



BEARockit Management Console

User Guide

Copyright

Copyright © 2005 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

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

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

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

Trademarks or Service Marks

BEA, BEA JRockit, BEA Liquid Data for WebLogic, BEA WebLogic Server, Built on BEA, Jolt, JoltBeans, SteelThread, Top End, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Manager, BEA MessageQ, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Enterprise Security, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Java Adapter for Mainframe, BEA WebLogic JDriver, BEA WebLogic JRockit, BEA WebLogic Log Central, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server Process Edition, BEA WebLogic WorkGroup Edition, BEA WebLogic Workshop, and Liquid Computing are trademarks of BEA Systems, Inc. BEA Mission Critical Support is a service mark of BEA Systems, Inc. All other company and product names may be the subject of intellectual property rights reserved by third parties.

All other trademarks are the property of their respective companies.

Contents

Introduction

What is New in the BEA JRockit Management Console.	1-1
Console Overhead	1-1
About this User Guide	1-2
Finding Additional Information	1-2

Using the Management Console to Monitor Applications

Architectural Overview	2-1
JMX specification	2-2
Platform Beans	2-2
Attributes Described	2-3
Viewing the Attributes	2-4
Icons Explained.	2-5
Attribute Subscriptions	2-5
Viewing and Editing the Attribute Subscriptions	2-6
Subscribing to Attributes.	2-6

Getting Started to Use the BEA JRockit Management Console

Connection to JRockit from the Management Console	3-1
Starting the BEA JRockit Management Console.	3-2
Enable the Management Server	3-2
Start the BEA JRockit Management Console.	3-2
Change the Port.	3-3

Advanced/Security Options	3-4
---------------------------------	-----

Touring the BEA JRockit Management Console Interface

Management Console Tabs.	4-2
Administrator Mode	4-3
Developer Mode	4-3

Setting up the BEA JRockit Management Console

Managing Connections	5-1
Creating a New Connection	5-1
Creating a New Folder	5-2
Connecting to BEA JRockit	5-3
Disconnecting from BEA JRockit.	5-3
Renaming a Connection or Folder	5-3
Removing a Connection or Folder	5-3
Hiding Disconnected Connections	5-4
About the Ignore List	5-4
Enabling Console Settings.....	5-4
Opening the Preferences Window	5-4
Mode of Operation	5-5
Graphics and E-mail Settings.....	5-5
Persistence Directory	5-6
Setting the Persistence Directory	5-6
Clearing All Attribute Subscription Logs.....	5-6
Communication Settings	5-7
Console Settings File.....	5-7
Locating consolesettings.xml.....	5-7

Using the BEA JRockit Management Console

Viewing Summary Information	6-1
Customizing the Displays	6-2
Customizing Gauges and Bars	6-3
Customizing Graphs	6-3
Changing Graph Attributes	6-4
Monitoring Memory Usage	6-5
Memory Tab Functionality	6-6
Monitoring Running Processes	6-7
Viewing Information about JRockit	6-7
Viewing and Creating Custom Alerts and Notifications	6-9
About Notification Triggers	6-9
Creating a New Rule	6-10
Creating Your Own Notification Actions and Constraints	6-13
Editing a Rule	6-13
Using a Rule to Monitor a Connection	6-14
Viewing a Notification Alert	6-14
Removing a Rule from a Connection	6-15
Removing a Rule from the Available Rules List	6-15
Getting Method Profiling Data	6-16
Starting and Stopping a Method Profiling Template	6-16
Creating a New Method Profiling Template	6-17
Removing a Method Profiling Template	6-17
Adding a Method to a Template	6-17
Removing a Method from a Template	6-19
Method Profiling Settings	6-19
Counting Exceptions in BEA JRockit	6-19

Adding an Exception	6-20
Starting, Stopping, and Removing an Exception Count	6-20

Using Advanced Features of the Management Console

Historical Data.	7-1
Viewing Historical Data.	7-1
Thread Stack Dump.	7-2
Viewing the Thread Stack Dump.	7-2
Starting and Running the Console in Headless Mode.	7-3
Running a Headless Management Console	7-4
Controlling the Console with Command Line Options	7-4
JRA Recordings.	7-5
Creating a JRA Recording	7-6

Adding Custom Notification Actions and Constraints

Creating a Custom Action.	A-1
Creating and Implementing a Notification Action—an Example.	A-2
Create the Action (Step 2)	A-2
Implementing <code>handleNotificationEvent()</code> (Step 3).	A-4
Creating the Action Editor (Step 4)	A-5
Implementing the Abstract Methods (Step 5)	A-6
Adding the New Action to the Deployment Entries (Step 6).	A-7
Displaying the New Action Editor (Steps 7 and 8)	A-8
Creating a Custom Constraint	A-8

Index

Introduction

The BEA JRockit Management Console is a JMX-compliant monitoring tool. It uses the extensive JMX instrumentation of the Java virtual machine to provide information on performance and resource consumption of applications running on BEA JRockit. This document contains procedures and information on how to monitor your running applications.

The following subjects are covered in this chapter:

- [What is New in the BEA JRockit Management Console](#)
- [Console Overhead](#)
- [About this User Guide](#)
- [Finding Additional Information](#)

What is New in the BEA JRockit Management Console

- This version of the BEA JRockit Management Console has full support of Java MBeans (JMX).
- Look and feel has been updated to create a better user experience.

