BEA JRockit SDK

Using BEA JRockit SDK

Version 1.4.2
June 2004

Contents

Introduction

What's In the User Guide? 1-1

Finding Additional Information 1-2

BEA JRockit Support 1-2

Supported Platforms 1-2

Tuning BEA JRockit 1-2

Documentation 1-2

Starting and Configuring BEA JRockit JVM

Before Starting BEA JRockit 2-1

Starting BEA JRockit 2-1

Sample Start-up Command 2-2

Configuring BEA JRockit 2-2

Using Standard Options 2-3

Setting the JVM Type 2-3

Setting General Information 2-3

Providing Information to the User 2-4

Using Non-standard Options 2-4

Setting Behavioral Options 2-5

Displaying Logging Information 2-5

Including a Timestamp with Logging Information 2-9

Protecting Systems by Using the Security Manager 2-10

Preventing BEA JRockit JVM (When Run as a Service) from Shutting Down After Receiving a Logoff Event 2-10

Special Instructions for Linux Users 2-10

Enabling Core Dumps on Red Hat AS 2-11

Overriding NPTL. 2-11

Using the BEA JRockit Memory Management System

The Mark-and-Sweep Collection Model 3-2

Garbage Collector Permutations 3-2

Generational 3-2

Single-spaced 3-3

Concurrent 3-3

Parallel 3-3

Running the Dynamic Garbage Collector 3-3

Using Static Garbage Collection Methods 3-4

Using Backward-compatible Garbage Collectors 3-5

Pros and Cons 3-5

Garbage Collector Selection Matrix 3-6

Setting the Default Garbage Collector 3-7

Overriding Garbage Collectors 3-8

Viewing Garbage Collection Activity 3-8

Thread-local Allocation. 3-9

Using the BEA JRockit Management Console

Console Overhead 4-1

Starting the Console. 4-2

Enable the Management Server 4-2

Attaching a Management Client 4-2

Start the JRockit Management Console 4-2

Starting the Management Server with a Security Manager. 4-3
Set the Port 4-4
Change the Number of Connections. 4-4
Parts of the Console. 4-4
Setting Up the Console 4-6
Making Connections. 4-6
Creating a New Folder 4-7
Creating a New Connection 4-7
Connecting a Connection to BEA JRockit JVM. 4-8
Disconnecting a Connection from BEA JRockit JVM 4-8
Renaming a Connection or Folder 4-9
Removing a Connection or Folder 4-10
Hiding Disconnected Connections 4-10
Enabling Console Settings 4-10
Setting the Operation Mode 4-10
Setting Other Preferences 4-11
Customizing the Display. 4-13
Using the Settings File 4-15
Using the Console 4-16
Information Tabs 4-16
Overview Tab 4-16
Memory Tab 4-17
Processor Tab 4-19
System Tab 4-20
Notification Tab. 4-22
View Historical Data 4-28
Using Advanced Features of the Console 4-30
View Thread Stack Dump 4-31

Method Profiling Tab 4-31

Exception Counting Tab 4-35

Creating a JRA Recording. 4-37

Closing the Console. 4-38

Starting and Running the Console in the Headless Mode 4-39

Running a Headless Management Console 4-39

Controlling the Console with Command-line Options. 4-39

Using the BEA JRockit Memory Leak Detector

Starting the Memory Leak Detector 5-2

Using the Memory Leak Detector 5-4

Overview of the Memory Leak Detection Process. 5-4

Getting Started 5-5

Memory Leak Detection. 5-6

An Example of How to Find a Real Memory Leak 5-9

Help Us Improve BEA JRockit 5-11

How will BEA Systems Use This Feedback 5-12

BEA JRockit Support for the Memory Leak Detector 5-12

Frequently Asked Questions 5-12

Does BEA Systems Guarantee the Accuracy of this tool's output?. 5-12

Does the Memory Leak Detector Cause Any Overhead?. 5-12

What Kind of Support is Available for the Memory Leak Detector? 5-13

Is There a Forum Where I can Discuss the Memory Leak Detector? 5-13

Known Issues. 5-13

Code Caching with BEA JRockit

Why Is Code Caching Helpful?. 6-2

What is the Cache? 6-2

How to Use Code Caching 6-2

Enabling Code Caching 6-3

Specifying a Cache Name 6-3

Code Caching in the Read/Write Mode 6-3

Code Caching in the Read-only Mode 6-3

Other Code Caching Arguments 6-4

Using Code Caching to Improve Performance 6-4

Setting the Verbosity Level 6-4

Enabling Code Caching by Using an Environment Variable 6-5

How Code Caching Works 6-5

What Happens When Code Caching Runs 6-5

Dealing with Code Changes 6-6

Dealing with Cache Cleanup 6-6

Removing Obsolete Methods 6-6

Cache File Validity 6-7

Error Recovery 6-7

Cleaning Up the Cache 6-7

Using BEA JRockit JVM with Other WebLogic Applications

Using BEA JRockit JVM with BEA WebLogic Server 7-1

Certified Versions 7-2

Verifying that BEA JRockit is Your JVM 7-2

Starting JRockit from the Node Manager 7-2

Enabling the Management Server from the Node Manager 7-2

Setting Options by Using the Node Manager 7-3

Tuning BEA JRockit for WebLogic Server 7-3

Monitoring BEA JRockit JVM from WebLogic Server 7-4

Switching to BEA JRockit JVM in WebLogic Server 7-6

Switching VMs When WebLogic Server is Running as a Service 7-6

Configuring JRockit for BEA WebLogic Workshop 7-7

Adding Custom Notification Actions and Constraints

Locating consolesettings.xml A-1

Creating a Custom Action A-2

Creating and Implementing an Action: Example A-2

Create the Action (Step 2) A-3

Implementing handleNotificationEvent() (Step 3). A-5

Creating the Action Editor (Step 4) A-5

Implementing the Abstract Methods (Step 5) A-7

Adding the New Action to the Deployment Entries (Step 6). A-8

Displaying the New Action Editor (Steps 7 and 8) A-8

Creating a Custom Constraint A-8

Using the Java Plugin

Supported Operating Systems and Browsers B-2

Installing the Plugin B-2

Note on Installing the BEA JRockit Plugin and Sun Plugin B-2

Implementing the Plugin B-3

Plugin Reference B-3

Tracing Thread Activity With Stack Dumps

Monitoring Information in Stack Dumps C-1

Detecting Deadlocks C-3

What is a “Lock Chain”? C-3

Lock Chain Types C-4

Open Chains C-4

Deadlock Chains C-4

Closed Chains C-4

Using Web Start with BEA JRockit

What You Can Do with Web Start D-1

Web Start Security. D-2

Installing and Launching Web Start D-2

Windows Implementations D-2

Linux Implementations. D-2

Comprehensive Web Start Documentation D-3

Index

Introduction

Welcome to Using BEA JRockit SDK. This document contains procedures and other information necessary for you to gain optimal performance from BEA Systems' industry-leading Java Virtual Machine, BEA JRockit.

This Introduction includes information on the following subjects:

- What's In the User Guide?
- Finding Additional Information

What's In the User Guide?

This user guide is organized as follows:

- Using the BEA JRockit Memory Management System describes how to implement the best memory management system—or garbage collection method—for your application. Garbage collection is the process of clearing dead objects from the heap, thus releasing that space for new objects.
- Starting and Configuring BEA JRockit JVM describes how to start BEA JRockit JVM and configure it to provide the best performance for your application.
- Using the BEA JRockit Management Console shows how to monitor and control running instances of BEA JRockit JVM using this graphic tool. The management console provides real-time information about the running application's characteristics, which is helpful both during development and in a deployed environment

- Using BEA JRockit JVM with Other WebLogic Applications discusses how to use the JVM configuration options to optimize BEA JRockit JVM performance with both BEA WebLogic Server and BEA WebLogic Workshop.
- Adding Custom Notification Actions and Constraints shows you how to create custom notification actions and constraints for the Management Console.

Finding Additional Information

You can find additional information about BEA JRockit throughout this documentation set. For a complete list of available documents, please refer to BEA JRockit SDK 1.4.2 Online Documentation. The following list cites the most commonly referenced information.

BEA JRockit Support

To get support for BEA JRockit, please refer to "BEA JRockit 1.4.2 SDK Support" in *Introduction to* BEA JRockit.

Supported Platforms

For a list of platforms supported by BEA JRockit, please refer to "Supported Platforms".

Tuning BEA JRockit

Tuning information can be found in *Tuning BEA JRockit 1.4.2 JVM*.

Documentation

Descriptions of all documents available with BEA JRockit and of the document conventions used is available in Using BEA JRockit 1.4.2 SDK Documentation.

Starting and Configuring BEA JRockit JVM

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

- Before Starting BEA JRockit
- Starting BEA JRockit
- Configuring BEA JRockit

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

Before Starting BEA JRockit

Before starting BEA 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 BEA JRockit

To start the BEA 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 BEA JRockit, below. See

Tuning BEA JRockit JVM for details on the tuning options available for this version of BEA 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\jrockit`*`xxx`*`\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 -Xgcprio:throughput -Xmx:256m -Xms:64m -Xns:24m myClass
```

In this example, the following options are set:

- `-verbose:memory`—Displays verbose output about memory usage.
- `-Xgcprio:throughput`—A dynamic garbage collector prioritized for memory throughput 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 BEA JRockit

When you start BEA 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 BEA JRockit. It contains information on the following subjects:

- Using Standard Options for:
  - Setting the JVM Type
  - Setting General Information
  - Providing Information to the User
- Using Non-standard Options for:
  - Setting Behavioral Options

  - Displaying Logging Information
  - Including a Timestamp with Logging Information
  - Protecting Systems by Using the Security Manager
  - Preventing BEA JRockit JVM (When Run as a Service) from Shutting Down After Receiving a Logoff Event
- Special Instructions for Linux Users
  - Enabling Core Dumps on Red Hat AS
  - Overriding NPTL

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 BEA JRockit JVM, Sun Microsystem's HotSpot JVM, or any other third-party JVM.

Setting the JVM Type

The following commands set the type of JVM you want to run, server-side or client-side:

- `-server`

  Starts BEA JRockit JVM as a server-side JVM. This value is the default.

- `-client`

  Starts BEA JRockit JVM as a client-side JVM. This option is helpful if you have a smaller heap and are anticipating shorter runtimes for your application.

You should be aware that setting the JVM type (or accepting the default) will also set the garbage collection algorithm that will be used during runtime. `-server` will start a single-spaced, parallel mark, parallel sweep collector while `-client` will start a a single-spaced, concurrent mark, concurrent sweep garbage collector. If you want to use a different collector, such as the dynamic, unified garbage collector or one of the specific fixed garbage collectors, you can override the default by using either the `-Xgcprio` or `-Xgc` command line options. For more information on garbage collection and the `-server` and `-client` options, please refer to "Using -server and -client to Set a Fixed Garbage Collector" in Using BEA JRockit Memory System.

Setting General Information

The following standard command line options set general information about BEA 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 2-1.

- `-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 in the following section).

Using Non-standard Options

Non-standard, or `-X`, command line options are options that are exclusive to BEA JRockit JVM that change the behavior of BEA 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 BEA 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 BEA 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 BEA 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 2-1 lists the parameters available for `-Xverbose`.

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

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

Table 2-1 -Xverbose Parameters

This Parameter...	Prints to the screen...

`codegen`	The names of each method that is being compiled. Verbose output for `codegen` might look like this:
	`[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`

`cpuinfo`	Any interesting information about your CPUs. Verbose output for `cpuinfo` might look like this:
	`[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`

`load`	The names of each loaded class. Verbose output for `load` might look like this:
	`[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+`

Table 2-1 -Xverbose Parameters

This Parameter…	Prints to the screen…
`memory;` `gc`	Information about the memory management system, including: • 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) The information displayed by `-Xverbose:memory` or `-Xverbose:gc` will vary depending upon the type of garbage collector you are using.
`memory;` `gc` with `gencon`	A report for a JVM running a generational concurrent collector (`-Xgc:gencon`) with `memory` or `gc` specified might look like this:

```
[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
```

Table 2-1 -Xverbose Parameters

This Parameter…	Prints to the screen…

`memory;` `gc` with `singlecon`	A report for a JVM running a single generation concurrent collector (`-Xgc:singlecon`) with `memory` or `gc` specified might look like this:
	`[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 2-1 -Xverbose Parameters

This Parameter...	Prints to the screen...

`memory;`
`gc`
with
`parallel`
	A report for a JVM running a parallel collector (`-Xgc:parallel`) with `memory` or `gc` specified might look like this:
	`[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`

`opt`	Information about all methods that get optimized. Verbose output for `opt` might look like this:
	`[opt ] 280 2434   0 ObjAlloc.main([Ljava.lang.String;)V: 0.00 ms`
	`[opt ] 0 : 9.8996 ms`

Including a Timestamp with Logging Information

`-Xverbosetimestamp`

You can force a timestamp to print out with other information generated by `-Xverbose` by using the command `-Xverbosetimestamp`. When you use this command, the time and date will precede the verbose information, as shown here:

```
[Wed Jan 14 16:51:57 2004][14578][load   ]created: java/lang/Integer : 1.4034ms
```

Protecting Systems by Using the Security Manager

BEA JRockit allows its own Management Console, JRA, and other tools, to connect to running processes through its own proprietary interface. You can open a port on the VM that can *only* connect to the tools that run locally on your system and not allow a connection from other tools running anywhere else on the network. This allows authorized users to access the production system and run the tool locally so they can monitor or attach to a running BEA JRockit instance, without exposing a security hole for non-authorized users to invade production systems.

To do this, you need to run the JVM with a security manager and grant access to the management server from the host(s) to which you choose to grant privileges. Use this procedure:

1. Run BEA JRockit with `-Djava.security.manager`.
2. In the policy file, grant the appropriate permissions, as shown in the following code sample:

```
grant {
   permission java.net.SocketPermission "localhost:7090", "*";
}
```

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

When BEA 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 BEA JRockit, the JVM does not watch for or process `CTRL_LOGOFF_EVENT` or `SIGHUP` events.

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

- Pressing Ctrl-Break to create a thread dump does not work.
- User code is responsible for causing shutdown hooks to run; for example by calling `System.exit()` when BEA JRockit is to be terminated.

Special Instructions for Linux Users

If you are running BEA JRockit on either a 32- or 64-bit Linux machine, two additional configuration options allow you to enable core dumps on Red Hat AS and to select one of two thread libraries for any Linux version. This information is described below.

Enabling Core Dumps on Red Hat AS

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 BEA 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.

Overriding NPTL

The Native POSIX Thread Library (NPTL) is a thread library option available for use instead of LinuxThreads with Red Hat Enterprise Linux 3.0. If you want to disable NPTL and use LinuxThreads, set `LD_ASSUME_KERNEL=2.4.1` in your environment.

Using the BEA JRockit Memory Management System

Memory management relies on effective "garbage collection," the process of clearing dead objects from the heap, thus releasing that space for new objects. Effective memory management ensures efficient processing. BEA JRockit's unified garbage collector allows you to select a dynamic garbage collector based solely upon one of two priorities: memory throughput or duration of pause time caused by collection. The unified garbage collector is *dynamic* in that, as it runs, it uses predefined heuristics to determine which collection algorithm to use and changing that algorithm as the individual case might warrant. You do not need to specify the actual algorithm to run this garbage collector.

In some instances, dynamic garbage collection might not be the most effective way to free-up memory. In those cases, BEA JRockit also provides a number of "static" collectors that can be started either by specifying the actual collector (`-Xgc:`*`<collectorName>`*) or by default, as determined by the JVM mode you select at startup.

This section describes how to use all of these garbage collection methods. It contains information on the following subjects:

- The Mark-and-Sweep Collection Model
- Garbage Collector Permutations
- Running the Dynamic Garbage Collector
- Using Static Garbage Collection Methods
- Overriding Garbage Collectors
- Viewing Garbage Collection Activity

- Thread-local Allocation

The Mark-and-Sweep Collection Model

The unified garbage collector is a *mark-and-sweep* collector that runs either as generational or single-spaced; that is, with or without a "nursery" (see Generational, below). Both mark-and-sweep options can implement either a concurrent or parallel algorithm. Please refer to Garbage Collector Permutations for more information on garbage collection options and algorithms.

A mark-and-sweep collector is a three-pass model that frees all unreferenced objects. It works as described in these steps:

1. The first pass clears all objects in the system that have their mark bit set. The mark bit identifies whether the block is in use if the block contains garbage.
2. The second pass, or "mark" phase, traverses all pointers, starting at the accessible roots of a program (conventionally, globals, the stack, and registers) and marks each object traversed.
3. The third pass, or "sweep" phase, re-walks the heap linearly and removes all objects that are not marked.

Garbage Collector Permutations

The unified garbage collector is comprised of various permutations—or "states"—of two garbage collector options:

- Generational
- Single-spaced

and two garbage collection algorithms:

- Concurrent
- Parallel

Generational

In generational garbage collection, the heap is divided into two sections: an old generation and a young generation—also called the "nursery." Objects are allocated in the nursery and when it is full, the JVM "stops-the-world" (that is, stops all threads) and moves the live objects in the young generation into the old generation. At the same time, an old collector thread runs in the background, marking live objects in the old generation and removing the dead objects, returning free space to JVM.

Single-spaced

The single-spaced option of garbage collection means that all objects live out their lives in a single space on the heap, regardless of their age. In other words, a single-spaced garbage collector does not have a nursery.

Concurrent

The concurrent garbage collection algorithm does its marking and/or sweeping “concurrently” with all other processing; that is, it does not stop Java threads to do the complete garbage collection.

Parallel

The parallel garbage collection algorithm stops Java threads when the heap is full and uses every CPU to perform a complete mark and/or sweep 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.

Running the Dynamic Garbage Collector

The unified garbage collector combines the options and algorithms described above within the mark-and-sweep model to perform collection. Depending upon the heuristics used, the collector will employ a generational or single-spaced collector with either a concurrent or parallel mark phase and a concurrent or parallel sweep phase.

The main benefit of a dynamic—or “optimizing”—garbage collector is that the only determination you need to make to run the collector best-suited to your needs is whether your application responds best to optimal memory throughput during collection or minimized pause times at that time. You do not need to understand the garbage collection algorithms themselves, or the various permutations thereof, just the behavior of your application.

To start the unified garbage collector, use the `-Xgcprio` command line option with either the `throughput` or `pausetime` parameter, depending upon which priority you want to use:

```
-Xgcprio:<throughput|pausetime>
```

Table 3-1 describes the priorities under which you can start a dynamic garbage collector and the parameters used to select that priority.

Table 3-1 -Xgcprio Option Priorities

Priority	Description	Parameter
Memory Throughput	Memory throughput is usually the most important priority for garbage collection and the one you will probably select most often. It 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.	`throughput`
Pause Time	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 longest pause time and the average pause time during an application run are extremely useful values for tuning the JVM. You will most commonly select pause time as your top priority when you know that your system is sensitive to lengthy pauses.	`pausetime`

Upon selecting the priority and starting the JVM, the dynamic garbage collector will then try to choose the garbage collection state that optimizes performance based upon the priority. It will seek modes that optimize throughput when `-Xgcprio:throughput` is set or that minimize the pause times (as much as possible) when `-Xgcprio:pausetime` is set.

Using Static Garbage Collection Methods

In some circumstances, you might not want to use a dynamic garbage collector. In those cases, you can use a static collector started by default or by selecting one of the backwardly-compatible garbage collectors provided in earlier versions of BEA JRockit. Unlike the dynamic garbage collector started with `-Xgcprio`, static collectors do not change from the algorithm originally selected to collect garbage. They will not attempt to optimize performance by changing algorithms.

This section contains information on the following subjects:

- Using Backward-compatible Garbage Collectors
- Setting the Default Garbage Collector

Using Backward-compatible Garbage Collectors

Three static garbage collectors originally available in earlier versions of BEA JRockit SDK are still available for your use in this version. In some circumstances, the performance of these collectors might meet your needs better than the unified collector or the default collectors available with the `-server` or `-client` flags. Additionally, if you want to use scripts written for the earlier versions of BEA JRockit that implement these collectors, those scripts will continue to work without requiring any modification—unless they use the generational copy garbage collector, which is no longer available (of course, your scripts can be modified to implement the unified garbage collector).

The available garbage collectors (and the command to start them) are:

- Single-spaced Concurrent (`-Xgc:singlecon`; this is the default garbage collector when BEA JRockit is run in the `-client` mode)
- Generational Concurrent (`-Xgc:gencon`)
- Parallel (`-Xgc:parallel`; this is the default garbage collector in the `-server` mode)

Pros and Cons

Table 3-2 lists the pros and cons of each garbage collector.

Table 3-2 Garbage Collector Pros and Cons

Garbage Collector	Pros	Cons
Single Spaced Concurrent	• Virtually removes all pauses. • Can handle gigabyte-sized heaps. • Default garbage collector when running with the -client option.	• 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.

Table 3-2 Garbage Collector Pros and Cons

Garbage Collector	Pros	Cons
Generational Concurrent	• 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.	• 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	• Uses all processors during collection, thus maximizing memory throughput. • Default garbage collector when running with the -server option (default behavior).	• "Stop the world" might cause a longer than desirable pause in processing.

Garbage Collector Selection Matrix

Table 3-3 is a matrix that you can use to determine which garbage collector is right for your BEA 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 Garbage Collector...** column. The third column lists the supported garbage collector that might suit your needs as well as—if not better than—an unsupported collector.

Table 3-3 Garbage Collector Selection Matrix

If You...	Select this Garbage Collector...	Or use...
• 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	`-Xgcprio:pausetime`

Table 3-3 Garbage Collector Selection Matrix

If You...	Select this Garbage Collector...	Or use...
• Cannot tolerate pauses of any length. • Employ gigabyte-sized heaps. • Willing to trade *some* memory thoughput for eliminating pauses. • Want better memory throughput than possible with Single Spaced Concurrent. • Are not sure that the other two methods would be adequate given how you've implemented BEA JRockit JVM.	Generational Concurrent	`-Xgcprio:pausetime`
• Using a machine with four CPUs or better or a single CPU machine with a lot of memory. • Can tolerate the occasional long pause • Need to maximize memory throughput	Parallel	`-Xgcprio:throughput`

Setting the Default Garbage Collector

If you don't set `-Xgcprio` or `-Xgc` at startup, BEA JRockit defaults to a preselected, static garbage collector based upon the JVM mode you select: server-side (`-server`; the default) or client-side (`-client`). These **are not** garbage collectors themselves, but JVM configuration options that start a static collector and set default initial and maximum heap sizes. Table 3-4 describes how these startup options set a garbage collector.

Table 3-4 Garbage Collectors Started by JVM Modes

JVM Mode	Description
`-server`	By definition, BEA JRockit is a server-side JVM, thus the `-server` option replicates BEA JRockit's default behavior. When you select `-server`, BEA JRockit will run Parallel garbage collector (equivalent to setting `-Xgc:parallel`). For information on how `-server` sets heap and nursery size, please refer to "Setting the Maximum Heap Size" and "Setting the Minimum Heap Size" in *Tuning BEA JRockit*.
`-client`	When you start BEA JRockit in the `-client` mode, you are asking it to behave as a client-side JVM. This mode is appropriate when you have a smaller heap and are anticipating shorter runtimes for your application. When you select `-client` at JVM startup, BEA JRockit will run a Single-spaced Concurrent garbage collector (equivalent to setting `-Xgc:singlecon`). For information on how `-client` sets heap and nursery size, please refer to "Setting the Maximum Heap Size" and "Setting the Minimum Heap Size" in *Tuning BEA JRockit*.

For more information on using the `-server` and `-client` options, please refer to Starting and Configuring BEA JRockit JVM.

Overriding Garbage Collectors

Setting `-Xgcprio` will override any settings associated with `-server` and `-client`. Setting `-Xgc` will override `-Xgcprio` and `-server` and `-client`.

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 BEA 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 -Xgcreport -Xgcpause myClass
```

Thread-local Allocation

Thread-local allocation removes object allocation contention and reduces the need to synchronize between threads allocating memory on the heap. It also increases cache performance on a multi-CPU system, because it reduces the risk of two threads running on different CPUs needing 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.

Using the BEA JRockit Management Console

The JRockit Management Console can be used to monitor and control running instances of BEA 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
- Creating a JRA Recording
- Closing the Console
- Starting and Running the Console in the Headless Mode

Console Overhead

The extra cost of running the JRockit Management Console against a running BEA 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 BEA 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 BEA JRockit JVM, the management server in the VM needs to be started. The server is disabled by default. To enable the management server, start BEA JRockit JVM with the `-Xmanagement` option:

```
-Xmanagement
```

Attaching a Management Client

You can use the class= and `classpath=` parameters with `-Xmanagement` to specify a management class and its classpath; for example:

```
-Xmanagement:class=<classname>,classpath=<path>
```

This option loads the class and causes its empty constructor to be called early in JVM startup. From the constructor, a new thread is then started, from which your management client is run. You should ensure that the constructor returns control quickly because this call is made early in BEA JRockit startup.

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 4-1 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 4-1 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 BEA 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:

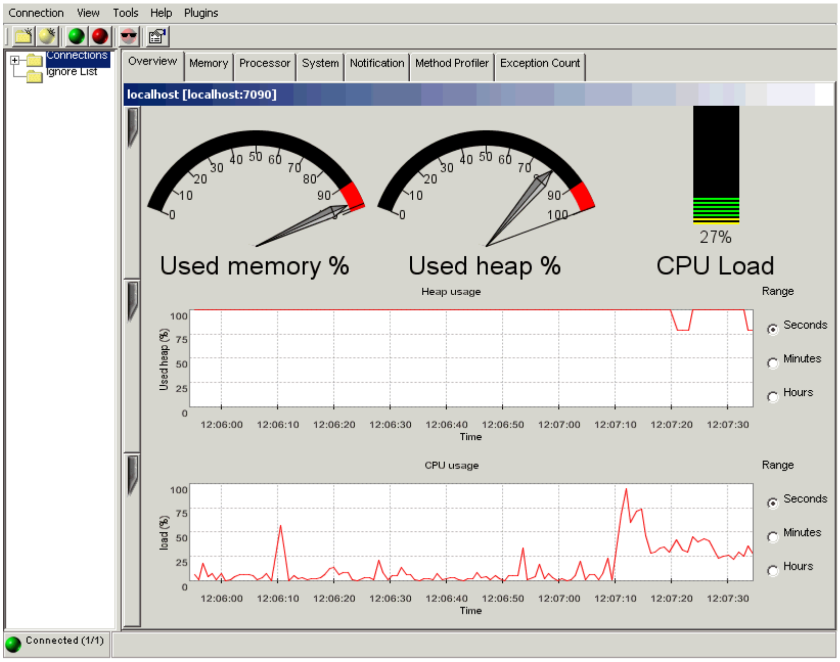
```
-Djrockit.managementserver.maxconnect=<maximum number of connections>
```

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 4-1:

Figure 4-1BEA JRockit JVM Management Console



The JRockit Management Console window is divided into two panes: a connection browser tree in the left pane (Figure 4-2) and a tabbed interface in the right pane (Figure 4-3).

Figure 4-2Connection Browser

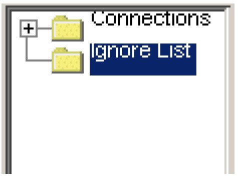
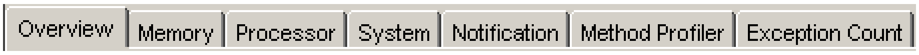


Figure 4-3Information Tabs (Administrator Mode)

The first tab shows an Overview of information for the selected BEA 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.

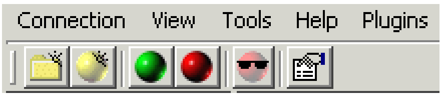
Figure 4-3 shows the information tabs available in the console's Administrator operation mode. When the console is in the Developer mode, additional tabs appear, as shown in Figure 4-4. These two operation modes are described in Setting the Operation Mode.

Figure 4-4Information Tabs (Developer Mode)



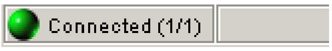
The console includes a toolbar that contains command buttons for some of the menu options (Figure 4-5). To toggle the Toolbar on or off, on the View menu select Tool Bar.

Figure 4-5Management Console Toolbar



The status bar (Figure 4-6) 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 BEA JRockit JVM implementations or not. To toggle the Status Bar on or off, on the View menu, select Status Bar.

Figure 4-6Status 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 BEA 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.

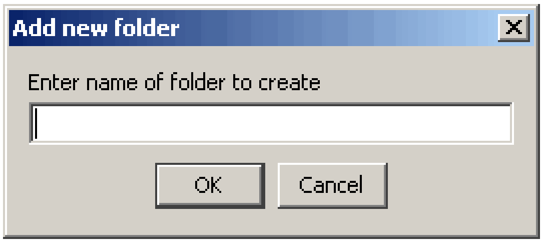
Creating a New Folder

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 4-7) appears:

Figure 4-7Add 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.

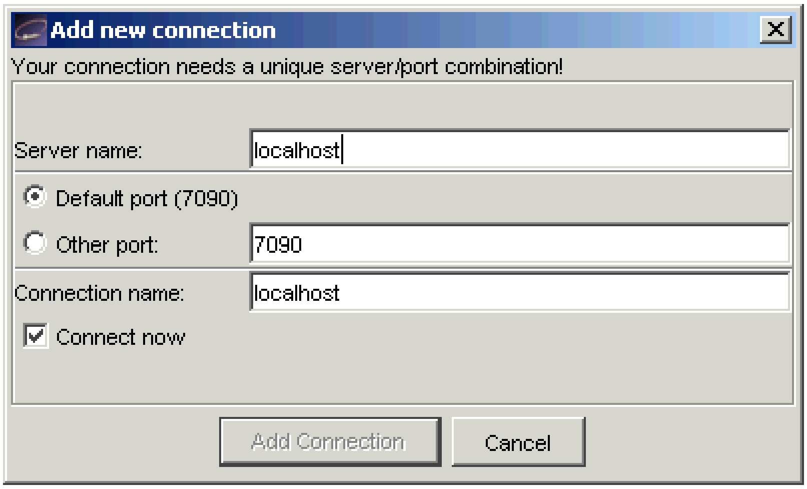
Creating a New Connection

To create a new connection to BEA 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 4-8) appears:

Figure 4-8Add 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 BEA JRockit JVM

To connect to BEA JRockit, do the following:

1. Select the BEA 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 BEA JRockit JVM

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

1. Select the BEA 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.

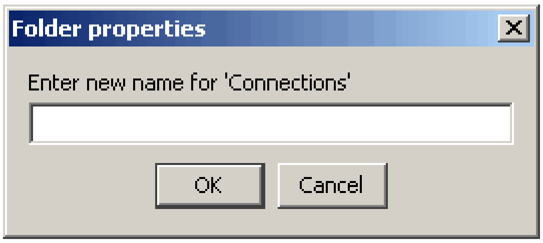
Renaming a Connection or Folder

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

1. Select the BEA 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 Properties.
   - 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 4-9) appears:

Figure 4-9Folder 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 BEA 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:

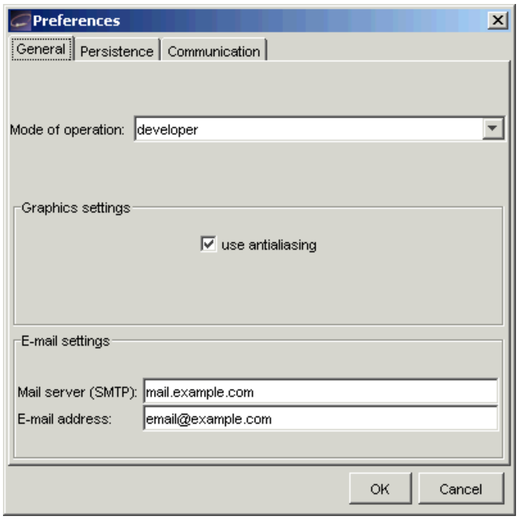
- **Administrator Mode**; This is the default mode, designed for system administrators who are interested in observing the state of the BEA JRockit JVM.
- **Developer Mode**; The developer mode is for developers and provides additional features such as a rudimentary method profiler and exception count 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 4-10) appears:

Figure 4-10Preferences Menu (General Tab)



Mode of operation: developer

Graphics settings

☑ use antialiasing

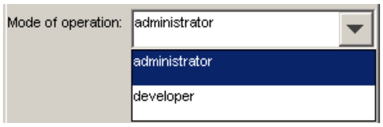
E-mail settings

Mail server (SMTP): mail.example.com

E-mail address: email@example.com

2. Click the Mode of operation drop-down control to display the list of operation modes (Figure 4-11).

Figure 4-11List of Operation Modes



Mode of operation: administrator

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 4-3 and Figure 4-4 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.

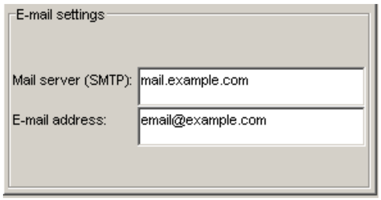
To change either of these values, open the Preferences dialog box from the Tools menu and proceed are described in the following sections:

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 4-12.
3. Click OK

Figure 4-12E-mail Preferences Panel



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).

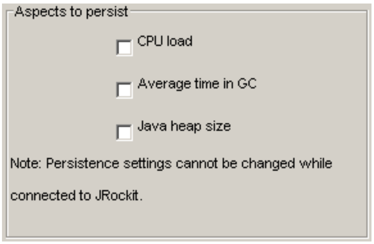
Selecting Aspects to Persist To set persistence preferences, do the following:

1. Disconnect any BEA 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 4-13):

Figure 4-13Aspects to Persist Panel


Aspects to persist

- [ ] CPU load
- [ ] Average time in GC
- [ ] Java heap size

Note: Persistence settings cannot be changed while connected to JRockit.

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 4-28.

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 BEA 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`.

