BEA WebLogic JRockit 7.0 Limited Availability for Red Hat Linux Advanced Server 2.1 - IPF

User Guide

Release 7.0 Limited Availability for Red Hat Linux Advanced Server 2.1 - IPF
November 2002

Contents

CHAPTER

1 Introduction to BEA WebLogic JRockit

Welcome to BEA WebLogic JRockit 7.0 Limited Availability for Red Hat Linux Advanced Server 2.1 - IPF, the first commercial server-side Java Virtual Machine. JRockit was developed uniquely for server-side applications and optimized for Intel architectures to ensure reliability, scalability, manageability, and flexibility for Java applications. This user guide provides instructions for using JRockit on the Window and Linux platforms.

This section contains information on the following subjects:

- What is JRockit?
- Why Should I Use JRockit?
- What Platforms Does JRockit Support?
- How to Print the Document
- Documentation Conventions
- JRockit Support

What is JRockit?

BEA WebLogic JRockit 7.0 Limited Availability for Red Hat Linux Advanced Server 2.1 - IPF is the only credible high performance Java Virtual Machine (JVM) developed uniquely for server-side applications and optimized for Intel architectures designed to ensure reliability, scalability, manageability, and flexibility for Java applications. A crucial component of the BEA WebLogic Platform, WebLogic JRockit delivers a new level of performance for Java applications deployed on any kind of hardware architecture at significantly lower costs to the enterprise. Furthermore, it is the only enterprise-class JVM optimized for Intel Architecture, providing seamless interoperability across multiple hardware and operating system configurations.

Why Should I Use JRockit?

There are two critical reasons for using BEA WebLogic JRockit:

- JRockit is designed for server-side applications.

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

- JRockit employs adaptive optimization to significantly improve runtime performance.

  Adaptive optimization is a JRockit feature that detects and removes bottlenecks in a running program, significantly improving program performance. Adaptive optimization can result in code with better performance than statically compiled code because it has access to more information regarding the running program. Another advantage is that the code does not lose any of the dynamic characteristics of interpreted languages.

What Platforms Does JRockit Support?

JRockit 7.0 Limited Availability for Red Hat Linux Advanced Server 2.1 - IPF, running with JS2E version 1.4.1, supports Red Hat Linux Advanced Server 2.1 IPF (Intel Itanium 2 servers).

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the JRockit 7.0 Limited Availability for Red Hat Linux Advanced Server 2.1 - IPF documentation Home page on the e-docs Web site. A PDF version of this document is also available in the documentation kit on the product CD. Or you can download the documentation kit from the JRockit portion of the BEA Download site. You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDF files, open the JRockit 7.0 Limited Availability for Red Hat Linux Advanced Server 2.1 - IPF documentation Home page, click the PDF files button and select the document you want to print.

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

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.

Convention	**Item**
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

Convention	Item
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

`...`	Indicates one of the following in a command line:
	■ That an argument can be repeated several times in a command line
	■ That the statement omits additional optional arguments
	■ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

JRockit Support

Should you experience any problems or find any bugs, please send us an e-mail at **support@bea.com**. We would appreciate if you could provide as much information as possible about the problem, for example:

- Hardware
- Version
- The program you ran
- Stack dumps (if any)
- A small code example that will reproduce the error.

CHAPTER

2 Understanding JRockit

This section includes information on the following subjects:

- Code Generation
- Memory Management
- Threads

Code Generation

Code generation is the process of converting Java code to assembly language. JRockit contains a very good optimizing JIT compiler with cutting-edge performance. It is unique because it doesn’t perform bytecode interpretation at all, instead everything is compiled. While this can lead to slightly longer startup times, the long-term result is faster performance for typical server-side applications.

Memory Management

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

- **Generational Stop 'n' Copy**, which divides the memory into two or more areas called “generations.” Instead of allocating objects in one single space and garbage collecting that whole space when it gets full, most of the objects are allocated in the "young generation", called the nursery.

- **Single-Spaced Concurrent**, one of two types of concurrent collectors, which does its work in parallel with ordinary processing; that is, it does not stop all Java threads to do the complete garbage collection. This is designed to support garbage collection without disruption and to improve multiprocessor garbage collection performance.
- **Generational Concurrent** is the second type of concurrent collector JRockit employs. Although very similar to a single-spaced concurrent collector, a generational concurrent garbage collector does its actual object allocation in a "nursery," reducing the need to do collection of the entire heap so often.
- **Parallel** garbage collection, which stops all Java threads and uses every CPU to perform a complete garbage collection of the entire heap.

To implement and configure a garbage collector, see Chapter 5, "Configuring the JRockit Memory Management System."

Threads

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

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

To implement and configure a Thread system, see Chapter 4, "Configuring the JRockit Thread System."

3 Configuring and Running JRockit

This section describes how to use JRockit's command line options. JRockit supports all the standard Java command line options plus numerous extended options.

This section includes information on the following subjects:

- Setting Up JRockit by Using Standard Command Line Options
- Setting Standard Output Options
- Changing JRockit Behavior for Other Java Applications by Using -X Options
- Configuring Garbage Collection and Thread Management by Using -X Options

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

Setting Up JRockit by Using Standard Command Line Options

The following are the standard command line options you can use with JRockit:

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

  Tells JRockit where to look for classes and resources.

Use a colon to separate entries on Linux. Alternately, you can use the option `-cp` to represent `-classpath`; for example:

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

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

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

Setting Standard Output Options

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 JRockit 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` and `codegen`. If no component is given, JRockit will display verbose information on everything.

- `-help`

  Tells JRockit to display a short help message.

- `-X`

  Tells JRockit to display a short help message on the extended options.

Changing JRockit Behavior for Other Java Applications by Using -X Options

The -X command line options are used to change the behavior of JRockit to better suit the needs of different Java applications. These options are non-standard and are subject to change at any time.

- `-Xnoopt`

  Tells JRockit not to optimize code.

- `-Xverify`

  Tells JRockit to do complete bytecode verification.

- `-Xquiet`

  Prevents JRockit from dumping output to `stdout` or `stderr`.

Configuring Garbage Collection and Thread Management by Using -X Options

Additional -X options, specific to garbage collection and threading, are described in Chapter 4, “Configuring the JRockit Thread System” and Chapter 5, “Configuring the JRockit Memory Management System.”

CHAPTER

4 Configuring the JRockit Thread System

JRockit supports two types of thread systems:

- Native Threads
- Thin Threads

These systems allow you to configure JRockit to take optimal advantage of the resources upon which you are running it. This section describes how to select and configure these thread systems. It includes information on the following subjects:

- Native Threads
- Thin Threads
- Choosing a Thread System
- Implementing a Thread System
- Configuring the Thread System

Native Threads

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

Thin Threads

Called the BEA JRockit High Performance Threading System, or Thin Threads, these Java threads are more specific and demand fewer resources than an operating system thread. Thus, using operating system threads to run Java threads can be a waste of resources.

Thin threads are independent of platform-specific (native) threads. They are implemented in JRockit and are not part of the operating system; that is, several threads are simulated by one (or more) native threads that represent the running program. Context switches and scheduling is done internally within JRockit, and is therefore much lighter-weight than context switching done within native threads by the OS.

In the High Performance Threading System, several Java threads are run in the same operating system thread. This allows JRockit to perform optimized thread scheduling, thread switching, and thread synchronization with less memory. This makes it possible to run a drastically higher number of threads at a higher speed than with any other Java VM.

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

Choosing a Thread System

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

Pros and Cons

Table 4-1 lists the pros and cons of each thread system.

Table 4-1 Thread System Pros and Cons

Thread System	Pros	Cons
Native Threads	■ Takes advantage of the operating system's thread scheduling and load balancing policies. ■ Are standard, giving you an application with native code, which relies upon the fact that each Java thread is mapped on to a operating system thread of its own, this is the only model that works (both DB2 and Oracle level 2 JDBC database drivers have been known to rely upon this). ■ On a multiprocessor system when the application has few active threads, the operating system scheduling system is better at utilizing the CPUs efficiently.	■ Context switching is more costly as it has to be done in the operating system instead of only in the JVM. ■ Every Java thread consumes more resources, because it requires an operating system thread of its own.
Thin Threads	■ Since several Java threads are run in the same operating system thread, JRockit can perform optimized thread scheduling, thread switching, and thread synchronization with less memory. ■ A variety of operating system threads can be used, splitting the Java threads among them. A Java thread is not bound to a specific operating system thread, so it can move between them to allow for optimal load balancing.	

Threading System Selection Matrix

Use the **If...** column in Table 4-2 to locate a condition that matches your JRockit implementation and select the thread system indicated in the **Select this Method...** column.

Table 4-2 Threading System Selection Matrix

If...	Select this Method...
You have in excess of a couple hundred threads	Thin Threads
You are running Linux on a single-CPU system	Thin Thread. This is because Linux threads are very expensive to use.

Implementing a Thread System

To implement a thread system, include one of the options listed in Table 4-3 when you start JRockit:

Table 4-3 Options Used to Implement Threading

To Use...	Include this Option...
Native Threads	`-Xnativethreads` This option is the default, therefore you only need to specify it when you want to change the method from thin threads.
Thin Threads	`-Xthinthreads` This option is not available on IA64.