Console Overhead

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

Note: We do not recommend that you run the Management Console on the same machine as the VM you are monitoring. If you run the Management Console on the same machine as the BEA JRockit you are monitoring, the Management Console GUI will steal valuable resources from the applications running on the JVM and you risk performance degradation as a result.

About this User Guide

This document shows you what information you can get from your Java Virtual Machine (JVM), how to get that information, and how to interpret it. This user guide assumes that you know what a JVM is.

Finding Additional Information

You can find additional information about BEA JRockit throughout the BEA JRockit documentation set. For a complete list of available documents, please refer to the [BEA JRockit JDK Online Documentation](#).

Using the Management Console to Monitor Applications

One of the biggest changes in the Management Console compared to the previous release is the addition of the JMX layer and the possibility of using MBeans. This chapter describes how the JMX layer and the MBeans are used to monitor your running Java applications. You will find information on the following subjects:

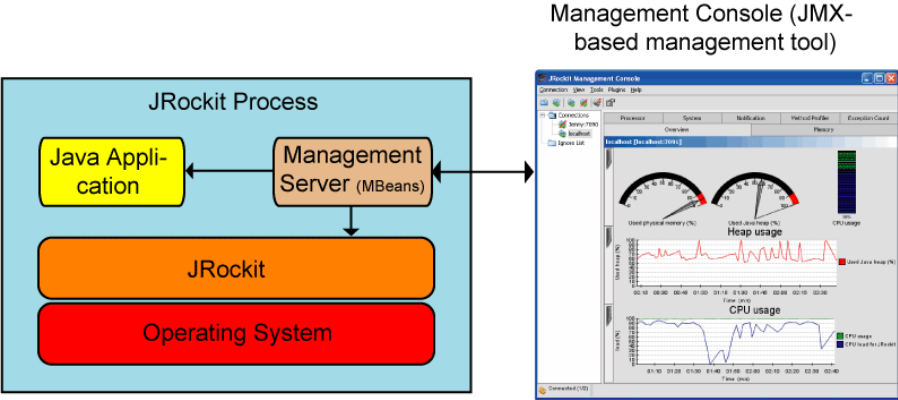
- [Architectural Overview](#)
- [Icons Explained](#)
- [Subscribing to Attributes](#)

Architectural Overview

[Figure 2-1](#) shows the architecture of the monitoring and management support. In this release, JRockit (JVM) is highly instrumented for monitoring and management. The platform instrumentation provides information on performance, resource consumption, and the JVM and logging settings of applications running on the Java platform.

JMX provides a standard way to instrument the Java runtime environment and applications; the JMX Remote API allows that instrumentation to be accessed remotely. The instrumentation is accessible through the JMX managed bean (MBean) interfaces, which are registered in the management server. Applications can also create their own MBeans and register them in the management server, which can serve as a single point for remote access. A JMX-compliant client, such as the BEA JRockit Management Console, can connect to the platform management server and manage the application (as well as the Java platform) using JMX technology.

Figure 2-1 Architecture of the JRockit Monitoring and Management Support



JMX specification

For a complete reference on the JMX standard, please see:

<http://java.sun.com/j2se/1.5.0/docs/api/javax/management/package-summary.html>

Platform Beans

The Java platform provides a set of platform MBeans for monitoring and managing the Java virtual machine and the logging facility. Table 2-1 describes the MBeans included in this release:

Table 2-1 MBeans Included in BEA JRockit 5.0

Platform MBean	Description
<code>java.lang.management.ClassLoadingMXBean</code>	Class loading system of the Java virtual machine.
<code>java.lang.management.CompilationMXBean</code>	Compilation system of the Java virtual machine.
<code>java.lang.management.MemoryMXBean</code>	Memory system of the Java virtual machine.
<code>java.lang.management.MemoryManagerMXBean</code>	Memory manager in the Java virtual machine.
<code>java.lang.management.MemoryPoolMXBean</code>	Memory pool in the Java virtual machine.
<code>java.lang.management.GarbageCollectorMXBean</code>	Garbage collector in the Java virtual machine.

Table 2-1 MBeans Included in BEA JRockit 5.0

Platform MBean	Description
<code>java.lang.management.ThreadMXBean</code>	Threading system of the Java virtual machine.
<code>java.lang.management.RuntimeMXBean</code>	Runtime system of the Java virtual machine.
<code>java.lang.management.OperatingSystemMXBean</code>	Operating system on which the Java virtual machine is running.
<code>java.util.logging.LoggingMXBean</code>	Logging facility.

JRockit has its own set of MBeans that inherit the functionality of the platform MBeans described in [Table 2-1](#). The correct names of the JRockit specific MBeans follow the same naming convention, except for the `jrockit` extension; for example,

`java.lang.management.ThreadMXBean` is the same as
`jrockit.management.ThreadMXBean`.

An MBean is a managed object that follows the design patterns conforming to the JMX specification. An MBean can represent a device, an application, or any resource that needs to be managed. The management interface of an MBean comprises a set of readable and/or writable attributes.

Each platform MBean has a set of attributes and operations such as memory usage, thread CPU usage, garbage collection statistics, and so on. Some might also emit notifications. We will explore a few platform MBeans in the following sections.

The Management Console can connect to the Management Server, which in turn allows you to monitor your application and view all the JMX objects. You need to be connected to the management server to view the objects. To start the Management Console, see [Starting the BEA JRockit Management Console](#).

