



BEA WebLogic JRockit

**Using WebLogic JRockit
8.1 SDK**

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

Introduction to BEA WebLogic JRockit 8.1 SDK

What is JRockit?	1-1
About the SDK	1-2
Why Should I Use JRockit?	1-2
What Platforms Does JRockit Support?	1-3
32-bit Platform Support	1-3
64-bit Platform Support	1-4
JRockit 8.1 SDK Support	1-4
If WebLogic JRockit JVM Crashes	1-4
Using JRockit 8.1 SDK Documentation	1-5
Printing These Documents	1-5
Understanding Documentation Conventions	1-6

Understanding WebLogic JRockit SDK

The BEA WebLogic JRockit Management Console	2-1
Code Generation and Optimization	2-2
Memory Management (Garbage Collection)	2-3
Threads	2-3

Starting and Configuring WebLogic JRockit JVM

Before Starting WebLogic JRockit	3-1
Starting WebLogic JRockit	3-2
Sample Start-up Command	3-2

Configuring WebLogic JRockit	3-2
Using Standard Options	3-3
Setting General Information	3-3
Providing Information to the User	3-3
Using Non-standard Options	3-4
Setting Behavioral Options	3-4
Displaying Logging Information	3-5
Preventing WebLogic JRockit JVM (When Run as a Service) from Shutting Down After Receiving a Logoff Event	3-10
Attaching a Debugger to a Process	3-11
Enabling Core Dumps on Linux	3-11

Selecting and Running a Memory Management System

Memory Management Terminology	4-1
WebLogic JRockit JVM Garbage Collectors	4-2
Generational Copying	4-2
Concurrent Garbage Collectors	4-3
Single Spaced Concurrent	4-3
Generational Concurrent	4-3
Parallel	4-3
Starting a Garbage Collector	4-4
Choosing a Garbage Collection Method	4-4
Pros and Cons	4-5
Garbage Collector Selection Matrix	4-6
Tuning for Garbage Collection	4-6
Viewing Garbage Collection Activity	4-7

Selecting and Running a Thread System

Native Threads	5-1
--------------------------	-----

Thin Threads	5-1
Starting the Thread System	5-2
Choosing a Thread System	5-3
Pros and Cons	5-4
Thread System Selection Matrix	5-4

Using the WebLogic JRockit Management Console

Console Overhead	6-1
Starting the Console.	6-2
Enable the Management Server	6-2
Start the JRockit Management Console	6-2
Starting the Management Server with a Security Manager.	6-2
Set the Port	6-3
Change the Number of Connections.	6-4
Parts of the Console.	6-4
Setting Up the Console	6-7
Making Connections.	6-7
Creating a New Folder	6-7
Creating a New Connection	6-8
Connecting a Connection to WebLogic JRockit JVM	6-8
Disconnecting a Connection from WebLogic JRockit JVM.	6-9
Renaming a Connection or Folder	6-9
Removing a Connection or Folder	6-10
Hiding Disconnected Connections	6-10
Enabling Console Settings	6-10
Setting the Operation Mode	6-11
Setting Other Preferences	6-12
Customizing the Display	6-14

Using the Settings File	6-16
Using the Console	6-16
Information Tabs	6-17
Overview Tab	6-17
Memory Tab	6-18
Processor Tab	6-20
System Tab	6-21
Notification Tab	6-22
View Historical Data	6-29
Using Advanced Features of the Console	6-31
View Thread Stack Dump	6-31
Method Profiling Tab	6-32
Exception Counting Tab	6-35
Closing the Console	6-37

Using WebLogic JRockit JVM with Other WebLogic Applications

Using WebLogic JRockit JVM with BEA WebLogic Server	7-1
Certified Versions	7-2
Verifying that WebLogic JRockit is Your JVM	7-2
Starting JRockit from the Node Manager	7-2
Enabling the Management Server from the Node Manager	7-3
Setting Options by Using the Node Manager	7-3
Tuning WebLogic JRockit for WebLogic Server	7-3
Monitoring WebLogic JRockit JVM from WebLogic Server	7-4
Running JRockit with Thin Threads on WebLogic Server	7-6
Switching to WebLogic JRockit JVM in WebLogic Server	7-6
Switching VMs When WebLogic Server is Running as a Service	7-7
Configuring JRockit for BEA WebLogic Workshop	7-7

What's in the WebLogic JRockit 8.1 SDK?

SDK Contents	A-1
Development Tools	A-1
Runtime Environment	A-2
Additional Libraries	A-2
C Header Files	A-2
The Management Console	A-2
File Differences Between WebLogic JRockit 8.1 SDK and Sun HotSpot SDK	A-3

Adding Custom Notification Actions and Constraints

Locating consolesettings.xml	B-1
Creating a Custom Action	B-2
Creating and Implementing an Action: Example	B-2
Create the Action (Step 2)	B-3
Implementing handleNotificationEvent() (Step 3)	B-5
Creating the Action Editor (Step 4)	B-5
Implementing the Abstract Methods (Step 5)	B-7
Adding the New Action to the Deployment Entries (Step 6)	B-8
Displaying the New Action Editor (Steps 7 and 8)	B-8
Creating a Custom Constraint	B-8

Index

Introduction to BEA WebLogic JRockit

8.1 SDK

BEA WebLogic JRockit 8.1 SDK provides tools, utilities, and a complete runtime environment for developing and running applications using the Java programming language. The WebLogic JRockit SDK includes the WebLogic JRockit Java Virtual Machine (JVM), the first commercial server-side JVM. The WebLogic JRockit JVM was developed uniquely for server-side applications and optimized for Intel architectures to ensure reliability, scalability, and manageability for Java applications.

This user guide provides instructions for using WebLogic JRockit SDK and JVM on the Windows and Linux platforms.

This section contains information on the following subjects:

- [What is JRockit?](#)
- [Why Should I Use JRockit?](#)
- [What Platforms Does JRockit Support?](#)
- [JRockit 8.1 SDK Support](#)
- [Understanding Documentation Conventions](#)

What is JRockit?

The WebLogic JRockit 8.1 JVM is the only credible high performance JVM developed uniquely for server-side applications and optimized for Intel architectures. It ensures reliability, scalability, manageability, and flexibility for Java applications. As a crucial component of the BEA WebLogic Platform, the WebLogic JRockit JVM delivers a new level of performance for Java

applications deployed on Intel 32-bit and 64-bit (Itanium) architecture at significantly lower costs to the enterprise. Furthermore, it is the only enterprise-class JVM optimized for Intel architectures, providing seamless interoperability across multiple hardware and operating system configurations. WebLogic JRockit 8.1 JVM makes it possible to gain optimal performance for your server-side application when running it on either the Windows and Linux operating systems and on either 32-bit and 64-bit architectures.

For more information on JVMs in general, see the [Introduction](#) to the JVM specification at.

<http://java.sun.com/docs/books/vmspec/2nd-edition/html/Introduction.doc.html#3057>

About the SDK

The WebLogic JRockit 8.1 JVM is one component of the WebLogic JRockit 8.1 software development kit (SDK). Along with the WebLogic JRockit 8.1 JVM, this kit is comprised of the Java Runtime Environment (JRE), which contains the JVM and Java class libraries (as specified by the Java 2 Platform API Specification), along with a set of development tools, such a compiler, a debugger, and so on. For more information on the contents of the WebLogic JRockit 8.1 SDK, please refer to [What's in the WebLogic JRockit 8.1 SDK?](#)

Why Should I Use JRockit?

There are three compelling reasons for using the WebLogic JRockit JVM:

- WebLogic JRockit SDK provides optimal running performance for applications developed for and running on BEA WebLogic Server.

WebLogic JRockit JVM is the default production JVM for WebLogic Server-based applications. During “typical” WLS installation, WebLogic JRockit SDK is automatically installed with the server. The Configuration Wizard creates startup scripts that invoke the WebLogic JRockit JVM, making its use simple and transparent.

- WebLogic JRockit JVM is designed for server-side applications.

If you are running server-side applications, the WebLogic JRockit JVM is the most effective VM you can use. A server-side JVM is designed with server applications in mind. Server-side applications have very different requirements than client-side applications and demand different behavior from a JVM. These applications generally do not use graphical user interfaces, run for longer periods of time, and are usually parallel and thread intensive. To fully exploit server-side applications and ensure optimal performance, your JVM should employ a server-specific design policy.

- WebLogic JRockit JVM employs adaptive optimization to significantly improve runtime performance.

Adaptive optimization is a WebLogic JRockit JVM feature that detects and removes bottlenecks in a running program, significantly improving program performance.

Adaptively optimized code usually performs better than statically compiled code because it has access to more information regarding the running program. Another advantage is that the code does not lose any of the dynamic characteristics of interpreted languages.

What Platforms Does JRockit Support?

WebLogic JRockit 8.1 SDK is certified to be compatible with J2SE 1.4.1.

32-bit Platform Support

The 32-bit versions WebLogic JRockit 8.1 SDK are supported on the following operating system releases:

Table 1-1 32-bit Platform Support

Operating Systems	Hardware Platforms	JRockit Support Status	Notes
Microsoft Windows 2000 SP2 (or higher)	Intel Pentium II or higher (and compatible)	Development and Production Support	None
Micorsoft Windows XP	Intel Pentium II or higher (and compatible)	Development Support	None
Red Hat Enterprise Linux AS / ES / WS 2.1	Intel Pentium II or higher (and compatible)	Development and Production Support for Red Hat Enterprise Linux AS & ES editions Development Support for Red Hat Enterprise Linux WS	Tested with kernel 2.4.9, glibc 2.2.4-31.7
SuSE SLES 8.0 (United Linux 1.0)	Intel Pentium II or higher (and compatible)	Development and Production Support	Tested with kernel 2.4.19, glibc 2.2.5

64-bit Platform Support

The 64-bit versions WebLogic JRockit 8.1 SDK are supported on the following operating system releases

Table 1-2 64-bit Platform Support

Operating Systems	Hardware Platforms	JRockit Support Status	Notes
Microsoft Windows Server 2003 EE	Intel Itanium 2 or higher	Development and Production Support	None
Red Hat Linux Advanced Server for Itanium Processor 2.1	Intel Itanium 2 or higher	Development and Production Support for Red Hat Advanced Server for Itanium Processor	Tested with kernel 2.4.18, glibc 2.2.4
Red Hat Linux Advanced Workstation for Itanium Processor 2.1		Development Support for Red Hat Advanced Workstation for Itanium Processor	Requires kernel 2.4.18-e.25 (or higher)
SuSE SLES 8.0 (United Linux 1.0)	Intel Itanium 2 or higher	Development and Production Support	Tested with kernel 2.4.19, glibc 2.2.5

JRockit 8.1 SDK Support

This section describes how to get support for WebLogic JRockit 8.1 SDK.

If WebLogic JRockit JVM Crashes

If WebLogic JRockit 8.1 crashes, it will dump information about the crash to `stderr` and create, in the directory where the VM was started, a file containing the same information, called `jrockit.<pid>.dump` (and, if you are using Windows, `jrockit.<pid>.mdmp`), where `<pid>` is the id of the process that crashed.

After a crash, send a copy of `jrockit.<pid>.dump` (and, if you are using Windows, `jrockit.<pid>.mdmp`), along with as much information as possible about your system setup and the application you were running when the VM crashed, to support@bea.com. You should provide the following information:

- Hardware
- Version of WebLogic JRockit JVM

- Operating system and its version
- The program you ran
- Stack dumps (if any)
- If possible, a small code example that will reproduce the error.

Using JRockit 8.1 SDK Documentation

This section provides hints for using this user guide. It includes information on the following subjects:

- [Printing These Documents](#)
- [Understanding Documentation Conventions](#)

Printing These Documents

You can print a copy of any WebLogic JRockit SDK document from a Web browser, one file at a time, by using the File→Print option on the browser.

PDF versions of all WebLogic JRockit 8.1 documents are available on the WebLogic JRockit 8.1 documentation pages on the e-docs Web site. You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access and print the PDFs, do the following:

1. Open the web page for the WebLogic JRockit 8.1 document you want to print and click the view as PDF icon.

A new browser launches, running the Adobe Acrobat Reader, which contains the PDF version of the document you selected.

2. Click the print button on the Adobe Acrobat Reader toolbar.

The Print dialog box appears.

3. Select the Print range (All, Current page, or Pages from) and click OK to print the document.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at <http://www.adobe.com/>.

Understanding Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
<i>italics</i>	Indicates emphasis or book titles.
monospace text	<div>Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.</div> <div><i>Examples:</i></div> <div>#include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</div>
monospace boldface text	<div>Identifies significant words in code.</div> <div><i>Example:</i></div> <div>void commit ()</div>
<i>monospace italic text</i>	<div>Identifies variables in code.</div> <div><i>Example:</i></div> <div>String <i>expr</i></div>
UPPERCASE TEXT	<div>Indicates device names, environment variables, and logical operators.</div> <div><i>Examples:</i></div> <div>LPT1 SIGNON OR</div>
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

Convention	Item
[]	<p>Indicates optional items in a syntax line. The brackets themselves should never be typed.</p> <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
	<p>Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.</p>
...	<p>Indicates one of the following in a command line:</p> <ul style="list-style-type: none"> • That an argument can be repeated several times in a command line • That the statement omits additional optional arguments • That you can enter additional parameters, values, or other information <p>The ellipsis itself should never be typed.</p> <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
.	<p>Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.</p>

Understanding WebLogic JRockit SDK

WebLogic JRockit SDK has a number of important features that separate it from other JVMs on the market today. This section provides high-level descriptions of some of the more critical SDK features to help you better understand what they can do. This section includes information on the following subjects:

- [The BEA WebLogic JRockit Management Console](#)
- [Code Generation and Optimization](#)
- [Memory Management \(Garbage Collection\)](#)
- [Threads](#)

For more detailed information on additional WebLogic JRockit SDK features, please refer to [What's in the WebLogic JRockit 8.1 SDK?](#)

The BEA WebLogic JRockit Management Console

The Management Console connects to the WebLogic JRockit JVM and provides real-time information about server behavior and resource availability, such as memory usage and profiling information. This gives you a powerful way of retrieving constant profile data about your application.

The Management Console provides a unique advantage when deploying a commercial Java solution because it gives you greater control of the complex set of interrelated variables that may affect your application in production. Administrators can monitor the WebLogic JRockit JVM operating characteristics and the Java application, and be automatically notified of changes in

resource availability or operating characteristics as they occur. Based on this information, administrators can identify bottlenecks in performance and change operating and environmental parameters to optimize performance and availability.

For more information on the Management Console, please refer to [Using the WebLogic JRockit JVM Management Console](#).

Code Generation and Optimization

WebLogic JRockit differs from most JVMs in that it compiles the code at first use by implementing a JIT ("Just In Time") compiler. This ensures desirable application performance from the outset, albeit at the cost of a slightly longer start-up time. To expedite start-up, however, WebLogic JRockit does not use all possible compiler optimizations. While doing so might lead to even better performance early in the application run, that would result in slower start-up.

Compilation time is part of application execution time, thus compiling all of the methods with all available optimizations also negatively impacts application performance. Therefore, WebLogic JRockit does not fully optimize all methods at start-up; in fact, it leaves many methods unoptimized throughout the entire application run. Instead, WebLogic JRockit chooses those functions whose optimization will most benefit application performance and only optimizes those methods.

WebLogic JRockit can thus be seen to have two distinct, but cooperating, code generators: a JIT compiler, which resolves data from bytecode, through three levels of intermediate representation, to native code (assembly language); and an optimizing compiler, which optimizes targeted methods at each level of intermediate representation.

WebLogic JRockit uses a “sampler thread” to identify which functions merit optimization. This thread wakes up at periodic intervals and checks the status of several application threads. It identifies what each thread is executing and notes some of the execution history. This information is tracked for all the methods and when it is perceived that a method is experiencing heavy use—in other words, is “hot”—that method is earmarked for optimization. Usually, a flurry of such optimization opportunities occur in the application's early run stages, with the rate slowing down as execution continues.

The optimizing compiler in WebLogic JRockit includes many of the best-known techniques for code generation, particularly for IA64 machines. This includes a sophisticated register allocator that takes full advantage of IA64's large register stacks.

Memory Management (Garbage Collection)

WebLogic JRockit JVM manages memory by employing four different garbage collectors. These collectors work during runtime to clear the memory heap of expired objects, or “garbage.” These four garbage collectors are:

- **Generational Copy**, which divides the memory into two areas called “generations.” Instead of allocating objects in one single space and garbage collecting that whole space when it gets full, most of the objects are allocated in the “young generation”, called the nursery.
- **Single-Spaced Concurrent**, one of two types of concurrent collectors, which does its work in parallel with ordinary processing; that is, it does not stop all Java threads to do the complete garbage collection. This is designed to support garbage collection without disruption and to improve multiprocessor garbage collection performance.
- **Generational Concurrent** is the second type of concurrent collector WebLogic JRockit employs. Although very similar to a single-spaced concurrent collector, a generational concurrent garbage collector does its actual object allocation in a “nursery,” reducing the need to do collection of the entire heap so often.
- **Parallel** garbage collection, which stops all Java threads and uses all CPUs to perform a complete garbage collection of the entire heap.

For information on selecting and using garbage collectors, see [Selecting and Running a Memory Management System](#).

Threads

WebLogic JRockit JVM can use one of two thread systems to maximize processing:

- **Native Threads** which maps Java threads directly to the operating system threads, directly taking advantage of the operating system's thread scheduling and load balancing policies. Native Threads is the default thread system for WebLogic JRockit JVM.
- **Thin threads**, wherein multiple Java threads are run on a single operating system thread. This allows WebLogic JRockit to optimize thread scheduling, thread switching, and thread synchronization, while using less memory.