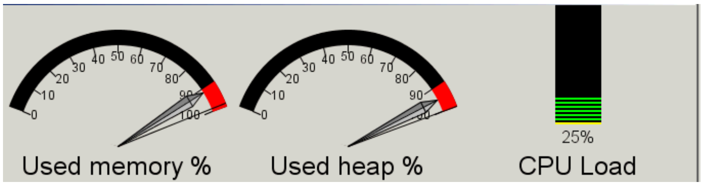
Customizing the Display

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

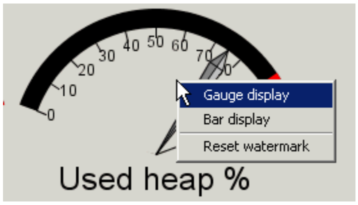
The gauges and bars are graphical devices showing memory and processor usage (Figure 4-14).

Figure 4-14Gauges and Bars



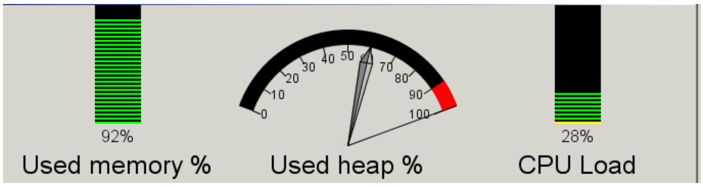
- To change from a gauge display to bar display, press the right mouse button when pointing at the gauge and select Bar display, as shown in Figure 4-15.

Figure 4-15Gauge Context Menu (Bar Display Selected)



The selected gauge will appear as a bar (Figure 4-16).

Figure 4-16Gauges and Bars with Gauge Converted to a Bar Display



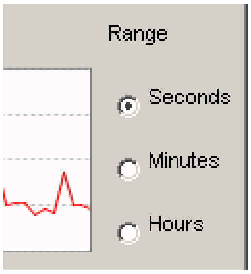
- To change back to a gauge, repeat the above, but select Gauge display.
- To reset the watermark—which indicates the highest level measured so far—press the right mouse button when pointing at the gauge or bar and select Reset Watermark.

Customizing Charts

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

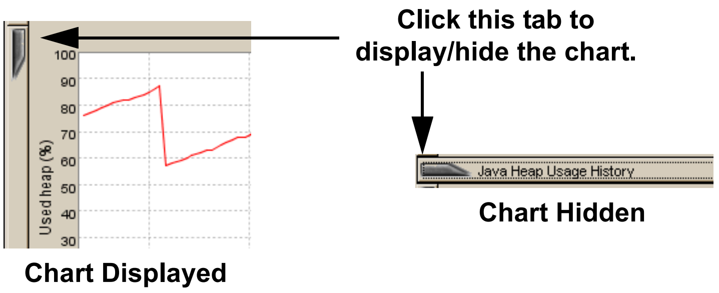
- To change scale on any of the chart, select the desired scale unit (seconds, minutes or hours) to the right of the chart (Figure 4-17) to be changed.

Figure 4-17Range 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 4-18).

Figure 4-18Hiding 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 BEA JRockit JVM. An aspect is data that can be measured; for example, used heap size or VM uptime.

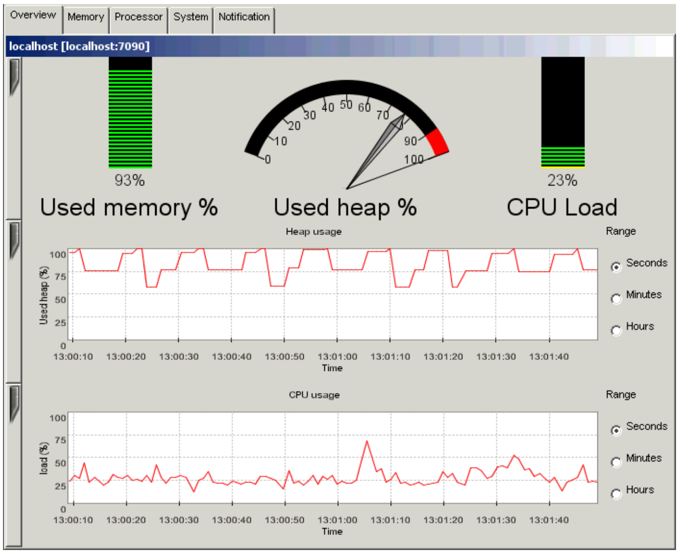
Information Tabs

Information tabs are pages containing details about different areas of the monitored BEA 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 4-19) 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 4-19Overview Tab

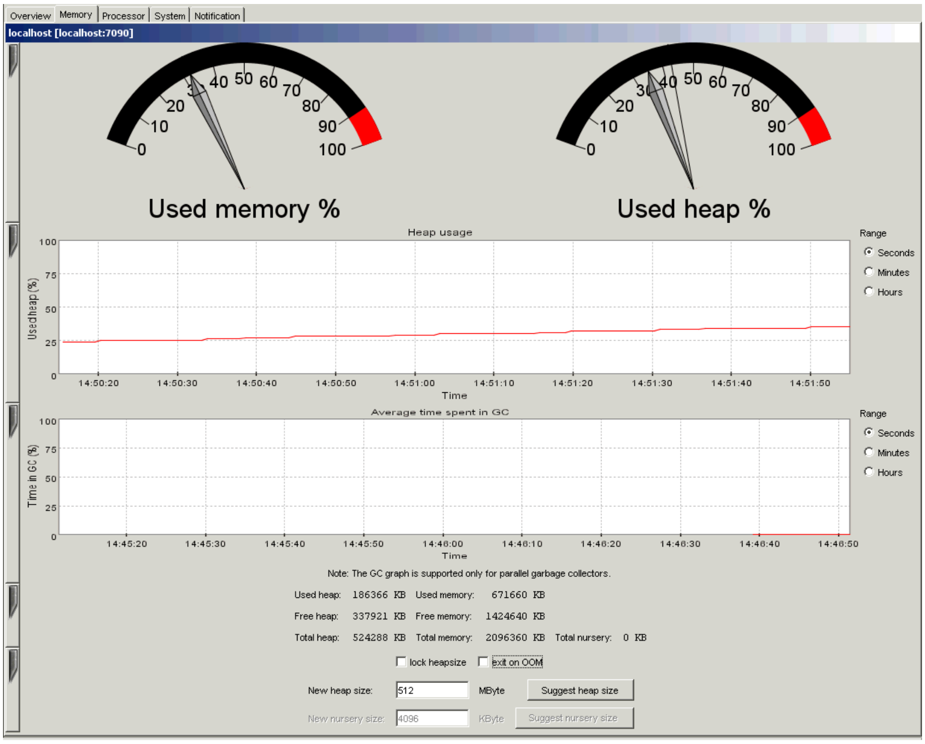


- The **Used Memory** gauge shows the percentage of occupied physical memory on the computer.
- The **Used Heap** gauge shows the percentage of occupied Java heap memory in the VM.
- The **CPU Load** bar shows the usage rate of the processor - or the average processor load on a multi-processor machine.
- 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 4-20) shows information about the memory status of the system, as shown.

Figure 4-20Memory Tab



Note: The GC graph is supported only for parallel garbage collectors.

Used heap: 186366 KB Used memory: 671660 KB

Free heap: 337921 KB Free memory: 1424640 KB

Total heap: 524288 KB Total memory: 2096360 KB Total nursery: 0 KB

☐ lock heapsize ☐ exit on OOM

New heap size: 512 MByte [Suggest heap size]

New nursery size: 4096 KByte [Suggest nursery size]

- 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 BEA 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.

Memory Tab Functionality

You can manipulate certain memory aspects of the JVM from the Memory Tab. These aspects are described in Table 4-1

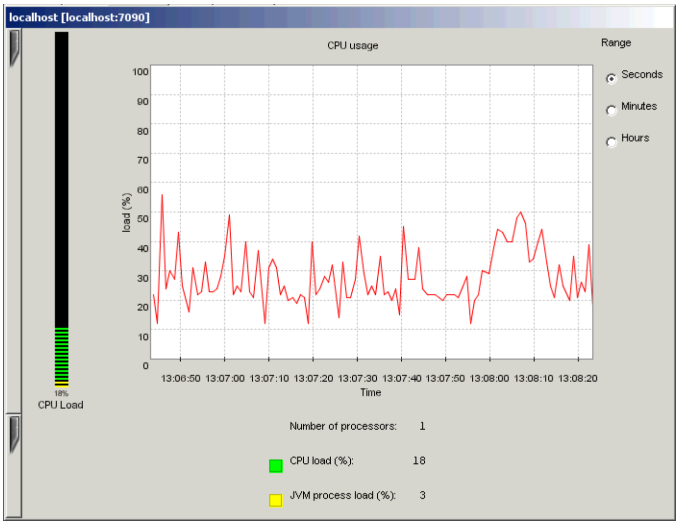
Table 4-1 Memory Tab Functionality

Function	Procedure
Manipulating the Heap Size	You can manipulate the heap size in any of the following ways: • To reset the size of your heap, enter a numeric value in the **New heap size** field and click **Suggest heap size**. The size set here, expressed in megabytes, represents the current heap size, not the maximum heap size. • To make the current size the maximum heap size, click **lock heap size**. Please refer to Setting the Heap Size for heap size requirements.
Changing the Nursery Size	If you are running a generational garbage collector, you can reset the nursery size by typing a new value in **New nursery size** and click **Suggest nursery size**. **Note:** If you are not running a generational collector, these fields will appear disabled. Please refer to Setting the Size of the Nursery for nursery size requirements.
Exiting on Out of Memory Errors	If you want to exit the JVM when you encounter an Out of memory (OOM) error, select **Exit on OOM**.

Processor Tab

The Processor tab (Figure 4-21) shows information about the processor status of the system.

Figure 4-21 Processor Tab



Range: [✓ Seconds] Minutes Hours

Number of processors: 1

🟩 CPU load (%): 18

🟨 JVM process load (%): 3

- 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.

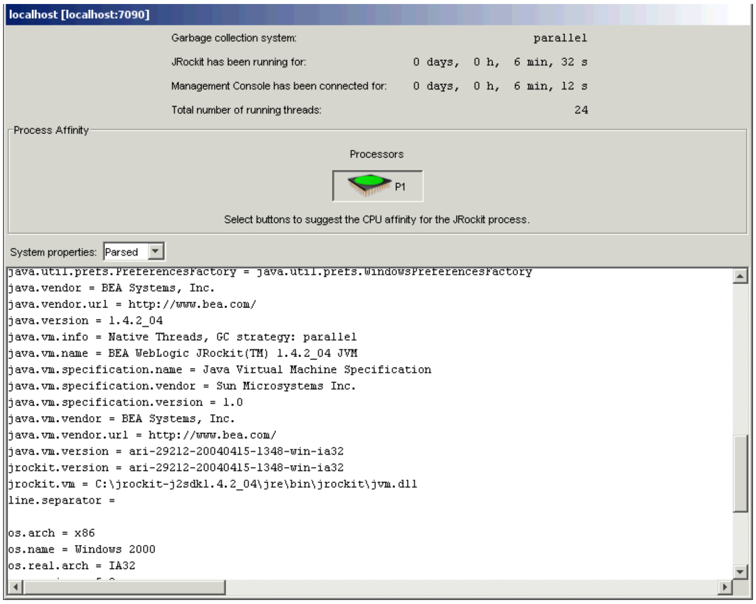
At the bottom of the page the following text information is displayed:

- **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 BEA JRockit JVM process(es), expressed as a percentage.

System Tab

The System tab (Figure 4-22) shows various information about the system status.

Figure 4-22System Tab



localhost [localhost:7090]

```
Garbage collection system:                                         parallel
JRockit has been running for:                      0 days,  0 h,  6 min, 32 s
Management Console has been connected for:         0 days,  0 h,  6 min, 12 s
Total number of running threads:                                         24
```

Process Affinity

Processors

P1

Select buttons to suggest the CPU affinity for the JRockit process.

System properties: Parsed

```
java.util.prefs.PreferencesFactory = java.util.prefs.WindowsPreferencesFactory
java.vendor = BEA Systems, Inc.
java.vendor.url = http://www.bea.com/
java.version = 1.4.2_04
java.vm.info = Native Threads, GC strategy: parallel
java.vm.name = BEA WebLogic JRockit(TM) 1.4.2_04 JVM
java.vm.specification.name = Java Virtual Machine Specification
java.vm.specification.vendor = Sun Microsystems Inc.
java.vm.specification.version = 1.0
java.vm.vendor = BEA Systems, Inc.
java.vm.vendor.url = http://www.bea.com/
java.vm.version = ari-29212-20040415-1348-win-ia32
jrockit.version = ari-29212-20040415-1348-win-ia32
jrockit.vm = C:\jrockit-j2sdk1.4.2_04\jre\bin\jrockit\jvm.dll
line.separator =

os.arch = x86
os.name = Windows 2000
os.real.arch = IA32
```

- **Garbage Collection System** shows which garbage collector BEA JRockit JVM is running. If you are using a dynamic garbage collector (`-Xgcprio`), this value will change when the garbage collector changes. For more information on the dynamic garbage collector, please refer to "Running the Dynamic Garbage Collector."
- **JRockit has been running for** shows how long BEA JRockit JVM has been running.
- **Management Console has been connected for** shows how long the currently displayed connection has been connected.
- **Total number of running threads** shows the number of active threads at any given time in the application run.
- **Process Affinity** contains buttons that correspond to processors. It displays a green icon if BEA JRockit JVM is running on this processor and a red icon if it is not. By selecting a button, the BEA 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.

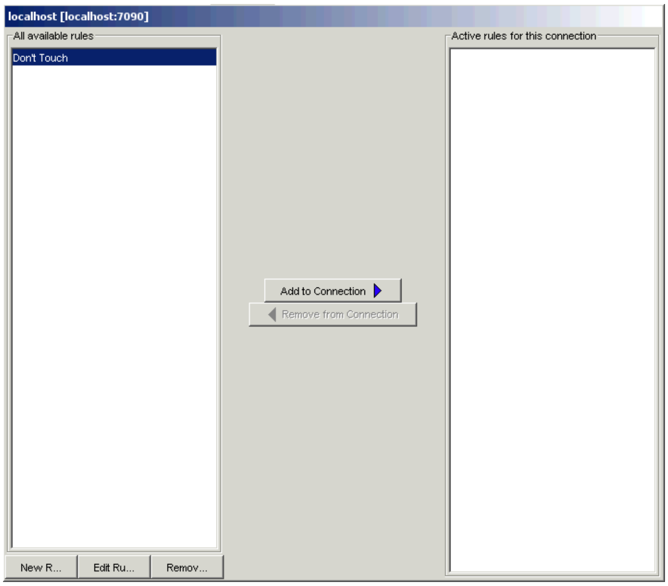
Notification Tab

Use the Notification tab (Figure 4-23) 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 programatically 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 4-23Notification Tab (No Rules Defined)



A notification trigger can be a certain event, for example, that the connection to BEA 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.

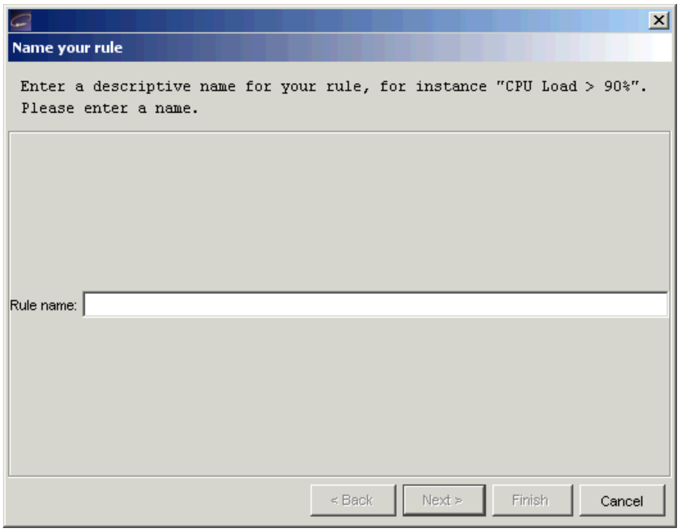
Creating a New Rule

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

1. Click New Rule.

   The Name your rule dialog box appears (Figure 4-24):

Figure 4-24Name Your Rule Dialog Box



Name your rule

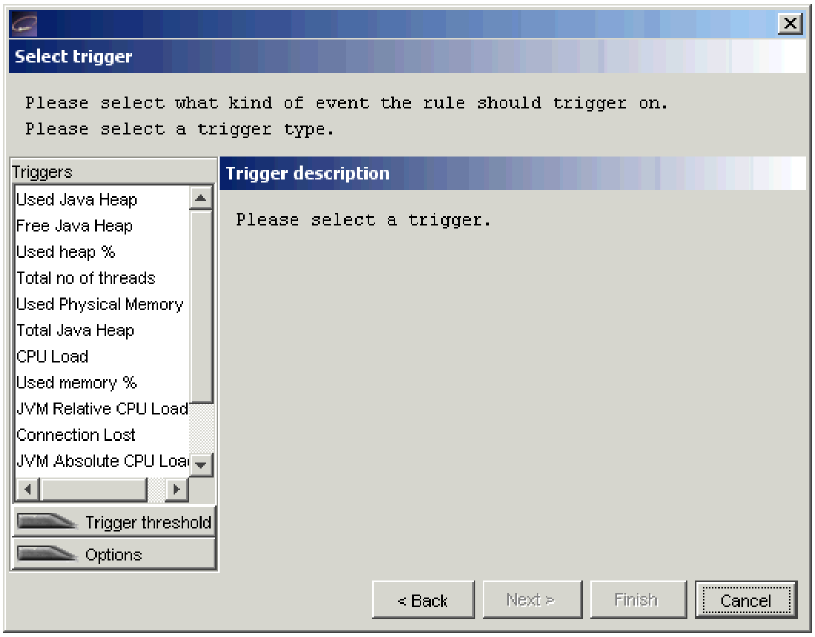
Enter a descriptive name for your rule, for instance "CPU Load > 90%".
Please enter a name.