Attributes Described

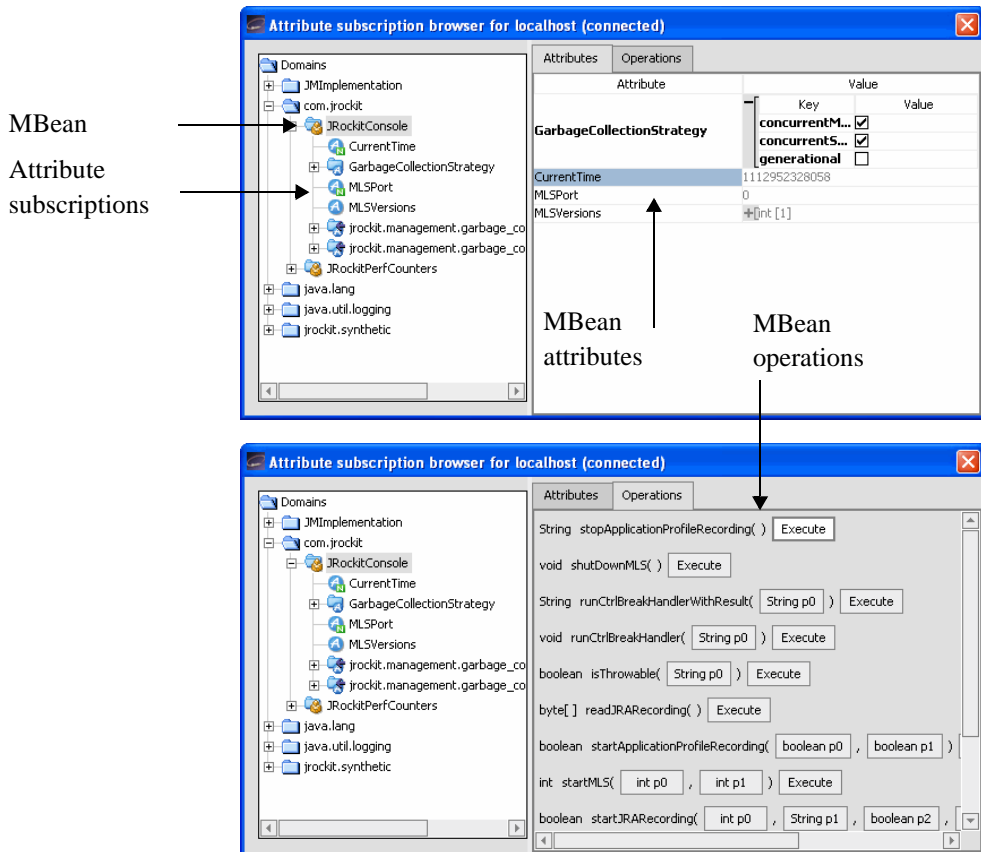
The registered MBeans and their attributes are all listed in the **Attribute Subscription Browser** ([Figure 2-3](#)). Each attribute has its own set of attribute subscriptions that can be used to monitor the different aspects of your application and connection.

Viewing the Attributes

1. Make sure you are connected to the management server (see [Enable the Management Server](#)) and that you are connected to a JRockit instance (see [Connecting to BEA JRockit](#)).
2. Click **Connection > Browse Attributes**.

The **Attribute subscription browser** appears, see [Figure 2-2](#).

Figure 2-2 Attribute Subscription Browser for Local Host









The tree to the left shows all available MBeans and their subscriptions on the management server

Icons Explained

[Table 2-2](#) describes the available icons in the **Attribute subscription browser**.

Table 2-2 Descriptions of icons.

Icon	Description
	The MBean itself
	A composite attribute. Open the folder and you display all attribute subscriptions for the composite attribute.
	An asynchronous JMX-notification based attribute. Open the folder and you display all attribute subscriptions for the composite value delivered with a notification event.
	Indicates that the attribute subscription is numerical.
	Indicates that the attribute subscription is a synthetic numerical. A synthetic attribute subscription does not exist on the server. It is either calculated on the client or derived from other attribute subscriptions.
	Indicates that the attribute subscription is neither synthetic nor numeric.

Attribute Subscriptions

The registered MBeans are all listed in the Attribute Subscription Browser ([Figure 2-3](#)). Each MBean has its own set of attribute subscriptions that can be used to monitor the different aspects of your application and connection. By subscribing to the attributes, you have a powerful way to monitor your application. You can subscribe to the attributes from many places in the Management Console; for example, you can add subscription attributes to the graphs on the Overview tab or you can select the **Enable Persistence** selection in the **Attribute subscription browser**.

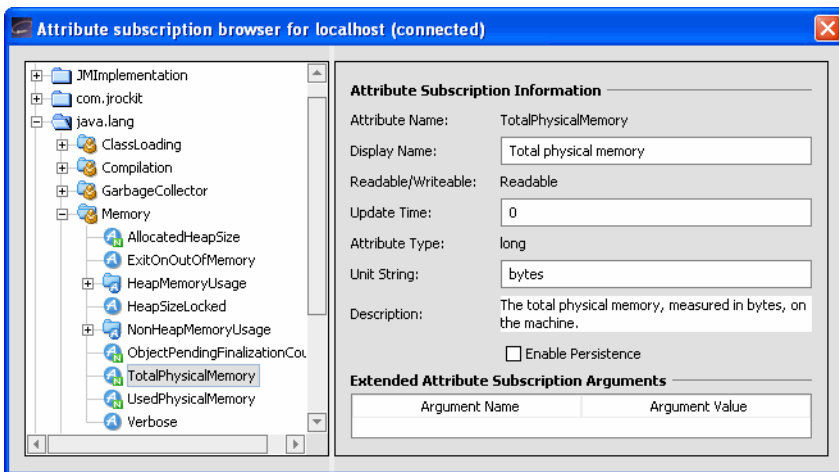
When you select the Enable Persistence option, the attributes are saved to a .csv file that is stored where you have specified. See [Persistence Directory](#) for more information on the persistence file.