Warning: Thin threads is experimental functionality in this version of WebLogic JRockit, and is not recommended for general use. This feature is subject to change without notice.

For information on selecting a thread system, please refer to [Selecting and Running a Thread System](#).

Starting and Configuring WebLogic JRockit JVM

This section describes how to start WebLogic JRockit and how to configure it by using standard and non-standard command line options. It includes information on the following subjects:

- [Before Starting WebLogic JRockit](#)
- [Starting WebLogic JRockit](#)
- [Configuring WebLogic JRockit](#)
- [Attaching a Debugger to a Process](#)
- [Enabling Core Dumps on Linux](#)

Note: If JRockit behaves in some unexpected way, please consult the [WebLogic JRockit Developers FAQ](#). If that doesn't solve your problem, please send an e-mail to support@bea.com.

Before Starting WebLogic JRockit

Before starting WebLogic JRockit JVM, ensure that you have the following directory set in your PATH environment variable:

- `<jrockit-install-directory>/bin` (for Linux)
- `<jrockit-install-directory>\bin` (for Windows)

Starting WebLogic JRockit

To start the WebLogic JRockit, at the command line enter the following:

```
java <configuration and tuning options> myClass
```

Where *<configuration and tuning options>* are the configuration and tuning options you want to use. The configuration options are described in [Configuring WebLogic JRockit](#), below. See [Tuning WebLogic JRockit JVM](#) for details on the tuning options available for this version of WebLogic JRockit.

Note: You can alternatively start JRockit with by fully qualifying the path to the file; for example, `/usr/local/java/bin/java` (depending on where it is installed) on Linux and `c:\bea\jrockitxxx\bin\java` (depending on where its installed) on Windows.

Sample Start-up Command

A sample start-up command, with some tuning options specified, might look like this:

```
java -verbose:memory -Xgc:gencopy -Xmx:256m -Xms:64m -Xns:24m myClass
```

In this example, the following options are set:

- `-verbose:memory`—Displays verbose output about memory usage.
- `-Xgc:gencopy`—A generational copy garbage collector will be used.
- `-Xmx:256m`—The maximum heap size is set to 256 megabytes.
- `-Xms:64m`—The initial heap size is set to 64 megabytes.
- `-Xns:24m`—The nursery size is set to 24 megabytes.
- `myClass`—Identifies the class that contains the `Main()` method.

For more information on the tuning options discussed above, please refer to

Configuring WebLogic JRockit

When you start WebLogic JRockit, you can set behavioral parameters by using either standard or non-standard command line options. This section describes these options and how to use them at startup to configure WebLogic JRockit. It contains information on the following subjects:

- [Using Standard Options](#) for:
 - [Setting General Information](#)

- Providing Information to the User
- Using Non-standard Options for:
 - Setting Behavioral Options
 - Displaying Logging Information
 - Preventing WebLogic JRockit JVM (When Run as a Service) from Shutting Down After Receiving a Logoff Event
- Enabling Core Dumps on Linux

Using Standard Options

Standard command line options work the same regardless of the JVM; in other words, these options work the same whether you are running WebLogic JRockit JVM, Sun Microsystem's HotSpot JVM, or any other third-party JVM.

Setting General Information

The following standard command line options set general information about WebLogic JRockit JVM:

- `-classpath <directories and zips/jars separated by : (Linux) or ; (Windows)>`

Tells the VM where to look for classes and resources.

Alternately, you can use the option `-cp` to represent `-classpath`; for example:

`-cp <directories and zips/jars separated by : or ;>`

- `-D<name>[=<value>]`

Tells the VM to set a Java system property. These can be read by a Java program, using the methods in `java.lang.System`.

Providing Information to the User

The following options determine if the system will provide messages to the operator and what the form and content of those messages should be.

- `-version`

Tells JRockit to display its product version number and then exit.

- `-showversion`

Tells the VM to display its product version number and then continue.

- `-verbose[:<components separated by ,>]`

Tells JRockit to display verbose output. This option is used mainly for debugging purposes and causes a lot of output to the console. Supported components are `memory`, `load`, `gc`, `opt`, and `codegen`. If no component is given, JRockit will display verbose information on everything. For more information on the components and the `-verbose` information they display, please refer to [Table 3-1](#).

- `-help`

Tells the VM to display a short help message.

- `-x`

Tells the VM to display a short help message on the extended options (do not confuse `-x` with the non-standard, or `-X`, options described below).

Using Non-standard Options

Non-standard, or `-x`, command line options are options that are exclusive to WebLogic JRockit JVM that change the behavior of WebLogic JRockit JVM to better suit the needs of different Java applications. These options are all preceded by `-x` and will not work on other JVMs (conversely, the non-standard options used by other JVMs won't work with WebLogic JRockit).

Note: Since these options are non-standard, they are subject to change at any time.

Setting Behavioral Options

The following non-standard options define general WebLogic JRockit JVM behavior:

- `-Xnoopt`

Tells the VM not to optimize code.

- `-Xverify`

Tells the VM to do complete bytecode verification.

- `-Xstrictfp`

Enables strict floating point arithmetics globally for all methods in all classes. This option is similar to the Java keyword `strictfp`. See the [Java Language Specification](#) for more details on `strictfp`.

Displaying Logging Information

`-Xverbose`

`-Xverbose` causes WebLogic JRockit to print to the screen specific information about the system. The information displayed depends upon the parameter specified with the option; for example, specifying the parameter `cpuinfo` displays information about your CPU and indicates whether or not the JVM can determine if hyper threading is enabled. [Table 3-1](#) lists the parameters available for `-Xverbose`.

Note: To use more than one parameter, separate them with a comma; for example:

```
-Xverbose:gc,opt
```

Table 3-1 -Xverbose Parameters

This Parameter...	Prints to the screen...
<code>codegen</code>	<p>The names of each method that is being compiled. Verbose output for <code>codegen</code> might look like this:</p> <pre>[codegen] 0 : 17.9411 ms [codegen] 0 68592131 1 java.lang.Object.unlockFatReal_jvmpi (Ljava.lang.Object;Ljava.lang.Thread;I)V: 17.94 ms [codegen] 1 : 2.0262 ms [codegen] 0 0 2 java.lang.Object.acquireMonitor(Ljava.lang.Object;II)I: 19.97 ms [codegen] 2 : 4.4926 ms [codegen] 0 10 3 java.lang.Object.unlockFat(Ljava.lang.Object;Ljava.lang.Thread;I) V: 24.46 ms [codegen] 3 : 0.3328 ms</pre>
<code>cpuinfo</code>	<p>Any interesting information about your CPUs. Verbose output for <code>cpuinfo</code> might look like this:</p> <pre>[cpuinfo] Vendor: GenuineInt [cpuinfo] Type: Original OEM [cpuinfo] Family: Pentium Pro [cpuinfo] Model: Pentium III/Pentium III Xeon [cpuinfo] Brand: Pentium III processor [cpuinfo] Supports: On-Chip FPU [cpuinfo] Supports: Virtual Mode Extensions [cpuinfo] Supports: Debugging Extensions [cpuinfo] Supports: Page Size Extensions</pre>

Table 3-1 -Xverbose Parameters

This Parameter...	Prints to the screen...
load	<p>The names of each loaded class. Verbose output for load might look like this:</p> <pre>[load] 0 1 java.lang.Object+ [load] 0 2 java.io.Serializable+ [load] 0 3 java.lang.Class+ [load] 0 5 java.lang.reflect.AccessibleObject+ [load] 0 6 java.lang.reflect.Member+ [load] 0 6 java.lang.reflect.Method+</pre>
memory; gc	<p>Information about the memory management system, including:</p> <ul style="list-style-type: none">• Start time of collection (seconds since JVM start)• End time of collection (seconds since JVM start)• Memory used by objects before collection (KB)• Memory used by objects after collection (KB)• Size of heap after collection (KB)• Total time of collection (seconds or milliseconds)• Total pause time during collection (milliseconds) <p>The information displayed by -Xverbose:memory or -Xverbose:gc will vary depending upon the type of garbage collector you are using.</p>

Table 3-1 -Xverbose Parameters

This Parameter...	Prints to the screen...
memory; gc	A report for a JVM running a generational concurrent collector (-Xgc:gencon) with memory or gc specified might look like this:
with gencon	<pre> [memory] Generational Concurrent collector [memory] nursery 20480K, heap 65536K, maximal heap 262144K [memory] <start>: Nursery GC <before>K-><after>K (<heap>K), <total> ms [memory] <s>-<end>: GC <before>K-><after>K (<heap>K), <total> s (<pause> ms) [memory] <s/start> - start time of collection (seconds since jvm start) [memory] <end> - end time of collection (seconds since jvm start) [memory] <before> - memory used by objects before collection (KB) [memory] <after> - memory used by objects after collection (KB) [memory] <heap> - size of heap after collection (KB) [memory] <total> - total time of collection (seconds or milliseconds) [memory] <pause> - total pause time during collection (milliseconds) Now running The GcList Test [memory] 0.860: Nursery GC 61615K->42008K (86016K), 11.400 ms [memory] 0.953: Nursery GC 62488K->42876K (86016K), 10.895 ms [memory] 1.031: Nursery GC 63356K->45303K (86016K), 30.156 ms [memory] 1.172: Nursery GC 65783K->46168K (86016K), 11.639 ms [memory] 1.250: Nursery GC 66648K->48596K (86016K), 31.189 ms The execution of The GcList Test took 0.578s </pre>

Table 3-1 -Xverbose Parameters

This Parameter...	Prints to the screen...
memory; gc	A report for a JVM running a single generation concurrent collector (-Xgc:singlecon) with memory or gc specified might look like this:
with singlecon	[memory] Single Generation Concurrent collector [memory] heap 65536K, maximal heap 262144K [memory] <s>-<end>: GC <before>K-><after>K (<heap>K), <total> s (<pause> ms) [memory] <s/start> - start time of collection (seconds since jvm start) [memory] <end> - end time of collection (seconds since jvm start) [memory] <before> - memory used by objects before collection (KB) [memory] <after> - memory used by objects after collection (KB) [memory] <heap> - size of heap after collection (KB) [memory] <total> - total time of collection (seconds or milliseconds) [memory] <pause> - total pause time during collection (milliseconds) Now running The GcList Test [memory] 1.016-1.172: GC 58543K->13906K (89716K), 0.156 s (3.420 ms) The execution of The GcList Test took 0.563s Now running The GcList Test [memory] 1.422-1.469: GC 102004K->389K (122816K), 0.047 s (5.048 ms)

Table 3-1 -Xverbose Parameters

This Parameter...	Prints to the screen...
memory; gc with parallel	<p>A report for a JVM running a parallel collector (-Xgc:parallel) with memory or gc specified might look like this:</p> <pre> [memory] Parallel collector [memory] heap 65536K, maximal heap 262144K [memory] <start>: GC <before>K-><after>K (<heap>K), <total> ms [memory] <start> - start time of collection (seconds since jvm start) [memory] <before> - memory used by objects before collection (KB) [memory] <after> - memory used by objects after collection (KB) [memory] <heap> - size of heap after collection (KB) [memory] <total> - total time of collection (milliseconds) Now running The GcList Test [memory] 1.016: GC 65536K->1463K (65536K) in 12.933 ms The execution of The GcList Test took 0.500s Now running The GcList Test [memory] 1.282: GC 65536K->1502K (65536K) in 11.046 ms [memory] 1.563: GC 65536K->1503K (65536K) in 12.119 ms The execution of The GcList Test took 0.484s Now running The GcList Test [memory] 1.782: GC 65536K->593K (65536K) in 9.365 ms The execution of The GcList Test took 0.125s </pre>

Table 3-1 -Xverbose Parameters

This Parameter...	Prints to the screen...
memory; gc with gencopy	<p>A report for a JVM running a Generational Copying collector (-Xgc:gencopy) with memory or gc specified might look like this:</p> <pre>[memory] Generational Copying collector [memory] nursery 640K, to/fromspace 32768K, pinnedspace 4K [memory] maximal heap 131712-262144K [memory] <start>: Nursery GC <before>K-><after>K (<heap>K), <total> ms [memory] <start>: GC <before>K-><after>K (<heap>K), <total> ms [memory] <start> - start time of collection (seconds since jvm start) [memory] <before> - memory used by objects before collection (KB) [memory] <after> - memory used by objects after collection (KB) [memory] <heap> - size of heap after collection (KB) [memory] <total> - total time of collection (milliseconds) Now running The GcList Test [memory] 0.828: Nursery GC 298K->212K (33596K), 1.561 ms [memory] 0.859: Nursery GC 852K->532K (33596K), 3.637 ms [memory] 0.859: Nursery GC 1172K->852K (33596K), 3.639 ms [memory] 0.875: Nursery GC 1492K->1172K (33596K), 3.600 ms [memory] 0.875: Nursery GC 1812K->1492K (33596K), 3.665 ms [memory] 0.891: Nursery GC 2132K->1812K (33596K), 3.620 ms [memory] 0.891: Nursery GC 2452K->2132K (33596K), 3.610 ms [memory] 0.906: Nursery GC 2772K->2452K (33596K), 3.720 ms [memory] 0.906: Nursery GC 3092K->2772K (33596K), 4.307 ms [memory] 0.922: Nursery GC 3412K->3092K (33596K), 3.794 ms [memory] 0.937: Nursery GC 3732K->3412K (33596K), 3.834 ms [memory] 0.937: Nursery GC 4052K->3732K (33596K), 3.640 ms [memory] 0.953: Nursery GC 4372K->4052K (33596K), 3.538 ms [memory] 0.953: Nursery GC 4692K->4372K (33596K), 3.621 ms</pre>
opt	<p>Information about all methods that get optimized. Verbose output for opt might look like this:</p> <pre>[opt] 280 2434 0 ObjAlloc.main([Ljava.lang.String;)V: 0.00 ms [opt] 0 : 9.8996 ms</pre>

Preventing WebLogic JRockit JVM (When Run as a Service) from Shutting Down After Receiving a Logoff Event

When WebLogic JRockit JVM is run as a service (for example, the servlet engine for a web server), it might receive CTRL_LOGOFF_EVENT or SIGHUP. Upon receiving such events, if the VM

tries to initiate shutdown, it will fail, since the operating system will not actually terminate the process. To avoid possible interference such as this, use the `-Xnohup` command-line option. When this option is used with WebLogic JRockit, the JVM does not watch for or process `CTRL_LOGOFF_EVENT` or `SIGHUP` events.

If you specify `-Xnohup`, be aware of the following:

- Ctrl-Break thread dumps are not available.
- User code is responsible for causing shutdown hooks to run; for example by calling `System.exit()` when WebLogic JRockit is to be terminated.

Attaching a Debugger to a Process

You can attach a debugger to a process so that you can see any error messages thrown by that process. To do so, use this procedure:

1. At the command line, enter the command:

```
-Djrockit.waitonerror
```

Note: `waitonerror` is analogous to Sun's `ShowMessageBoxOnError` parameter.

When a fatal error occurs, the dumps are produced as usual and a dialog box appears saying “An error occurred. Attach the debugger now, then press Yes to debug or No to exit.”

- a. At this point, if you want a thread dump, press `[Ctrl]-[Break]`. Be aware that this might not always work.
 - b. Attach the debugger to the process; be sure not to break into the process at this time.
2. Click Yes in the dialog box to have the debugger to catch the error and break into the process.

If you want better symbol information you should replace the default `.pdb` file with the private one. You may have problems in getting the debugger to display source code. If this happens, use `windbg`, as it allows you to set the paths.

Enabling Core Dumps on Linux

If you are using Red Hat AS and want to ensure that a core/javacore file is created in the working directory in the event WebLogic JRockit crashes, you need to enable core dumps. To do this, set the `ulimit -c` value to something greater than zero, but no greater than a value your filesystem can accommodate; for example, `ulimit -c 10000000`. These values are measured in blocks,

with each block equaling one kilobyte. You can set the `ulimit` value either from the command line, in the `*.profile` file, or in a shell script.

Selecting and Running a Memory Management System

Memory management, known as “garbage collection” is the process of clearing “dead” objects from the heap, thus releasing that space for new objects. Effective memory management ensures efficient processing. This section includes information on the following subjects:

- [Choosing a Garbage Collection Method](#)
- [WebLogic JRockit JVM Garbage Collectors](#)
- [Choosing a Garbage Collection Method](#)
- [Tuning for Garbage Collection](#)

Memory Management Terminology

Before continuing, there are some terms you should understand. You may already be familiar with some of the terms, especially if you have read any other documents about garbage collectors.

Thread-local allocation

Thread-local allocation removes object allocation contention and reduces the need to synchronize between thread performing allocations on the heap. It also gives increased cache performance on a multi-CPU system, because it reduces the risk of two threads running on different CPUs having to access the same memory pages at the same time.

Thread-local allocation is not the same thing as thread-local objects, but many people tend to confuse the two terms. Thread-local allocation does not determine whether the objects can be accessed from a single thread only (that is, thread-local objects); thread-local allocation means that the thread has an area of its own where no other thread will create

new objects. The objects that the thread creates in that area may still be reached from other threads.

Pause time

Garbage collector pause time is the length of time that the garbage collector stops all Java threads during a garbage collection. The longer the pause, the more unresponsive your system will be. The worst pause time and the average pause time are the two most interesting values you can use for tuning the system.

Memory throughput

Memory throughput measures the time between when an object is no longer referenced and the time that it's reclaimed and returned as free memory. The higher the memory throughput the shorter is the time between the two events. Moreover, the higher the memory throughput the smaller the heap you will need.