Configuring the Thread System

To provide the optimal out-of-the-box experience with JRockit comes with default values that adapt automatically to the specific platform on which you are running JRockit. You can modify these values to configure your thread system by specifying at the command line any of the options listed in this section.

Setting the Type of Thread Allocation

```
-Xallocationtype:<global|local>
```

`-Xallocationtype` sets the type of thread allocation. The allocation type local is recommended for the vast majority of applications. However, if the maximum heap size is very small (less then 128 MB) or if the number of threads used by the application is very high (several hundred) the allocation type global might work better, particularly on single CPU systems. The reason for this is that every thread-local area consumes a fixed amount of memory (approximately 2 kilobytes). If the number of threads is very high or the heap size very small when using thread-local allocation the potential waste of space could cause excess fragmentation of the heap. This leads to more frequent garbage collections and may cause the application to run out of memory prematurely.

Setting the Thread Stack Size

```
-Xss<size>[k|K]
```

`-Xss<size>[k|K]` sets the thread stack size in kilobytes.

In addition to setting the thread stack size, if the number of threads is high, you should use `-Xallocationtype:global`, as suggested in “Setting the Type of Thread Allocation” to reduce heap fragmentation.

CHAPTER

5 Configuring the JRockit Memory Management System

Have you ever seen strange pauses in your application that you haven’t been able to explain? Have you seen one or all CPUs pegged on 100% utilization and all the others on 0% and still very few transactions in your system? If you answered yes to either of these two questions, your application might have been suffering from the effects of a poorly performing garbage collector. Some fairly simple tuning of the memory management system can improve performance dramatically for many applications.

This section includes information on the following subjects:

- Memory Management Terminology
- JRockit Garbage Collectors
- Choosing a Garbage Collection Method
- Implementing a Garbage Collector
- Configuring the Garbage Collector
- System Defaults

Memory Management Terminology

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

Thread-local allocation

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

Pause time

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

Memory throughput

Memory throughput measures the time it takes between when an object is no longer referenced and the time it gets 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.

JRockit Garbage Collectors

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

- Generational Copying Garbage Collectors
- Concurrent Garbage Collectors
- Parallel Garbage Collectors

Generational Copying Garbage Collectors

The first type of JRockit garbage collector is the generational copying garbage collector (`-Xgc:gencopy`).

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

A generational copying garbage collector is specifically designed as a lightweight alternative for use on single CPU systems with a small (less then 128 mB) heap. It is suitable for testing applications on your desktop machine; however for a deployment environment another garbage collector would in most cases be more efficient.

Concurrent Garbage Collectors

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

JRockit can employ two types of concurrent garbage collectors:

- Single Spaced Concurrent
- Generational Concurrent

Single Spaced Concurrent

The second type of JRockit garbage collector is the single spaced concurrent garbage collector (`-Xgc:singlecon`). What is unique about the concurrent garbage collectors is that they remove garbage collection pauses completely. Using these garbage collectors, the heaps can be gigabyte-size and there will be no long pauses. However, keep in mind that concurrent garbage collectors trade memory throughput for reduced pause time. It takes longer between the time the object is referenced the last time and the system detects and reclaims it; in other words it takes longer for the object to die. The natural consequence of this is that you will most likely need a larger heap with a concurrent garbage collector than you need with any other. In addition, if your ordinary Java threads create more garbage than the concurrent garbage collector manages to collect, there will be pauses while the Java threads are waiting for the concurrent garbage collector to complete its cycle.

Generational Concurrent

The third type of JRockit garbage collector is the generational concurrent garbage collector (`-Xgc:gencon`). In this garbage collector, objects are allocated in the young generation. When the young generation (called a nursery) is full, JRockit "stops-the-world" and moves the objects that are still live in the young generation to the old generation. An old collector thread runs in the background all the time; it marks objects in the old space as live and removes the dead objects, returning them to JRockit as free space.

The advantage of the generational concurrent garbage collector compared to the single spaced concurrent garbage collector is that it has a higher memory throughput.

Parallel Garbage Collectors

The fourth type of JRockit garbage collector is the parallel garbage collector (`-Xgc:parallel`). When the heap is full all Java threads are stopped and every CPU is used to perform a complete garbage collection of the entire heap. This behavior causes longer pause times than for the concurrent collectors but maximizes memory throughput. On a machine with four CPUs or better this is the recommended garbage collector, provided that your application can tolerate the slightly longer pause times.

Choosing a Garbage Collection Method

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

Pros and Cons

Table 5-1 lists the pros and cons of each garbage collector.

Table 5-1 Garbage Collector Pros and Cons

Garbage Collector	Pros	Cons
Generational Copying	■ Works well with single CPU systems with a heap smaller than 128mB. ■ Good for testing on a single machine.	■ Not effective with large (>128mB) heaps. ■ Not recommended in a deployment environment.
Single Spaced Concurrent	■ Virtually removes all pauses. ■ Can handle gigabyte-sized heaps.	■ 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.
Generational Concurrent	■ Virtually removes all pauses. ■ Has a higher memory throughput than single spaced concurrent garbage collector	■ 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.

Table 5-1 Garbage Collector Pros and Cons

Garbage Collector	Pros	Cons
Parallel	■ Maximizes memory throughput.	■ "Stop the world" might cause a longer than desirable pause in processing.

Garbage Collector Selection Matrix

Table 5-2 is a matrix that you can use to determine which garbage collector is right for your JRockit implementation. Use the **If...** column to locate a condition that matches your JRockit implementation and select the garbage collector indicated in the **Select this Method...** column.

Table 5-2 Garbage Collector Selection Matrix

If You...	Select this Garbage Collector...
■ Want lightweight alternative for use on single CPU systems with a small (less then 128 mB) heap. ■ Are testing applications on your desktop machine.	Generational Copying
■ Cannot tolerate pauses of any length. ■ Employ gigabyte-sized heaps. ■ Willing to trade memory thoughput for eliminating pauses.	Single Spaced Concurrent
■ 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	Generational Concurrent
■ Using a machine with four CPUs or better. ■ Can tolerate the occasional long pause ■ Need to maximize memory throughput	Parallel

Implementing a Garbage Collector

To implement a garbage collector, simply enter at the command line the option for the desired collector. Table 5-3 lists these options:

Table 5-3 Garbage Collector Implementation Options

Garbage Collector	Option
Generational Copying	`-Xgc:gencopy`
Single Spaced Concurrent	`-Xgc:singlecon`
Generational Concurrent	`-Xgc:gencon`
Parallel	`-Xgc:parallel`

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

Configuring the Garbage Collector

To provide the optimal out-of-the-box experience with JRockit comes with default values that adapt automatically to the specific platform on which you are running JRockit (see “System Defaults” on page 9). You can modify these values to tune your garbage collector by specifying at the command line any of the options listed in this section.

Printing a Comprehensive Report

```
-Xgcreport
```

`-Xgcreport` causes JRockit to print a comprehensive garbage collection report at program completion. The option `-Xgcpause` causes JRockit to print a line each time Java threads are stopped for garbage collection. Combining the two is a very good way of examining the memory behavior of your application.

Setting the Initial and Maximum Heap Size

```
-Xmx:<size>/-Xms<size>
```

`-Xmx` sets the maximum size of the heap. The general recommendation is to set this as high as possible, but not so high that it causes page-faults for the application or for some other application on the same computer. Set it to something less than the amount of memory in the machine. If you have multiple applications running on the computer at the same time the value could be much lower. The general recommendation is to set the initial heap size (`-Xms`) to the same size as the maximum heap size.

Setting the Size of the Nursery

```
-Xns:<size>
```

`-Xns` sets the size of the young generation (nursery). If you are creating a lot of temporary objects you should have a large nursery. Generally, the larger you can make the nursery while keeping the GC-pause times acceptably low, the better. You can see the nursery pause times in JRockit by starting the JVM with `-Xgcpause`, but you have to decide yourself what is an acceptable GC pause time before your system becomes unresponsive.

Setting the Type of Thread Allocation

```
-Xallocationtype:<global|local>
```

`-Xallocationtype` sets the type of thread allocation. The allocation type `local` is recommended for the vast majority of applications. However, if the maximum heap size is very small (less then 128 MB) or if the number of threads used by the application is very high (several hundred) the allocation type `global` might work better, particularly on single CPU systems. The reason for this is that every thread-local area consumes a fixed amount of memory (approximately 2 kilobytes). If the number of threads is very high or the heap size very small when using thread-local allocation the potential waste of space could cause excess fragmentation of the heap. This leads to more frequent garbage collections and may cause the application to run out of memory prematurely.

Defining When a Memory Space will be Cleared

```
-Xcleartype:<gc|local|alloc>
```

`-Xcleartype` defines when the memory space occupied by an object that has been garbage collected will be cleared. It can be done during the garbage collection (`gc`), when a thread-local area is allocated (`local`) or when that space is allocated for a new object (`alloc`). You should use `local` or `alloc`. `alloc` might work better if the objects allocated are very large (1 to 2 kilobytes).

Note these additional conditions:

- The `-Xcleartype:local` option is available only if the `-Xallocationtype` is set to `local`.
- The `alloc` parameter is not available on IA64 systems.