Rule name:

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

The Select trigger dialog box appears (Figure 4-25):

Figure 4-25Select Trigger Dialog Box

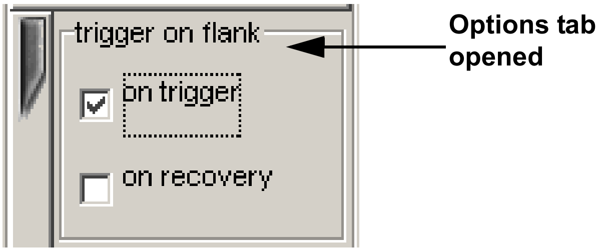


Please select what kind of event the rule should trigger on.
Please select a trigger type.

Please select a trigger.

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 4-26; this box will be marked either Min value or Max value, depending on the type of trigger selected.

Figure 4-26Trigger Threshold and Options Text Boxes



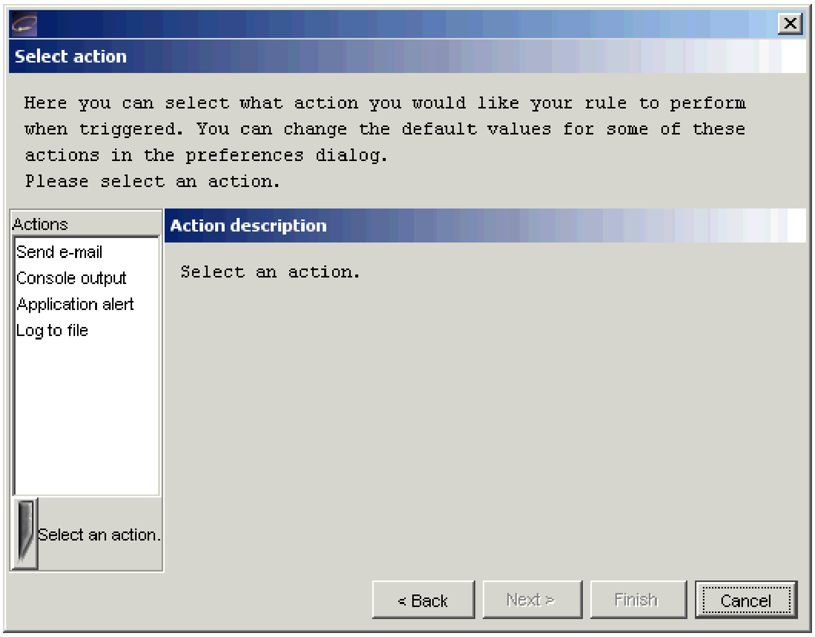
5. Select further options under the Option tab. For example, in Figure 4-26, you need to select what kind of aspect value change will trigger the notification:

- 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 4-27):

Figure 4-27Select Action Dialog Box



Select action

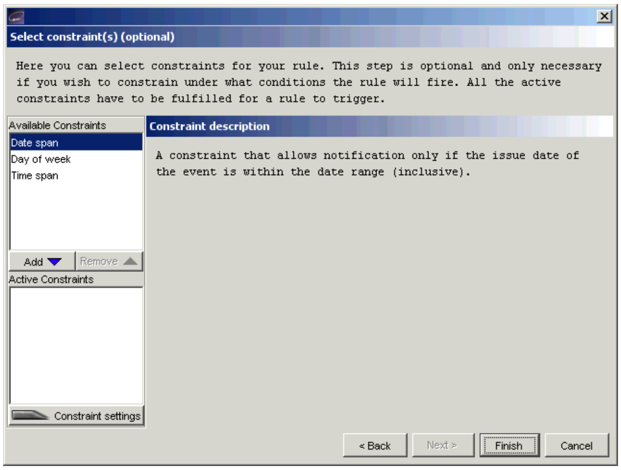
Here you can select what action you would like your rule to perform when triggered. You can change the default values for some of these actions in the preferences dialog.
Please select an action.

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):

   a. Click Next.

      The Select Constraint dialog box appears (Figure 4-28):

Figure 4-28Select Constraint Dialog Box



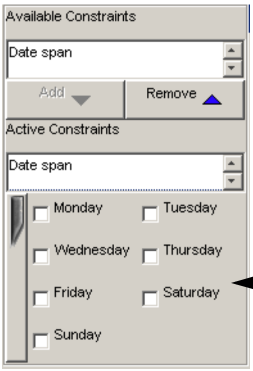
```
Here you can select constraints for your rule. This step is optional and only necessary
if you wish to constrain under what conditions the rule will fire. All the active
constraints have to be fulfilled for a rule to trigger.
```

```
A constraint that allows notification only if the issue date of
the event is within the date range (inclusive).
```

b. Select a constraint and click Add.

The constraint name will appear in the add list, as shown in Figure 4-29.

Figure 4-29Constraint Added

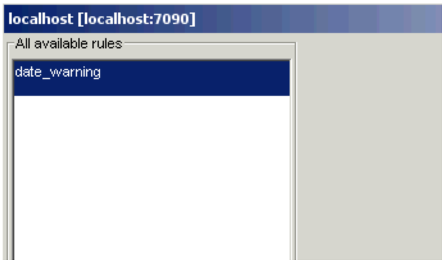


c. Enter constraint settings in the text fields under the list of constraints (Figure 4-29).

9. Click Finish.

The new rule appears in the All available rules list on the Notification tab, as shown in Figure 4-30.

Figure 4-30New Rule in List



10. Add the rule to your connection as described in Add a Rule to BEA 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.

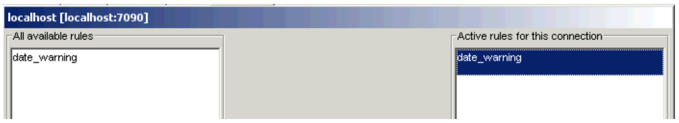
Add a Rule to BEA JRockit JVM

To add a rule to BEA 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 4-31.

Figure 4-31Rule Added to Active rules for This Connection List



Remove a Rule from BEA JRockit JVM

To remove a rule from BEA 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.

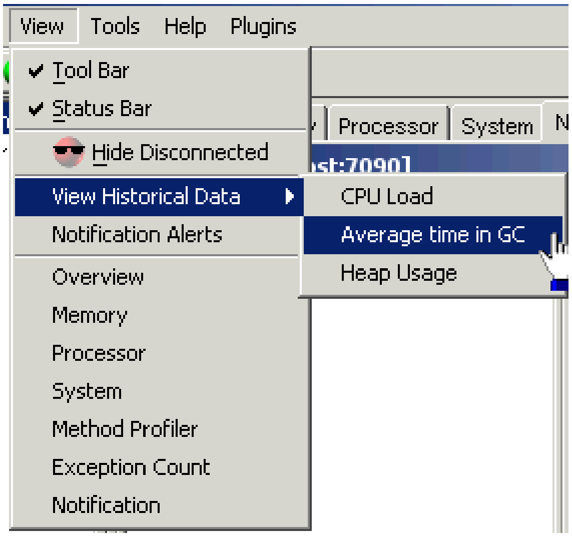
View Historical Data

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 BEA JRockit JVM has its peak loads.

To open this window, do the following:

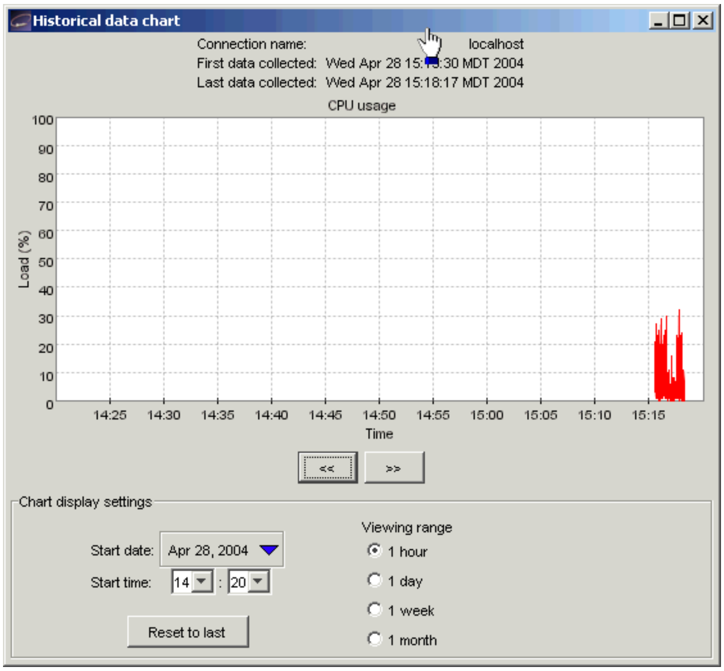
1. Select the connection for which you want to view data.
2. Open the View menu and select View Historical Data.
3. Select the aspect for which you want to view historical data, as shown in Figure 4-32.

Figure 4-32View Menu with Historical Data Submenu Open



Historical data for the selected aspect appears (Figure 4-33).

Figure 4-33Historical Data (CPU Load Selected)



Connection name: localhost
First data collected: Wed Apr 28 15:[illegible]3:30 MDT 2004
Last data collected: Wed Apr 28 15:18:17 MDT 2004

Chart display settings

Start date: Apr 28, 2004
Start time: 14 : 20

Viewing range
[✓ 1 hour]
1 day
1 week
1 month

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 BEA 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

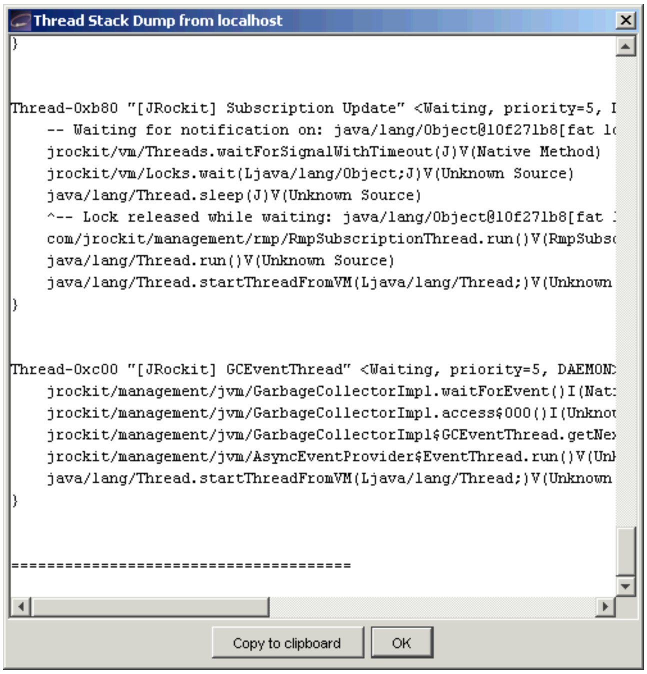
This section describes the more advanced features of the Management Console. Some of these are only available when running in the Developer mode, described in Setting the Operation Mode.

View Thread Stack Dump

The stack dump contains a list of all running threads in BEA 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 4-34).

Figure 4-34Thread Stack Dump



```
}


Thread-0xb80 "[JRockit] Subscription Update" <Waiting, priority=5, I
    -- Waiting for notification on: java/lang/Object@10f271b8[fat lo
    jrockit/vm/Threads.waitForSignalWithTimeout(J)V(Native Method)
    jrockit/vm/Locks.wait(Ljava/lang/Object;J)V(Unknown Source)
    java/lang/Thread.sleep(J)V(Unknown Source)
    ^-- Lock released while waiting: java/lang/Object@10f271b8[fat l
    com/jrockit/management/rmp/RmpSubscriptionThread.run()V(RmpSubsc
    java/lang/Thread.run()V(Unknown Source)
    java/lang/Thread.startThreadFromVM(Ljava/lang/Thread;)V(Unknown
}


Thread-0xc00 "[JRockit] GCEventThread" <Waiting, priority=5, DAEMON>
    jrockit/management/jvm/GarbageCollectorImpl.waitForEvent()I(Nat
    jrockit/management/jvm/GarbageCollectorImpl.access$000()I(Unknow
    jrockit/management/jvm/GarbageCollectorImpl$GCEventThread.getNex
    jrockit/management/jvm/AsyncEventProvider$EventThread.run()V(Unk
    java/lang/Thread.startThreadFromVM(Ljava/lang/Thread;)V(Unknown
}


====================================
```

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.

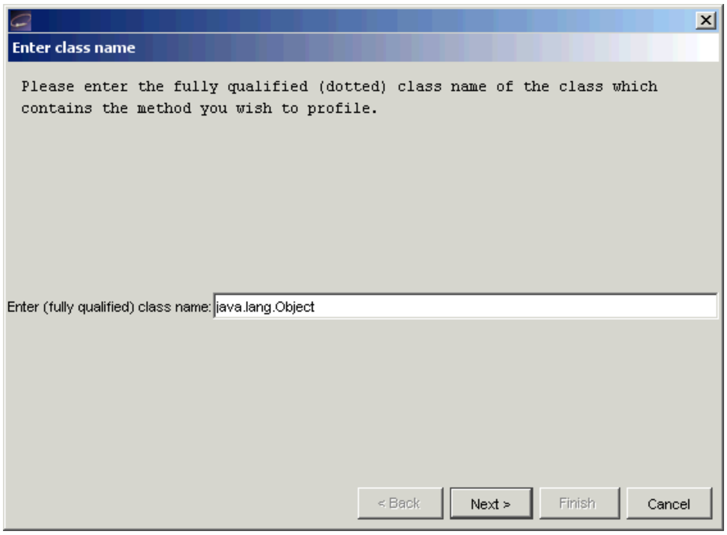
Adding a Method to a Template

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 4-35).

Figure 4-35Enter Class Name Dialog Box



Enter class name

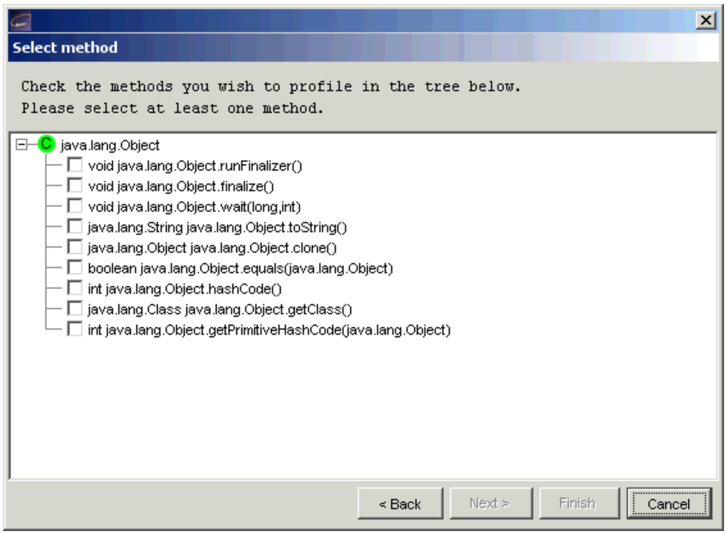
Please enter the fully qualified (dotted) class name of the class which contains the method you wish to profile.

Enter (fully qualified) class name: java.lang.Object

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 4-36):

Figure 4-36Select Method Dialog Box


Select method

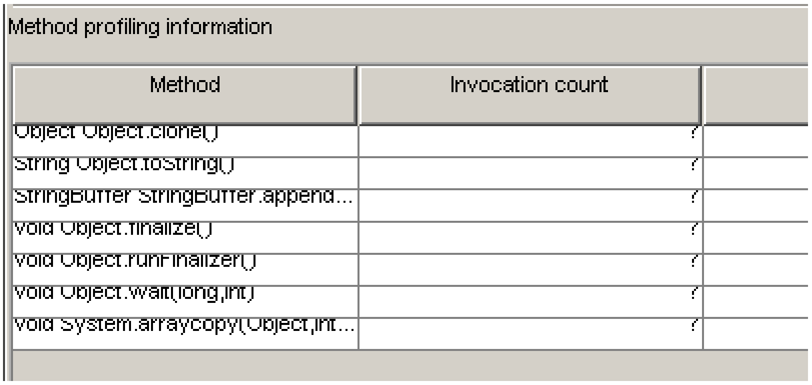
Check the methods you wish to profile in the tree below.
Please select at least one method.