WebLogic JRockit JVM Garbage Collectors

This section describes the four garbage collectors available in WebLogic JRockit JVM. These collectors are:

- [Generational Copying](#)
- [Concurrent Garbage Collectors](#)
- [Parallel](#)

Generational Copying

A generational copying garbage collector divides the memory into two or more areas called “generations”. Instead of allocating objects in one single space and garbage collecting that whole space when it gets full, most of the objects are allocated in the “young generation”, called the nursery. As most objects die young, most of the time it will be sufficient to collect only the nursery and not the entire heap.

A generational copying garbage collector is specifically designed as a lightweight alternative for use on single CPU systems with a small (less than 128 MB) heap. It is suitable for testing applications on your desktop machine; however for a deployment environment with multiple processors and/or a large heap size (in excess of 128 MB), another garbage collector would in most cases be more efficient.

Concurrent Garbage Collectors

A concurrent garbage collector does its work in parallel with ordinary work; that is, it does not stop all Java threads to do the complete garbage collection. Most garbage collectors today are “stop-the-world,” or parallel, collectors and are not very efficient; for example, if you have a large heap and use a parallel collector, if you need to collect the whole heap, you might experience pauses lasting up to several seconds, depending on the heap size. Concurrent garbage collectors are designed to rectify this condition.

WebLogic JRockit can employ two types of concurrent garbage collectors:

- [Single Spaced Concurrent](#)
- [Generational Concurrent](#)

Single Spaced Concurrent

A single spaced concurrent garbage collector (`-Xgc:singlecon`) completely eliminates garbage collection pauses. If you use these garbage collectors, the heaps can be gigabyte-size and still long pauses will not occur. However, to achieve this elimination of pauses, concurrent garbage collectors trade memory throughput; that is, it takes longer between the time the object is referenced the last time and the system detects and reclaims it. The natural consequence of this is that you will most likely need a larger heap with a concurrent garbage collector than you need with any other. In addition, if your ordinary Java threads create more garbage than the concurrent garbage collector manages to collect, unanticipated pauses will occur while the Java threads are waiting for the garbage collector to complete its cycle.

Generational Concurrent

With generational concurrent garbage collectors (`-Xgc:gencon`), objects are allocated in the young generation (the nursery). When the nursery is full, WebLogic JRockit JVM “stops-the-world” and moves the live objects in the young generation to the old generation. An old collector thread runs in the background all the time; it marks objects in the old space as live and removes the dead objects, returning them to WebLogic JRockit JVM as free space. The advantage of the generational concurrent garbage collector compared to the single spaced concurrent garbage collector is that it has a higher memory throughput.

Parallel

Parallel garbage collectors (`-Xgc:parallel`) stop all Java threads when the heap is full and use every CPU to perform a complete garbage collection of the entire heap. A parallel collector can

have longer pause times than concurrent collectors, but it maximizes throughput. Even on single CPU machines, this maximized performance makes parallel the recommended garbage collector, provided that your application can tolerate the longer pause times.

Starting a Garbage Collector

To start a garbage collector, simply include at the command line the `-Xgc` option and the type of collector you want to use. [Table 4-1](#) lists these arguments:

Table 4-1 Garbage Collector Implementation Options

Garbage Collector	Option
Generational Copying	<code>-Xgc:gencopy</code>
Single Spaced Concurrent	<code>-Xgc:singlecon</code>
Generational Concurrent	<code>-Xgc:gencon</code>
Parallel	<code>-Xgc:parallel</code>

When started, JRockit will run with the specified garbage collector.

Default

If the garbage collector has not been set and the maximum heap size (set by using `-Xmx` or using the default as described above) is less than 128 MB, the default garbage collector is generational copying (`-Xgc:gencopy`); otherwise the default is generational concurrent (`-Xgc:gencon`).

Choosing a Garbage Collection Method

Each of the four garbage collectors has its benefits and its drawbacks. In an effort to help you choose the collector that best suits your needs, this section discusses the pros and con of each collector and provides a matrix to help you decide which one to use.

Pros and Cons

Table 4-2 lists the pros and cons of each garbage collector.

Table 4-2 Garbage Collector Pros and Cons

Garbage Collector	Pros	Cons
Generational Copying	<ul style="list-style-type: none"> • Works well with single CPU systems with a heap smaller than 128mB. • Packs small objects together during each collection. • Good for testing on a single machine. • Default garbage collector for heaps less than 128 mB. 	<ul style="list-style-type: none"> • Not effective with large (>128mB) heaps. • Lacks speed and scalability. • Moves objects back and forth between the two semispaces so it doesn't fully utilize the whole heap. • Not recommended in a deployment environment.
Single Spaced Concurrent	<ul style="list-style-type: none"> • Virtually removes all pauses. • Can handle gigabyte-sized heaps. 	<ul style="list-style-type: none"> • Trades memory for fewer pauses. • If ordinary Java threads create more garbage than this collector can collect, pauses occur while these threads are waiting for the collector to complete its cycle. • Only effective so long as the program doesn't run out of memory during collection.
Generational Concurrent	<ul style="list-style-type: none"> • Virtually removes all pauses. • Has a higher memory throughput than single spaced concurrent garbage collector. • Reduces the risk of running out of allocatable memory during collection because the old space is not filled at the same speed. 	<ul style="list-style-type: none"> • Trades memory for fewer pauses. • If ordinary Java threads create more garbage than this collector can collect, pauses occur while these threads are waiting for the collector to complete its cycle.
Parallel	<ul style="list-style-type: none"> • Uses all processors during collection, thus maximizing memory throughput. 	<ul style="list-style-type: none"> • "Stop the world" might cause a longer than desirable pause in processing.

Garbage Collector Selection Matrix

Table 4-3 is a matrix that you can use to determine which garbage collector is right for your WebLogic JRockit JVM implementation. Use the **If...** column to locate a condition that matches your implementation and select the garbage collector indicated in the **Select this Method...** column.

Table 4-3 Garbage Collector Selection Matrix

If You...	Select this Garbage Collector...
<ul style="list-style-type: none">Want lightweight alternative for use on single CPU systems with a small (less then 128 mB) heap.Are testing applications on your desktop machine.	Generational Copying
<ul style="list-style-type: none">Cannot tolerate pauses of any length.Employ gigabyte-sized heaps.Willing to trade memory thoughput for eliminating pauses.Have a single CPU machine with a lot of memory.	Single Spaced Concurrent
<ul style="list-style-type: none">Cannot tolerate pauses of any length.Employ gigabyte-sized heaps.Willing to trade <i>some</i> memory throughput for eliminating pauses.Want better memory throughput than possible with Single Spaced Concurrent.Are not sure that the other three methods would be adequate given how you’ve implemented WebLogic JRockit JVM.	Generational Concurrent
<ul style="list-style-type: none">Using a machine with four CPUs or better or a single CPU machine with a lot of memory.Can tolerate the occasional long pauseNeed to maximize memory throughput	Parallel

Tuning for Garbage Collection

The effectiveness of the garbage collector depends upon a number of memory management parameters you can set; for example, heap size and thread allocation. To provide the optimal out-of-the-box experience, WebLogic JRockit comes with default values for these—and other—settings that adapt automatically to the specific platform on which you are running the JVM.

However, you can modify these values to improve performance by specifying at the command line any of the options listed in [Tuning WebLogic JRockit JVM](#) in *Tuning WebLogic JRockit JVM*.

Viewing Garbage Collection Activity

To observe garbage collection activity, use one or both of the options described here. Using these options will help you evaluate the effectiveness of the selected garbage collector and make necessary tuning decisions.

- If you want to see a comprehensive report of garbage collection activity, enter the `-Xgcreport` option at startup. This option causes WebLogic JRockit JVM to print a comprehensive garbage collection report at program completion.
- If you want to see garbage collection activity when it occurs, enter the `-Xgcpause` option. This option causes the VM to print a line each time Java threads are stopped for garbage collection.

Combining these two options is a very good way of examining the memory behavior of your application; for example:

```
-java -Xgcparallel -Xgcreport -Xgcpause myClass
```

Selecting and Running a Memory Management System

Selecting and Running a Thread System

WebLogic JRockit's thread systems allow the JVM to take optimal advantage of the underlying operating system. WebLogic JRockit JVM supports two types of thread systems: native threads and thin threads. This section describes these thread systems and how to select and run them with your application. It includes information on the following subjects:

- [Native Threads](#)
- [Thin Threads](#)
- [Starting the Thread System](#)
- [Choosing a Thread System](#)

Native Threads

This is the WebLogic JRockit JVM's default thread system. It maps Java threads directly to the operating system threads, taking advantage of the operating system's thread scheduling and load balancing policies.

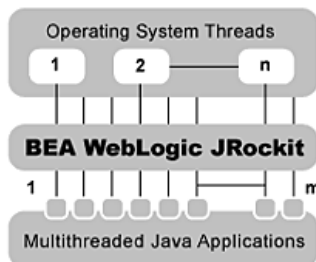
Thin Threads

Warning: Thin threads is experimental functionality in this version of WebLogic JRockit, and is not recommended for general use. This feature is subject to change without notice.

Thin threads simulate several Java threads with one (or more) native threads that represent the running program ([Figure 5-1](#)). Context switches and scheduling is done internally within

WebLogic JRockit, and is therefore much light-weight than context switching done within native threads by the operating system.

Figure 5-1 Thin Threads Model



Thin threads are independent of native threads and are implemented in WebLogic JRockit JVM, thus they are not part of the operating system. This means WebLogic JRockit JVM uses less memory to perform optimized thread scheduling, thread switching, and thread synchronization. This makes it possible to run a significantly higher number of threads at a higher speed than with any other JVM.

To fully utilize system resources on a multi-processor system, WebLogic JRockit JVM is not restricted to running all Java threads in the same operating system thread. A variety of operating system threads can be used, which splits the Java threads among them. A Java thread is not bound to a specific operating system thread; it can move between them to allow for optimal load balancing.

Starting the Thread System

Note: For information on selecting the best thread system for your application, please refer to [Choosing a Thread System](#).

To start a thread system, include one of the options listed in [Table 5-1](#) when you start WebLogic JRockit JVM:

Table 5-1 Options Used to Implement Threading

To Use...	Use this Option...
Native Threads	<p>-Xnativethreads</p> <p>This option is the default and only needs to be specified if you are switching from native threads.</p>
Thin Threads	<p>-Xthinthreads</p> <p>This option is not available on IA64. Additionally, thin threads is experimental functionality in this version of WebLogic JRockit and it is recommended that you do not use it. It is subject to change at any time.</p>

Choosing a Thread System

This section includes some tools that will help you determine which thread system is right for your WebLogic JRockit JVM implementation. It contains a list of the pros and cons of each method and a decision matrix that will help you identify the optimal threading model.

Pros and Cons

Table 5-2 lists the pros and cons of each thread system.

Table 5-2 Thread System Pros and Cons

Thread System	Pros	Cons
Native Threads	<ul style="list-style-type: none">• Takes advantage of the operating system's thread scheduling and load balancing policies.• Are standard for native applications, which relies upon the fact that each Java thread is mapped to an operating system thread of its own, this is the only model that works (both DB2 and Oracle level 2 JDBC database drivers have been known to rely upon this).• On a multiprocessor system when the application has few active threads, the operating system scheduling system is better at utilizing the CPUs efficiently.	<ul style="list-style-type: none">• Context switching is more costly as it has to be done in the operating system instead of only in the JVM.• Every Java thread consumes more resources, because it requires an operating system thread of its own.
Thin Threads	<ul style="list-style-type: none">• Since several Java threads are run in the same operating system thread, JRockit can perform optimized thread scheduling, thread switching, and thread synchronization with less memory.• A variety of operating system threads can be used, splitting the Java threads instead of the operating system threads among them. A Java thread is not bound to a specific operating system thread, so it can move between them to allow for optimal load balancing.	<ul style="list-style-type: none">• Not fully supported by or tested with this version of WebLogic JRockit.• Adds complexity because bridging against the operating system becomes less obvious.• Requires WebLogic JRockit JVM, instead of the operating system, to schedule threads, which can negatively impact performance.• Not an effective on multiprocessing systems that require a small number of threads (a few hundred or less).• Not designed to run with IA64 machines.

Thread System Selection Matrix

Because thin threads is experimental functionality and subject to change, BEA Systems recommends that you always use native threads as your threading system. However, under certain

circumstances, thin threads might provide optimal performance for your WebLogic JRockit JVM implementation. Use the **If...** column in [Table 5-3](#) to locate a condition that matches your implementation and select the thread system indicated in the **Select this Method...** column..

Table 5-3 Threading System Selection Matrix

If...	Select this Method...
You have a relatively small number of threads (less than 100)	Native Threads
Require a proven thread system	Native Threads
You are using an IA64 machine	Native Threads
You have in excess of a couple hundred threads	Thin Threads
You are running Linux on a single-CPU system	Thin Thread. This is because Linux threads are very expensive to use.

Selecting and Running a Thread System

Using the WebLogic JRockit Management Console

The JRockit Management Console can be used to monitor and control running instances of WebLogic JRockit JVM. It provides real-time information about the running application's characteristics, which can be used both during development—for example, to find where in an application's life cycle it consumes more memory—and in a deployed environment—for example, to monitor the system health of a running application server.

This section includes information on the following subjects:

- [Console Overhead](#)
- [Parts of the Console](#)
- [Setting Up the Console](#)
- [Using the Console](#)

Console Overhead

The extra cost of running the JRockit Management Console against a running WebLogic JRockit JVM is very small and can almost be disregarded. This provides for a very low cost monitoring and profiling of your application.

Note: It is not recommended that you run the Management Console on the same machine as the VM you are monitoring. If you run the Console on the same machine as the WebLogic JRockit you are monitoring, the Management Console GUI will steal valuable resources from the application running on the JVM and you risk performance degradation as a result.

Starting the Console

Starting the Management Console is a two-step process:

1. [Enable the Management Server](#)
2. [Start the JRockit Management Console](#)

Additionally, you might want to also complete these tasks as part of the start-up process:

- [Set the Port](#)
- [Change the Number of Connections](#)

Enable the Management Server

Before the Management Console can connect to WebLogic JRockit JVM, the management server in the VM needs to be started. The server is disabled by default. To enable the management server, start WebLogic JRockit JVM with the `-Xmanagement` option:

```
-Xmanagement
```

Start the JRockit Management Console

Start the JRockit Management Console from the command prompt by typing:

```
console
```

Note: Before starting the Management Console, you must specify the JRE path and the classpath to the `.jar` file.

You can also start the Management Console without using the launcher. At the command line, enter:

```
java -jar <jrockit-install-directory>/console/ManagementConsole.jar
```

Starting the Management Server with a Security Manager

If you try to start the management server (`-Xmanagement` option) with a security manager running (`-Djava.security.manager` option) the management server might not start and you will get error messages such as the following:

```
"ERROR: failed to initialize class com.jrockit.management.rmp.  
RmpSocketListener."
```

To allow the management server to run under a security manager, add the text shown in [Listing 6-1](#) to your policy file. The standard location of the policy file is:

- `java.home/lib/security/java.policy` (Linux)
- `java.home\lib\security\java.policy` (Windows)

For more information on policy files please refer to:

<http://java.sun.com/products/jdk/1.2/docs/guide/security/PolicyFiles.html>

Listing 6-1 Code for Starting the Management Server with a Security Manager

```
/* --- Permissions for the JRockit management Server --- */

/* TODO 1: Locate the installed managementserver.jar in JAVA_HOME/jre/lib */
grant codeBase "file:C:/MY_JAVA_HOME/jre/lib/managementserver.jar" {

    /* TODO 2: Add permissions for your console client to connect. */
    permission java.net.SocketPermission "my-console-client.com", "accept,
    resolve";

    /* TODO 3: Add permissions for the management server to listen for
    connections. */
    permission java.net.SocketPermission "localhost:7090", "listen,
    resolve";

    /* Add permissions for management server standard operations. */
    permission com.bea.jvm.ManagementPermission "createInstance";
    permission java.lang.RuntimePermission "modifyThreadGroup";
    permission java.lang.RuntimePermission "modifyThread";
    permission java.lang.RuntimePermission "shutdownHooks";
    permission java.util.PropertyPermission "*", "read, write";
};
```

Set the Port

When WebLogic JRockit JVM is started with the `-Xmanagement` option set—and provided the VM is not running in “quiet” mode—it should print out a short message following the command

line indicating that the management server is running and which port it is using. You can optionally choose which port to use by setting, as a command line argument, the port number in the `port` property:

```
java -Djrockit.managementserver.port=<portnumber>
```

The default port the management server uses to connect is 7090. It is strongly recommended that you block this port in your firewall, otherwise unauthorized users might access the management server.