Printing the Pause Times

```
-Xgcpause
```

`-Xgcpause` prints pause times caused by the garbage collector.

System Defaults

This section describes the default values for the JRockit memory management system. To provide the best out-of-the-box performance possible these defaults adapt automatically to the specific platform on which JRockit is running.

Heap Size

If the initial heap size (`-Xms`) is not set, the default initial heap size depends on the maximum heap size (`-Xmx`) value and on the amount of available physical memory. If the maximum heap size or the available physical memory is less than 128 MB the initial heap size will be 16 MB. If the maximum heap size is larger than 128 MB the initial heap size will be 75% of the available physical memory up to 64 MB.

Garbage Collector

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

Nursery Size

If the nursery size (`-Xns`) has not been set the default size depends on the number of CPUs. For the generational copying (`gencopy`) garbage collector the default nursery size is 640 KB times the number of CPUs and for the generational concurrent (`gencon`) garbage collector the default nursery size is 10 MB times the number of CPUs.

Thread Stack Size

If the thread stack size (`-Xss`) has not been set the default value depends on the threading system and the platform you are running on. When using thin threads the minimum thread stack size is 8 kilobytes and the default is 64 kilobytes. When using native threads the minimum thread stack size is 16 kilobytes. For Linux the default thread stack size when using native threads is 128 kilobytes.

Note: If `-Xss` is set below the minimum value, thread stack size will default to the minimum value automatically.

Allocation Type

If the allocation type (`-Xallocationtype`) is not set, the default is `global` for the generational copying (`gencopy`) garbage collector and `local` for all others (`singlecon`, `gencon`, and `parallel`).

Clear Type

If the clear type (`-Xcleartype`) is not set the default is `alloc` on IA32 systems and `gc` on IA64 systems.

Note: The option `alloc` is not available on IA64 systems.

CHAPTER

6 Using the JRockit Management Console

The JRockit Management Console can be used to supervise and monitor running instances of JRockit. 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
- Starting the Console
- Parts of the Console
- Setting Up the Console
- Using the Console

Console Overhead

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

Starting the Console

Starting the JRockit 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 JRockit, the management server in the virtual machine needs to be started. The server is disabled by default. To enable the management server, start JRockit set the `-Xmanagement` option:

```
-Xmanagement
```

Start the JRockit Management Console

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

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

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

Set the Port

When JRockit is started with the `-Xmanagement` option set—and provided JRockit 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 the port in the `port` property:

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

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

Change the Number of Connections

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

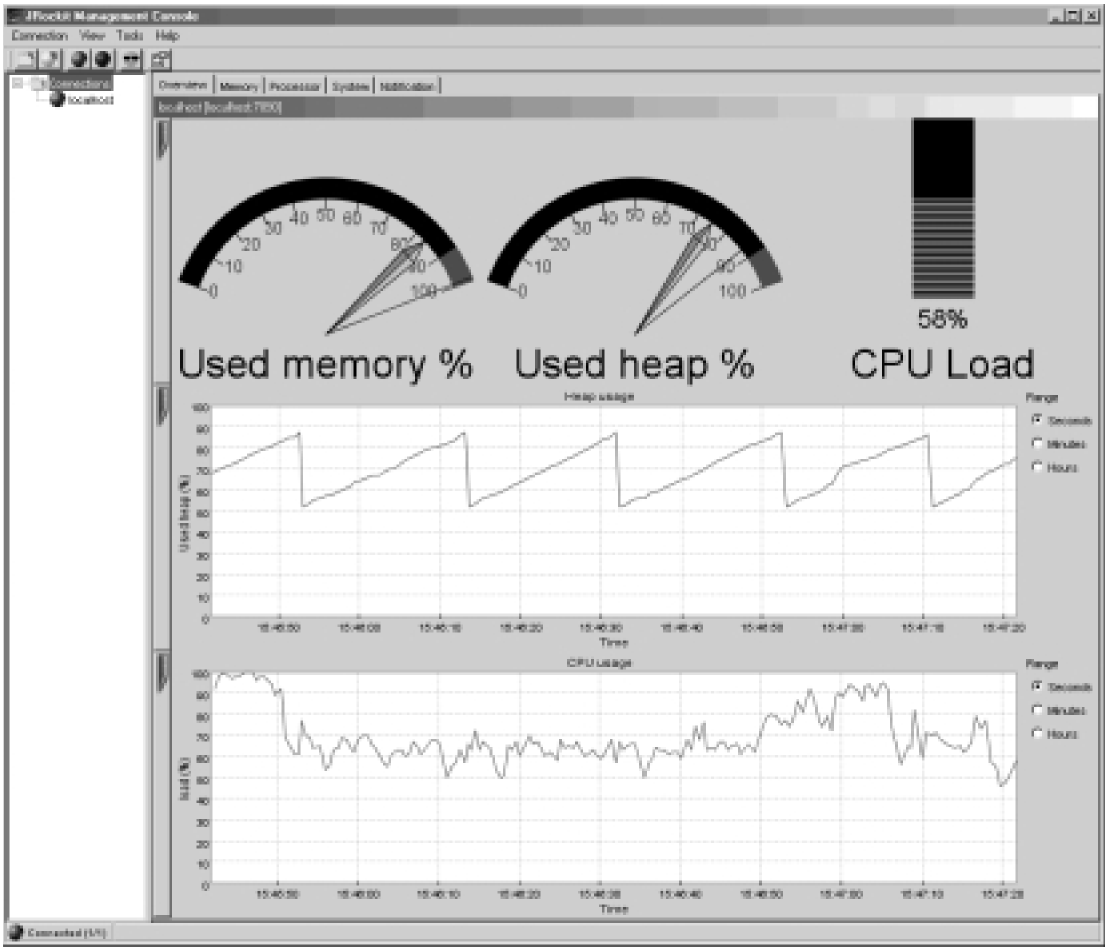
```
-Djrockit.managementserver.maxconnect=<maximum number of
connections>
```

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

Parts of the Console

When the JRockit Management Console window appears, the console has started, as shown in Figure 6-1:

Figure 6-1 JRockit Management Console



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

Figure 6-2 Connection Browser

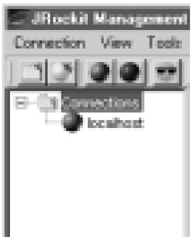
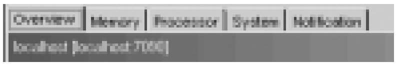


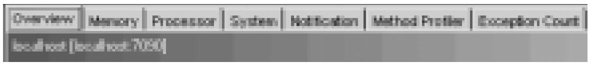
Figure 6-3 Information Tabs (Administrator Mode)



The first tab shows an Overview of information for the selected JRockit connection(s) (as highlighted in the connection browser pane). The other tabs contain detailed information about different areas of JRockit, as will be described in "Information Tabs" on page 16.

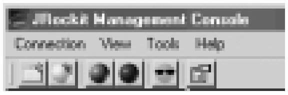
Figure 6-3 shows the information tabs available in the console's Administrator operation mode. When the console is in the Developer mode, additional tabs appear, as shown in Figure 6-4. These two operation modes are described in "Setting the Operation Mode" on page 10.

Figure 6-4 Information Tabs (Developer Mode)



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

Figure 6-5 Management Console Toolbar



The status bar (Figure 6-6) at the bottom of the window displays information 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 JRockit implementations or not. To toggle the Status Bar on or off, on the View menu, select Status Bar.

Figure 6-6 Status Bar

Connected (1/1) Add a new connection in the selected folder

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 JRockit 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 JRockit are indicated by a green icon; those disconnected are indicated by a red icon.

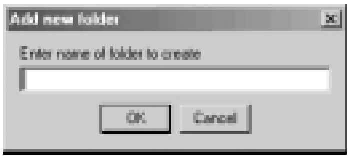
Creating a New Folder

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

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

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

Figure 6-7 Add New Folder Dialog Box



Enter name of folder to create

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

   The new folder will appear in the connection browser.

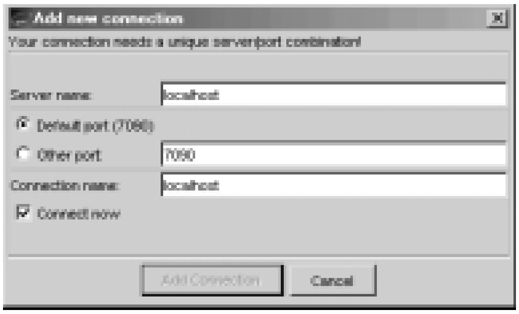
Creating a New Connection

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

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

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

Figure 6-8 Add New Connection Dialog Box



Your connection needs a unique server/port combination!

Server name: localhost

(●) Default port (7090)

( ) Other port: 7090

Connection name: localhost

[✓] Connect now

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 JRockit

To connect to JRockit, do the following:

1. Select the JRockit 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 JRockit

To disconnect a connection from JRockit, do the following:

1. Select the JRockit 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

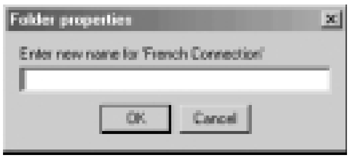
To rename a connection or a folder of connection, do the following:

1. Select the JRockit connection or folder to rename.

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

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

Figure 6-9 Folder 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.