Viewing and Editing the Attribute Subscriptions

1. Make sure you are connected to the management server (see [Enable the Management Server](#)) and that you are connected to a JRockit instance (see [Connecting to BEA JRockit](#)).
2. Click **Connection > Browse Attributes**.

The **Attribute subscription browser** appears, see [Figure 2-3](#).

Figure 2-3 Attributes Subscription Browser



The tree to the left shows all available MBeans on the management server. These MBeans are organized according to their domains (see [Table 2-2](#) for a description of the icons).

3. Select an attribute subscription you want to change or view.

You can change the properties of the attribute subscription under Attribute Subscription Information. This information is saved in your settings file (see [Persistence Directory](#) for more information on the settings file).

4. Close the window when you are done to return to the Management Console.

Subscribing to Attributes

1. Open the **Attribute subscription browser** window.
2. Select an attribute subscription.
3. Select **Enable Persistence**.

4. Choose a different attribute or close the window.

Using the Management Console to Monitor Applications

Getting Started to Use the BEA JRockit Management Console

The Management Console is a JMX-compliant monitoring tool. It uses the extensive JMX instrumentation of JRockit to provide information on performance and resource consumption of applications running on the Java platform.

This section includes information on the following subjects:

- [Connection to JRockit from the Management Console](#)
- [Starting the BEA JRockit Management Console](#)
- [Change the Port](#)

Connection to JRockit from the Management Console

With the BEA JRockit Management Console you monitor and control running instances of the BEA JRockit JVM. It provides real-time information about the running Java application's characteristics, such as memory consumption and CPU usage. This information 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.

The Management Console is connected to the running instance of JRockit via the management server, as shown in [Figure 2-1](#). The JRockit process contains JRockit, the running application, and a management server. The machine running the Management Console is preferably on a machine outside the machine running JRockit, that way it is the least intrusive. In [Figure 2-1](#) the Management Console is listening to the JRockit process on port 7091 (the default port).

Starting the BEA JRockit Management Console

The starting of the Management Console is a two-step process:

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

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, like this:

```
java -Xmanagement [your application]
```

Start the BEA JRockit Management Console

The Management Console executable is located in `JRockit_JDK/bin`, where `JRockit_JDK` is the directory where the JDK is installed. If this directory is on your system path, you can start the tool by simply typing `console` in a command (shell) prompt. Otherwise, you have to type the full path to the executable file.

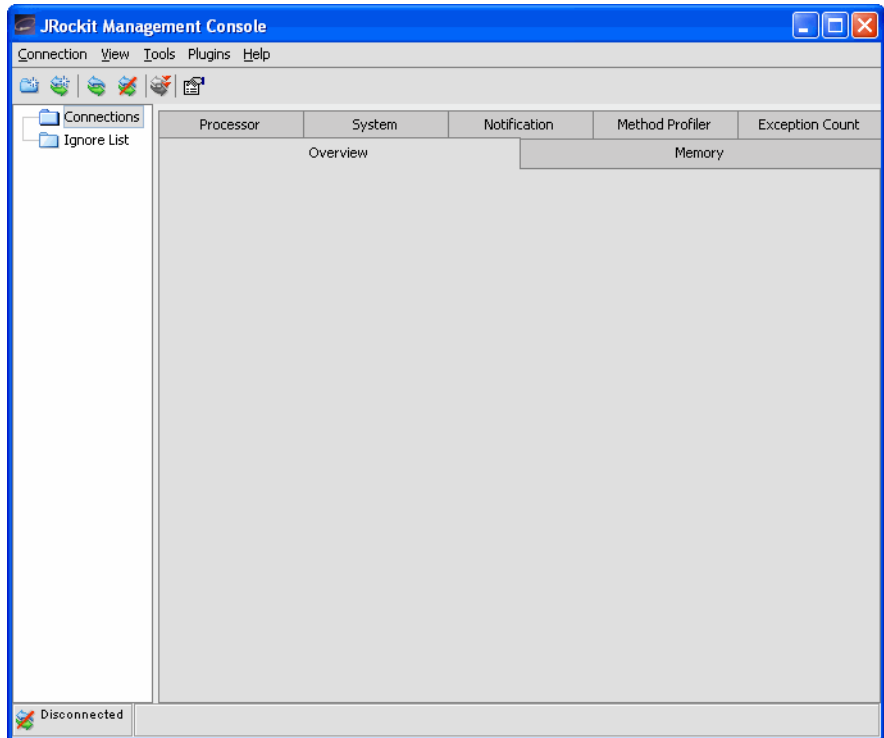
Start the BEA JRockit Management Console from the command prompt by typing:

```
console
```

You can also start the Management Console without using the launcher. From the command prompt, type:

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

When the console is started the Management Console window appears, see [Figure 3-1](#).