Change the Number of Connections

You can change the number of connections allowed to the server by setting the `maxconnect` property:

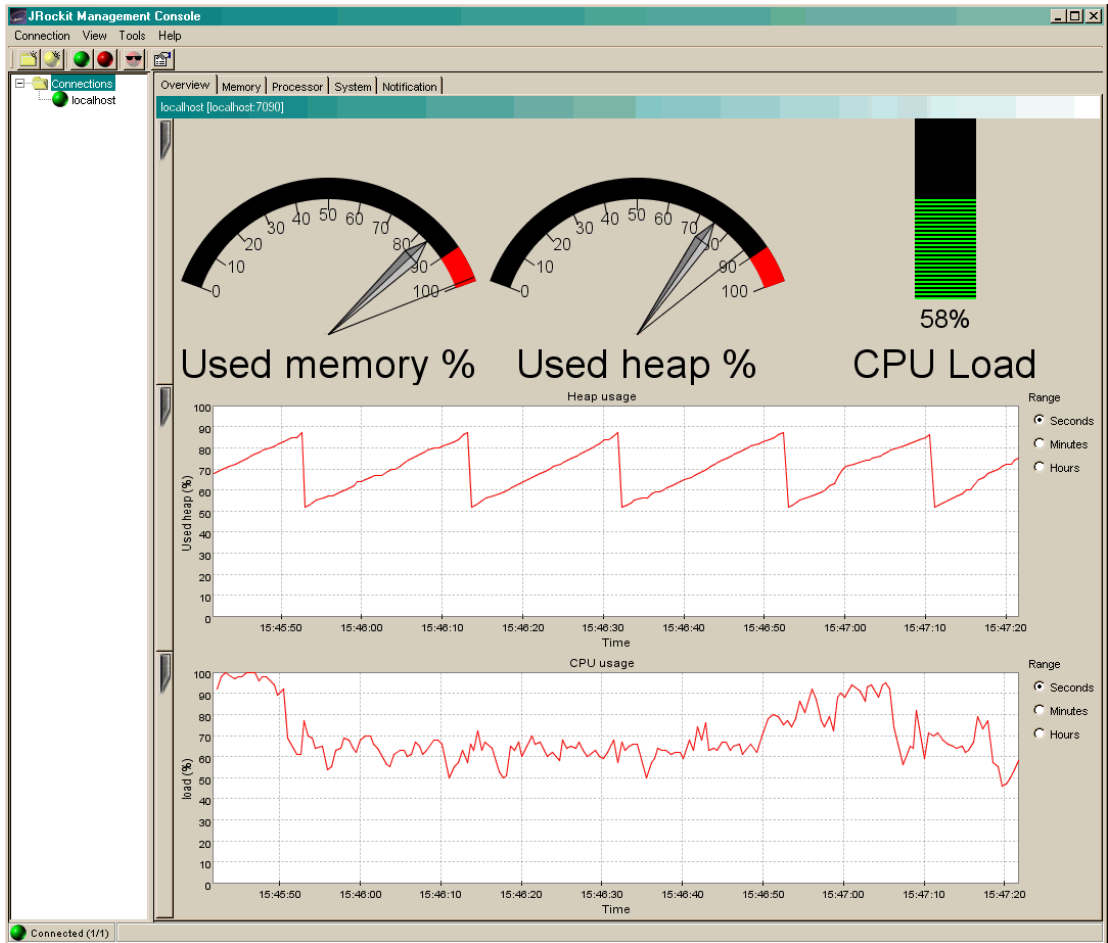
```
-Djrockit.managementserver.maxconnect=<maximum number of connections>
```

The default limit is four concurrent connections. While this should be enough for most users, you can change it, if necessary. The connection limit protects against Denial of Service (DoS) attacks by intruders.

Parts of the Console

When the JRockit Management Console window appears, the console has started, as shown in [Figure 6-1](#):

Figure 6-1 WebLogic JRockit JVM Management Console



The JRockit Management Console window is divided into two panes: a connection browser tree in the left pane (Figure 6-2) and a tabbed interface in the right pane (Figure 6-3).

Figure 6-2Connection Browser



Figure 6-3Information Tabs (Administrator Mode)



The first tab shows an Overview of information for the selected WebLogic JRockit JVM connection(s) (as highlighted in the connection browser pane). The other tabs contain detailed information about different areas of the VM, as will be described in [Information Tabs](#).

[Figure 6-3](#) shows the information tabs available in the console's Administrator operation mode. When the console is in the Developer mode, additional tabs appear, as shown in [Figure 6-4](#). These two operation modes are described in [Setting the Operation Mode](#).

Figure 6-4Information Tabs (Developer Mode)



The console includes a toolbar that contains command buttons for some of the menu options ([Figure 6-5](#)). To toggle the Toolbar on or off, on the View menu select Tool Bar.

Figure 6-5Management Console Toolbar



The status bar ([Figure 6-6](#)) at the bottom of the window displays informational messages and tool tips when you hover over a toolbar button or select something in a menu. It also indicates whether the JRockit Management Console is connected to one or several WebLogic JRockit JVM implementations or not. To toggle the Status Bar on or off, on the View menu, select Status Bar.

Figure 6-6Status Bar



Setting Up the Console

Once the console is running, you will need to configure it to suit your needs. Configuring—or “setting up”—the console includes these tasks:

- [Making Connections](#)
- [Enabling Console Settings](#)

Making Connections

The connection browser displays a collection of saved connections to WebLogic JRockit JVM organized in folders. If necessary, you can add your own folders and connection nodes to the tree structure. Active connections currently connected to a running VM are indicated by a green icon; those disconnected are indicated by a red icon.

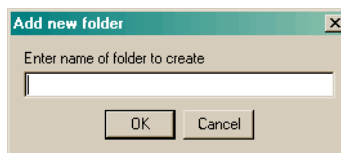
Creating a New Folder

To create your own folder in the connection browser, do the following:

1. Select an existing folder (for example, Connections) for which you want to create a subfolder.
2. Open the New Folder dialog box by doing one of the following:
 - Choose Connection→New Folder.
 - Press the right mouse button to open a context menu and select New Folder.
 - Press `Ctrl+N`.
 - Click the New Folder button on the toolbar.

The Add new folder dialog box ([Figure 6-7](#)) appears:

Figure 6-7Add New Folder Dialog Box



3. Enter the name of the new folder in the text field and click OK.

The new folder will appear in the connection browser.

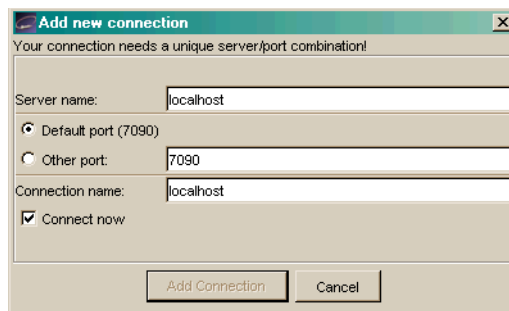
Creating a New Connection

To create a new connection to WebLogic JRockit JVM in the connection browser, do the following:

1. Select the folder in which the connection should be placed
2. Open the New Connection dialog box by doing one of the following:
 - Open the Connection menu and select New Connection.
 - Press the right mouse button to open a context menu and select New Connection.
 - Click the New Connection button on the toolbar.

The Add new connection dialog box ([Figure 6-8](#)) appears:

Figure 6-8Add New Connection Dialog Box



3. Enter the name of the server, the port and the new connection in the appropriate text fields or retain the default values. Then, select or deselect Connect now and click Add Connection.

Connecting a Connection to WebLogic JRockit JVM

To connect to WebLogic JRockit, do the following:

1. Select the WebLogic JRockit JVM connection to connect, a subfolder of connections to connect, or the folder Connections to connect all existing connections.
2. Do one of the following to connect the selected connection(s):
 - Open the Connection menu and select Connect.
 - Press the right mouse button to open a context menu and select Connect.
 - Press `Ctrl+O`.

- Click the Connect button on the toolbar.

When the connection is made, the status bar will read “Connected” and activity on the console will commence.

Disconnecting a Connection from WebLogic JRockit JVM

To disconnect a connection from WebLogic JRockit JVM, do the following:

1. Select the WebLogic JRockit JVM connection to connect, a subfolder of connections to connect, or the folder Connections to disconnect all existing connections.
2. Do one of the following to disconnect the selected connection(s):
 - Open the Connection menu and select Disconnect.
 - Press the right mouse button to open a context menu and select Disconnect.
 - Press `Ctrl+D`.
 - Click the Disconnect button on the toolbar.

The connection will be lost and the status bar will indicate that you’ve been disconnected. All activity on the console will cease.

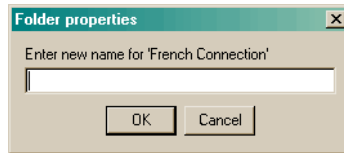
Renaming a Connection or Folder

To rename a connection or a folder of connection, do the following:

1. Select the WebLogic JRockit JVM connection or folder to rename.
2. Do one of the following to rename the selected connection or folder:
 - Open the Connection menu and select Rename.
 - Press the right mouse button to open a context menu and select Properties.
 - Press `F2`.
 - Click the name label of the item (see **Note**, below).

The Folder properties dialog box ([Figure 6-9](#)) appears:

Figure 6-9Folder Properties Dialog Box



3. Enter a new name into the text field and click OK

Note: If you select the last option (click the item label), the Folder properties dialog box will not appear. Instead, the label itself will be enabled for direct editing. Simply type the new name over the old and click away from the label or press Enter.

Removing a Connection or Folder

To remove a connection or folder, do the following:

1. Select a connection or a subfolder to remove.
2. Do one of the following to remove the selected item:
 - Open the Connection menu and select Remove.
 - Press the right mouse button to open a context menu and select Remove.
 - Press Delete.
3. Click Yes on the confirmation dialog box that appears.

The selected item disappears from the connection browser.

Hiding Disconnected Connections

Sometimes you might want to show just information about active WebLogic JRockit JVM connections. To hide information about disconnected connections, do one of the following:

- Open the View menu and select Hide Disconnected.
- Click the Hide Disconnected button on the toolbar.

To show the information about disconnected connections again, simply deselect Hide Disconnected in same way that you made the selection.

Enabling Console Settings

This section describes how to enable various JRockit Management Console settings.

Setting the Operation Mode

The Management Console can be run in two different operating modes:

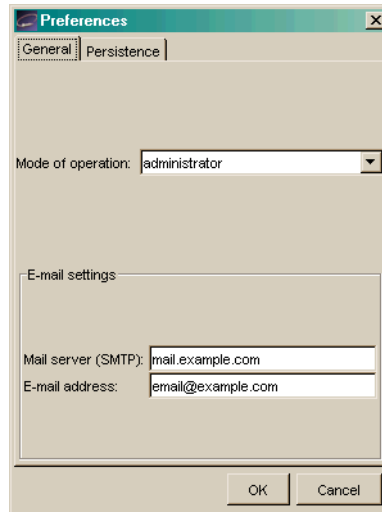
- **Administrator Mode**; This is the default mode, designed for system administrators who are interested in observing the state of the WebLogic JRockit JVM.
- **Developer Mode**; The developer mode is for developers and provides additional features such as a rudimentary method profiler and exception count functionality. Additional pages appearing in the developer mode are the Method Profiler page and the Exception Count page.

To set the operation mode, do the following:

1. From the Tools menu, select Preferences...

The Preferences dialog box (Figure 6-10) appears:

Figure 6-10 Preferences Menu (General Tab)



2. Click the Mode of operation drop-down control to display the list of operation modes (Figure 6-11).

Figure 6-11 List of Operation Modes



3. Select the mode you want to use and click OK.

Depending upon the mode to which you are toggling, the tabs on the console will change. See [Figure 6-3](#) and [Figure 6-4](#) for examples.

Setting Other Preferences

In addition to setting the operation mode, you can use the Preferences dialog box to change these settings:

- Default e-mail settings for the notification system (please refer to [Notification Tab](#)).
- Persistence behavior.

To change either of these values, open the Preferences dialog box from the Tools menu and proceed as described in the following sections:

Setting E-mail Preferences

To change e-mail preferences, do the following:

1. Display the General tab on the Preferences dialog box
2. In the appropriate text fields, enter the new e-mail information (SMTP server and E-mail address), as shown in [Figure 6-12](#).
3. Click OK

Figure 6-12E-mail Preferences Panel

A screenshot of a dialog box titled "E-mail settings". It contains two text input fields. The first field is labeled "Mail server (SMTP):" and contains the text "mail.example.com". The second field is labeled "E-mail address:" and contains the text "email@example.com".

E-mail settings	
Mail server (SMTP):	mail.example.com
E-mail address:	email@example.com

Enabling Persistence

Enabling the persistence means that aspect values are saved to a file and can be reviewed in charts by opening the View menu and selecting View Historical Data ([View Historical Data](#)).

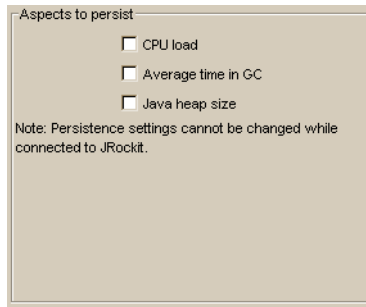
Selecting Aspects to Persist To set persistence preferences, do the following:

1. Disconnect any WebLogic JRockit JVM connections.

Note: If you have not disconnected the connections and attempt to use this dialog box, you will be prompted to disconnect.

The checkboxes in the Aspects to persist panel become enabled ([Figure 6-13](#)):

Figure 6-13 Aspects to Persist Panel



2. Select the aspects you want to persist.
3. Click OK.

The selected aspect values are saved to a file that you can review in charts as described in [“View Historical Data” on page 6-29](#).

Specifying the Persistence Directory In addition to setting preferences for the aspects to persist, you can also specify where to save the file that contains the aspect value (the “Persistence directory”). To do so:

1. Click Choose (next to the Persistence directory field).

If you are still connected to WebLogic JRockit JVM, you will be prompted to disconnect; click Yes to proceed. A standard Open dialog box appears.

2. Locate the directory where you want to save the file and click Open.

The Open dialog box closes, returning you to the Preferences dialog box.

3. Click OK.

The new Persistence directory will appear in that field.

Erasing Persistence Value Logs Finally, you can erase all persistence value logs by clicking Clear all aspect logs. You will see a confirmation message to which you should respond Yes. Be aware that, if you delete all persistence value logs by clicking this button, you will also delete any other files stored in the `<USER_HOME>/console/data` directory.

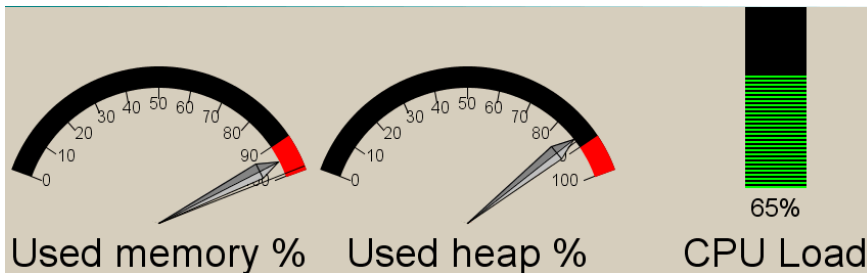
Customizing the Display

You can customize the console and change the way some of the monitoring data is displayed, as described in this section.

Customizing Gauges and Bars

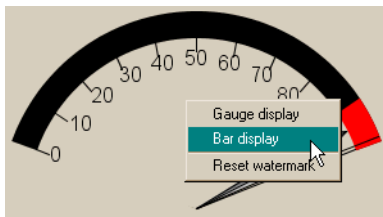
The gauges and bars are graphical devices showing memory and processor usage ([Figure 6-14](#)).

Figure 6-14Gauges and Bars

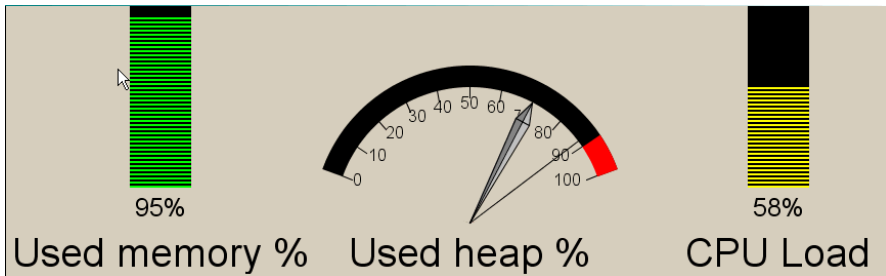


- To change from a gauge display to bar display, press the right mouse button when pointing at the gauge and select Bar display, as shown in [Figure 6-15](#).

Figure 6-15Gauge Context Menu (Bar Display Selected)



The selected gauge will appear as a bar ([Figure 6-16](#)).

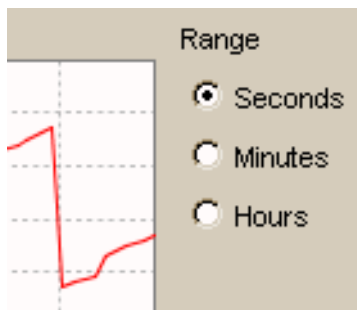
Figure 6-16 Gauges and Bars with Gauge Converted to a Bar Display

- To change back to a gauge, repeat the above, but select Gauge display.
- To reset the watermark—which indicates the highest level measured so far—press the right mouse button when pointing at the gauge or bar and select Reset Watermark.

Customizing Charts

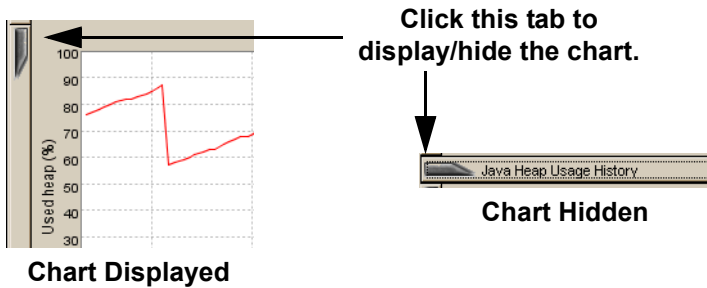
Charts appear on the JRockit Management Console to show specified information about WebLogic JRockit.

- To change scale on any of the chart, select the desired scale unit (seconds, minutes or hours) to the right of the chart ([Figure 6-17](#)) to be changed.