- java.lang.Object
  - [ ] void java.lang.Object.runFinalizer()
  - [ ] void java.lang.Object.finalize()
  - [ ] void java.lang.Object.wait(long,int)
  - [ ] java.lang.String java.lang.Object.toString()
  - [ ] java.lang.Object java.lang.Object.clone()
  - [ ] boolean java.lang.Object.equals(java.lang.Object)
  - [ ] int java.lang.Object.hashCode()
  - [ ] java.lang.Class java.lang.Object.getClass()
  - [ ] int java.lang.Object.getPrimitiveHashCode(java.lang.Object)

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 4-37.

Figure 4-37Method Profiling Information List with Method Added


Method profiling information

Method	Invocation count	
Object Object.clone()	?	
String Object.toString()	?	
StringBuffer StringBuffer.append...	?	
void Object.finalize()	?	
void Object.runFinalizer()	?	
void Object.wait(long,int)	?	
void System.arraycopy(Object,int...	?	

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.

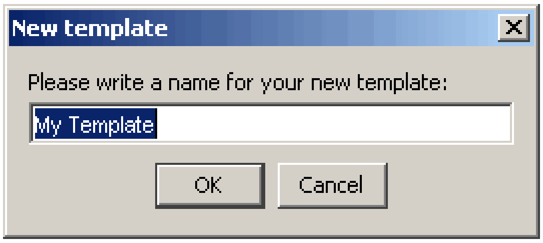
Creating a New Template

To create a new template, do the following:

1. Click New template.

   The New template dialog box appears (Figure 4-38).

Figure 4-38New 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

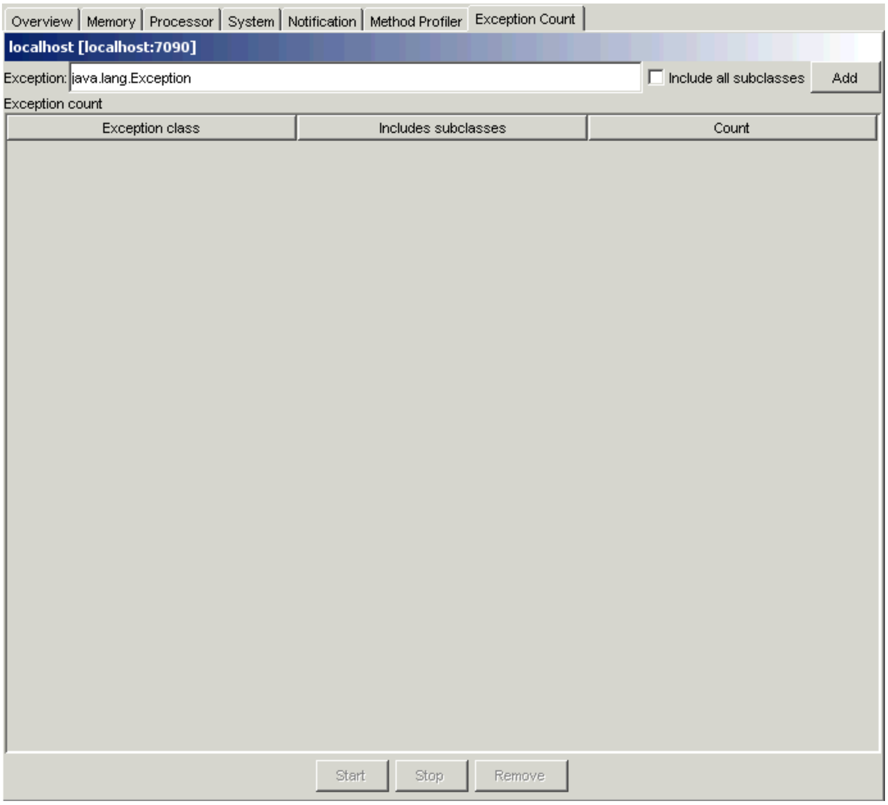
You can switch between using qualified method names or short names in the method profiling table.

- 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 4-39) shows exceptions thrown in BEA JRockit JVM. It counts the number of exceptions of a certain type thrown.

Figure 4-39Exception Counting Tab



Exception: java.lang.Exception

Exception class	Includes subclasses	Count

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 BEA 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.

Creating a JRA Recording

The BEA JRockit Runtime Analyzer (JRA) is an internal tool used by the BEA JRockit development team to analyze runtime performance of BEA JRockit and Java applications running on it. This tool provides information on internals in BEA JRockit that are useful to the development team and BEA JRockit in general.

One part of the JRA runs inside the JVM, recording information about it and the Java application currently running. This tool is launched from the Management Console, as described in the following procedure. The recorded information is saved to a file which you can view in the analyzer tool, as described in "Using the BEA JRockit Runtime Analyzer."

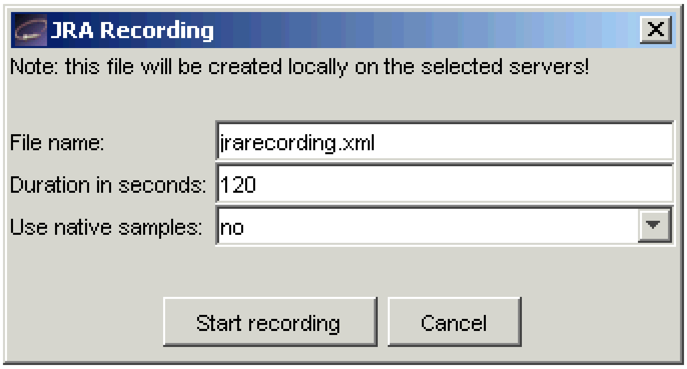
To make a recording, use this procedure:

Note: Before you can make a recording, you need to be working in the Developer mode, as described in "Setting the Operation Mode."

1. Open the Plugins menu and select Make a JRA recording.

   The JRA Recording dialog box appears (Figure 4-40).

Figure 4-40 JRA Recording Dialog Box


JRA Recording

Note: this file will be created locally on the selected servers!

File name: jrarecording.xml

Duration in seconds: 120

Use native samples: no

2. Type a descriptive name for the recording in the Filename field. This will be the name by which the file is saved.

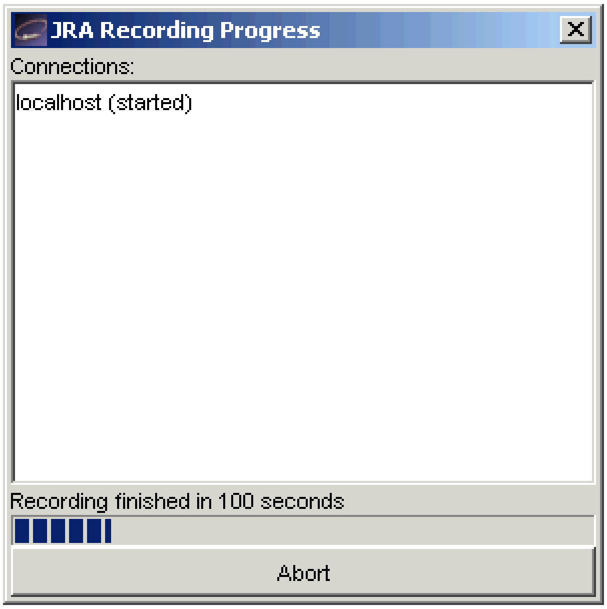
Optionally, you can also select

- The duration of the recording (in seconds)
- Whether or not to use native samples.

3. Click Start recording.

   The JRA Recording Progress box appears (Figure 4-41).

Figure 4-41JRA Recording Progress Box



localhost (started)

Recording finished in 100 seconds

4. When the recording is complete, click Done.

To view the recording, use the analyzer tool, as described in "Using the BEA JRockit Runtime Analyzer."

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.

Starting and Running the Console in the Headless Mode

You can run the Management Console, its notification subsystem, and the user actions without using a GUI. This function is referred to running the console in a "headless" mode and can greatly reduce the amount of system overhead required to run BEA JRockit.

Running a Headless Management Console

To run the Console in the headless mode, start the console as you normally would (see "Start the JRockit Management Console" for details) but add the -headless command-line option; for example:

```
java -jar ManagementConsole.jar -headless
```

You can control the console's behavior by using the command-line options described in Table 4-2.

As it runs, the JVM statistics normally associated with the Management Console can be written to file. The file to which statistics are written will be automatically created, but only if you choose to save, or "persist" data. It will be created in a directory of your choosing.

You can control which JVM statistics are persisted by specifying them in an XML settings file. The settings file is also created automatically, when you exit the application *when it is running in GUI mode*. By default, it will be created in the `<user_home>/.ManagementConsole` directory. You can specify another file at another location by using the `-settings` command-line option.

Controlling the Console with Command-line Options

You can use one of the command-line options listed in Table 4-2to control the behavior of the headless Management Console.

Note: These options are not specific to running the Console in the headless mode; they are also valid when running it with a GUI.

Table 4-2 Headless Management Console Command-line Options

Option	Description
`-headless`	Starts the console in the headless mode (won't load GUI related classes).

Table 4-2 Headless Management Console Command-line Options

`-settings <settings file>`	Starts the console using the specified settings file. If you are starting in the GUI mode and this file doesn't exist, it will be created when you close the application.
`-connectall`	Makes all connections available in the settings file (that is, previously added by using the GUI).
`-connect <connection 1> <connection 2> <...>`	Connects to the named connections available in the settings file previously added by using the GUI.
`-autoconnect`	Automatically connects to any JVM running the management server with JDP turned on.
`-uptime <time in seconds>`	Runs the Console for the specified amount of time, and then automatically shut it down.
`-jrockitmode`	Starts the Console in the JRockit Mode (only makes sense in GUI mode).
`-useraction <name> <delay in seconds> <period (optional)>`	Runs the named user action after the specified delay. If no period has been specified, the action will be run once. If the period has been specified it will be run every `<period (optional)>` seconds.
`-version`	Prints the version of the ManagementConsole and then exits.

For example:

```
java -jar ManagementConsole.jar -headless -settings
   C:\Headless\consolesettings.xml -connectall -autoconnect -uptime 3600
   -useraction ctrlbreak 30 60
```

This example

- Starts the management console in headless mode (`-headless`).
- Reads the specified settings file (`-settings C:\Headless\consolesettings.xml`).
- Tries to connect to all previously specified JVMs (`-connectall`).

- Actively searches for new connections using JDP (`-autoconnect`).
- After running 30 seconds, it will start issuing control breaks to all connected JVMs every minute (`-useraction ctrlbreak 30 60`).
- After an hour it will automatically shut down (`-uptime 3600`).

All notification rules that have been previously added to specific connections will be active.

Using the BEA JRockit Memory Leak Detector

This product is provided "as-is," without any expressed or implied warranties or support by BEA Systems, Inc. This product, which may or may not become an officially supported product from BEA Systems, may contain errors and/or inaccuracies. Use of this product is left solely upon the discretion of the user without any endorsement from BEA Systems. The Memory Leak Detector functionality may or may not be available in future BEA JRockit versions. Questions and problems may be reported via online BEA JRockit newsgroups at `http://newsgroups.bea.com`.

The BEA JRockit Memory Leak Detector is a tool to detect memory leaks within Java applications running on BEA JRockit. A memory leak means application code holding on to memory which is not actually used by the application any more. The BEA JRockit Memory Leak Detector is a real-time profiling tool that gives information about what type of objects are allocated, how many, of what size and how they relate to each other. Unlike other similar tools, there is no need to create full heap dumps to be analyzed at a later stage. The data presented is fetched directly from the running JVM and the JVM can continue to run with a relatively small overhead. When the analysis is done, the tool can be disconnected, and the JVM will run at full speed again. This makes the tool viable for use in a production type environment.

The purpose of this tool is to display memory leaking object types and provide help to track the source of the problem. Another purpose of this tool is to help the developer by increased understanding and knowledge to avoid similar programming errors in future projects.

Note: To access the full version of the BEA JRockit Memory Leak Detector, JRockit JRockit 1.4.2_05 or higher is required.

This section describes the BEA JRockit Memory Leak Detector (from now on referred to as Memory Leak Detector) and how to use it to detect memory leaks. It includes information on the following subjects:

- Starting the Memory Leak Detector
- Using the Memory Leak Detector
- Help Us Improve BEA JRockit
- BEA JRockit Support for the Memory Leak Detector
- Frequently Asked Questions
- Known Issues

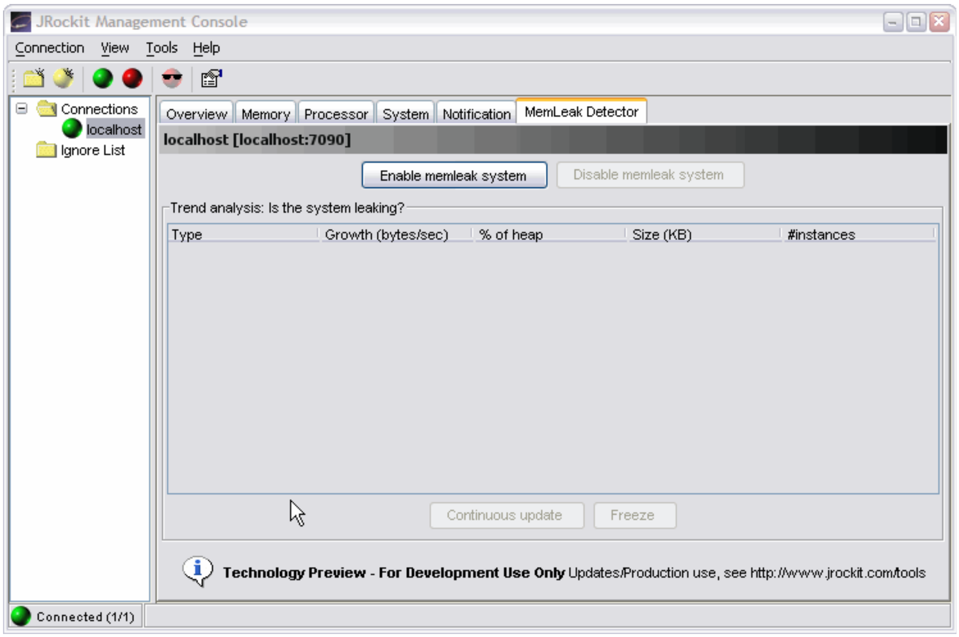
Starting the Memory Leak Detector

To start the Memory Leak Detector you need to start the BEA JRockit Management Console (as from BEA JRockit 1.4.2_05):

1. Start your Java application with BEA JRockit as usual, but add the `-Xmanagement` option to the command line.
2. Start the Management Console and connect to the BEA JRockit you just started. (See Start the JRockit Management Console for details on how to do this.)
3. Click on the tab named MemLeak Detector.

   Figure 5-1 displays the content of this tab.

Figure 5-1 MemLeak Detector Tab in the BEA JRockit Management Console


JRockit Management Console
Connection View Tools Help
Connections
localhost
Ignore List
Overview Memory Processor System Notification MemLeak Detector
localhost [localhost:7090]
Enable memleak system
Disable memleak system
Trend analysis: Is the system leaking?
Type Growth (bytes/sec) % of heap Size (KB) #instances
Continuous update
Freeze
Connected (1/1)

Technology Preview - For Development Use Only Updates/Production use, see http://www.jrockit.com/tools

4. Click Enable memleak system. This automatically starts the trend analysis, which causes information about different object types to be displayed in the Trend analysis table.

   In Table 5-1 you can find detailed explanations of what each column stands for. When starting the memory leak data collection you also get a message from JRockit that the “ManagementServer started trend analysis”.

Table 5-1 Column Descriptions

Column Title	Description
Type	The type of object
Growth	How much memory (in bytes) is allocated for this type of object per second.
% of heap	How large percentage of the heap is occupied by this type of object.
Size	What size in kB does that percentage correspond to.
#instances	How many objects of this type is currently referenced.

Using the Memory Leak Detector

This section describes how to use the Memory Leak Detector. You will find the following topics:

- Overview of the Memory Leak Detection Process
- Getting Started
- Memory Leak Detection
- An Example of How to Find a Real Memory Leak

Overview of the Memory Leak Detection Process

The memory leak detection process consists of three phases:

1. trend analysis
2. studying object type relations
3. instance investigation

Trend analysis means to observe continuously updated object type related information and try to discover object types with suspicious memory growth. These object types should then be studied in the next phase of the memory leak detection process. The information in the trend analysis table will be updated each time a garbage collection is performed.

Studying object type relations means repeatedly following reference paths between object types, i.e. classes. The goal is to find interesting connections between growing object types and what types of objects points to them. Finding the object type guilty of the unusual memory growth will lead to the third and final phase of the memory leak detection process.