Figure 3-1 The main Management Console window.

Change the Port

When BEA JRockit JVM is started with the `-Xmanagement` option set, 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 that the management server uses to connect is 7091. It is strongly recommended that you block this port in your firewall, otherwise unauthorized users might access the management server.

Advanced/Security Options

The `-Xmanagement` option is the same as using the following startup options from Sun Microsystems:

- `-Dcom.sun.management.jmxremote.port=7091`
- `-Dcom.sun.management.jmxremote.authenticate=false`
- `-Dcom.sun.management.jmxremote.ssl=false`

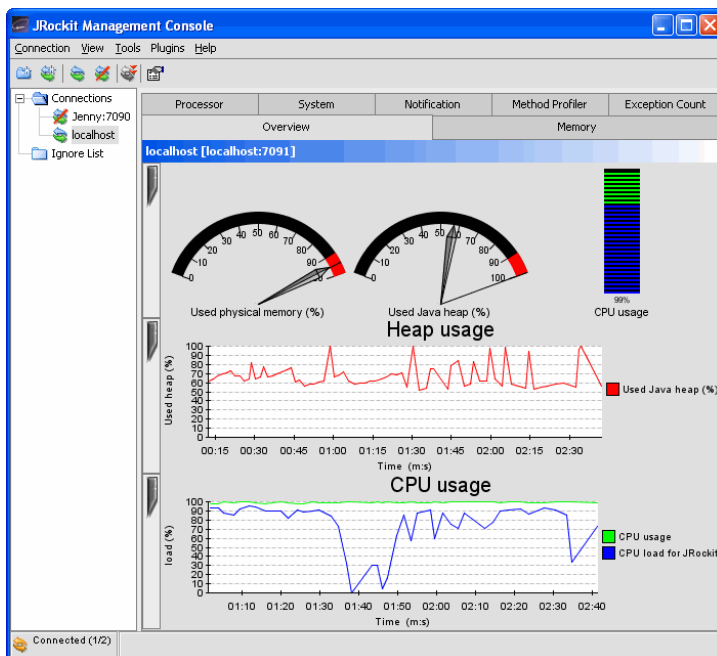
If you want to start the management server using alternate startup options, see:

<http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html#remote> for more information on remote monitoring and management.

Touring the BEA JRockit Management Console Interface

This section introduces the graphical elements of this interface (see [Figure 4-1](#)) and explains the basic elements of the interface.

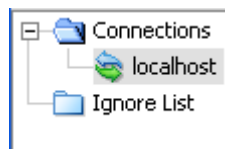
Figure 4-1 BEA JRockit JVM Management Console Window (when connected to JRockit)



The BEA JRockit Management Console window is divided into two panes: a connection browser tree in the left pane ([Figure 4-2](#)) and a tabbed information pane in the right ([Figure 4-5](#)).

In the connection browser ([Figure 4-2](#)) you administrate the different connections that you have to running JRockits. The Ignore List contains connections to other VMs that you are connected to but do not want information from at the moment.

Figure 4-2 Connection Browser



The toolbar ([Figure 4-3](#)) contains the most used tools of the Management Console: new connections folder, new connection, connect, disconnect, and preferences.

Note: You need to create a connection before you see a list of connections. See [Creating a New Connection](#) for information on how to create a connection.

Figure 4-3 Management Console Toolbar



The status bar ([Figure 4-4](#)) 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 or not the BEA JRockit Management Console is connected to one or several BEA JRockit JVM implementations.

Figure 4-4 Management Console Status Bar



To turn off the Status Bar, select **View > Status Bar**.

Management Console Tabs

You can run the Management Console in two different modes: **Administrator** (see [Figure 4-5](#)) and **Developer** (see [Figure 4-6](#)) mode. The default mode is the Developer mode. To switch between the two modes, see [Mode of Operation](#).

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 [Using the BEA JRockit Management Console](#).

Figure 4-5 Information Tabs (Administrator Mode)

Overview	Memory	Processor	System	Notification
----------	--------	-----------	--------	--------------

Developer Mode

When the console is in the Developer mode, additional tabs appear, as shown in [Figure 4-6](#). For a full description of the different tabs, see [Using the BEA JRockit Management Console](#).

Figure 4-6 Information Tabs (Developer Mode)

System	Notification	Method Profiler	Exception Count
Overview	Memory	Processor	

Touring the BEA JRockit Management Console Interface

Setting up the BEA JRockit Management Console

Before you start running and using the Management Console, it is a good idea to set up the different connections for the JVMs you want to view. The Management Console is best run remotely on a computer connected to the management server. That way the console is the least intrusive on the JRockit that is running in your system.

Configuring—or “setting up”—the Management Console includes these tasks:

- [Managing Connections](#)—deciding which connections to make.
- [Enabling Console Settings](#)—various settings about the console.

Managing Connections

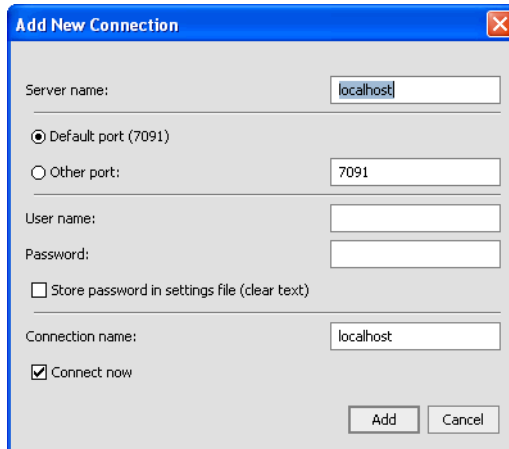
The connection browser ([Figure 4-2](#)) displays a collection of saved connections to BEA JRockit JVMs organized in folders. If necessary, you can add your own folders and connection nodes to the tree structure.