Figure 6-17 Range Selection Radio Buttons

- To hide a chart click the vertical tab at the left of the diagram you want to hide. When the diagram is hidden, the tab appears horizontally ([Figure 6-18](#)).

Figure 6-18 Hiding a Chart



- To show the diagram again, click the horizontal tab again.

Using the Settings File

When you exit the JRockit Management Console, your settings are automatically saved in a file called `consolesettings.xml`. This file is located in the folder:

```
<user home directory>\ManagementConsole
```

The exact path to the user home directory will vary on different platforms. On Windows it is usually something like `\Documents and Settings\<username>`; for example:

```
C:\Documents and Settings\jsmith\ManagementConsole
```

If no settings file exists in this directory it will be automatically created the next time the Management Console is closed.

Warning: Do not edit this file by hand! Doing so can make it unusable and may cause the Management Console to crash on startup.

If you are experiencing problems with the settings file, you can always delete it and let the Management Console create a new one for you.

Using the Console

The JRockit Management Console monitors different “aspects” of WebLogic JRockit JVM. An aspect is data that can be measured; for example, used heap size or VM uptime.

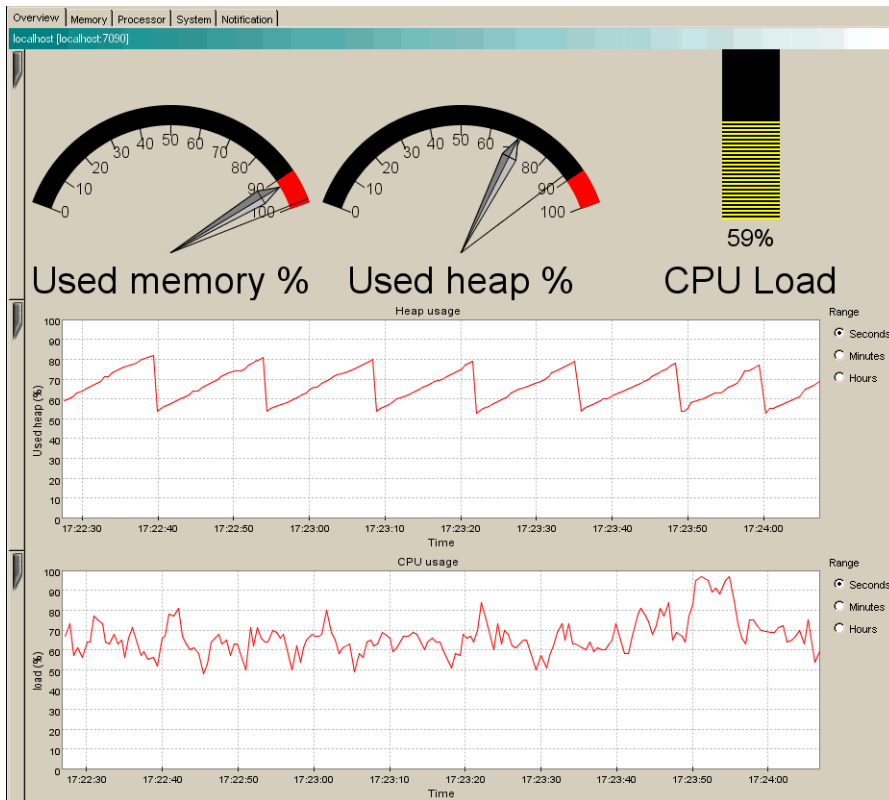
Information Tabs

Information tabs are pages containing details about different areas of the monitored WebLogic JRockit JVM. Display a tab by clicking it or by accessing the View menu. This section describes the tabs available on the JRockit Management Console.

Overview Tab

The Overview tab ([Figure 6-19](#)) shows an overview of selected connections. To select more than one connection, select the folder containing the connections you want to view. They will appear simultaneously. The page is divided into a “dashboard” with gauges in the upper part and charts in the lower part.

Figure 6-19 Overview Tab

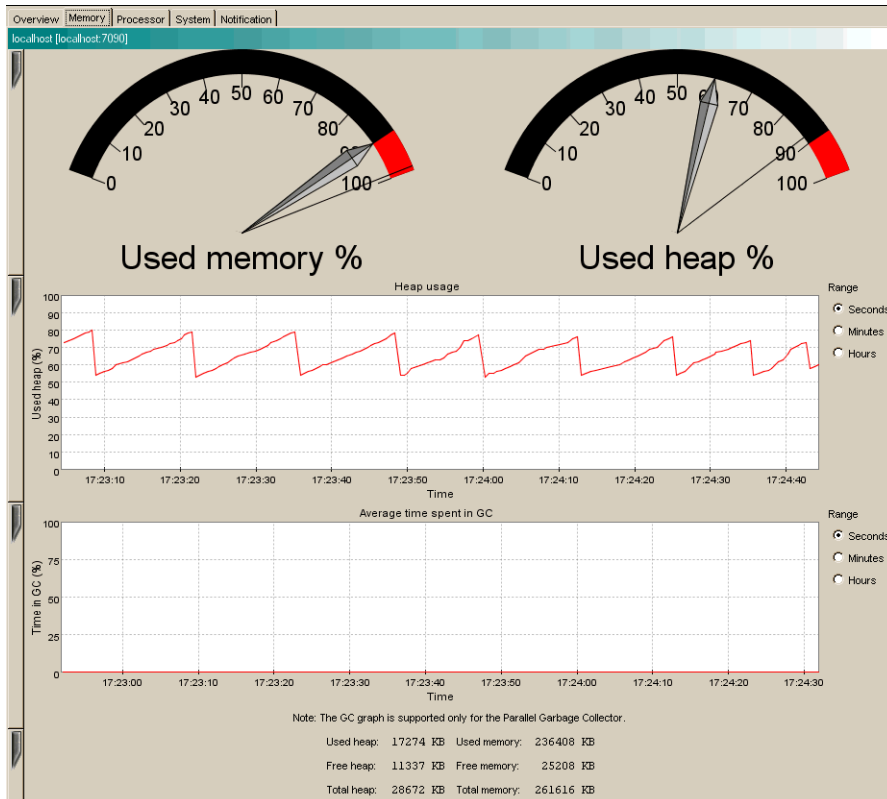


- The **Used Memory** gauge shows the percentage of occupied physical memory on the computer.
- The **Used Heap** gauge shows the percentage of occupied Java heap memory in the VM.
- The **CPU Load** bar shows the usage rate of the processor - or the average processor load on a multi-processor machine.
- The **Heap Usage** chart shows the percentage of used Java heap over time.
- The **CPU Usage** chart shows the average usage rate of the processor(s) over time.

Memory Tab

The Memory tab ([Figure 6-20](#)) shows information about the memory status of the system, as shown.

Figure 6-20Memory Tab



- The **Used Memory** gauge shows the percentage of machine memory in use.
- The **Used Heap** gauge shows the percentage of occupied Java heap.
- The **Heap Usage** chart shows the percentage of occupied heap over time.
- The **Time in GC** chart shows the average time spent on garbage collection over time. This chart is only updated when running WebLogic JRockit JVM with the Parallel garbage collector, and an actual garbage collection occurs.

At the bottom of the page the following text information is displayed (in kilobytes):

- **Used Heap** shows the occupied heap space.
- **Free Heap** shows the free heap space.

- **Total Heap** shows the heap size.
- **Used Memory** shows the amount of occupied physical memory.
- **Free Memory** shows the amount of free physical memory.
- **Total Memory** shows the total physical memory size.

Processor Tab

The Processor tab (Figure 6-21) shows information about the processor status of the system.

Figure 6-21 Processor Tab



- The **CPU Load** bar shows the average processor load as a percentage. The overall load is displayed in green while the load of the JVM process(es) is displayed in yellow.

- The **CPU Usage** chart shows the average processor load as a percentage over time.

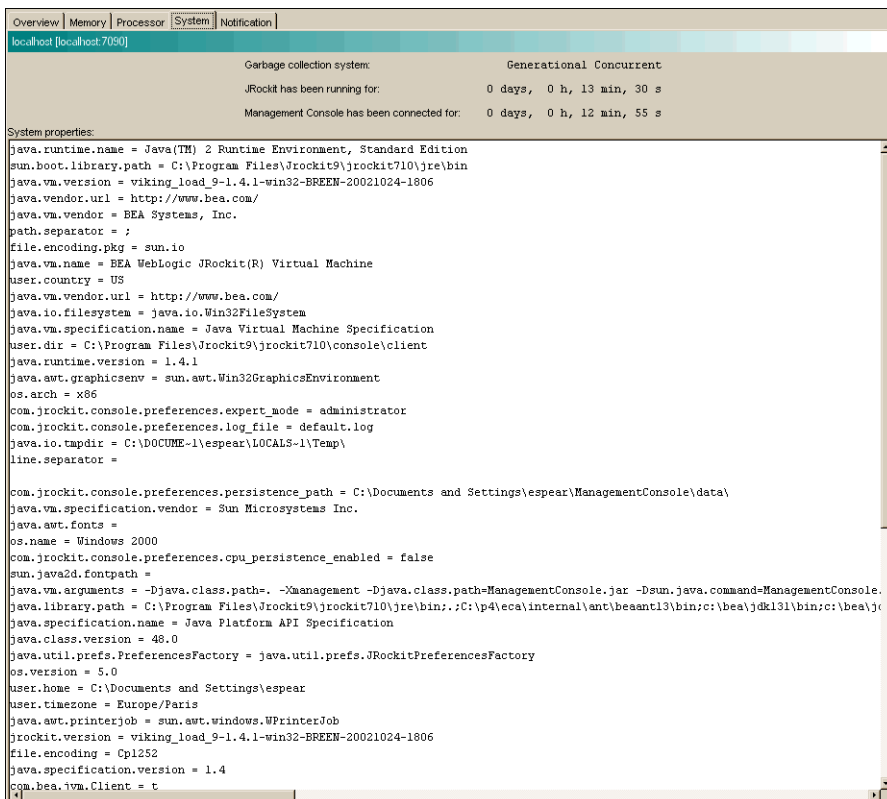
At the bottom of the page the following text information is displayed:

- **Number of Processors** shows the number of processors.
- **CPU Load** shows the overall processor load as a percentage.
- **JVM Process Load** shows the load of the WebLogic JRockit JVM process(es), expressed as a percentage.

System Tab

The System tab (Figure 6-22) shows various information about the system status.

Figure 6-22System Tab



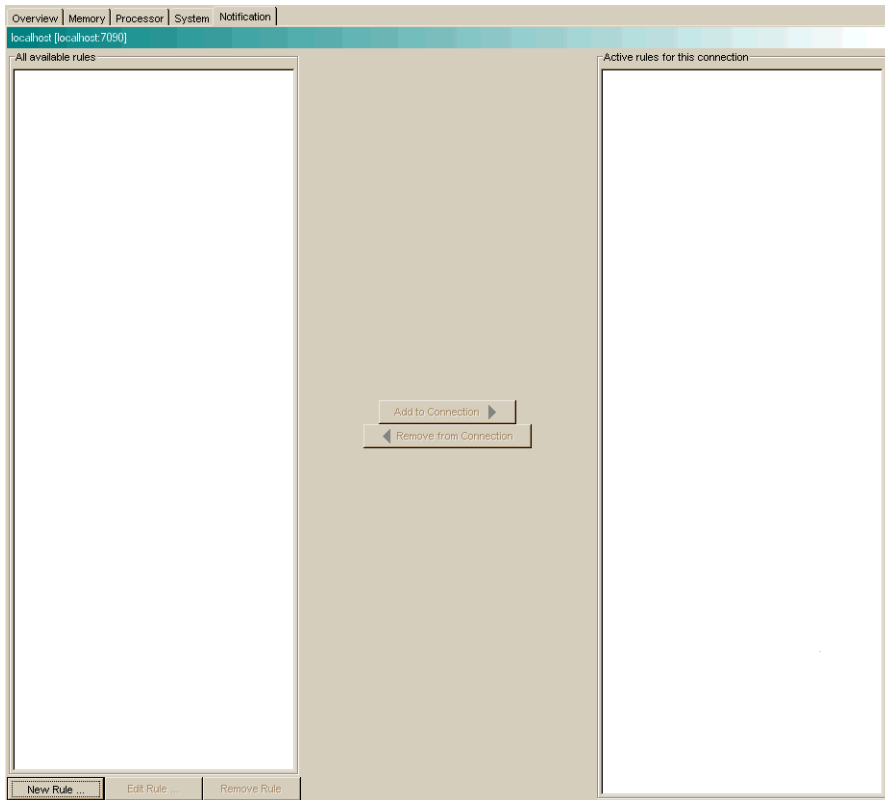
- **Garbage Collection System** shows which garbage collector WebLogic JRockit JVM is running.
- **JRockit Uptime** shows how long WebLogic JRockit JVM has been running.
- **Connection Uptime** shows how long the currently displayed connection has been connected.
- **Process Affinity** contains buttons that correspond to processors. It displays a green icon if WebLogic JRockit JVM is running on this processor and a red icon if it is not. By selecting a button, the WebLogic JRockit JVM process can be bound to one or more processors. The VM might be released from such a connection by deselecting the button again. This is only a suggested affinity: the operating system might not follow the suggestion. Changing the process affinity is a feature that is only available when monitoring a VM instance running on the Windows platform. The Process Affinity display is only activated when the Management Console is in the Developer mode, described in [Setting the Operation Mode](#).
- **System Properties** shows the Java System Properties loaded in the VM.

Notification Tab

Use the Notification tab ([Figure 6-23](#)) to define alerts that notify users when certain events occur. You can create your own notification rules based on different triggers, with optional constraints, that alert you with a prescribed notification. This section describes how to create these rules.

Creating Custom Actions and Constraints

After starting the Management Console for the first time, a file named `consolesettings.xml` will be created in the `\ManagementConsole` directory in your `<user_home>` directory. Among the contents of this file are the entries for the default actions and constraints. You can programmatically create custom notification actions and constraints, which are also stored in this file. Once added, these actions and constraints will appear on the Notifications tab of the Management Console. For complete information on creating custom notification actions and constraints, see [“Adding Custom Notification Actions and Constraints.”](#)

Figure 6-23 Notification Tab (No Rules Defined)

A notification trigger can be a certain event, for example, that the connection to WebLogic JRockit JVM was lost, or that an aspect reaches a certain value, for example, the used memory reaches 95%. A notification constraint can limit when a rule is triggered for example by not sending alerts at night or on certain dates.

The notification action is how the alert is communicated to the user. It can be one of the following:

- E-mail shows an e-mail when the notification is sent to the specified address by using the specified SMTP server.
- System out action displays the notification in the command window where you started the JRockit Management Console.
- Application alert displays the notification in an alert dialog in the Management Console.

- Log to file logs the notification to the specified file.

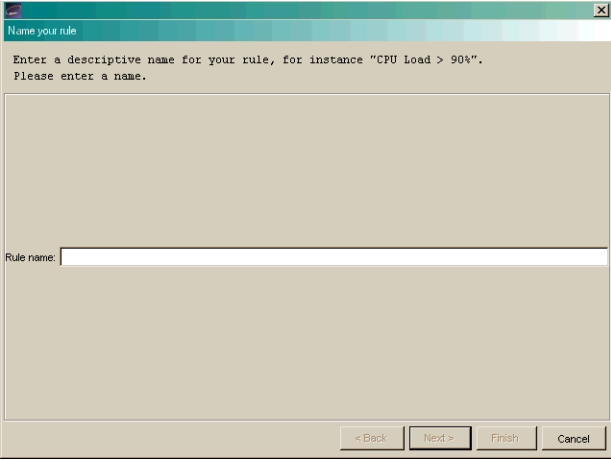
Creating a New Rule

Rules determine when and how to issue a notification. To create a new rule, do the following:

1. Click New Rule.

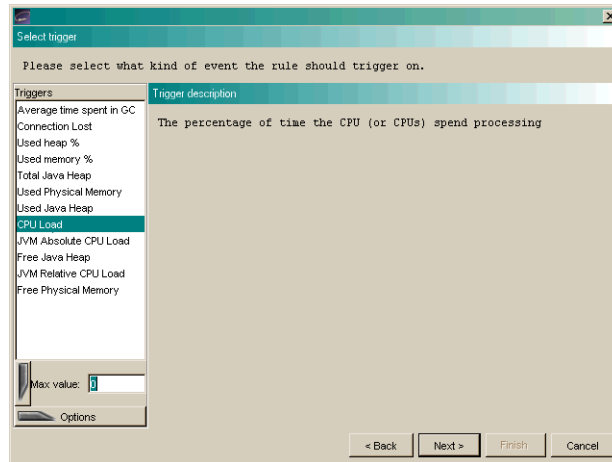
The Name rule dialog box appears (Figure 6-24):

Figure 6-24Name Rule Dialog Box

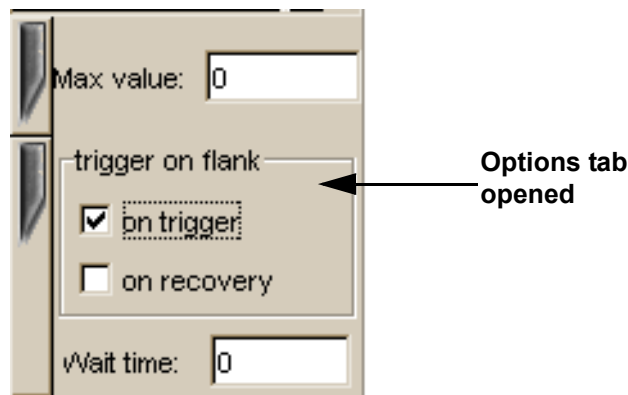
A screenshot of a Java Swing dialog box titled "Name your rule". The dialog has a light blue header bar with the title and a close button. Below the header, the main area has a light tan background. It contains the text "Enter a descriptive name for your rule, for instance 'CPU Load > 90%'. Please enter a name." followed by a large empty text area. At the bottom left, there is a label "Rule name:" followed by a single-line text input field. At the bottom right, there are four buttons: "< Back", "Next >", "Finish", and "Cancel".

