit.codecoverage.verbose
```

The `verbose` option causes code coverage information to display on the screen. This option is useful when you want to see textual differences between coverage files. All information appears in plain text.

Appending an Output File

```
-Djrockit.codecoverage.appendoutput
```

The `appendoutput` option allows you to append code coverage results to the output file, rather than overwrite it.

Code Coverage Output

Code coverage results are output to a filter file. These files will appear as plain text unless you specify `-Djrookit.codecoverage.verbose` at startup, in which case, the format will be verbose.

Events Described

The file shows one event per line for six different event types, as identified by the following prefixes:

- `f`: information about the filter used
- `c`: class load
- `m`: method load (happens together at class load time)
- `g`: method code generated (happens at first execution of the method)
- `l`: code generated for a line of source code (only reported for source lines actually containing code)
- `x`: line executed

Filter File Rules

The following rules apply to the filter file:

- Events are written to the file in exactly the order they occur in the JVM, with no sorting or buffering.
- The file is flushed and closed upon termination of the JVM.
- Abnormal exit may result in incomplete data at the end of the file.
- Every line begins with `<event>:<testid>`.

File Format

[Table 1](#) describes the filter file format in both the plain text and verbose modes.

Table 1 Filter File Format by Event Type

Event Type	File Format
Filter	<p>Plain text:</p> <p>f:<testid>:<filterspec>:<timestamp></p> <p>Verbose:</p> <p>Same as plain text</p>
Class load	<p>Plain text:</p> <p>c:<testid>:<classname>:<source filename>:<type></p> <p>Type can be one of the following</p> <ul style="list-style-type: none"> • c - regular class • a - abstract class • i - interface <p>Verbose:</p> <p>Same</p>
Method load	<p>Plain text:</p> <p>m:<testid>:<classname>.<methodname><descriptor>:<methodid></p> <p>The method id is an unique identifier for this method valid for this run only.</p> <p>Verbose:</p> <p>m:<testid>:<classname>.<methodname><descriptor></p>
Method generated (excecuted) event	<p>Plain text:</p> <p>g:<testid>:<methodid></p> <p>Verbose:</p> <p>g:<testid>:<classname>.<methodname><descriptor></p>

Table 1 Filter File Format by Event Type

Line generated event	<p>Plain text: <code>l:<testid>:<methodid>:<lineNo>,<lineNo>,...</code> Line numbers are in no specific order, so duplicates might exist.</p> <p>Verbose: <code>l:<testid>:<classname>.<methodname><descriptor>:<lineNo></code> Only one line reported per event.</p>
Line executed (covered) event	<p>Plain text: <code>x:<testid>:<methodid>:<lineNo>,<lineNo>,...</code></p> <p>Verbose: <code>x:<testid>:<classname>.<methodname><descriptor>:<lineNo></code></p> <ul style="list-style-type: none"> • Only one line reported per event. • Line numbers are exactly the order they are executed, so duplicates might exist.