Creating a New Connection

1. Select the folder in which the connection should be placed
2. Click **Connection > New Connection**.

The Add new connection dialog box ([Figure 5-1](#)) appears:

Figure 5-1 Add New Connection Dialog Box

The image shows a Windows-style dialog box titled "Add New Connection" with a blue title bar and a red close button. The dialog has a light gray background. It contains several fields and options: "Server name:" with a text box containing "localhost"; a radio button group with "Default port (7091)" selected; "Other port:" with a text box containing "7091"; "User name:" and "Password:" with empty text boxes; a checkbox "Store password in settings file (clear text)" which is unchecked; "Connection name:" with a text box containing "localhost"; and a checkbox "Connect now" which is checked. At the bottom right are "Add" and "Cancel" buttons.

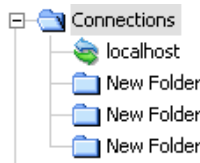
3. Type in a name of the server, the port (the port number of the enabled JMX server), and the new connection in the appropriate text fields or retain the default values.
4. Type in a user name for the connection and a password. If you want to store the password in the settings file, select the store option (the password is stored unencrypted).
5. Type in a connection name.
6. Select or deselect **Connect now** depending on if you want to monitor your JRockit VM at once or not.
7. Click **Add**.

The connection will now appear in the Connections folder.

Creating a New Folder

1. Select the existing **Connections** folder.
2. Click **Connection > New Folder**.

The new folder appears in the connection browser (Figure 5-2).

Figure 5-2 New Folders

3. Select the new folder once to activate the folder, then click again to highlight the name.
4. Type in a name for the folder and press **Enter**.

Connecting to BEA JRockit

1. From your list of connections, select the BEA JRockit JVM connection you wish to activate.
To connect an entire folder, select the folder.
2. Click **Connection > Connect**.

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

Disconnecting from BEA JRockit

1. From your list of connections, select the BEA JRockit JVM connection you want to disconnect.
To disconnect an entire folder, select the folder.
2. Click **Connection > Disconnect**.

The connection will be lost and the status bar will indicate that you’ve been disconnected.

Renaming a Connection or Folder

1. Select the BEA JRockit JVM connection or folder to rename.
2. Click once more to change the name.
3. Type in a new name and press **Enter**.

Removing a Connection or Folder

1. Select the connection subfolder you want to remove.
2. Click **Connection > Remove**.

The selected connection disappears from the connection browser.

Hiding Disconnected Connections

Sometimes you only want to display information about the active BEA JRockit JVM connections. To hide information about disconnected connections:

- 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.

Note: If you have one connection only, it will be visible at all times.

About the Ignore List

In the Ignore List you hide those connections that you do not want to access. See

Enabling Console Settings

The Management Console has different preferences that can be set. These preferences are grouped into the following main groups: General, Persistence, and Communication (see [Figure 5-3](#)). The Communication preference is only visible in developer mode.

This section is divided into the following subsections:

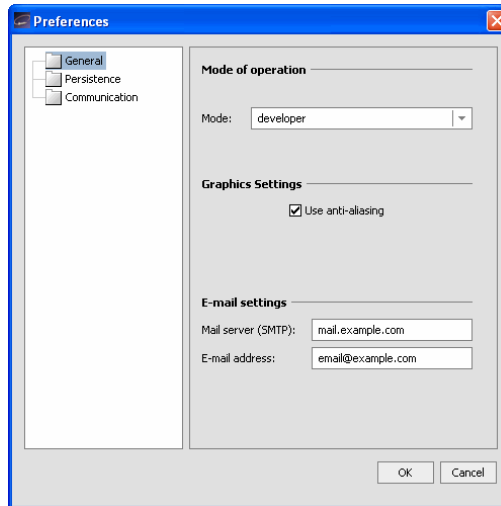
- [Mode of Operation](#)
- [Graphics and E-mail Settings](#)
- [Persistence Directory](#)
- [Communication Settings](#)

Opening the Preferences Window

- Click **Tools > Preferences**.

The Preferences dialog box ([Figure 5-3](#)) appears.

Figure 5-3 Preferences Window



Mode of Operation

The Management Console can be run in two different operating modes depending on the type of information you want to access:

- **Developer Mode:** This is the default mode and provides more features (compared to the Administrator mode) such as a rudimentary method profiler and exception count functionality. Additional tabs are also available: the **Method Profiler** tab and the **Exception Count** tab.
- **Administrator Mode:** This mode is designed for system administrators who are mostly interested in observing the state of the BEA JRockit JVM.

Graphics and E-mail Settings

In addition to setting the operation mode, you use **General Preferences** to change the following:

- Turn off the anti-aliasing setting. This is good if you are running the Management Console on a Linux system. Using anti-aliasing over a remote X connection over Linux slows down the GUI considerably.
- Change the default e-mail settings for the notification system, for example, the system sends an e-mail to this address when a certain heap size is reached.