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 JRockit 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:

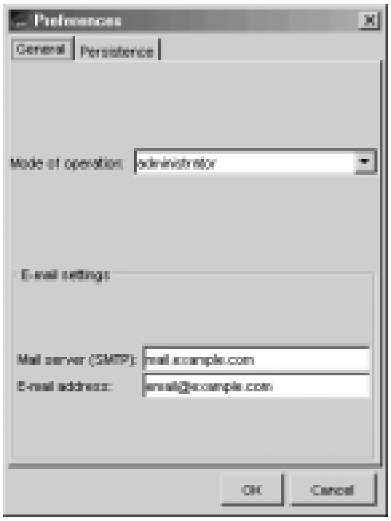
- **Administrator Mode**; This is the default mode, designed for system administrators who are interested in observing the state of the JRockit system.
- **Developer Mode**; The developer mode is for developers and provides additional features such as a rudimentary method profiler and exception count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

To set the operation mode, do the following:

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

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

Figure 6-10 Preferences Menu (General Tab)



Mode of operation: administrator

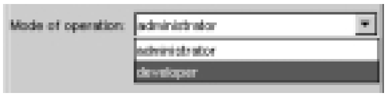
E-mail settings

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 6-11).

Figure 6-11 List 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 6-3 and Figure 6-4 for examples.

Setting Other Preferences

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

- Default e-mail settings for the notification system
- Persistence behavior.

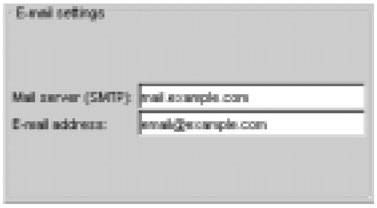
To change either of these values, open the Preferences dialog box from the Tools menu and proceed are described in the following sections:

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 6-12.
3. Click OK

Figure 6-12 E-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” on page 29).

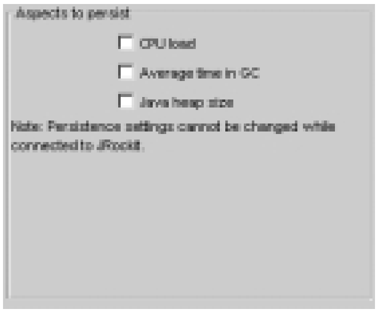
Selecting Aspects to Persist To set persistence preferences, do the following:

1. Disconnect any JRockit connections.

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

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

Figure 6-13 Aspects to Persist Panel



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 29.

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

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

   If you are still connected to JRockit, 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.

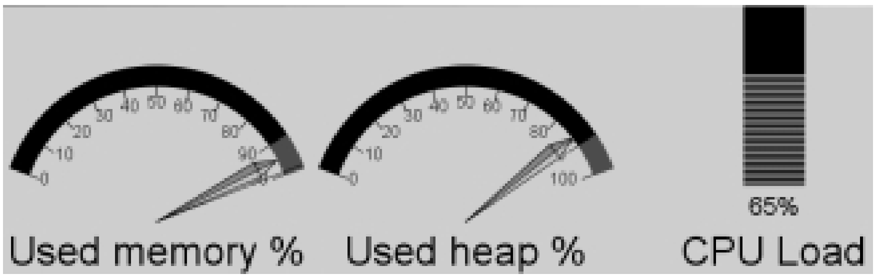
Customizing the Display

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

Customizing Gauges and Bars

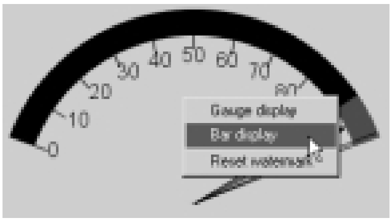
The gauges and bars are graphical devices showing memory and processor usage (Figure 6-14).

Figure 6-14 Gauges and Bars



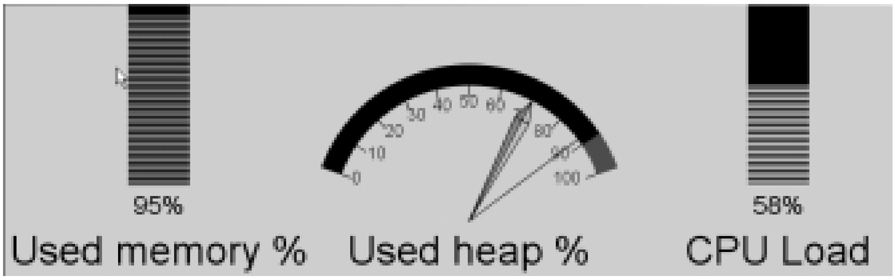
- To change from a gauge display to bar display, press the right mouse button when pointing at the gauge and select Bar display, as shown in Figure 6-15.

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



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

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



- To change back to a gauge, repeat the above, but select Gauge display.

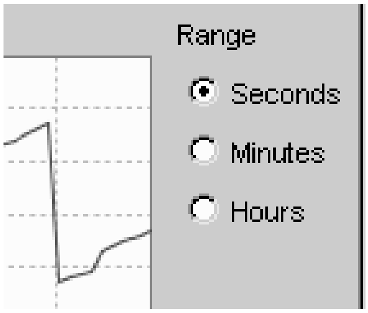
- To reset the watermark—which indicates the highest level measured so far—press the right mouse button when pointing at the gauge or bar and select Reset Watermark.

Customizing Charts

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

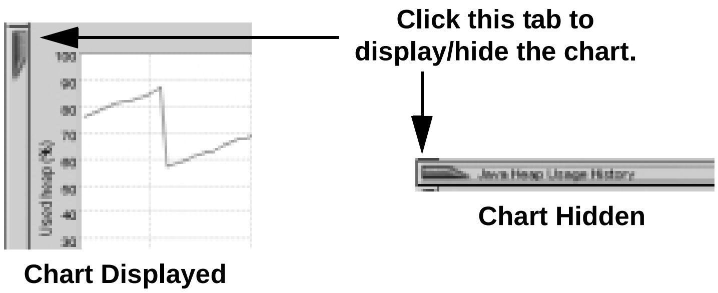
- To change scale on any of the chart, select the desired scale unit (seconds, minutes or hours) to the right of the chart (Figure 6-17) to be changed.

Figure 6-17 Range Selection Radio Buttons



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

Figure 6-18 Hiding a Chart



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

Using the Settings File

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

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

The exact path to the user home directory will vary on different platforms.

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 JRockit. An aspect is data that can be measured; for example, used heap size or JRockit uptime.