2. Enter the name of the new rule in Rule name: and click Next.

The Select trigger dialog box appears (Figure 6-25):

Figure 6-25 Select Trigger Dialog Box

3. Select a trigger (the individual triggers are described in the right panel).
4. Enter a threshold in the text box below the trigger list, if required (Figure 6-26; this box will be marked either Min value or Max value, depending on the type of trigger selected).

Figure 6-26 Trigger Threshold and Options Text Boxes

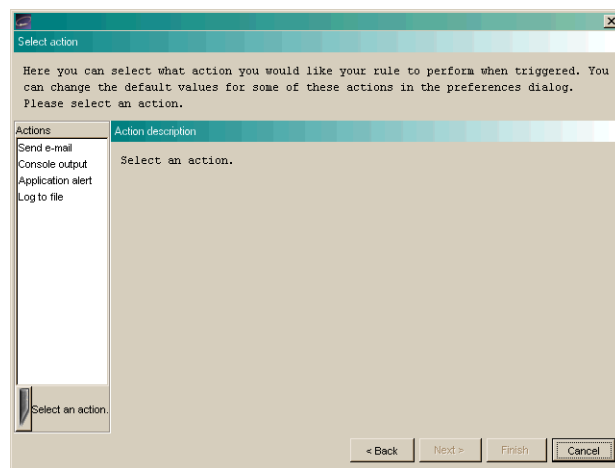
5. Select further options under the Option tab. For example, in Figure 6-26, you need to select what kind of aspect value change will trigger the notification:
 - on trigger, which triggers the notification when the aspect reaches the trigger value from a lower value (for example, if the trigger is 80 and the aspect value moves up from 75).

- on recovery, which triggers the notification when the aspect reaches the trigger value from a higher value (for example, if the trigger is 80 and the aspect value moves down from 85).

6. Click Next.

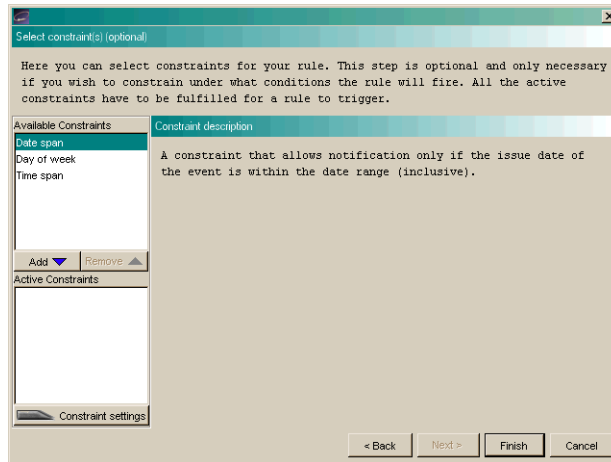
The Select Action dialog box appears (Figure 6-27):

Figure 6-27Select Action Dialog Box



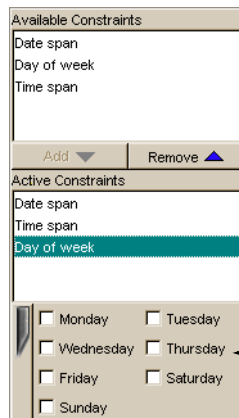
7. Select an action and enter settings data, if required.
8. If necessary, add a constraint to the rule (this step is optional; if you don't want to add a constraint, go to step 8):
- Click Next.

The Add Constraint dialog box appears (Figure 6-28):

Figure 6-28Add Constraint Dialog Box

- b. Select a constraint and click Add.

The constraint name will appear in the add list, as shown in [Figure 6-29](#).

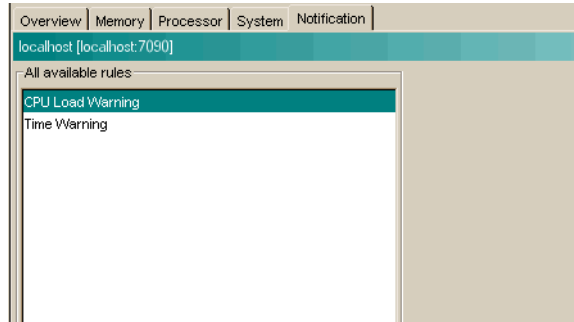
Figure 6-29Constraint Added

**Constraint settings;
Day of week
selected.**

- c. Enter constraint settings in the text fields under the list of constraints ([Figure 6-29](#)).
9. Click Finish.

The new rule appears in the All available rules list on the Notification tab, as shown in [Figure 6-30](#).

Figure 6-30New Rule in List



10. Add the rule to your connection as described in [Add a Rule to WebLogic JRockit JVM](#).

Editing a Rule

To edit a rule, do the following:

1. In the Available rules list, select the rule to be edited and click Edit Rule.
2. Check the name of the rule, edit it, if necessary, and click Next.
3. Check the trigger and trigger settings, edit them, if necessary, and click Next.
4. Check the action and the action settings and edit them if necessary.
5. To continue editing the rule, the do the following (optional; if you don't want to add a constraint, go to step 6):
 - a. Click Next.
 - b. Check the constraints and the constraint settings. Edit them, if necessary.
6. To finish the editing a rule, click Finish.

Add a Rule to WebLogic JRockit JVM

To add a rule to WebLogic JRockit JVM, do the following:

1. Select the rule to be added in the Available rules list.
2. Click Add to JRockit.

The rule appears in the Active rules for this connection list, as shown in [Figure 6-31](#).

Figure 6-31 Rule Added to Active rules for This Connection List

Remove a Rule from WebLogic JRockit JVM

To remove a rule from WebLogic JRockit JVM, do the following:

1. Select the rule to be removed in the Active rules for this connection list.
2. Click Remove from JRockit.

The rule will now be removed from the Active rules for this connection list.

Remove a Rule

To remove a rule from the Available rules list, do the following:

1. Select the rule to be removed.
2. Click Remove Rule.

A removal confirmation dialog box appears.

3. Click Yes
4. The rule disappears from the Available rules list.

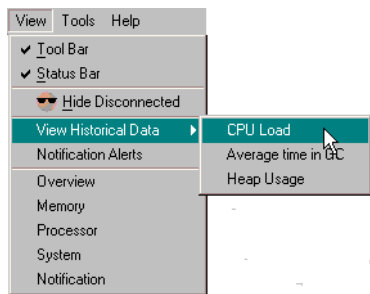
View Historical Data

The historical data window displays a chart where historical data for an aspect can be viewed. This is useful for observing trends over time and, for example, finding when a server running with WebLogic JRockit JVM has its peak loads.

To open this window, do the following:

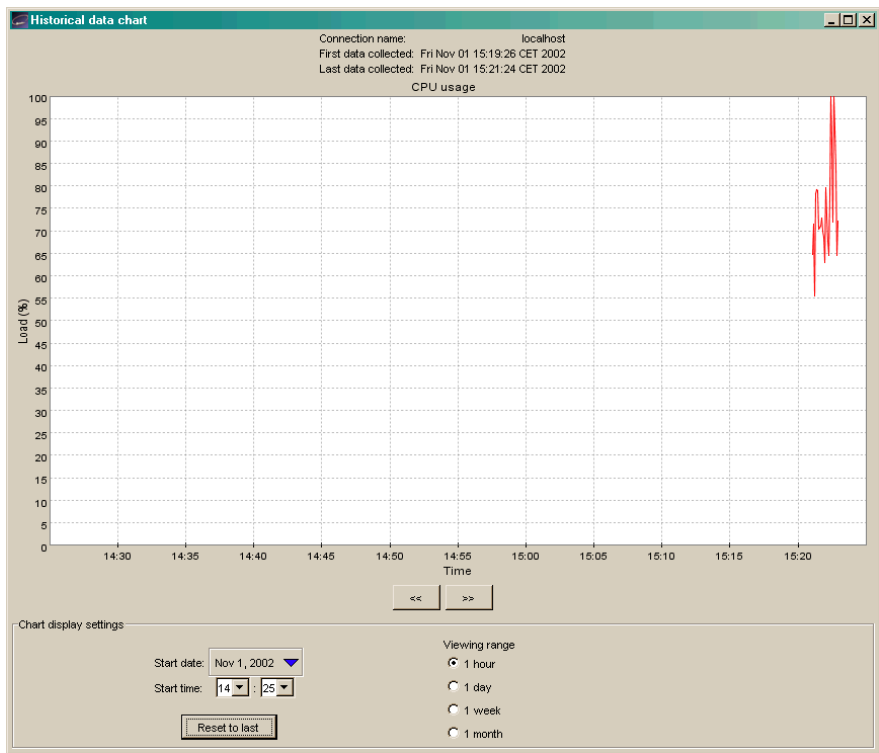
1. Select the connection for which you want to view data.
2. Open the View menu and select View Historical Data.
3. Select the aspect for which you want to view historical data, as shown in [Figure 6-32](#).

Figure 6-32View Menu with Historical Data Submenu Open



Historical data for the selected aspect appears (Figure 6-33).

Figure 6-33Historical Data (CPU Load Selected)



4. Navigate through time either by using the arrows or changing the start time in the Chart display settings.

To be able to observe historical data, aspect data from WebLogic JRockit JVM must first have been persisted, that is, written to file. See [Setting Other Preferences](#) to enable or disable persistence. The following aspects are possible to persist, and thus display, historical data for:

- Used heap (as a percentage)
- CPU load (as a percentage)
- Average time spent garbage collecting (as a percentage)

As soon as data has been created by a connected connection, it is available for historical observation.

Using Advanced Features of the Console

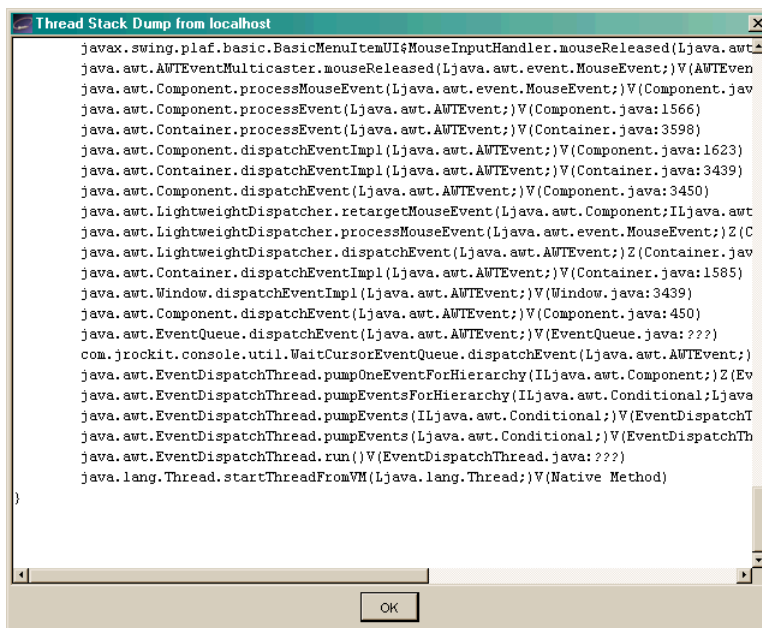
This section describes the more advanced features of the Management Console. Some of these are only available when running in the Developer mode, described in [Setting the Operation Mode](#).

View Thread Stack Dump

The stack dump contains a list of all running threads in WebLogic JRockit JVM with a method call stack trace for each thread.

To view the thread stack dump, open the Tool menu and select View Thread Stack Dump. A dialog box containing the stack dump appears ([Figure 6-34](#)).

Figure 6-34 Thread Stack Dump



Method Profiling Tab

Note: You must be in the developer operation mode before you can perform the tasks described in this section. For more information on entering the developer operation mode, see [Setting the Operation Mode](#).

The Method Profiler tab allows the developer to monitor method execution in a non-intrusive way. The Method Profiler can provide information about the average time spent in selected methods and the number of times methods are invoked.

Method Templates are collections of methods that can be re-used on different connections. There is a Default template, but the user may also create new templates.

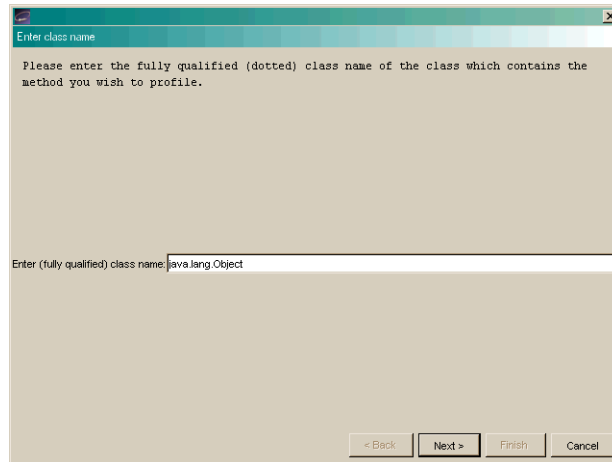
Adding a Method to a Template

To add a method to a template, do the following:

1. Select the template to be modified from the Select template list.
2. Click Add Method.

The Enter class name dialog box appears (Figure 6-35).

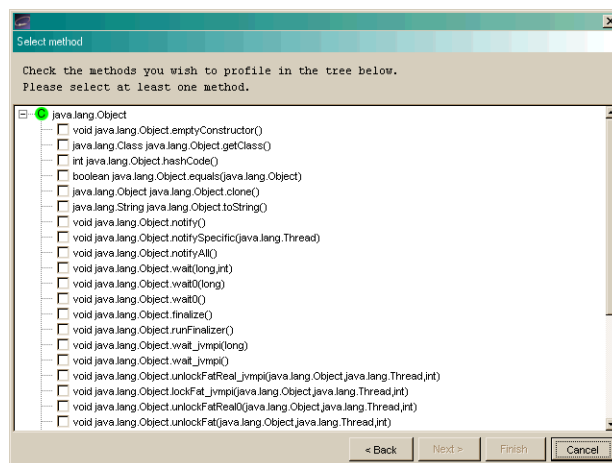
Figure 6-35Enter Class Name Dialog Box



3. Enter a fully qualified class name, for example, `java.util.Vector`, in the text field and click Next.

The Select method dialog box appears (Figure 6-36):

Figure 6-36Select Method Dialog Box



4. Select the methods to be added to the template and press Finish.

The method name will appear on the Method profiling information list, as shown in [Figure 6-37](#).

Figure 6-37Method Profiling Information List with Method Added

Method profiling information	
Method	Invocation count
StringBuffer StringBuffer.append(String)	
void Object.wait0(long)	
void System.arraycopy(Object,int,Object,int,i...	

Removing a Method from a Template

To remove a method from a template, do the following:

1. From the Select template list, select the template you want to modify.
2. From the Method Profiling Information list, select the method(s) to be removed from the template.
3. Click Remove Method.

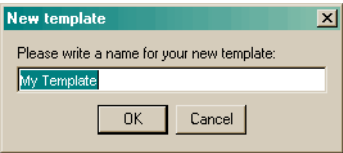
Creating a New Template

To create a new template, do the following:

1. Click New template.

The New template dialog box appears ([Figure 6-38](#)).

Figure 6-38New Template Dialog Box



2. Enter a name for the new template in the text field.
3. Click OK.

Removing a Template

To remove a template, do the following:

1. From the Select template list select the template to be removed.

2. Click Remove.

A confirmation dialog box appears.

3. Click Yes.

Starting and Stopping Method Profiling

To start the method profiling, do the following:

1. From the Select template list, select the template to be started.
2. Click Start/Stop.

If you select Start, numbers in the Invocation count cells for each method begin to increment as method calls are made. If you select Stop, this activity will cease.

Method Profiling Settings

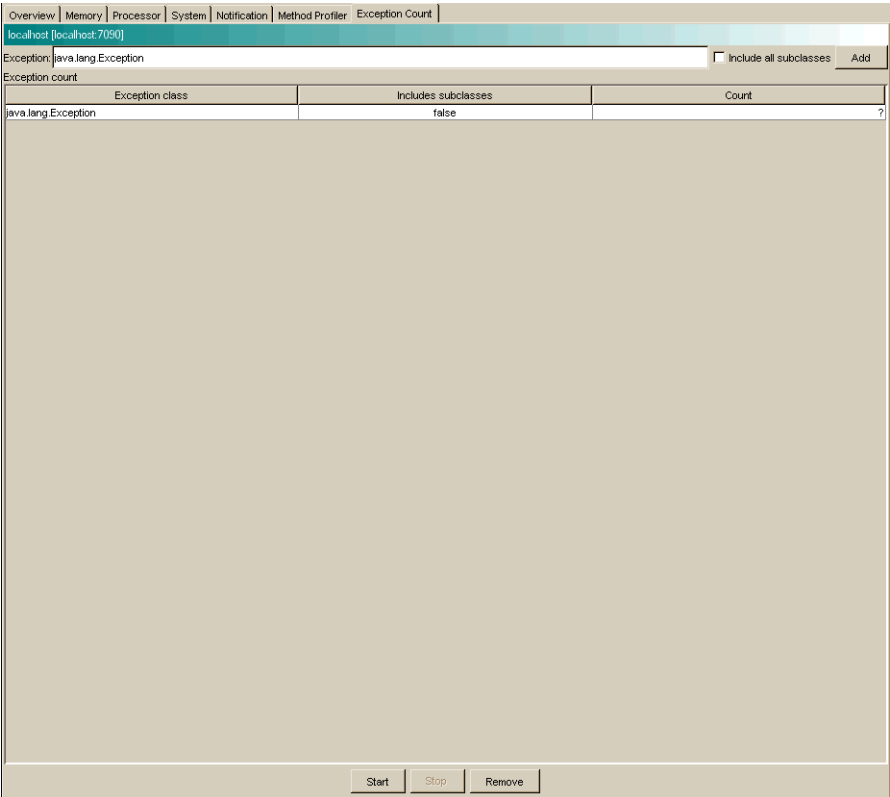
You can switch between using qualified method names or short names in the method profiling table.