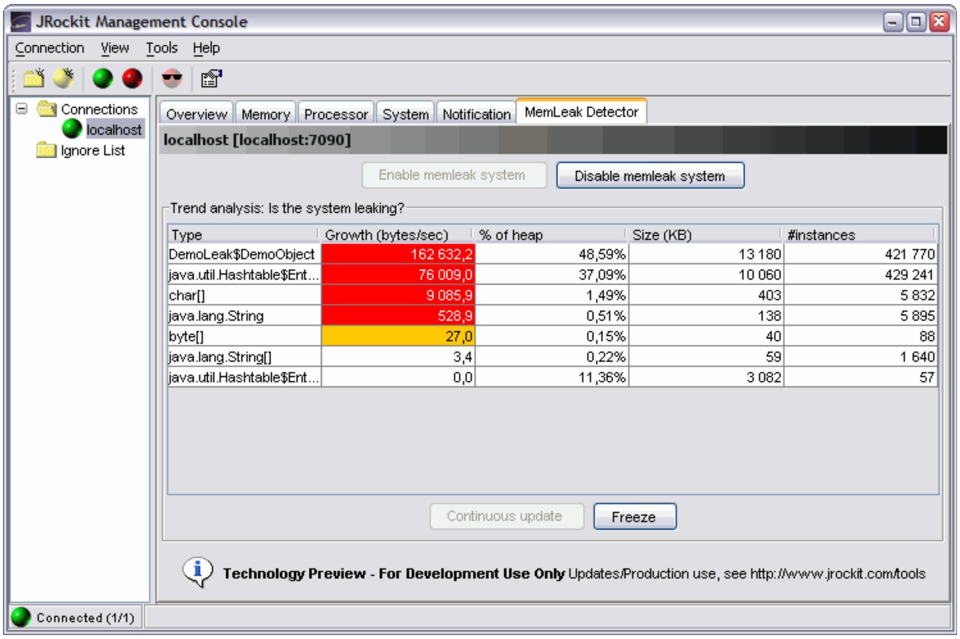
Instance investigation consists of finding an instance of abnormal memory size or holding an abnormal amount of references and then inspect that instance. When inspecting an instance, values will be displayed, e.g. field names, field types, and field values. These values will hopefully lead you to the correct place for the error in the application code, i.e. where that particular instance of that particular object type is allocated, modified, or removed—depending on what the situation implies. Minimizing the problem areas to the ones connected to the suspected instance will most likely lead you on the right track to finding the actual problem causing the memory leak and fix it.

Getting Started

To analyze an application you need to start the Memory Leak Detector (see Starting the Memory Leak Detector).

1. Click Enable memleak system to start the continuous update of profiling data, the result should look something like Figure 5-2.

Figure 5-2 Memory Leak System Enabled


JRockit Management Console
Connection View Tools Help
Connections
localhost
Ignore List
Overview Memory Processor System Notification MemLeak Detector
localhost [localhost:7090]
Enable memleak system
Disable memleak system
Continuous update
Freeze
Connected (1/1)

Trend analysis: Is the system leaking?

Type	Growth (bytes/sec)	% of heap	Size (KB)	#instances
DemoLeak$DemoObject	162 632,2	48,59%	13 180	421 770
java.util.Hashtable$Ent...	76 009,0	37,09%	10 060	429 241
char[]	9 085,9	1,49%	403	5 832
java.lang.String	528,9	0,51%	138	5 895
byte[]	27,0	0,15%	40	88
java.lang.String[]	3,4	0,22%	59	1 640
java.util.Hashtable$Ent...	0,0	11,36%	3 082	57

Technology Preview - For Development Use Only Updates/Production use, see http://www.jrockit.com/tools

The growth values marked in red show the object types that grow more than 100 bytes/sec. The areas marked in yellow indicate object types that grow between 10 and 100 bytes/sec. Object types that grow less than that are white.

2. Click Freeze to stop the continuous update of the data. By doing so you enable the possibility to analyze the data displayed.

   If you want the table to start collecting data again, click Continuous update.

   Note: The trend analysis will be reset to zero when starting continuous update, i.e. once you click Continuous update, the frozen data is lost.

Memory Leak Detection

The following sections will guide you on how to use the Memory Leak Detector to help you with the memory leak detection process.

1. Click Freeze to stop the continuous update of the memory leak data collection.

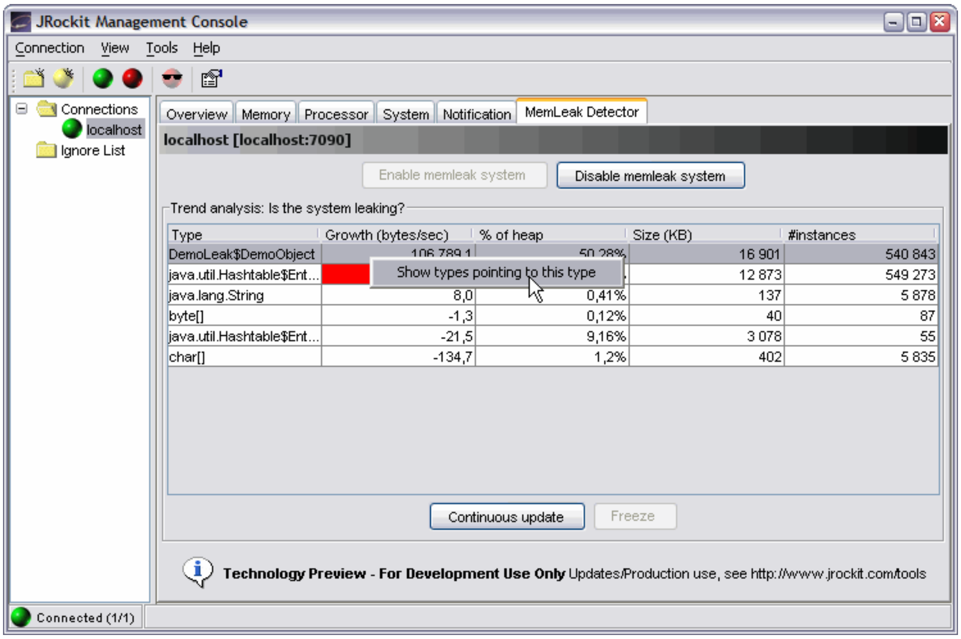
   The JRockit process announces that the "ManagementServer stopped trend analysis".

2. Mark any object type you find interesting. Probably one with a high growth value (most likely marked red or yellow).

3. Right-click on the selection.

   A menu appears, see Figure 5-3.

Figure 5-3 Marking Suspect Object Type


JRockit Management Console
Connection View Tools Help
Connections
localhost
Ignore List
Overview Memory Processor System Notification MemLeak Detector
localhost [localhost:7090]
Enable memleak system
Disable memleak system
Show types pointing to this type
Continuous update
Freeze
Connected (1/1)

Trend analysis: Is the system leaking?

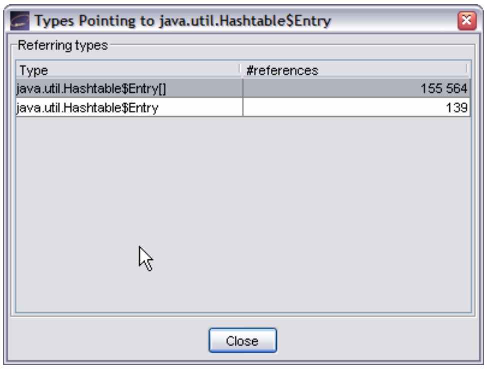
Type	Growth (bytes/sec)	% of heap	Size (KB)	#instances
DemoLeak$DemoObject	106 789,1	50,28%	16 901	540 843
java.util.Hashtable$Ent...	🟥		12 873	549 273
java.lang.String	8,0	0,41%	137	5 878
byte[]	-1,3	0,12%	40	87
java.util.Hashtable$Ent...	-21,5	9,16%	3 078	55
char[]	-134,7	1,2%	402	5 835

Technology Preview - For Development Use Only Updates/Production use, see http://www.jrockit.com/tools

4. Select Show types pointing to this type.

   The Referring Types Window appears that displays a list of the object types pointing to that particular type of object, see Figure 5-4.

Figure 5-4 Referring Types Window: Investigating Suspicious Object Types



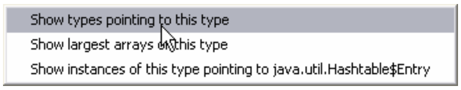
Types Pointing to java.util.Hashtable$Entry

Referring types

Type	#references
java.util.Hashtable$Entry[]	155 564
java.util.Hashtable$Entry	139

5. Mark an object type you wish to study and right-click, the different action alternatives appear, see Figure 5-5.

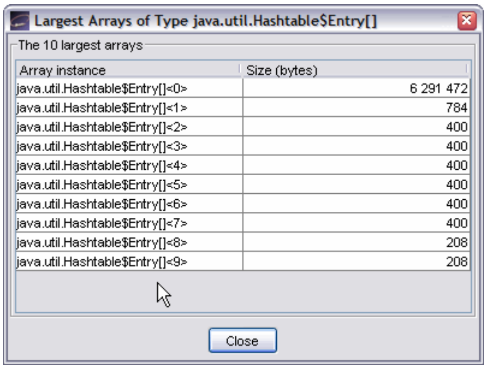
Figure 5-5 Available Action Alternatives


Show types pointing to this type
Show largest arrays of this type
Show instances of this type pointing to java.util.Hashtable$Entry

6. Select the option you wish to study.

- *Show types pointing to this type* alternative (see how to in section 4.).
- *Show largest array of this type* if the selected object type happens to be an array. This function lists the ten largest array instances of this type in the Largest Arrays Window, see Figure 5-6.
- *Show instances of this type pointing to <TYPE>*. This alternative lists—if it is not too large—all instances of the selected object type pointing to objects of type *TYPE* in the Referring Instances Window, see Figure 5-7.

Figure 5-6 Largest Array Window: Looking at the Largest Array Instances

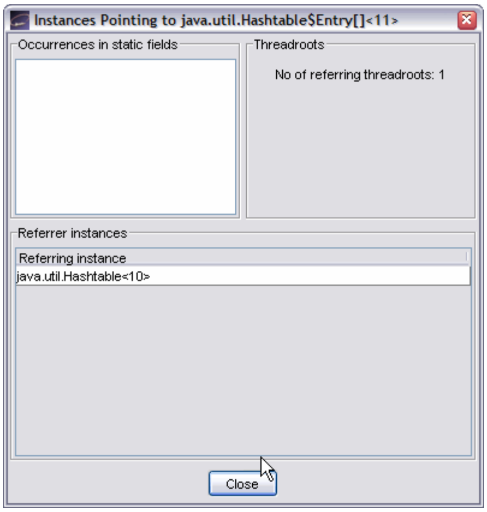


Array instance	Size (bytes)
java.util.Hashtable$Entry[]<0>	6 291 472
java.util.Hashtable$Entry[]<1>	784
java.util.Hashtable$Entry[]<2>	400
java.util.Hashtable$Entry[]<3>	400
java.util.Hashtable$Entry[]<4>	400
java.util.Hashtable$Entry[]<5>	400
java.util.Hashtable$Entry[]<6>	400
java.util.Hashtable$Entry[]<7>	400
java.util.Hashtable$Entry[]<8>	208
java.util.Hashtable$Entry[]<9>	208

7. Follow the suspicious instances in Figure 5-6 by selecting and right-clicking the selection and choosing *Show instances pointing to this array.*

   In the new Referring Instances Window that appears, see Figure 5-7, you can see static fields and how many thread roots are referring to the instance in question.

Figure 5-7 Referring Instances Window: Instances Pointing to an Array Instance.


Instances Pointing to java.util.Hashtable$Entry[]<11>
Occurrences in static fields
Threadroots
Referrer instances
Close

No of referring threadroots: 1

Referring instance
java.util.Hashtable<10>

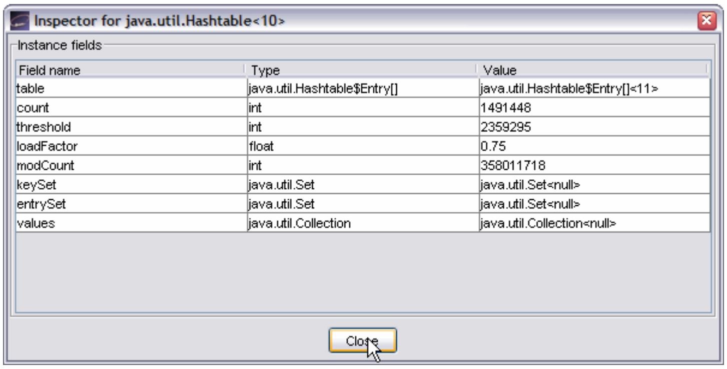
8. Select a referring instance and right-click.

9. Click Inspect. The Inspection Window appears, see Figure 5-8.

   When inspecting an instance you get all necessary data to hopefully track the particular instance in the application code. You get the field names, their types, and their values.

Trying to map these values to the code will help you discover the source of the memory leak.

Figure 5-8 Inspection Window: Inspecting an Instance



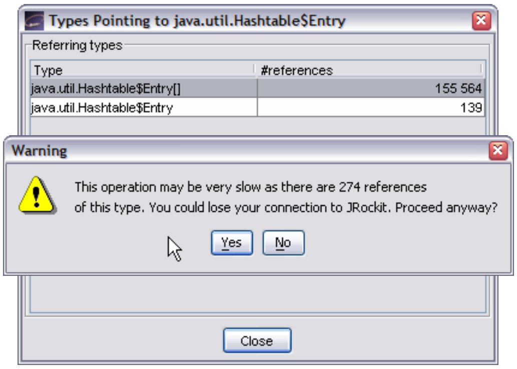
Inspector for java.util.Hashtable<10>

Instance fields

Field name	Type	Value
table	java.util.Hashtable$Entry[]	java.util.Hashtable$Entry[]<11>
count	int	1491448
threshold	int	2359295
loadFactor	float	0.75
modCount	int	358011718
keySet	java.util.Set	java.util.Set<null>
entrySet	java.util.Set	java.util.Set<null>
values	java.util.Collection	java.util.Collection<null>

In some cases the lists that is displayed may be very long. In those cases you will be notified and informed about the consequences of displaying such a list, see Figure 5-9.

Figure 5-9 Trying to display a very long list



Types Pointing to java.util.Hashtable$Entry

Referring types

Type	#references
java.util.Hashtable$Entry[]	155 564
java.util.Hashtable$Entry	139

Warning

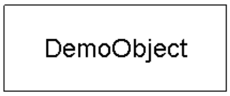
This operation may be very slow as there are 274 references of this type. You could lose your connection to JRockit. Proceed anyway?

An Example of How to Find a Real Memory Leak

Below follows an example of how to find a memory leak by narrowing down the search space. Once the search space is narrowed down, you will hopefully find the exact problem area and then be able to solve the problem by changing you application.

After starting the Memory Leak Detector, choose to investigate the object type that grows the most and which is not expected to grow, considering the design and the purpose of the application. In this example it turns out to be the DemoObject object type (Figure 5-10). Select the row corresponding to the suspected object type, right-click on it and choose the menu option *Show types pointing to this type.*

Figure 5-10 Trend Analysis Gave the Object Type DemoObject



A Referring Types Window (Figure 5-4) appears displaying the one object type pointing to the DemoObject. It turns out to be HashTable$Entry. To pursue this suspected memory leak path, select the corresponding row, right-click on the selection, and choose *Show types pointing to this type* once again.

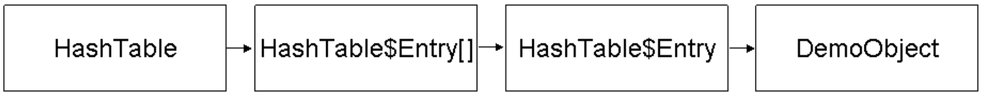
In the new Referring Types Window that appears, note the two object types pointing to the previous object type. One is HashTable$Entry[ ] and the other is HashTable$Entry. Notice that the number of references from the HashTable$Entry[ ] type is much larger than expected and choose to investigate this array type further. Select the row corresponding to the array type, right-click the selection, and choose *Show types pointing to this type.* By selecting this option, you find that the object type HashTable is pointing to the array in the new Referring Types Window that appeared.

Return to the previous window and select the suspected array type again. Right-click on the selection, but this time choose *Show largest arrays of this type.* A Largest Array Window (Figure 5-6) appears with a list of the ten largest instances of this array, with the largest listed on top. This information tells you that it is one single array instance being responsible for the large memory occupation.

Instead of choosing the other alternative: *Show instances of this type pointing to <TYPE>*, where TYPE in this case is HashTable$Entry the new window that appears presents a warning for a large amount of references maybe causing the connection to the JRockit process to be lost. This means that the HashTable$Entry[ ] consumes unexpected amounts of memory and is holding on to an enormous amount of references to HashTable$Entry.

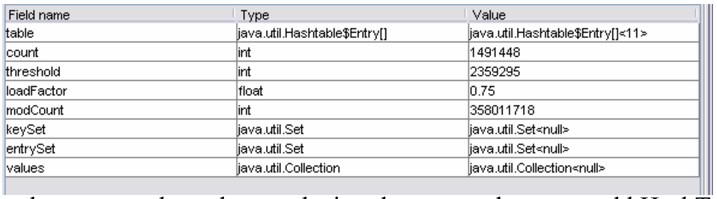
Return to the Largest Array Window. Select and right-click the suspected alternative, i.e. the instance occupying the largest amount of memory. Select *Show instances pointing to this array.* A Referring Instances Window (Figure 5-7) appears with a list of instances pointing to the array. It is a HashTable instance (Figure 5-11).