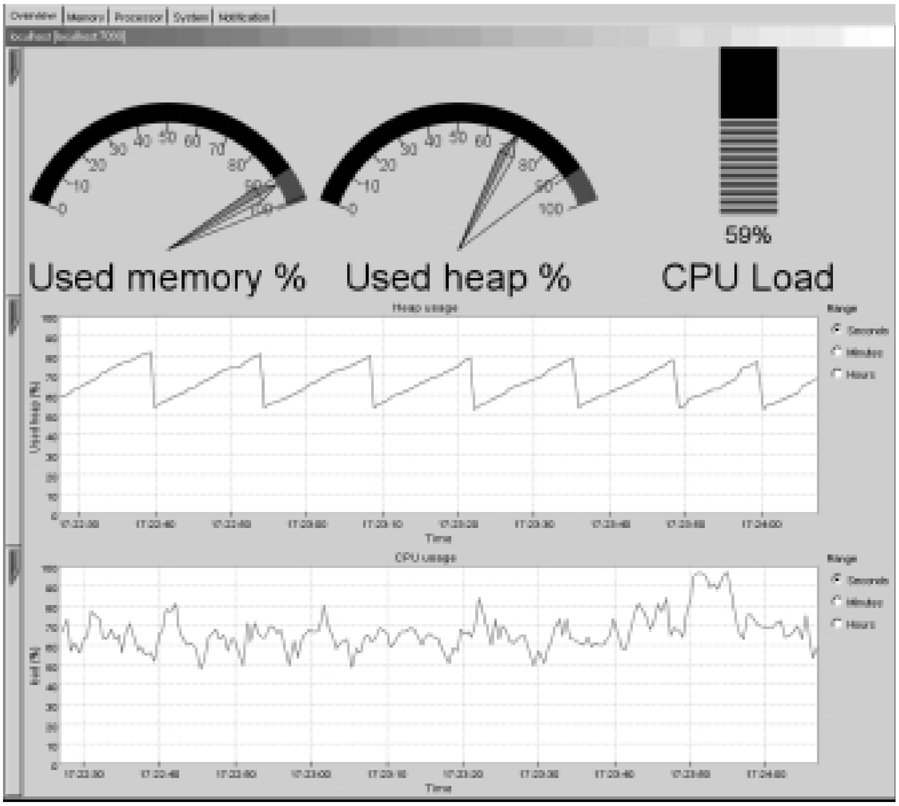
Information Tabs

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

Overview Tab

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

Figure 6-19 Overview Tab

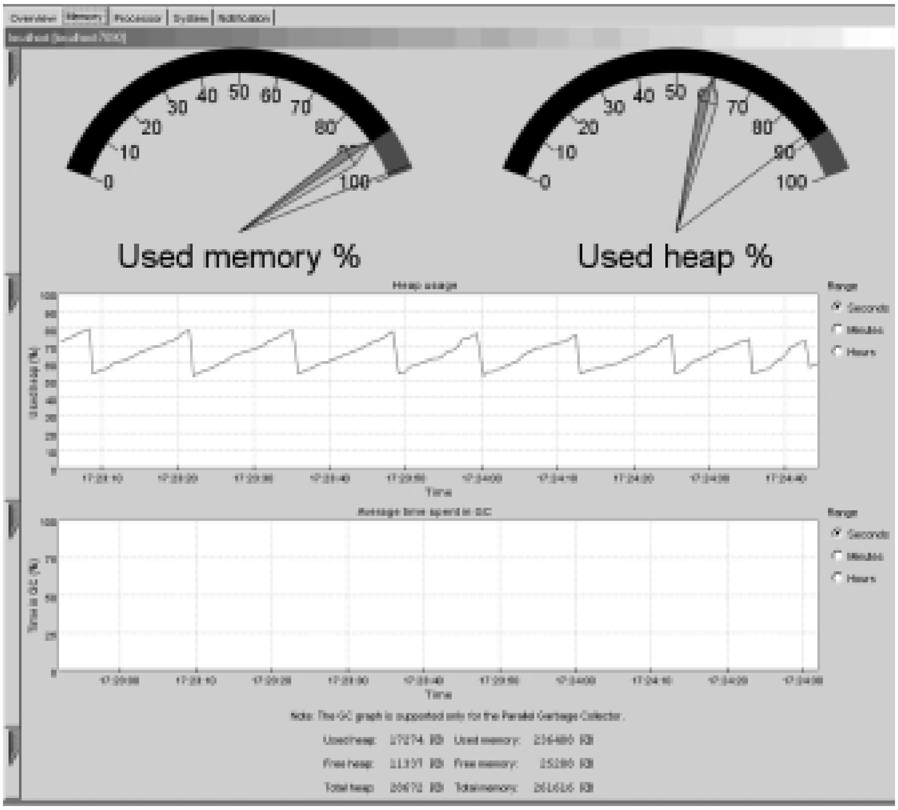


- The **Used Memory** gauge shows the percentage of occupied physical memory on the computer.
- The **Used Heap** gauge shows the percentage of occupied Java heap memory in the virtual machine.
- The **CPU Load** bar shows the occupancy rate of the processor - or the average processor load on a multi-processor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

Memory Tab

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

Figure 6-20 Memory Tab



Note: The GC graph is supported only for the Parallel Garbage Collector.

```
Used heap:   17274 KB   Used memory:  236480 KB
Free heap:   11397 KB   Free memory:   25280 KB
Total heap:  28672 KB   Total memory: 261616 KB
```

- The **Used Memory** gauge shows the percentage of occupied memory.
- 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 JRockit with the Parallel garbage collector, and an actual garbage collection occurs.

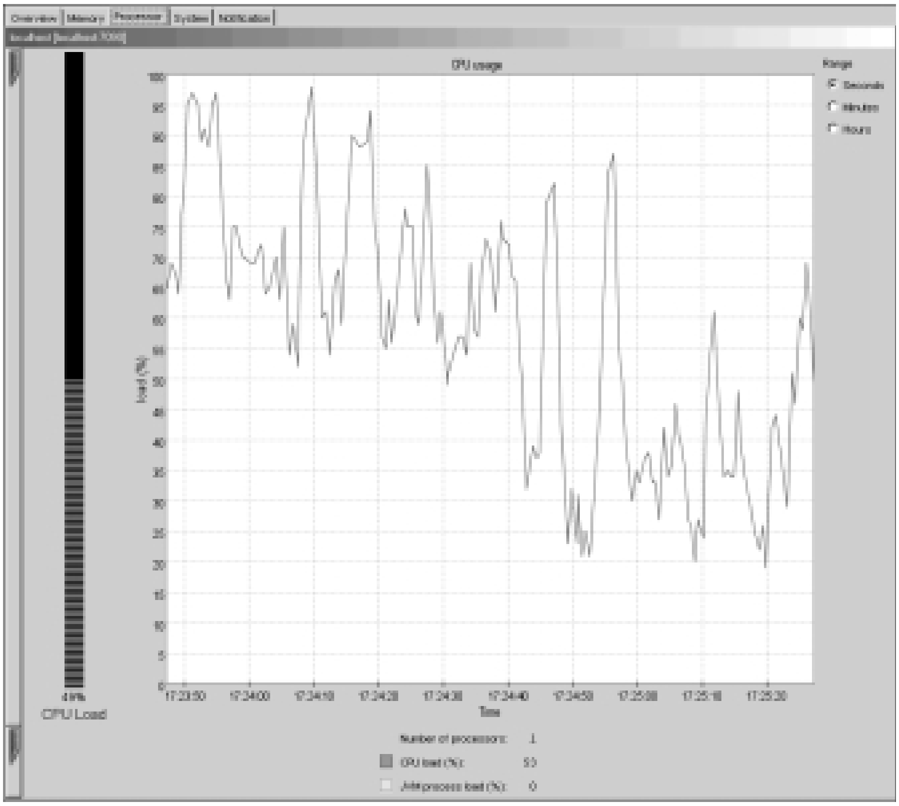
At the bottom of the page the following text information is displayed (in kilobytes):

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

Processor Tab

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

Figure 6-21 Processor Tab



Number of processors: 1
▨ CPU load (%): 53
□ JVM process load (%): 0

- 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.

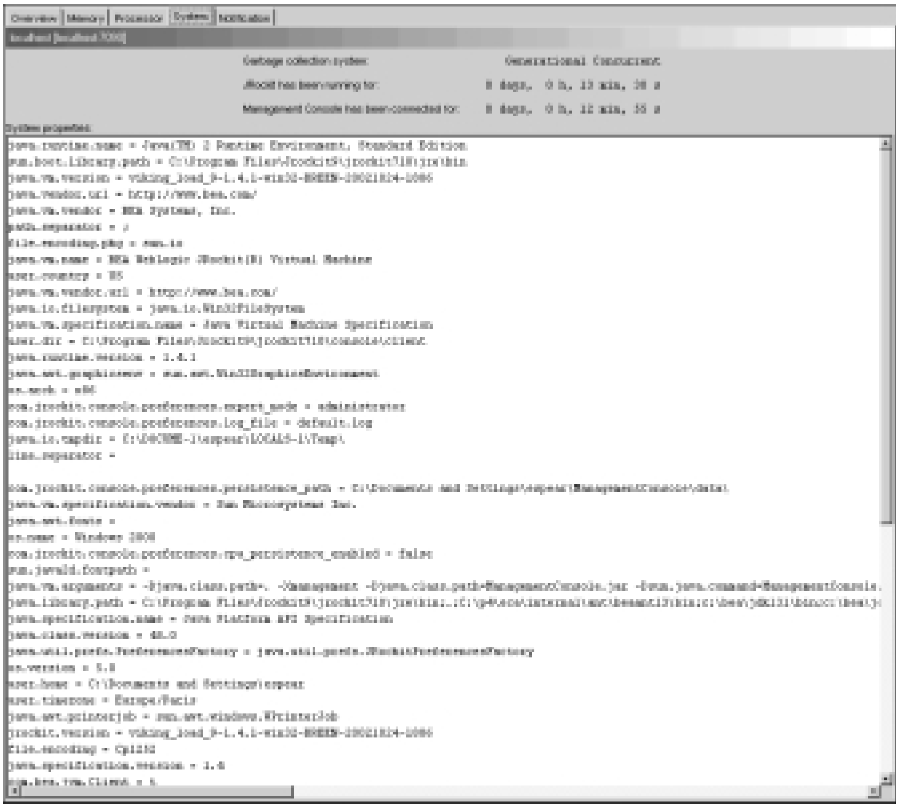
At the bottom of the page the following text information is displayed:

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

System Tab

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

Figure 6-22 System Tab



```
Garbage collection system:                       Generational Concurrent
JRockit has been running for:                    0 days,  0 h, 13 min, 38 s
Management Console has been connected for:       0 days,  0 h, 12 min, 55 s
System properties:
java.runtime.name = Java(TM) 2 Runtime Environment, Standard Edition
sun.boot.library.path = C:\Program Files\JRockit\jrockit71\jre\bin
java.vm.version = viking_load_9-1.4.1-win32-BREEN-20021024-1086
java.vendor.url = http://www.bea.com/
java.vm.vendor = BEA Systems, Inc.
path.separator = ;
file.encoding.pkg = sun.io
java.vm.name = BEA WebLogic JRockit(R) Virtual Machine
user.country = US
java.vm.vendor.url = http://www.bea.com/
java.io.filesystem = java.io.Win32FileSystem
java.vm.specification.name = Java Virtual Machine Specification
user.dir = C:\Program Files\JRockit\jrockit71\console\client
java.runtime.version = 1.4.1
java.awt.graphicsenv = sun.awt.Win32GraphicsEnvironment
os.arch = x86
com.jrockit.console.preferences.expert_mode = administrator
com.jrockit.console.preferences.log_file = default.log
java.io.tmpdir = C:\DOCUME~1\aspear\LOCALS~1\Temp\
line.separator =

com.jrockit.console.preferences.persistence_path = C:\Documents and Settings\aspear\ManagementConsole\data\
java.vm.specification.vendor = Sun Microsystems Inc.
java.awt.fonts =
os.name = Windows 2000
com.jrockit.console.preferences.rpe_persistence_enabled = false
sun.jnu.fontpath =
java.vm.arguments = -Djava.class.path=. -Xmanagement -Djava.class.path=ManagementConsole.jar -Dsun.java.command=ManagementConsole.
java.library.path = C:\Program Files\JRockit\jrockit71\jre\bin;.;C:\p4\core\internal\mc\bea\bin;c:\bea\jdk131\bin;c:\bea\j
java.specification.name = Java Platform API Specification
java.class.version = 48.0
java.util.prefs.PreferencesFactory = java.util.prefs.JRockitPreferencesFactory
os.version = 5.0
user.home = C:\Documents and Settings\aspear
user.timezone = Europe/Paris
java.awt.printerjob = sun.awt.windows.WPrinterJob
jrockit.version = viking_load_9-1.4.1-win32-BREEN-20021024-1086
file.encoding = Cp1252
java.specification.version = 1.4
com.bea.jvm.Client = 1
```