- To enable invocation count, select the Invocation count checkbox at the bottom of the page.
- To enable timing, select the Timing checkbox at the bottom of the page.

Exception Counting Tab

The Exception Count tab ([Figure 6-39](#)) shows exceptions thrown in WebLogic JRockit JVM. It counts the number of exceptions of a certain type thrown.

Figure 6-39Exception Counting Tab



Add an Exception

To add an exception to observe, do the following:

- 1. Enter the fully qualified name of the exception into the text field at the top of the page, e.g., “java.io.IOException”.
- 2. Choose whether or not all subclasses of that exception should be included in the count by selecting or deselecting the Include subclasses checkbox.
- 3. Click Add. You can only add subclasses of java.lang.Throwable which are loaded in WebLogic JRockit JVM and you can only add exceptions while connected.

The exception should now be displayed in the table.

Starting, Stopping, and Removing an Exception Count

To start the exception count, click Start. The results should now appear next to the name of the exception being counted. Similarly, to stop the exception count, click Stop.

To remove an exception from the count, select the exception to be removed and click Remove.

Closing the Console

To close the JRockit Management Console and disconnect all connections, open the Connection menu and select Exit. Clicking X in the top right corner of the window will also close the JRockit Management Console.

Using the WebLogic JRockit Management Console

Using WebLogic JRockit JVM with Other WebLogic Applications

The configuration options described elsewhere in this user guide can be set to optimize WebLogic JRockit JVM performance with both BEA WebLogic Server and BEA WebLogic Workshop. This chapter defines these optimal settings and discussed how to use the JVM with these applications. It includes information on the following subjects:

- [Using WebLogic JRockit JVM with BEA WebLogic Server](#)
- [Configuring JRockit for BEA WebLogic Workshop](#)

Using WebLogic JRockit JVM with BEA WebLogic Server

BEA WebLogic JRockit JVM is certified for use with BEA WebLogic Server. This section includes information on the following subjects:

- [Certified Versions](#)
- [Verifying that WebLogic JRockit is Your JVM](#)
- [Starting JRockit from the Node Manager](#)
- [Enabling the Management Server from the Node Manager](#)
- [Tuning WebLogic JRockit for WebLogic Server](#)
- [Setting Options by Using the Node Manager](#)
- [Monitoring WebLogic JRockit JVM from WebLogic Server](#)
- [Running JRockit with Thin Threads on WebLogic Server](#)

- [Switching to WebLogic JRockit JVM in WebLogic Server](#)
- [Switching VMs When WebLogic Server is Running as a Service](#)

Certified Versions

For details on certified and supported platform combinations of WebLogic Server with WebLogic JRockit 8.0, please refer to the following Web pages:

<http://www.bea.com/products/weblogic/server/>

or

<http://www.bea.com/products/weblogic/jrockit/>

Verifying that WebLogic JRockit is Your JVM

WebLogic JRockit is the default production JVM shipped with WebLogic Server, although you can use another VM, such as Sun Microsystem's HotSpot JVM as a development VM. To ensure that WebLogic JRockit is the JVM running with your instance of WebLogic Server, at the command line, type:

```
java -showversion
```

If WebLogic JRockit is running, the system will respond:

```
java version "1.4.1_05"  
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_05)  
BEA WebLogic JRockit(R) Virtual Machine (build  
8.1-1.4.1_05-win32-CROSIS-20030317-1550, Native Threads, Generational  
Concurrent Garbage Collector)
```

Note: This example assumes you are using the native thread method (the default) and generational concurrent garbage collector (default when maximum heap size is larger than 128 MB).

Starting JRockit from the Node Manager

If you are starting WebLogic JRockit JVM from the WLS Node Manager, you need to enter the fully-qualifying path, as shown above, in the Java Home field on the Remote Start Page; for example:

```
\bea\jrockit81_141\bin\java
```


Enabling the Management Server from the Node Manager

You can enable the management server from the WLS Node Manager by doing the following:

1. Start the Node Manager as described in [Starting Node Manager with Commands or Scripts](#) and navigate to the Remote Start page.
2. Ensure that you have specified an absolute pathname to WebLogic JRockit JVM's top-level directory in the Java Home field
3. In Arguments, type `-Xmanagement`.

For more information on using the Node Manager, please refer to the [Overview of Node Manager](#) in [Configuring and Managing WebLogic Server](#).

Setting Options by Using the Node Manager

If you started the server or cluster of servers with the Node Manager and specified an absolute pathname to WebLogic JRockit JVM's top-level directory in the Java Home field on the Node Manager's Remote Start page, you can set any option from this page, too. Simply enter the option and any arguments in the Arguments field.

For more information on using the Node Manager, please refer to the [Overview of Node Manager](#) in [Configuring and Managing WebLogic Server](#).

Tuning WebLogic JRockit for WebLogic Server

To use the WebLogic JRockit JVM instead of the Sun JVM, you need to increase the initial heap size to 64 MB (`-Xms:64m`) and the maximum heap size to at least 200 MB (`-Xmx:200m`). In addition, the following defaults are used:

- `-Xnativethreads`
- `-Xallocationtype:local`

These settings are normally used for initial development. If you want to improve WebLogic JRockit performance, you can try one of the following, bearing in mind that all applications are different and you need to verify which settings give the best performance in each case:

- Increase the heap initial and maximum size (`-Xms` and `-Xmx`).
- Change the garbage collector to single spaced concurrent (`-Xgc:singlecon`) or parallel (`-Xgc:parallel`). Note that if you select parallel as your garbage collector, the `-Xns` setting will have no effect on processing (see [Setting the Size of the Nursery](#)).

For more information on tuning WebLogic JRockit, please refer to [Tuning WebLogic JRockit JVM](#).

Monitoring WebLogic JRockit JVM from WebLogic Server

If you run WebLogic Server with WebLogic JRockit JVM, you can use the WebLogic Server Administration Console to view runtime data about the VM and the memory and processors on the computer hosting it.

To monitor WebLogic JRockit JVM, do the following:

1. Start WebLogic Server with WebLogic JRockit JVM as the VM.
2. In the left pane of the Administration Console, expand the Servers folder.
3. Click a server that is using the WebLogic JRockit JVM.
4. In the right pane, click the Monitoring tab. Then click the JRockit tab.

The JRockit tab displays monitoring information.

Table 7-1 WebLogic JRockit Attributes Monitored by the WebLogic Server Administration Console

Attribute	Description
Total Nursery Size	Indicates the amount (in bytes) of memory that is currently allocated to the nursery. The nursery is the area of the Java heap where objects are initially allocated. Instead of garbage collecting the entire heap, generational garbage collectors focus on the nursery. Because most objects die young, most of the time it is sufficient to garbage collect only the nursery and not the entire heap. If you are not using a generational garbage collector, the nursery size is 0.
Max Heap Size	Indicates the maximum amount of memory (in bytes) that the VM can allocate for its Java heap. This number is fixed at startup time of the VM, typically by the <code>-Xmx</code> option.
Gc Algorithm	Indicates the type of garbage collector that WebLogic JRockit JVM is using.
Total Garbage Collection Count	Indicates the number of garbage collection runs that have occurred since the VM was started.

Table 7-1 WebLogic JRockit Attributes Monitored by the WebLogic Server Administration Console

Attribute	Description
GCHandles Compaction	<p>Indicates whether the VM's garbage collector compacts the Java heap. Usually the heap is scattered throughout available memory. A garbage collector that compacts the heap defragments the memory space in addition to deleting unused objects.</p> <p>Values:</p> <ul style="list-style-type: none"> • true • false
Concurrent	<p>Indicates whether JRockit's garbage collector runs in a separate Java thread concurrently with other Java threads.</p> <p>Values:</p> <ul style="list-style-type: none"> • true • false
Generational	<p>Indicates whether JRockit's garbage collector uses a nursery space. Instead of garbage collecting the entire heap, generational garbage collectors focus on the nursery. Because most objects die young, most of the time it is sufficient to garbage collect only the nursery and not the entire heap.</p> <p>Values:</p> <ul style="list-style-type: none"> • true • false
Incremental	<p>Indicates whether JRockit's garbage collector collects only a small portion of the heap during each old collection (incremental) or collects the whole heap during each collection (non-incremental).</p> <p>Values:</p> <ul style="list-style-type: none"> • true • false
Parallel	<p>Indicates whether the JRockit's garbage collector is able to run in parallel on multiple processors if multiple processors are available.</p> <p>Values:</p> <ul style="list-style-type: none"> • true • false
Number Of Processors	<p>Displays the number of processors on JRockit's host computer. If this is not a Symetric Multi Processor (SMP) system, the value will be 1.</p>

Table 7-1 WebLogic JRockit Attributes Monitored by the WebLogic Server Administration Console

Attribute	Description
Total Number Of Threads	Indicates the number of Java threads (daemon and non-daemon) that are currently running on JRockit across all processors.
Number Of Daemon Threads	Indicates the number of daemon Java threads currently running on JRockit across all processors.

To view additional data about WebLogic JRockit, such as how long it spends in a specific method, use the WebLogic JRockit Management Console, as described in [Using the WebLogic JRockit JVM Management Console](#).

Running JRockit with Thin Threads on WebLogic Server

Warning: Thin threads is experimental functionality in this version of JRockit, and is not recommended for general use. This feature is subject to change without notice.

The JRockit high performance thread system ([Thin Threads](#), `-Xthinthreads`) and the native I/O system of WebLogic Server are incompatible as they both use asynchronous I/O. To avoid problems you must disable the native I/O system of WebLogic Server when running JRockit using thin threads. The native I/O is disabled automatically in WebLogic Server if JRockit is using thin threads, even if it is turned on in the corresponding WebLogic Server configuration file. In their respective default setups, WebLogic JRockit JVM does not use thin threads and WebLogic Server uses native I/O.

Switching to WebLogic JRockit JVM in WebLogic Server

When you switch to WebLogic JRockit JVM in WebLogic Server, any changes to the VM and start-up setting, should be handled by the WLS Configuration Wizard. Additionally, if any installation-wide scripts must be updated due to the switch, these will also be handled by the WLS Configuration Wizard.

Among information that needs to be changed when switching to WebLogic JRockit JVM are:

- The value for the `JAVA_HOME` variable needs to be changed to the absolute pathname to the top BEA directory; for example, `c:\bea\jrockit81`.

You can also change the `JAVA_HOME` variable from the [Node Manager's](#) Remote Start page by entering the absolute pathname in the Java Home field.

- Change the value of the `JAVA_VENDOR` variable to `BEA`.

You will also need to restart any servers that are currently running.

For complete details on switching to WebLogic JRockit JVM from another JVM, please refer to [Migrating to WebLogic JRockit](#). For more information on using the Configuration Wizard when switching to WebLogic JRockit, please refer to [Changing the JVM that Runs Servers](#).

Switching VMs When WebLogic Server is Running as a Service

To switch the virtual machine when WebLogic Server is running as a service, do the following:

1. Stop the service.
2. Start `regedit` and find the service keys corresponding to your service
`(HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/{ServiceName})`.
3. In the Parameters folder, change the value of the key `JavaHome` from the default VM to your WebLogic JRockit SDK directory.
4. Here you can also alter the arguments sent to the VM by editing the values of the key `CmdLine`.
5. Restart the service.

Configuring JRockit for BEA WebLogic Workshop

If you are running JRockit with BEA WebLogic Workshop, we recommend that you use the same configuration parameters specified for WebLogic Server in [Tuning WebLogic JRockit for WebLogic Server](#).

What's in the WebLogic JRockit 8.1 SDK?

WebLogic JRockit 8.1 SDK is very similar to the Sun JDK, except that it includes a new JRE with the WebLogic JRockit JVM and some changes to the Java class libraries (however, all of the class libraries have the same behavior in WebLogic JRockit as in the Sun JDK). For a more detailed description of the differences between the two SDKs, please refer to [File Differences Between WebLogic JRockit 8.1 SDK and Sun HotSpot SDK](#).

This section describes the contents of the WebLogic JRockit 8.1 SDK and compares a WebLogic JRockit SDK installation to a comparable Sun SDK installation. It includes information on the following subjects:

- [SDK Contents](#)
- [File Differences Between WebLogic JRockit 8.1 SDK and Sun HotSpot SDK](#)

SDK Contents

This section describes the various components that make up the WebLogic JRockit 8.1 SDK. It also identifies the folder in which these components reside.

Development Tools

Found in: `/bin`

Development tools and utilities help you develop, execute, debug, and document programs written in the Java programming language. The WebLogic JRockit 8.1 SDK includes the standard tools commonly distributed with the typical Java SDKs. While most of these are standard JDK tools and are proven to work well with Java development projects, you are free to use any other

third party tools, compilers, debuggers, IDEs, and so on that might work best in your situation. The tools included with WebLogic JRockit 8.1 SDK are:

- `Javac` compiler
- `Jdb` debugger
- Javadoc, which is used to create an HTML documentation site for the JVM API

For more information on these tools, please refer to Sun Microsystem's Java™ 2 SDK Tools and Utilities website at:

<http://java.sun.com/j2se/1.4/docs/tooldocs/tools.html>

Runtime Environment

Found in: `/jre`

The BEA WebLogic JRockit implementation of the Java 2 runtime environment for use by the SDK. The runtime environment includes the BEA WebLogic JRockit JVM, class libraries, and other files that support the execution of programs written in the Java programming language.

Additional Libraries

Found in: `/lib`

Additional class libraries and support files required by the development tools.

C Header Files

Found in: `/include`

Header files that support native-code programming using the Java Native Interface, the Java Virtual Machine Debugger Interface, the Java Virtual Machine Profiler Interface and other functionality of the Java 2 Platform.

The Management Console

Found in: `/console`

The WebLogic JRockit Management Console is used to monitor and control running instances of WebLogic JRockit JVM. It provides real-time information about the running application's characteristics, which can be used both during development—for example, to find where in an

application's life cycle it consumes more memory-and in a deployed environment-for example, to monitor the system health of a running application server.

File Differences Between WebLogic JRockit 8.1 SDK and Sun HotSpot SDK

This section describes how WebLogic JRockit SDK differs from Sun Microsystems' HotSpot SDK. Each table below lists, by component and operating system (O/S), files that either exist in HotSpot SDK 1.4.1 or WebLogic JRockit SDK.

Be aware of these variables:

- `$ARCH` = i386 on linux32, but `$ARCH` = ia64 on linux64
- `mydir[/*]` means `mydir` and `mydir/*`

The following files are included in the 8.1 SP1 SDK by accident and should not be considered part of the JRockit 8.1 SP1 SDK.

Table A-1 Included Files Not Part of WebLogic JRockit

O/S	Files
win32	jre/bin/jpicpl32.cpl
linux32	jre/lib/\$ARCH/libjawt.so
linux64	jre/lib/\$ARCH/libjsig.so

Table A-2 Files Contained Only in Sun SDK; Not Supported by WebLogic JRockit

Component	O/S	File	Notes
Sun Plugin Support	win32	bin/HtmlConverter.exe lib/htmlconverter.jar jre/bin/ActPanel.dll jre/bin/jpi*. * jre/bin/NP*. * jre/bin/eula.dll	
	linux32	bin/ControlPanel jre/bin/ControlPanel jre/ControlPanel.html bin/HtmlConverter lib/htmlconverter.jar jre/lib/i386/libjavaplugin_jni.so jre/lib/javaplugin.jar jre/lib/locale[/ *] jre/plugin[/ *]	
Sun Java Web Start	win32	jre/javaws-1_2_0_03-windows-i586-i.exe	
	linux32	jre/javaws-1_2_0_03-linux-i586-i.zip	
Sun AWT Native Interface	win32	include/jawt.h	
	win64	include/win32/jawt_md.h jre/bin/jawt.dll jre/lib/jawt.lib lib/jawt.lib	
	Linux32	include/jawt.h	
	linux64	include/linux/jawt_md.h jre/lib/\$ARCH/libjawt.so	
Sun VM Native Interface	win32	lib/jvm.lib	This is an empty file in WebLogic JRockit that only exists to enable OptimizeIt detection
	win64		

Table A-2 Files Contained Only in Sun SDK; Not Supported by WebLogic JRockit

Component	O/S	File	Notes
Sun JCOV	win32	jre/bin/jcov.dll	
	win64	jre/lib/jvm.jcov.txt	
	linux32	jre/lib/\$ARCH/libjcov.so	
	linux64	jre/lib/jvm.jcov.txt	

Table A-3 Files Used Only in Sun SDK; Not used by WebLogic JRockit

Component	O/S	Files	Notes
SUN Hotspot VM support files	win32	jre/bin/msvcrt.dll	
	win64	jre/bin/msvcrt.dll	win64 only
		jre/bin/net.dll	
		jre/bin/nio.dll	
		jre/bin/zip.dll	
		jre/bin/hpi.dll	
		jre/bin/client [/*]	win32 only
		jre/bin/server [/*]	
	linux32	jre/lib/\$ARCH/libnet.so	
	linux64	jre/lib/\$ARCH/libnio.so	
		jre/lib/\$ARCH/libzip.so	
		jre/lib/\$ARCH/libjsig.so*8	
		jre/lib/\$ARCH/client [/*]	linux32 only
		jre/lib/\$ARCH/server [/*]	

Table A-4 Files Contained Only in Sun SDK; Removed from WebLogic JRockit to Reduce Package Size

Component	O/S	Files	Notes
SUN Demo sources	win32	demo [/*]	
	win64		
	linux32		
	linux64		

What's in the WebLogic JRockit 8.1 SDK?