Figure 5-11 Studying Object Type Relations Resulted In the Following Schedule



The Inspection Window (Figure 5-8) appears. In that window you can see different field names, their types, and their values (Figure 5-12). Pretty soon you will probably be able to map these fields and values to a certain point in the application code.

Figure 5-12 Instance Inspection Helps Mapping the Problem to Corresponding Code



Field name	Type	Value
table	java.util.Hashtable$Entry[]	java.util.Hashtable$Entry[]<11>
count	int	1491448
threshold	int	2359295
loadFactor	float	0.75
modCount	int	358011718
keySet	java.util.Set	java.util.Set<null>
entrySet	java.util.Set	java.util.Set<null>
values	java.util.Collection	java.util.Collection<null>

From this example, you can draw the conclusion that somewhere you add HashTable$Entry instances into the HashTable$Entry[ ] which is kept alive by a HashTable. You can also read that your application never seems to remove them, since the memory occupied by these type of objects is continuously growing. To confirm the beliefs, investigate the code thoroughly at the place where the instance field info has taken you.

The source of the confirmed memory leak turned out to be in a place in the code where, after HashTable$Entry objects are added to the HashTable$Entry[ ] in a HashTable. The application removed all of the HashTabe$Entry objects except one; it missed the last instance due to an off-by-one error (a very common error causing memory leaks of this sort).

Help Us Improve BEA JRockit

The Memory Leak Detector provides an easy way to capture information about object type allocation statistics. It is designed to help developers to easier find memory leaks and to better understand critical points of program engineering.

If you have any suggestions relevant for this purpose on how to improve this tool or information on how it is most commonly used in development environments, we would be grateful to receive your input. This information would contribute to our understanding on how to best further improve this tool in the future.

Please, send an email with feedback and your ideas on how to use it to:

jrockit-improve@bea.com

How will BEA Systems Use This Feedback

The feedback will be considered by the development team designing the Memory Leak Detector. We will look at collected ideas and improve the tools of BEA JRockit to make them even easier to use. Our goal with the development of this tool is to simplify the difficult task of finding memory leaks in the future and help the developer work more efficiently.

BEA JRockit is already providing a lot of appreciated manageability tools, and to stay appreciated and to keep a close dialogue with developers using Java Runtime Environments, BEA Systems is always trying to find ways to improve BEA JRockit. This is one of the ways.

BEA JRockit Support for the Memory Leak Detector

Only more recent versions of BEA JRockit fully support the Memory Leak Detector: BEA JRockit 1.4.2_05.

Frequently Asked Questions

Following are some questions we have frequently been asked about the Memory Leak Detector:

- Does BEA Systems Guarantee the Accuracy of this tool's output?
- Does the Memory Leak Detector Cause Any Overhead?
- What Kind of Support is Available for the Memory Leak Detector?
- Is There a Forum Where I can Discuss the Memory Leak Detector?

Does BEA Systems Guarantee the Accuracy of this tool's output?

Since this is not a supported product, we cannot make any guarantees about the correctness of the data we show or the stability of the product when using the Memory Leak Detector.

Does the Memory Leak Detector Cause Any Overhead?

During the first phase of the memory leak detection process the data presented is continuously updated; however, the overhead during this phase is very small. During the second and third phase the only overhead that will be caused is some additional garbage collections which in most cases is negligible. Overall, there is practically no overhead and it should not affect the speed or results of your application.

What Kind of Support is Available for the Memory Leak Detector?

The Memory Leak Detector functionality is currently being provided as-is for your convenience and to help with memory leak detection and is not supported by BEA Support.

Is There a Forum Where I can Discuss the Memory Leak Detector?

If you have any questions you are welcome to share them in the BEA JRockit general interest newsgroup, which is monitored by our engineering team. To access the newsgroup, go to:

http://newsgroups.bea.com

Known Issues

Sometimes static fields and the number of thread roots are not correctly displayed in the window displaying instances referring to an other instance. This can be helped by starting the memory leak detection process one again (i.e. unfreezing and freezing the memory leak system). However, if there is data displayed, it is the correct values.

Code Caching with BEA JRockit

Code caching is being introduced as an experimental feature in this version of BEA JRockit. In a future release of BEA JRockit, this feature is planned to be supported when used only with BEA WebLogic Server. For all other Java applications, it is provided "as-is" without any expressed or implied warranties or support by BEA Systems, Inc. and might contain errors and/or inaccuracies. Use of this feature with all other Java applications other than WebLogic Server is left solely to the discretion of the user without any endorsement from BEA Systems. Questions and problems may be reported via online BEA JRockit 1.4.2 newsgroups at `http://newsgroups.bea.com`.

Code caching—or code persistence—is the process of storing generated machine code to disk for retrieval when that code is required in a subsequent instance of the JVM. Since cached code is already generated, the time that code generation would require on subsequent startups is lessened and—usually—execution time is reduced. Persisting code should not be mistaken for hibernation, which means storing not only code but also objects and the entire heap.

This section describes how BEA JRockit's code caching feature works and shows you how to run it. It includes information on the following subjects:

- Why Is Code Caching Helpful?
- What is the Cache?
- How to Use Code Caching
- How Code Caching Works

- Error Recovery
- Cleaning Up the Cache

Why Is Code Caching Helpful?

Startup time is a concern to many people, especially during development. For a typical run of the `javac` compiler, code generation in the JVM represents a majority of the startup time. By using code caching, startup time—and thus execution time—can be improved significantly.

What is the Cache?

The "cache" is a directory in your file system that stores previously generated code. The cache is shared among all instances of BEA JRockit that are started on the machine. The directory itself is comprised of a number of different files. The cache is logically divided into parts. Each part consists of a code file (with a `.code` file extension) and the index file (with a `.ndx` extension). Each part is numbered and the number is the same for the code and index file. Thus you will find pairs of files called, for example, `1.code` and `1.ndx`.

How to Use Code Caching

When BEA JRockit is started with code caching enabled, JIT-ed code is written to the cache. The next time you run BEA JRockit, it reads this file and instantiates any necessary cached methods as compiled code, rather than as bytecode that would require compilation.

This section describes how to use code caching by running these functions:

- Enabling Code Caching
- Specifying a Cache Name
- Code Caching in the Read/Write Mode
- Code Caching in the Read-only Mode
- Other Code Caching Arguments
- Using Code Caching to Improve Performance
- Setting the Verbosity Level
- Enabling Code Caching by Using an Environment Variable

Enabling Code Caching

Enable code caching by using the `-XXcodecache` command line option with the appropriate arguments, as needed; for example:

```
-XXcodecache:[dir=directory],[readonly],[clobber]
```

Specifying a Cache Name

By default, the cached code is written to a per-user directory referred to as the "cache." The default locations are as follows (on windows the exact path name is different on different locales):

- Windows:

```
C:\Documents and Settings\username>\Local Settings\Application Data\JRockit
CodeCache\
```

- Linux:

```
/tmp/<username>_jrockit_codecache/
```

To use a different cache directory, use the `-XXcodecache:dir=` command to specify the new cache name:

```
-XXcodecache:dir=/path/to/myCacheDirectory
```

For information on the cache directory, please refer to How Code Caching Works.

Code Caching in the Read/Write Mode

By default, code caching runs in read/write mode. This means that when BEA JRockit encounters a new method, it first checks the cache to see if a compiled version of the method is available in the cache. If so, that version is read into memory and used. If the method is not available from the cache, it is generated and then stored to cache.

Code Caching in the Read-only Mode

You can also instruct BEA JRockit to read the cache but not write newly compiled code to it. This is helpful, for example, when you have deployed your application in a production environment and want to ensure nothing changes in the working cache. Using `readonly` will prevent any updating. A typical command line invoking `readonly` might look like this:

```
-XXcodecache:file=myCacheFile,readonly
```

Other Code Caching Arguments

Table 6-1 list additional code caching arguments that you can use along with `dir=` and `readonly`.

Table 6-1 Other Code Caching Arguments

Argument	Description
`clobber`	Unconditionally overwrites code cache. This option is useful to force the recreation of a cache from scratch. Otherwise, by default the VM will attempt to append to an existing cache. This option is overridden by the `readonly` option.
`exitonerror`	Exit BEA JRockit when a code caching error occurs. By default BEA JRockit will recover from an error in the code cache by disabling the cache.

Using Code Caching to Improve Performance

The simplest way of using code caching to improve the startup time performance of an application is to run the application once to build an initial cache, then to rerun the same application; for example:

```
java -XXcodecache HelloWorld.
```

This will create the cache and add to that cache any compiled methods.

```
java -XXcodecache HelloWorld
```

This will use the *previously* created cache and thus run faster since no code will need to be compiled.

Setting the Verbosity Level

You can use the option `-Xverbose:[codecache,codecache2]` to set verbosity mode for status messages. `codecache` will print basic information about the cache as well as information about methods that could not be saved or loaded. Verbose level 2 (`codecache2`) will print more detailed messages about each method as it is being saved or loaded.

Enabling Code Caching by Using an Environment Variable

Instead of enabling code caching on each command line, you can set the environment variable `JR_CODECACHE` to enable code caching for all invocations of BEA JRockit. The value of the environment variable should be the same as the arguments to `-XXcodecache`. Since empty environment variables are not supported by all operating system, it is possible to set the variable to the value `enable` if all you want is the default options.

For example:

```
set JR_CODECACHE=enable
set JR_CODECACHE=ro,dir=/path/to/myCodeCache
```

How Code Caching Works

This section describes how code caching works. It includes descriptions of the following functionality:

- What Happens When Code Caching Runs
- Dealing with Code Changes
- Dealing with Cache Cleanup
- Removing Obsolete Methods
- Cache File Validity

What Happens When Code Caching Runs

When you start a Java program with code caching enabled, BEA JRockit access the cache directory and either opens an existing cache file or creates a new empty file. As new methods are generated, they are written to one of these files (if it is an existing file, the information is appended to the existing information). Finally, during shutdown, a new cache index is created and the cache directory is closed. If a fatal error occurred that would result in the creation of an invalid cache, the corrupted cache is automatically removed.

A code cache is not only application specific, but also application usage specific. This is because methods can be generated differently depending on class initialization order, static initializers, assertion status, and other conditions which may change if the application is run differently. Generally, as long as use of the application does not change significantly, these assumptions—which are checked when a method is loaded—will remain valid. If not, some of the methods stored in the cache will become invalid and will have to be regenerated. The newly generated methods are appended to the cache and override the previous definitions, which will still remain in the cache but are inaccessible.

Dealing with Code Changes

The codecache functionality will prevent cached code that is no longer valid from being used by the JVM. If changes are made to the class files after the code is stored to the cache, the code caching system prevents that stored code from being retrieved during subsequent runs. Many methods have dependencies to other classes and methods because of in-lining and other optimizations; these dependencies must also be stored and if they are no longer valid at retrieval time the stored method cannot be used.

To ensure that changed code is not retrieved, BEA JRockit stores classes based on the secure `checksum` (MD5 version) of the class bytes. This information is used as the index for a class when BEA JRockit looks up stored methods. Classes that have been changed are detected by a change in `checksum`, and any stored methods which depend on the old version of the class will be invalidated and will have to regenerated. A dependency check is run when loading a method to determine if the method has any dependencies on classes that have been invalidated.

Dealing with Cache Cleanup

This version of BEA JRockit has no provision for cleaning up the cache. It is therefore a good practice to occasionally delete the cache and recreate it by rerunning the application. This is also true if the application is being currently developed, in which case classes will be recompiled causing cached methods to be replaced. For more information, please refer to Cleaning Up the Cache.

Removing Obsolete Methods

Once the application development cycle has completed and the application usage has stabilized, it is probably a good idea to regenerate the cache to remove obsolete methods. Following that, the best way to deploy the application would be to run the application at install time and generate a cache, then subsequently run the application with this cache in read only mode as suggested previously. This will prevent new methods from being added to the cache, and allow the application cache to be shared.

Cache File Validity

The cache generated is very dependent on machine-specific factors as well as some JVM options. If any of these constraints are not satisfied when loading a cache, it is considered invalid. When the cache is invalid and you are running code caching in the read-only mode, the JVM code caching is disabled. If you are running it in the read/write mode, an attempt will be made to overwrite the cache. In this version of code caching, the file is dependent on machine architecture and operating system type.

Error Recovery

When code caching is enabled, BEA JRockit will always attempt to recover from a serious error specific to code caching. Examples of such errors are running out of memory, invalid code cache format, I/O errors while reading or writing from the cache. In these cases, the VM will continue running the application but code caching will be disabled and an error message will be displayed on the screen.

On the other hand, being unable to store or load an individual method because loading or saving constraints have changed is not considered an error and code caching will continue without saving or loading that particular method. If no verbosity is chosen, nothing will be reported. `-Xverbose:codecache` will result in messages about specific methods that could not be saved or loaded.

Cleaning Up the Cache

Currently, BEA JRockit does not perform any sort of cache cleanup. This means that any unused classes and methods will continue to persist in the cache even if they are never used again. This is helpful if you later decide to use a previous version of a class as it might still be in the cache. But to clean up a cache in this version of BEA JRockit, you will have to rebuild it when you have finished developing the application.

CHAPTER 7

Using BEA JRockit JVM with Other WebLogic Applications

The configuration options described elsewhere in this user guide can be set to optimize BEA 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 BEA JRockit JVM with BEA WebLogic Server
- Configuring JRockit for BEA WebLogic Workshop

Using BEA JRockit JVM with BEA WebLogic Server

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

- Certified Versions
- Verifying that BEA JRockit is Your JVM
- Starting JRockit from the Node Manager
- Enabling the Management Server from the Node Manager
- Tuning BEA JRockit for WebLogic Server
- Setting Options by Using the Node Manager
- Monitoring BEA JRockit JVM from WebLogic Server
- Switching to BEA 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 BEA 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 BEA JRockit is Your JVM

BEA 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 BEA JRockit is the JVM running with your instance of WebLogic Server, at the command line, type:

```
java -version
```

If BEA JRockit is running, the system will respond:

```
java version "1.4.2_04"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_04-b05)
BEA JRockit(TM) 1.4.2_04 JVM (build ari-29212-20040415-1348-win-ia32,
Native Threads, GC strategy: parallel)
```

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 BEA 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 BEA 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 BEA 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 BEA JRockit for WebLogic Server

To use the BEA JRockit JVM instead of the Sun JVM, you need to increases 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 BEA 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 affect on processing (see Setting the Size of the Nursery).

For more information on tuning BEA JRockit, please refer to *Tuning BEA JRockit JVM*.

Monitoring BEA JRockit JVM from WebLogic Server

If you run WebLogic Server with BEA 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 BEA JRockit JVM, do the following:

1. Start WebLogic Server with BEA 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 BEA 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 BEA 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 `-Xmx` option.
Gc Algorithm	Indicates the type of garbage collector that BEA JRockit JVM is using.
Total Garbage Collection Count	Indicates the number of garbage collection runs that have occurred since the VM was started.
GCHandles Compaction	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. Values: • true • false

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

Attribute	Description
Concurrent	Indicates whether JRockit's garbage collector runs in a separate Java thread concurrently with other Java threads. Values: • true • false
Generational	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. Values: • true • false
Incremental	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). Values: • true • false
Parallel	Indicates whether the JRockit's garbage collector is able to run in parallel on multiple processors if multiple processors are available. Values: • true • false
Number Of Processors	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.
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 BEA JRockit, such as how long it spends in a specific method, use the BEA JRockit Management Console, as described in Using the BEA JRockit JVM Management Console.

Switching to BEA JRockit JVM in WebLogic Server

When you switch to BEA 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 BEA 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 BEA JRockit JVM from another JVM, please refer to Migrating to BEA JRockit. For more information on using the Configuration Wizard when switching to BEA 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 BEA 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 BEA JRockit for WebLogic Server.

Adding Custom Notification Actions and Constraints

After starting the BEA 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 A-1. This class will receive the `NotificationEvents`.

Listing A-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() + "======");
   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 A-2. This method acts on the incoming event.

Listing A-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 A-3 shows how this is done.

Listing A-3 Creating the Action Editor

```
package com.example.actions;
```

```
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 A-4.

Listing A-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 A-5.

Listing A-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 A-6:

Listing A-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)
{
```

Using the Java Plugin

Popular web browsers, such as Netscape Navigator and Microsoft Internet Explorer, come with a default JRE, under which applets and Java beans are run, already installed. If you want to run applets under BEA JRockit JRE, you can implement the Java plug-in that ships with this product.

Available only on IA32 implementations, the BEA JRockit Java plug-in extends the functionality of your web browser, allowing applets and Java beans run under it rather than its default JRE. The Java Plug-in is part of the BEA JRockit JRE and is installed when the JRE is installed on a computer. It works with both Netscape and Internet Explorer.

This section includes information on the following subjects:

- Supported Operating Systems and Browsers
- Installing the Plugin
- Implementing the Plugin
- Plugin Reference

Supported Operating Systems and Browsers

Table B-1 lists the operating systems and browsers supported by the BEA JRockit Java plug-in.