- **Garbage Collection System** shows which garbage collector JRockit is running.
- **JRockit Uptime** shows how long JRockit has been running.
- **Connection Uptime** shows how long the currently displayed connection has been connected.
- **Process Affinity** contains buttons that correspond to processors. It displays a green icon if JRockit is running on this processor and a red icon if it is not. By selecting a button, the JRockit process can be bound to one or more processors.

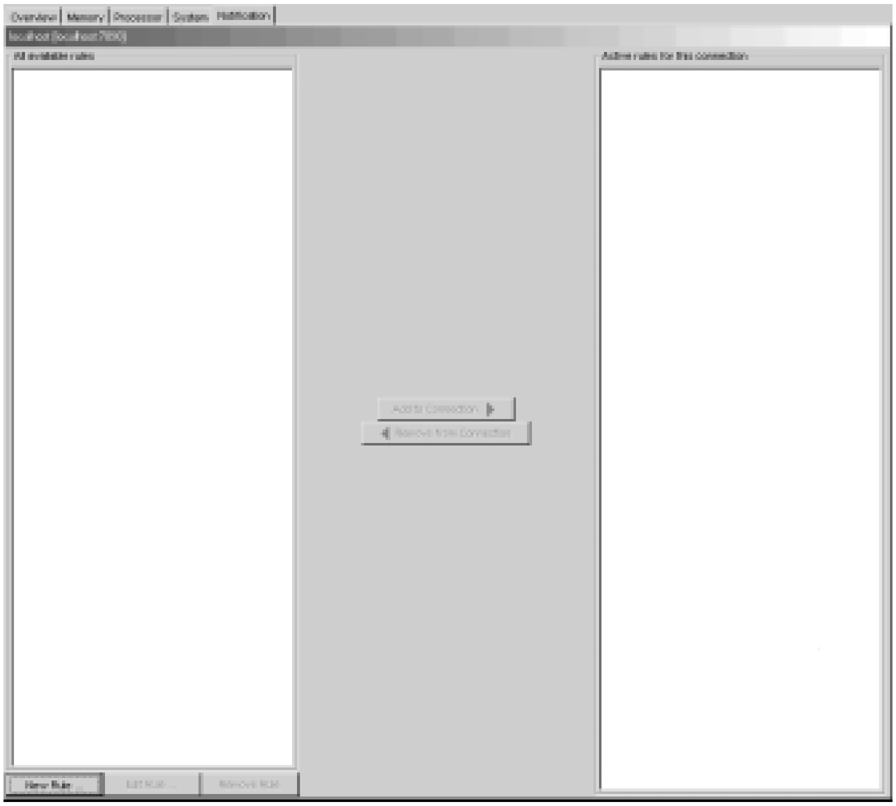
JRockit might be released from such a connection by deselecting the button again. This is only a suggested affinity: the operating system might not follow the suggestion. The Process Affinity display is only activated when the Management Console is in the Developer mode, described in "Setting the Operation Mode" on page 10.

- **System Properties** shows the Java System Properties loaded in JRockit.

Notification Tab

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

Figure 6-23 Notification Tab (No Rules Defined)



A notification trigger can be a certain event, for example, that the connection to JRockit 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 with the notification is sent to the specified address by using the specified SMTP server.
- System out action displays the notification in the command window where you started the JRockit Management Console.

- Application alert displays the notification in an alert dialog in the Management Console.
- Log to file logs the notification to the specified file.

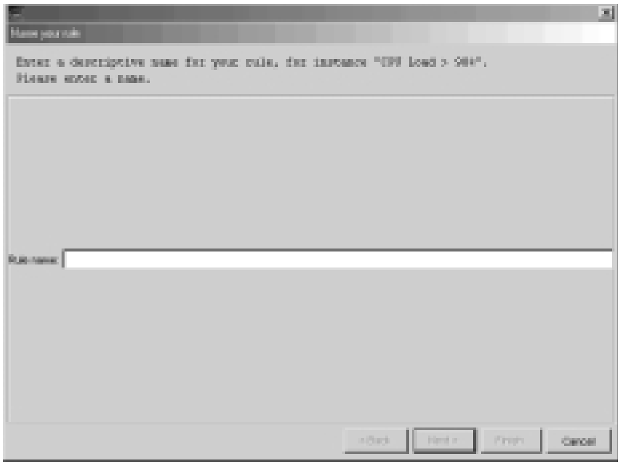
Creating a New Rule

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

1. Click New Rule.

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

Figure 6-24 Name Rule Dialog Box

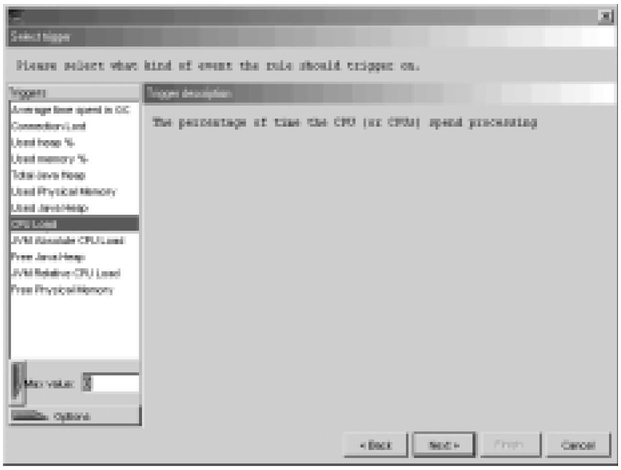


Enter a descriptive name for your rule, for instance "CPU load > 90%".
Please enter a name.

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

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

Figure 6-25 Select Trigger Dialog Box



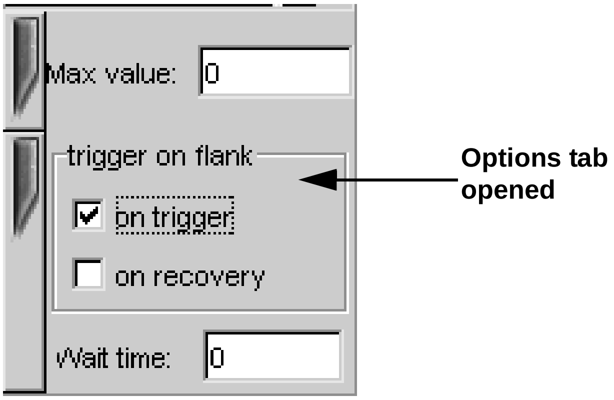
Please select what kind of event the rule should trigger on.

Average time spent in GC
Connection Load
Used heap %
Used memory %
Total Java Heap
Used Physical Memory
Used Java Heap
CPU Load
JVM Absolute CPU Load
Free Java Heap
JVM Relative CPU Load
Free Physical Memory

The percentage of time the CPU (or CPUs) spend processing

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

Figure 6-26 Trigger Threshold and Options Text Boxes



Max value: 0

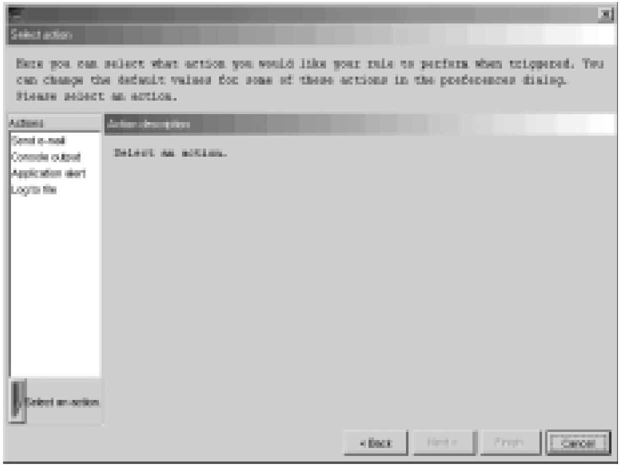
trigger on flank
[✓] on trigger
[ ] on recovery

Wait time: 0

5. Click Next.

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

Figure 6-27 Select Action Dialog Box



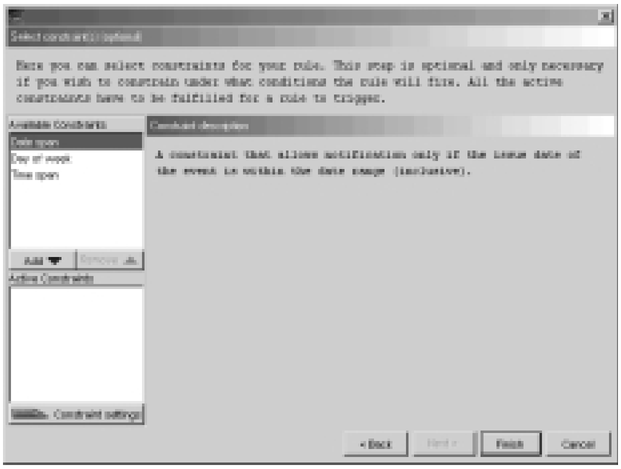
Here you can select what action you would like your rule to perform when triggered. You can change the default values for some of these actions in the preferences dialog. Please select an action.