Table A-4 Files Contained Only in Sun SDK; Removed from WebLogic JRockit to Reduce Package Size

SUN Java API sources	win32	src.zip
	win64	
	linux32	
	linux64	
SUN Manual pages	linux32	man [/*]
	linux64	

Table A-5 Files Only in WebLogic JRockit SDK; Not Used or Supported by Sun

Component	O/S	Files	Notes
WebLogic JRockit Management Console	linux32	bin/console [.exe]	
	linux64	console [/*]	
	win32		
	win64		
WebLogic JRockit Management API	win32	jre/lib/managementapi.jar	
	win64		
	linux32		
	linux64		
WebLogic JRockit JVM support files	win32	jre/bin/dbghelp.dll	
	win64	jre/bin/jrockit [/*]	
		jre/lib/managementserver.jar	
	linux32	jre/lib/\$ARCH/jrockit [/*]	
	linux64	jre/lib/managementserver.jar	

Adding Custom Notification Actions and Constraints

After starting the WebLogic JRockit JVM Management Console for the first time, a file named `consolesettings.xml` will be created in the `\ManagementConsole` directory in your home directory. Among other entries, this file contains the deployment entries for the default actions and constraints. You can create custom notification actions and constraints for the Management Console, which are also stored in this file. Once added, these actions and constraints will appear on the Notifications tab of the Management Console.

This appendix includes information on the following subjects:

- [Locating consolesettings.xml](#)
- [Creating a Custom Action](#)
- [Creating and Implementing an Action: Example](#)
- [Creating a Custom Constraint](#)

Locating consolesettings.xml

The `consolesettings.xml` file is located in your home directory, under the `\ManagementConsole` folder. If you are using Windows, the path should be:

`C:\Documents and Settings\<user_name>\ManagementConsole`

(where `<user_name>` is the user name under which you are running the Management Console)

If you are using Linux, the path will normally be:

`/home/<user_name>/ManagementConsole`

(where `<user_name>` is the user name under which you are running Management Console)

Creating a Custom Action

The following procedure walks you through the steps necessary to create and implement a custom action. In this procedure, you will be creating a print action.

1. Add the `ManagementConsole.jar` to your build path.

You can find this `.jar` in the `<jrockit_home>/console` directory.

2. Create a subclass of `AbstractNotificationAction`. This class will receive the `NotificationEvents`.

3. Implement `handleNotificationEvent`:

```
public void handleNotificationEvent(NotificationEvent event)
```

You can also override the `exportToXml` and `initializeFromXml` methods to store your action settings to XML.

4. Create a subclass of `AbstractNotificationActionEditor` to create the graphical editor used to edit the settings. If you have no editable settings for your action, you can just use the `com.jrockit.console.notification.ui.NotificationActionEditorEmpty`.

5. Implement the abstract methods:

```
protected void storeToObject(Describable object);  
protected void initializeEditor(Describable object);
```

6. Edit the `consolesettings.xml` file to include your new action under the `<registry_entry>` element.

7. Add your new classes in the classpath.

8. Run the console.

The new action will be available in the new rule dialog box in the notification section of the Management Console (see [Notification Tab](#)).

Creating and Implementing an Action: Example

This section shows a real-life example of how an action is created and implemented. Once implemented, a text field where you can enter a parameter will appear on the Notification tab.

The step numbers that appear in headings below refer to the steps in the procedures under [Creating a Custom Action](#).

Note: This example assumes that `ManagementConsole.jar` has been added to the build path (Step 1).

Create the Action (Step 2)

First, we create a subclass of `AbstractNotificationAction`, as shown in [Listing B-1](#). This class will receive the `NotificationEvents`.

Listing B-1 Building the Parameterized Action

```
package com.example.actions;
import org.w3c.dom.Element;

import com.jrockit.console.notification.*;
import com.jrockit.console.util.XmlToolkit;

/**
 * Test class showing how to build a parameterized action.
 *
 * @author Marcus Hirt
 */
public class MyTestAction extends AbstractNotificationAction
{
    private final static String TEST_SETTING = "test_param";
    public final static String DEFAULT_VALUE = "default value";
    private String m_parameter = DEFAULT_VALUE;

    /**
     * @see com.jrockit.console.notification.NotificationAction#
     * handleNotificationEvent(NotificationEvent)
     */
    public void handleNotificationEvent(NotificationEvent event)
    {
        System.out.println("===MyTestAction with param: " +
            getParameter() + "=====");
    }
}
```

Adding Custom Notification Actions and Constraints

```
        System.out.println(NotificationToolkit.prettyPrint(event));
    }

    /**
     * @see com.jrockit.console.util.XmlEnabled#exportToXml
     * (Element)
     */
    public void exportToXml(Element node)
    {
        XmlToolkit.setSetting(node, TEST_SETTING, m_parameter);
    }

    /**
     * @see com.jrockit.console.util.XmlEnabled#initializeFromXml
     * (Element)
     */
    public void initializeFromXml(Element node)
    {
        m_parameter = XmlToolkit.getSetting(node, TEST_SETTING,
            DEFAULT_VALUE);
    }

    /**
     * Returns the parameter.
     *
     * @return some parameter.
     */
    public String getParameter()
    {
        return m_parameter;
    }

    /**
     * Sets the parameter.
     *
     * @param parameter the value to set the parameter to.
     */
    public void setParameter(String parameter)
```



```

    {
        m_parameter = parameter;
    }
}

```

Implementing handleNotificationEvent() (Step 3)

While creating the subclass of `AbstractNotificationAction` created, we implemented `handleNotificationEvent()`, as shown in [Listing B-2](#). This method acts on the incoming event.

Listing B-2 Implementing handleNotificationEvent

```

public class MyTestAction extends AbstractNotificationAction
{
    private final static String TEST_SETTING = "test_param";
    public final static String DEFAULT_VALUE = "default value";
    private String m_parameter = DEFAULT_VALUE;

    /**
     * @see com.jrockit.console.notification.NotificationAction#
     * handleNotificationEvent(NotificationEvent)
     */
    public void handleNotificationEvent(NotificationEvent event)
    {

```

Creating the Action Editor (Step 4)

Next, we create a subclass of `AbstractNotificationActionEditor` to create the graphical editor used to edit the settings. [Listing B-3](#) shows how this is done.

Listing B-3 Creating the Action Editor

```

package com.example.actions;

```

Adding Custom Notification Actions and Constraints

```
import java.awt.*;
import javax.swing.*;

import com.jrockit.console.notification.Describable;
import com.jrockit.console.notification.ui.AbstractNotification
    ActionEditor;

/**
 * Simple test editor. Displays a text field where you can enter a
 * parameter.
 * (Note that you'd get better layout results using a GridbagLayout.)
 *
 * @author Marcus Hirt
 */
public class MyTestActionEditor extends AbstractNotificationActionEditor
{
    private JTextField m_parameterField = new
        JTextField(MyTestAction.DEFAULT_VALUE);

    /**
     * Constructor for MyTestActionEditor.
     */
    public MyTestActionEditor()
    {
        super();
        setName("MyTestAction settings");
        add(new JLabel("Param:"), BorderLayout.WEST);
        add(m_parameterField, BorderLayout.CENTER);
        setMinimumSize(new Dimension(140,0));
    }

    /**
     * @see com.jrockit.console.notification.ui.Abstract
     * Editor#initializeEditor(com.jrockit.console.notification.
     * Describable)
     */
    protected void initializeEditor(Describable action)
    {
        m_parameterField.setText(((MyTestAction) action).
            getParameter());
    }
}
```

```

    }
    /**
     * @see com.jrockit.console.notification.ui.AbstractEditor#
     * storeToObject(com.jrockit.console.notification.Describable)
     */
    protected void storeToObject(Describable action)
    {
        ((MyTestAction) action).setParameter(m_parameterField.
            getText());
    }
}

```

Implementing the Abstract Methods (Step 5)

When we created the action editor above, we implemented the abstract methods `initializeEditor()` and `storeToObject()`, as shown in [Table B-4](#).

Listing B-4 Implementing the Abstract Methods

```

    */
    protected void initializeEditor(Describable action)
    {
        m_parameterField.setText(((MyTestAction) action).
            getParameter());
    }
    /**
     * @see com.jrockit.console.notification.ui.AbstractEditor#
     * storeToObject(com.jrockit.console.notification.Describable)
     */
    protected void storeToObject(Describable action)
    {
        ((MyTestAction) action).setParameter(m_parameterField.
            getText());
    }
}

```

Adding the New Action to the Deployment Entries (Step 6)

Before the action and editor can appear on the Management Console, you need to add it to the deployment entries in `consolesettings.xml`, under the `<registry_entry>` element, as shown in [Listing B-5](#).

Listing B-5 Adding the New Action to the Deployment Entries

```
<registry_entry>
  <entry_class>
    com.company.actions.MyTestAction
  </entry_class>
  <entry_name>
    Test action
  </entry_name>
  <entry_description>
    Test action, dynamically added.
  </entry_description>
  <entry_editor_class>
    com.company.actions.MyTestActionEditor
  </entry_editor_class>
</registry_entry>
```

Displaying the New Action Editor (Steps 7 and 8)

Finally, add the new classes to your classpath and start the console. When you navigate to the Notifications tab, you'll see the new editor on the tab.

Creating a Custom Constraint

Create custom constraints by using the same procedure described in [Creating a Custom Action](#), except that you must implement:

```
boolean validate(NotificationEvent event)
```

instead of:

```
void handleNotificationEvent(NotificationEvent event)
```

as shown in [Listing B-6](#):

Listing B-6 Code Change for Creating a Customer Constraint

```
public class MyTestAction extends AbstractNotificationAction
{
    private final static String TEST_SETTING = "test_param";
    public final static String DEFAULT_VALUE = "default value";
    private String m_parameter = DEFAULT_VALUE;

    /**
     * @see com.jrockit.console.notification.NotificationAction#
     * handleNotificationEvent(NotificationEvent)
     */

    boolean validate(NotificationEvent event)
```

Adding Custom Notification Actions and Constraints

Index

Symbols

../tuning/config.html 4-7

A

adaptive optimization 1-3
Administrator mode 6-11
Adobe Acrobat Reader 1-5
architecture 1-2
aspect 6-16
aspect value change 6-25
 add constraint 6-27
 on trigger 6-25
asynchronous I/O 7-6

C

command line option
 -Xstrictfp 3-4
 -Xverify 3-4
command line options
 -classpath 3-3
 -D 3-3
 -Djrockit.managementserver.maxconnect
 6-4
 -Djrockit.managementserver.port 6-4
 -help 3-4
 -showversion 3-3
 -verbose 3-4
 -version 3-3
 -X 3-4
 -Xallocationtype 7-3
 -Xgc 4-4

 -Xgcpause 4-7
 -Xmanagement 6-2
 -Xnativethreads 5-3, 7-3
 -Xnoop 3-4
 -Xthinthreads 5-3, 7-6
 -Xverbose 3-5

command-line option 3-11
configuration options 7-1
consolesettings.xml 6-16
context switching 5-1, 5-2, 5-4
conventions
 documentation 1-6
CTRL_LOGOFF_EVENT 3-10

D

dead objects 4-3
default values, thread system 4-6
Developer Mode 6-11
documentation
 conventions 1-6
 PDF version 1-5
 printing 1-5
 to print 1-5

E

extended options
 -Xnohup 3-11

G

garbage collection 7-4
 benefits of type 4-4

choosing 4-4, 4-6

concurrent 4-3

- generational concurrent 4-3

- generational concurrent 4-4, 4-5, 4-6

- single spaced concurrent 4-3, 4-4, 4-5, 4-6

drawbacks of type 4-4

generational copying 4-2, 4-4, 4-5, 4-6

nursery 4-2

old generation 4-3

parallel 4-3, 4-4, 4-5, 4-6

pause time 4-2

pauses 4-3, 4-5

young generation 4-3

garbage collector 2-3, 4-2

- Generational Concurrent 2-3

- Generational Stop 'n' Copy 2-3

- Parallel 2-3

- Single-Spaced Concurrent 2-3

Generational Concurrent 2-3

Generational Copy 2-3

generational copying 4-2

H

heap 7-4, 7-5

- size 4-2, 4-3, 4-5

high performance thread system 7-6

High Performance Threading System (thin threads) 2-3

I

IA64, thread system limitations 5-3, 5-4

Information tabs

- Overview tab 6-17

Intel Architecture 1-2

Intel architecture 1-1

J

Java heap memory 6-18

Java thread 4-3, 4-5, 4-7, 5-2, 5-4, 7-5, 7-6

Java Virtual Machine (JVM) 1-1

java.lang.System 3-3

JDBC database drivers 5-4

jrockit.dump 1-4

M

Management Console 6-1, 6-16

- adding an exception 6-36

- Administrator mode 6-6

- advanced features

- Exception Count tab 6-35

- Method Profiler tab 6-32

- method templates 6-32

- changing the number of connections 6-4

- command buttons 6-6

- connecting a connection to JRockit 6-8

- connection browser 6-5, 6-7

- connection node 6-7

- Connection Uptime 6-22

- CPU Load 6-21

- CPU Load bar 6-18

- CPU Usage chart 6-18

- customizing 6-14

- charts 6-15

- gauges and bars 6-14

- settings file 6-16

- dash board 6-17

- disconnecting a connection from JRockit 6-9

- enabling console settings 6-10

- enabling the management server 6-2

- Exception Count 6-11

- Free Heap 6-19

- Free Memory 6-20

- Garbage Collection System 6-22

- Heap Usage chart 6-18, 6-19

- hiding a disconnected connection 6-10

- information tabs 6-17

- Memory tab 6-18
- Notification tab 6-22, 6-27
- Processor tab 6-20
- System tab 6-21
- JRockit Uptime 6-22
- JVM Process Load 6-21
- Method Profiler 6-11
- new connection 6-8
- Number of Processors 6-21
- parts of 6-4
- Process Affinity 6-22
- removing a connection 6-10
- removing a folder 6-10
- renaming a connection 6-9
- renaming a folder 6-9
- setting the operation mode 6-11
- setting the port 6-3
- setting up 6-7
- starting 6-2
- starting, stopping, and removing an
 - exception count 6-37
- status bar 6-6
- System Properties 6-22
- tabbed interface 6-5
- thread stack dump 6-31
 - viewing 6-31
- Time in GC chart 6-19
- Total Heap 6-20
- Total Memory 6-20
- Used Heap 6-19
- Used Heap gauge 6-18, 6-19
- Used Memory 6-20
- Used Memory gauge 6-19
- Used Memory gauge 6-18
- management console
 - Developer mode 6-6
- memory management 2-3
- memory throughput 4-2, 4-5
- Method Profiling
 - Method Profiling Information List 6-34
 - settings 6-35

- starting and stopping 6-35
- Method Templates 6-32
 - adding a method 6-32
 - creating a new template 6-34
- Method Profiling Information List 6-34
- removing 6-34
- removing a method 6-34

N

- native threads 5-1, 5-2, 5-4
- notification action 6-23
 - Application alert 6-23
 - creating a new rule 6-24
 - editing a rule 6-28
 - E-mail 6-23
 - Log to file 6-24
 - System out 6-23
- notification constraint 6-23
- notification trigger 6-23
- nursery 4-2, 7-4, 7-5

O

- object allocation 4-1, 4-2
- operating system thread 5-2, 5-4
- operation mode 6-6, 6-11, 6-12
 - Developer mode 6-11

P

- pause time 4-2
- performance optimization 7-1
- persistence value log 6-13
- printing product documentation 1-5
- product version 3-3, 3-4

S

- SIGHUP 3-10
- Single-Spaced Concurrent 2-3
- Support 1-4

Symmetric Multi Processor (SMP) system, 7-5
system property, Java 3-3

T

thin threads 5-4, 7-6
thread scheduling 2-3, 5-1, 5-2, 5-4
thread stack dump
 viewing 6-31
thread switching 2-3, 5-2, 5-4
thread synchronization 2-3, 5-2, 5-4
thread system 2-3, 5-1, 5-3
 choosing 5-3
 default 5-1
 default values 4-6
 High Performance Threading System (thin
 threads) 2-3
 Native Threads 2-3
 native threads 5-1, 5-2, 5-3, 5-4
 operating system threads 5-1, 5-4
 selection matrix 5-4
 thin threads 5-1, 5-3, 5-5
 thread allocation
 local 4-1
thread-local object 4-1

V

verbose output 3-2, 3-4

W

WebLogic 1-1
WebLogic Platform 1-1
WebLogic Server 7-1, 7-6
 Administration Console 7-4
 Concurrent 7-5
 Gc Algorithm 7-4
 GCHandles Compaction 7-5
 Generational 7-5
 Incremental 7-5
 Max Heap Size 7-4

Number Of Daemon Thread 7-6
Number Of Processors 7-5
Parallel 7-5
Total Garbage Collection Count 7-4
Total Number Of Threads 7-6
Total Nursery Size 7-4
 configuring JRockit for 7-3
WebLogic Workshop 7-1, 7-7

X

-Xnohup 3-11

Y

young generation 4-2