Table B-1 Java-in Plugin O/S and Browser

Operating System Support	For a list of supported operating systems, see the list of supported AI32 platforms at BEA JRockit 1.4.2 SDK Platform Support.
Browser Support	Netscape 4.7.x, 6.2.2, 7 Mozilla 1.2.1, 1.3, 1.4, 1.4.1 Internet Explorer 5.5 (SP2+), 6.x For information on the latest browser support see: http://java.sun.com/j2se/1.4.2/system-configurations.html **Note:** Netscape 4.79 will only run applets that are specifically tagged to be run by 1.4.2_04. You can find examples of these applets in the linux32 and win32 demos available in `demo/plugin/applet/*` at: `http://edocs.bea.com/wljrockit/docs142/demo_src.html`. Download either `jrockit-j2sdk1.4.2_04-win32-demo.zip` or `jrockit-j2sdk1.4.2_04-linux32-demo.zip` (depending upon your operating system) and extract the necessary demos.

Installing the Plugin

The Java plugin is installed automatically for Win32 machines when you install BEA JRockit JRE, as described in "Installing the JRE." For Linux32 machines, you can install it as described in either of these documents from Sun Microsystems:

- Manual Installation/Registration of Java Plug-in—Linux (manual installation and registration)
- Control Panel Script Options for Plug-in Registration (automatic installation and registration)

Note on Installing the BEA JRockit Plugin and Sun Plugin

If you installing the Sun JRE after installing the BEA JRockit JRE, the Sun JRE will become the default Java Plugin on the system. If this happens, you should uninstall and reinstall the BEA JRockit JRE again.

Implementing the Plugin

You can implement the BEA JRockit Java plug-in the same way you would implement a similar product from Sun Microsystems; that is, by using one of these two different methods:

- Including the conventional APPLET tag in a web page, as described in "Using the Conventional APPLET Tag."
- Replacing the APPLET tag with the OBJECT tag for Internet Explorer or by replacing the APPLET tag with the EMBED tag for Netscape 4. Be aware that the OBJECT and EMBED tags must conform to a special format as described in the Sun Microsystems document, "Using OBJECT, EMBED and APPLET Tags in Java Plug-in."

Plugin Reference

Generally, once the plug-in is installed, its behavior will be transparent and require little, if any, user intervention. However, there are many other related topics that you may want to understand. Sun Microsystems provides helpful information on their Java Plugin that is helpful to BEA JRockit Java plug-in users. You can find this information at:

`http://java.sun.com/products/plugin/reference/docs/index.html`

Detailed information of particular interest to developers can be found in Sun Microsystems' Java Plug-in 1.4.2 Developer Guide. This document contains such information as

- How proxy configuration works in Java Plugin.
- What protocols does the Java Plugin support.
- Discussions on cookie support and caching.
- Behavior of Java Plugin and how set options via the Java Plugin Control Panel.
- How to deploy the Java Plugin on the Internet, within an intranet, via Java Server Pages, and so on.
- Plugin security, including RSA signed applet verification.
- Java Plugin debugging support for applets.
- Supporting multiple JREs in the same environment, Java-to-JavaScript communication, how to persist applets across browser sessions, and other advance topics.

Tracing Thread Activity With Stack Dumps

Stack dumps, or "stack traces," reveal information about an application's thread activity that can help you diagnose problems and better optimize application and JVM performance; for example, stack dumps can show the occurrence of "deadlock" conditions, which can seriously impact application performance.

Stack dumps usually occur when certain errors are thrown. You can also create a stack dump by invoking a control break (usually by pressing `Ctrl-Break` or `Ctrl-\`; or `SIGQUIT` on Linux). This section provides information on working with stack dumps. It includes information on these subjects:

- Monitoring Information in Stack Dumps
- Detecting Deadlocks

Monitoring Information in Stack Dumps

When printing stack traces with `Control-Break`, BEA JRockit also shows the status of active locks (monitors). For each thread, BEA JRockit prints the following information if the thread is in a waiting state:

If the thread is trying to take a lock (to enter a synchronized block), but the lock is already held by another thread, this is indicated at the top of the stack trace, as "Blocked trying to get lock".

If the thread is waiting on a notification on a lock (by calling `Object.wait()`), this is indicated at the top of the stack trace as "Waiting for notification".

If the thread has taken any locks, this is shown in the stack trace. After a line in the stack trace describing a function call is a list of the locks taken by the thread in that function. This is described as `^-- Holding lock` (where the `^--` serves as a reminder that the lock is taken in the function written above the line with the lock).

Caution: The lines with the lock information might not always be correct, due to compiler optimizations. This means two things:

- If a thread, in the same function, takes first lock A and then lock B, the order in which they are printed is unspecified.
- Sometimes, if a thread, in method `foo()` calls method `bar()`, and takes a lock A in `bar()`, the lock might be printed as being taken in `foo()`.

Normally, this shouldn't be a problem. The order of the lock lines should never move much from their correct position. Also, lock lines will never be missing—you can be assured that all locks taken by a thread are shown in the stack dump.

The semantics for waiting (for notification) on an object in Java is somewhat complex. First you must take the lock for the object, and then you call `wait()` on that object. In the wait method, the lock is released before the thread actually goes to sleep waiting for a notification. When it receives a notification, wait re-takes the lock before returning. So, if a thread has taken a lock, and is waiting (for notification) on that lock, the line in the stack trace that describes when the lock was taken is not shown as “Holding lock,” but as “Lock released while waiting.”

All locks are described as `Classname@0xLockID[LockType]`; for example:

```
java/lang/Object@0x105BDCC0[thin lock]
```

Where:

- `Classname@0xLockID` describe the object the to which the lock belongs. The classname is an exact description, the fully qualified class name of the object. `LockID`, on the other hand, is a temporary ID which is only valid for a single thread stack dump. That is, you can trust that if a thread A holds a lock `java/lang/Object@0x105BDCC0`, and a thread B is waiting for a lock `java/lang/Object@0x105BDCC0`, in a single thread stack dump, then it is the same lock. If you do any subsequent stack dumps however, `LockID` is not comparable and, even if a thread holds the same lock, it might have a different `LockID` and, conversely, the same `LockID` does not guarantee that it holds the same lock.
- `LockType` describes the kind of BEA JRockit internal lock type the lock is. Currently, three kinds of locks exist:

– fat locks: locks with a history of contention, or that have been waited on (for notification).

– thin locks: locks that have no contention (several threads trying to take the lock simultaneously).

– recursive locks: locks occur when a thread takes a lock it already holds.

Listing C-7 shows an example of what a stack trace for a single thread can look like.

Listing C-7 Example: Stack Trace for a Single Thread

```
"Open T1" prio=5 id=0x680 tid=0x128 waiting
   -- Waiting for notification on: java/lang/Object@0x1060FFC8[fat lock]
   at jrockit/vm/Threads.waitForSignalWithTimeout(Native Method)@0x411E39C0
   at jrockit/vm/Locks.wait(Locks.java:1563)@0x411E3BE5
   at java/lang/Thread.sleep(Thread.java:244)@0x41211045
   ^-- Lock released while waiting: java/lang/Object@0x1060FFC8[fat lock]
   at test/Deadlock.loopForever(Deadlock.java:67)@0x412304FC
   at test/Deadlock$LockerThread.run(Deadlock.java:57)@0x4123042E
   ^-- Holding lock: java/lang/Object@0x105BDCC0[recursive]
   ^-- Holding lock: java/lang/Object@0x105BDCC0[thin lock]
   at java/lang/Thread.startThreadFromVM(Thread.java:1690)@0x411E5F73
   --- End of stack trace
```

Detecting Deadlocks

After the normal stack dumps, BEA JRockit performs a deadlock detection. This is done by finding “lock chains” in the Java application. If a lock chain is found to be circular, the application is considered caught in a deadlock.

What is a “Lock Chain”?

Although they appear somewhat complex, lock chains are fairly straightforward; they can be defined as follows:

- Threads A and B form a lock chain if Thread A holds a lock that Thread B is trying to take. If A is not trying to take a lock, then the lock chain is “open.”
- If A->B is a lock chain, and B->C is a lock chain, then A->B->C is a more complete lock chain.

- If a Thread D doesn't exist, meaning lock chain C->D doesn't exist, then A->B->C is a complete and open lock chain.

Lock Chain Types

BEA JRockit analyzes the threads and forms complete lock chains. There are three possible kinds of lock chains: Open, Deadlock and Closed lock chains.

Open Chains

Open lock chains represent a straight dependency, as described in What is a "Lock Chain"?. Thread A is waiting for B which is waiting for C, and so on.

Deadlock Chains

Deadlock (circular) chains are similar to an open lock chain, except that the first element is waiting for the last element, in the simplest case: A is waiting for B, which is waiting for A. Note that a deadlocked chain has no head. BEA JRockit selects an arbitrary thread to display as the first element in the chain.

Closed Chains

Closed chains are like open chains, but the first element in the chain is waiting for a lock in another chain. This other chain may be open, deadlocked or closed. If the other chain is deadlocked, then the closed chain is also deadlocked. Note that the division between a closed chain and the other chain is arbitrary.

Closed chains arise whenever two different threads are blocked trying to take the same lock; for example: Thread A holds lock Lock A while Thread B is waiting for Lock A; Thread C is also waiting for Lock A. BEA JRockit will interpret this in one of the following ways:

- B > A as an open lock chain.
- C > A as a closed lock chain.
- C > A as an open lock chain.
- B > A as a closed lock chain.

The only item you might find of interest is if you have a deadlocked lockchain. This can never be resolved, and the application will be stuck waiting indefinitely. Also, if you have long (but open) lock chains, your application might be spending unnecessary time waiting for locks.

Using Web Start with BEA JRockit

This version of BEA JRockit includes an implementation of Java Web Start, a tool that allows you to start Java applications with a single click in your browser. With Web Start, you can download and launch applications directly from the browser and avoid complex and time-consuming installation procedures. Any Java application can be started by using Web Start.

This section includes information on the following subjects:

- What You Can Do with Web Start
- Web Start Security
- Installing and Launching Web Start
- Comprehensive Web Start Documentation

What You Can Do with Web Start

With Java Web Start, you launch applications simply by clicking on a Web page link. If the application is not present on your computer, Java Web Start automatically downloads all necessary files. It then caches the files on your computer so the application is always ready to be relaunched anytime you want—either from an icon on your desktop or from the browser link. And no matter which method you use to launch the application, the most current version of the application is always presented to you.

Web Start Security

Java Web Start includes the security features of the Java platform to ensure the integrity of your data and files. It also enables you to use the latest Java 2 technology with any browser.

Installing and Launching Web Start

Java Web Start is installed as part of the public JRE installation (see Installing the BEA JRockit 1.4.2 JRE).

Windows Implementations

Upon installation, a new icon will appear on your desktop (Figure D-1) and a new selection will appear on your Start menu, under Programs.

Figure D-1 Java Web Start Icon

Use either of these to launch Java Web Start (you can also launch it from the command line by typing:

```
<jre_home>/javaws/javaws
```

(where *<jre_home>* is you JRE home directory; for example:

`C:/jrockit-j2sdk1.4.2_03/jre`).

Linux Implementations

The Linux installation does not change with Web Start added; however you can only launch Web Start from the command line. Do so by entering the command:

```
<jre_home>/javaws/javaws
```

Note: JPackage RPMs will install Java Web Start and you can start by using the same command used for other Linux implementations.

Comprehensive Web Start Documentation

Java Web Start is a Sun Microsystems product and the BEA JRockit implementation is no different than Sun's. Please refer to the following documents for more complete information on using this feature:

- Java Web Start Developers Section:

`http://java.sun.com/products/javawebstart/developers.html`

- Java Web Start API Specification:

`http://java.sun.com/products/javawebstart/reference/api/index.html`

- Code Samples and Applications:

`http://java.sun.com/products/javawebstart/reference/codesamples/index.html`

- Technical Articles & Tips:

`http://java.sun.com/products/javawebstart/reference/techart/index.html`

- FAQs:

`http://java.sun.com/products/javawebstart/faq.html`

Index

A

Administrator mode 4-10
aspect 4-16
aspect value change 4-24
    add constraint 4-26
    on trigger 4-25

C

client-side JVM 2-3
command line options
    -classpath 2-4
    -Client 2-3
    -D 2-4
    -Djrockit.managementserver.maxconnect 4-4
    -Djrockit.managementserver.port 4-4
    -help 2-4
    -Server 2-3
    -showversion 2-4
    -verbose 2-4
    -version 2-4
    -X 2-4
    -Xallocationtype 7-3
    -Xgcpause 3-8
    -Xmanagement 4-2
    -Xnativethreads 7-3
    -Xnoop 2-5
    -Xstrictfp 2-5
    -Xverbose 2-5
    -Xverify 2-5
command-line option 2-10
configuration options 7-1
consolesettings.xml 4-15
CPU Load bar 4-20
CPU Usage chart 4-20
CTRL_LOGOFF_EVENT 2-10

D

dead objects 3-3
Developer Mode 4-10

E

extended options
    -Xnohup 2-10

G

garbage collection 2-3, 7-4
    choosing 3-6
    concurrent
        generational concurrent 3-6, 3-7
        single spaced concurrent 3-5, 3-6, 3-7
    dynamic 3-4
    generational 3-2
    old generation 3-2
    parallel 3-6, 3-7
    pauses 3-5, 3-6
    single-spaced 3-2
    young generation 3-2
garbage collector 3-5
    unified 2-3
    unsupported 3-5

H

heap 7-4
    size 3-4, 3-5

I

Information tabs
    Overview tab 4-16

J

Java heap memory 4-17
Java thread 3-3, 3-5, 3-6, 3-8, 7-5
java.lang.System 2-4

M

Management Console 4-1, 4-16
    adding an exception 4-36
    Administrator mode 4-6
    advanced features
        Exception Count tab 4-35
        Method Profiler tab 4-31
        method templates 4-31
    changing the number of connections 4-4
    command buttons 4-6
    connecting a connection to JRockit 4-8
    connection browser 4-5, 4-6
    connection node 4-6
    CPU Load 4-20
    CPU Load bar 4-17
    CPU Usage chart 4-17
    customizing 4-13
        charts 4-14
        gauges and bars 4-14
        settings file 4-15
    dash board 4-16
    disconnecting a connection from JRockit 4-8
    enabling console settings 4-10
    enabling the management server 4-2
    Exception Count 4-10
    Free Heap 4-18
    Free Memory 4-19
    Garbage Collection System 4-21
    Heap Usage chart 4-17, 4-18
    hiding a disconnected connection 4-10
    information tabs 4-16
        Memory tab 4-17
        Notification tab 4-22, 4-26
        Processor tab 4-19
        System tab 4-20
    JRockit Uptime 4-21
    JVM Process Load 4-20
    Method Profiler 4-10
    new connection 4-7
    Number of Processors 4-20
    parts of 4-4
    Process Affinity 4-21
    removing a connection 4-10
    removing a folder 4-10
    renaming a connection 4-9
    renaming a folder 4-9
    setting the operation mode 4-10
    setting the port 4-4
    setting up 4-6
    starting 4-2
    starting, stopping, and removing an exception count 4-37
    status bar 4-6
    System Properties 4-22
    tabbed interface 4-5
    thread stack dump 4-31
        viewing 4-31
    Time in GC chart 4-18
    Total Heap 4-18
    Total Memory 4-19
    Used Heap 4-18
    Used Heap gauge 4-17, 4-18
    Used Memory 4-18
    Used Memory gauge 4-18
    Used Memory guage 4-17
management console

Developer mode 4-6

Memory Throughput 3-4

memory throughput 3-4, 3-6

Method Profiling

Method Profiling Information List 4-33

settings 4-35

starting and stopping 4-34

Method Templates 4-31

adding a method 4-32

creating a new template 4-34

Method Profiling Information List 4-33

removing 4-34

removing a method 4-33

N

notification action 4-23

Application alert 4-23

creating a new rule 4-23

editing a rule 4-27

E-mail 4-23

Log to file 4-23

System out 4-23

notification constraint 4-23

notification trigger 4-22

nursery 7-4, 7-5

O

object allocation 3-9

operation mode 4-6, 4-10, 4-11

Developer mode 4-10

P

Pause Time 3-4

performance optimization 7-1

persistence value log 4-13

Processor tab

CPU Load 4-20

CPU Usage 4-20

product version 2-4

R

Red Hat AS 2-11

using 2-11

S

server-side JVM 2-3

SIGHUP 2-10

Symetric Multi Processor (SMP) system, 7-5

system property, Java 2-4

T

thread stack dump

viewing 4-31

thread system

thread allocation

local 3-9

thread-local object 3-9

U

unified garbage collector 2-3

V

verbose output 2-2, 2-4

W

WebLogic Server 7-1, 7-6

Administration Console 7-4

Concurrent 7-5

Gc Algorithm 7-4

GCHandles Compaction 7-4

Generational 7-5

Incremental 7-5

Max Heap Size 7-4

Number Of Daemon Thread 7-5

Number Of Processors 7-5

Parallel 7-5

Total Garbage Collection Count 7-4

Total Number Of Threads 7-5
Total Nursery Size 7-4
configuring JRockit for 7-3
WebLogic Workshop 7-1, 7-7

X

-Xnohup 2-10