Select an action.

6. Select an action and enter settings data, if required.
7. 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 Add Constraint dialog box appears (Figure 6-28):

Figure 6-28 Add Constraint Dialog Box



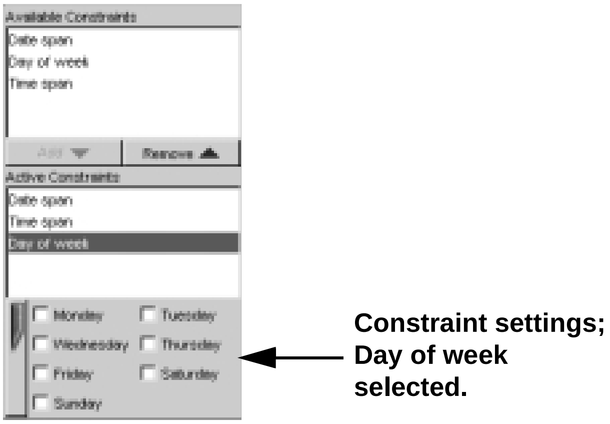
Here you can select constraints for your rule. This step is optional and only necessary if you wish to constrain under what conditions the rule will fire. All the active constraints have to be fulfilled for a rule to trigger.

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 6-29.

Figure 6-29 Constraint Added

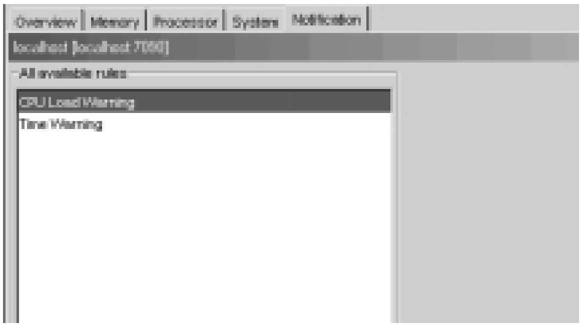


c. Enter constraint settings in the text fields under the list of constraints (Figure 6-29).

8. Click Finish.

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

Figure 6-30 New Rule in List



9. Add the rule to your connection as described in “Add a Rule to JRockit” on page 28

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.

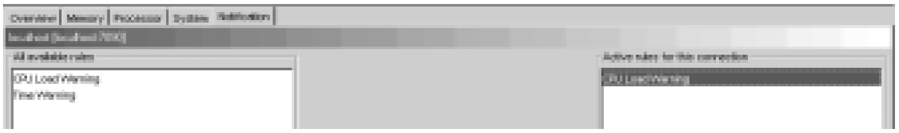
Add a Rule to JRockit

To add a rule to JRockit, do the following:

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

   The rule appears in the Active rules for this connection list, as shown in Figure 6-31.

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



Remove a Rule from JRockit

To remove a rule from JRockit, 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.

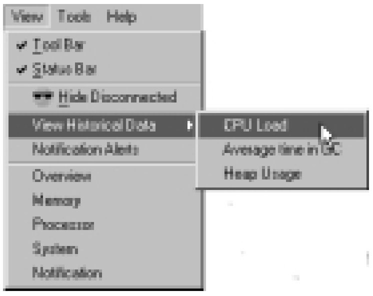
View Historical Data

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 JRockit has its peak loads.

To open this window, do the following:

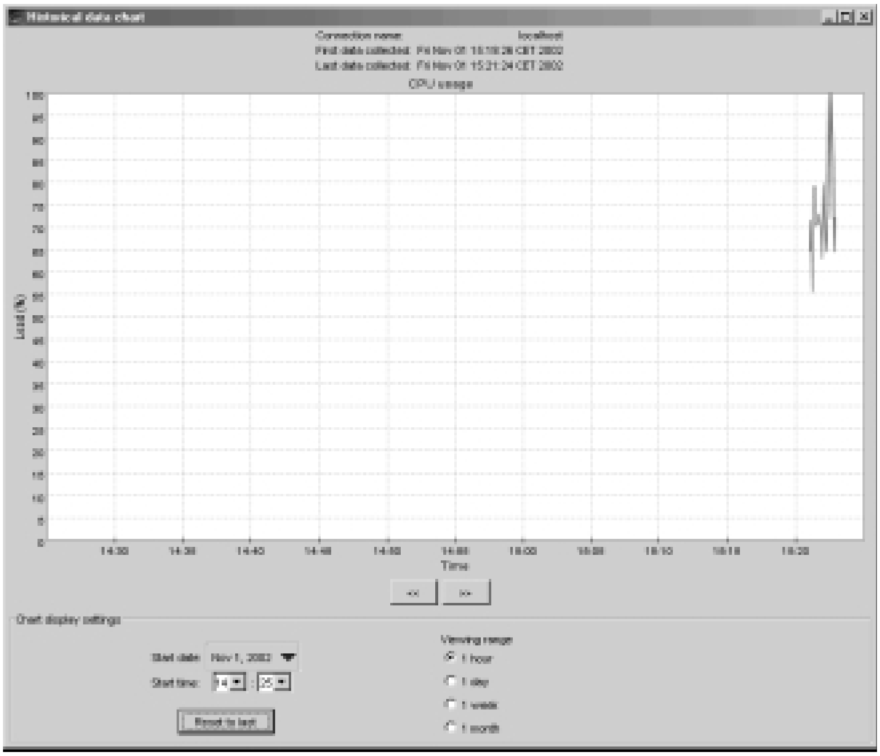
1. Select the connection for which you want to view data.
2. Open the View menu and select View Historical Data.
3. Select the aspect for which you want to view historical data, as shown in Figure 6-32.

Figure 6-32 View Menu with Historical Data Submenu Open



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

Figure 6-33 Historical Data (CPU Load Selected)



Connection name: localhost
First data collected: Fri Nov 01 15:18:28 CET 2002
Last data collected: Fri Nov 01 15:21:24 CET 2002

Chart display settings

Start date: Nov 1, 2002
Start time: 14 : 25

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 JRockit must first have been persisted, that is, written to file. See “Setting Other Preferences” on page 11 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

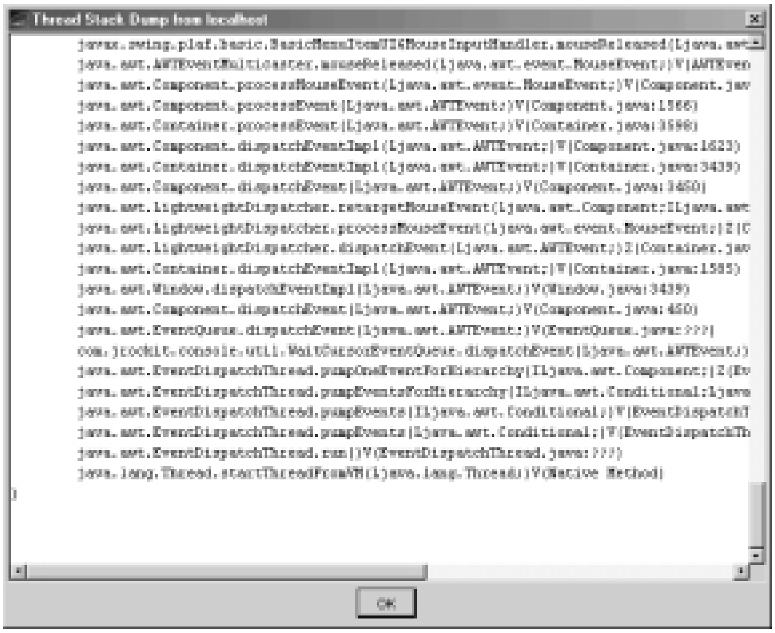
This section describes the more advanced features of the Management Console. Some of these are only available when running in the Developer mode, described in "Setting the Operation Mode" on page 10.