Persistence Directory

The persistence directory is where the Management Console saves the persisted values of attributes (in the attribute subscription log). Values that are saved are, for example, any changes that you make to MBeans, i.e., name changes or such. The attribute subscription logs are saved upon exiting the Management Console. If there, for some reason, arises problems with the logs, you can clear the logs by deleting them and then have the Management Console create a new one for you (see [Clearing All Attribute Subscription Logs](#)).

Setting the Persistence Directory

1. Click **Tools > Preferences**.
2. Select the **Persistence** option.
3. Click **Choose** to find another location for the attribute subscription logs.
4. Click **Yes** when you are asked to disconnect JRockit connections.
5. Select a location for the files.

The default location for the files are `<User_Home>\.ManagementConsole\data\`.

6. Click **Open**.

Every time you exit the Management Console, a new aspect value log is saved in the specified directory. It is not recommended to change these logs by hand.

7. Close the **Preferences** window.

Clearing All Attribute Subscription Logs

1. Click **Tools > Preferences**.
2. Select the **Persistence** option.
3. Click **Clear all attribute subscription logs**.
4. Click **Yes** when you are asked to disconnect JRockit connections.
5. Close the **Preferences** window.

Note: If you delete all persistence logs by clicking this button, you will also delete any other files stored in the `<User_Home>\.ManagementConsole\data\`.

Communication Settings

The possibility to change communication settings is on if you have set the Management Console to Developer mode (see [Mode of Operation](#)). The available communication settings are:

- Socket timeout
- Reply timeout
- Core update interval

Use precaution when changing these values.

Console Settings File

For the Management Console to remember your specific settings, for example, your connections, it automatically creates a settings file when you exit. This file is called `consolesettings.xml` and 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\<username>\.ManagementConsole
```

If you do not currently see such a file in the above directory, one has probably not been created yet. As soon as you exit the Management Console one is created.

Warning: Do not edit the settings file by hand! Doing so 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.

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:

Setting up the BEA JRockit Management Console