View Thread Stack Dump

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

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

Figure 6-34 Thread Stack Dump



```
        javax.swing.plaf.basic.BasicMenuItemUI$MouseInputHandler.mouseReleased(Ljava.awt
        java.awt.AWTEventMulticaster.mouseReleased(Ljava.awt.event.MouseEvent;)V(AWTEven
        java.awt.Component.processMouseEvent(Ljava.awt.event.MouseEvent;)V(Component.jav
        java.awt.Component.processEvent(Ljava.awt.AWTEvent;)V(Component.java:1566)
        java.awt.Container.processEvent(Ljava.awt.AWTEvent;)V(Container.java:3598)
        java.awt.Component.dispatchEventImpl(Ljava.awt.AWTEvent;)V(Component.java:1623)
        java.awt.Container.dispatchEventImpl(Ljava.awt.AWTEvent;)V(Container.java:3439)
        java.awt.Component.dispatchEvent(Ljava.awt.AWTEvent;)V(Component.java:3450)
        java.awt.LightweightDispatcher.retargetMouseEvent(Ljava.awt.Component;ILjava.awt
        java.awt.LightweightDispatcher.processMouseEvent(Ljava.awt.event.MouseEvent;)Z(C
        java.awt.LightweightDispatcher.dispatchEvent(Ljava.awt.AWTEvent;)Z(Container.jav
        java.awt.Container.dispatchEventImpl(Ljava.awt.AWTEvent;)V(Container.java:1585)
        java.awt.Window.dispatchEventImpl(Ljava.awt.AWTEvent;)V(Window.java:3439)
        java.awt.Component.dispatchEvent(Ljava.awt.AWTEvent;)V(Component.java:450)
        java.awt.EventQueue.dispatchEvent(Ljava.awt.AWTEvent;)V(EventQueue.java:???)
        com.jrockit.console.util.WaitCursorEventQueue.dispatchEvent(Ljava.awt.AWTEvent;)
        java.awt.EventDispatchThread.pumpOneEventForHierarchy(ILjava.awt.Component;)Z(Ev
        java.awt.EventDispatchThread.pumpEventsForHierarchy(ILjava.awt.Conditional;Ljava
        java.awt.EventDispatchThread.pumpEvents(ILjava.awt.Conditional;)V(EventDispatchT
        java.awt.EventDispatchThread.pumpEvents(Ljava.awt.Conditional;)V(EventDispatchTh
        java.awt.EventDispatchThread.run()V(EventDispatchThread.java:???)
        java.lang.Thread.startThreadFromVM(Ljava.lang.Thread;)V(Native Method)
}
```

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" on page 10

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

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

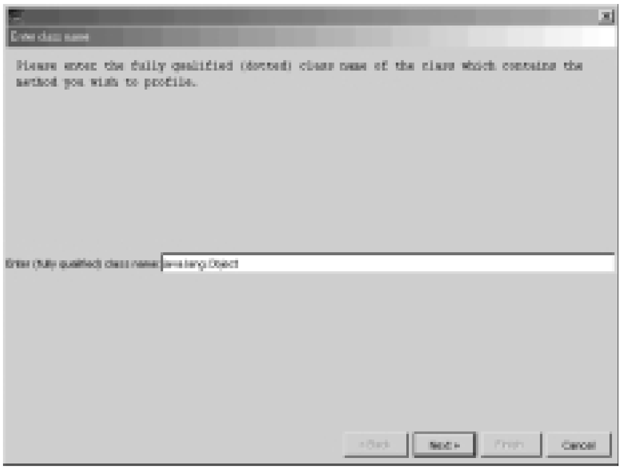
Adding a Method to a Template

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

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

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

Figure 6-35 Enter Class Name Dialog Box



Enter class name

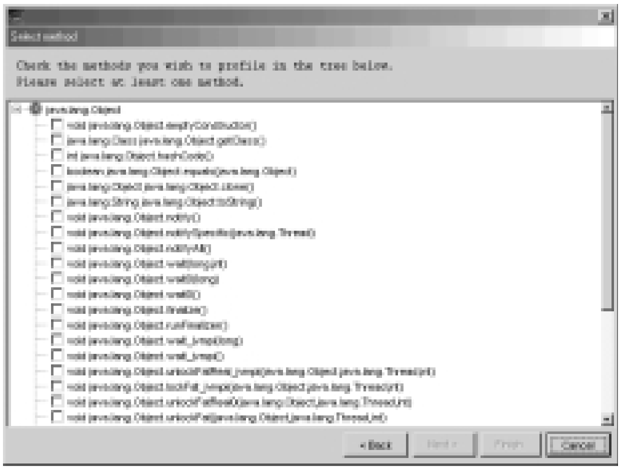
Please enter the fully qualified (dotted) class name of the class which contains the method you wish to profile.

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 6-36):

Figure 6-36 Select Method Dialog Box



Select method

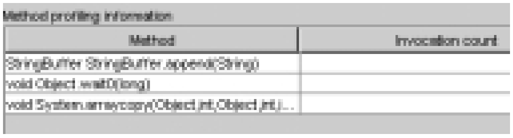
Check the methods you wish to profile in the tree below.
Please select at least one method.

- java.lang.Object
  - void java.lang.Object.emptyConstructor()
  - java.lang.Class java.lang.Object.getClass()
  - int java.lang.Object.hashCode()
  - boolean java.lang.Object.equals(java.lang.Object)
  - java.lang.Object java.lang.Object.clone()
  - java.lang.String java.lang.Object.toString()
  - void java.lang.Object.notify()
  - void java.lang.Object.notifySpecific(java.lang.Thread)
  - void java.lang.Object.notifyAll()
  - void java.lang.Object.wait(long,int)
  - void java.lang.Object.wait0(long)
  - void java.lang.Object.wait0()
  - void java.lang.Object.finalize()
  - void java.lang.Object.runFinalizer()
  - void java.lang.Object.wait_impl(long)
  - void java.lang.Object.wait_impl()
  - void java.lang.Object.unlockFatFirst_impl(java.lang.Object,java.lang.Thread,int)
  - void java.lang.Object.lockFat_impl(java.lang.Object,java.lang.Thread,int)
  - void java.lang.Object.unlockFatRest0(java.lang.Object,java.lang.Thread,int)
  - void java.lang.Object.unlockFat(java.lang.Object,java.lang.Thread,int)

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

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

Figure 6-37 Method Profiling Information List with Method Added



Method profiling information

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

Removing a Method from a Template

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

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

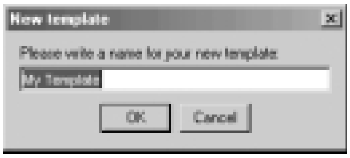
Creating a New Template

To create a new template, do the following:

1. Click New template.

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

Figure 6-38 New 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. If you select Stop, this activity will cease.

Method Profiling Settings

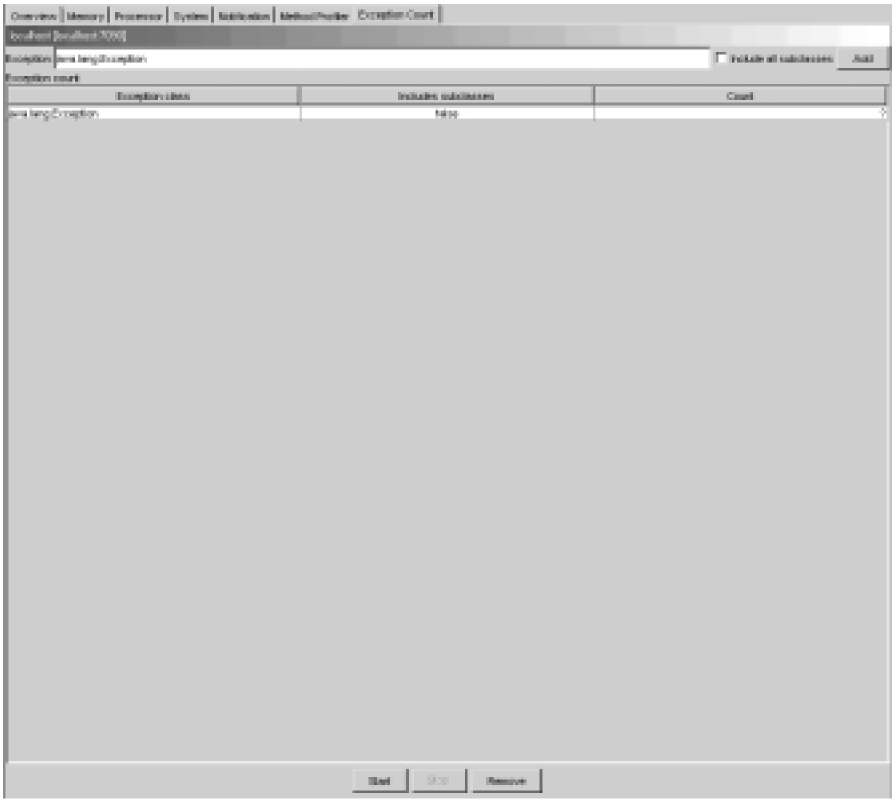
You can switch between using qualified method names or short names in the method profiling table.

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

Exception Counting Tab

The Exception Count Tab (Figure 6-39) shows exceptions thrown in JRockit. It counts the number of exceptions of a certain type thrown.

Figure 6-39 Exception Counting Tab



Exception: java.lang.Exception

Exception count

Exception class	Includes subclasses	Count
java.lang.Exception	false	

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 JRockit and you can only add exceptions while connected.

The exception should now be displayed in the table.

Starting, Stopping, and Removing an Exception Count

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

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

Closing the Console

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