```
/home/<user_name>/ManagementConsole
```

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

Using the BEA JRockit Management Console

To help you navigate in the Management Console, there are several information tabs (the number of tabs depends on which mode of operation you are using, see [Developer Mode](#)). You can also customize the display to fit your personal preferences. This section contains information on the following subjects:

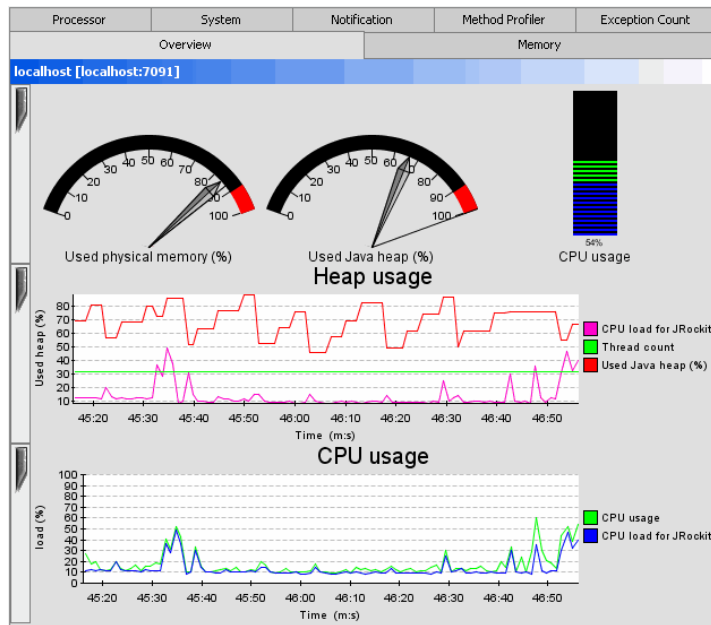
- [Viewing Summary Information](#)
- [Customizing the Displays](#)
- [Monitoring Memory Usage](#)
- [Monitoring Running Processes](#)
- [Viewing Information about JRockit](#)
- [Viewing and Creating Custom Alerts and Notifications](#)

Viewing Summary Information

The **Overview** tab ([Figure 6-1](#)) shows an summary 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 graphs in the lower part.

Note: The figure shows the **Overview** tab when the console is connected to JRockit, see [Creating a New Connection](#).

Figure 6-1 Overview Tab



- The **Used physical memory** gauge shows the percentage of occupied physical memory on the computer.
- The **Used Java heap** gauge shows the percentage of occupied Java heap memory in the VM.
- The **CPU usage** gauge shows the usage rate of the processor—or the average processor load—on a multi-processor machine.
- The **Heaps 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.

Customizing the Displays

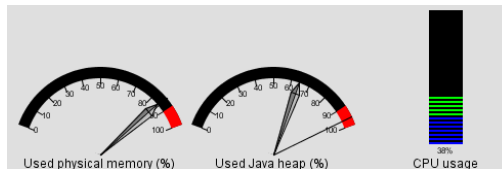
You can customize the console and change the way some of the monitoring data is displayed, as described in the following sections:

- [Customizing Gauges and Bars](#)
- [Customizing Graphs](#)

Customizing Gauges and Bars

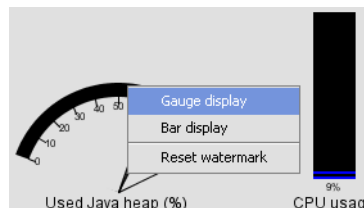
The gauges and bars are graphical devices showing memory and processor usage ([Figure 6-2](#)).

Figure 6-2 Gauges and Bars



- To change from a gauge display to a bar display, right-click when pointing at the gauge and select Bar display, as shown in [Figure 6-3](#).

Figure 6-3 Gauge Context Menu (Bar Display Selected)



The selected gauge will appear as a bar ([Figure 6-3](#)).

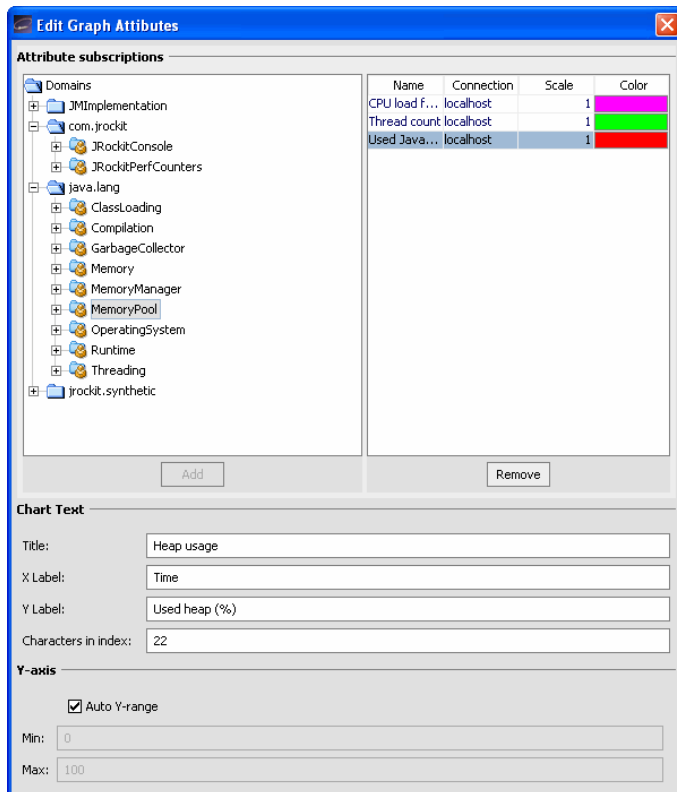
- To change back to a gauge, repeat the above, but select **Gauge display**.
- To reset the watermark—which indicates the highest level measured so far—right-click when pointing at the gauge or bar and click **Reset Watermark**.

Customizing Graphs

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

- **To change a graph** on any of the charts, right-click the graph and choose: Show Hours, Show Minutes, or Show Seconds.
- **To hide a graph** click the vertical tab at the left of the diagram you want to hide. When the diagram is hidden, the tab appears horizontally.
- **To view more graph attributes**, right-click the graph and select **Edit Attributes**, the Edit Graph Attributes appears (see [Figure 6-4](#)).

Figure 6-4 The Edit Graph Attributes window



Changing Graph Attributes

1. Right-click on a graph and select **Edit Attributes**. The Edit Graph Attributes window appears (see Figure 6-4).

In this window, you can do the following:

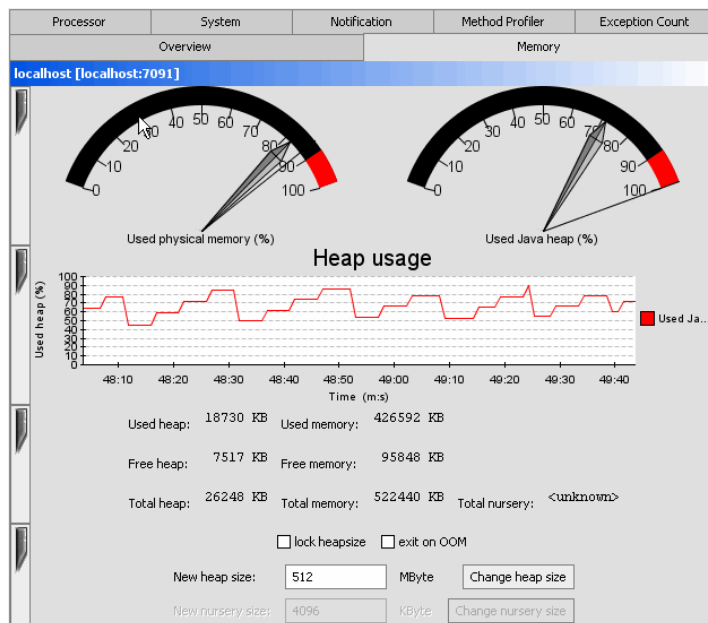
- Add attribute subscriptions to a graph. Highlight the attribute subscription and click **Add**.
- Change color of an active attribute subscription.
- Remove an attribute subscription. Highlight the attribute subscription and click **Remove**.
- Change the texts, such as the title of the graph, the X and Y labels, and the number of characters in the index name.

- Set the Y-axis to an auto range, which is useful if you have attributes plotted over a wide spectra.
2. Close the window when you are done with the settings.
- If you have trouble seeing all attributes on a graph, try to rescale the attribute in the **Edit Graph Attribute** window.

Monitoring Memory Usage

The **Memory** tab (Figure 6-5) shows information about the memory status of the system, as shown.

Figure 6-5 Memory Tab



- The **Used physical memory** gauge shows the percentage of machine memory in use.
- The **Used Java heap** gauge shows the percentage of occupied Java heap.
- The **Heap usage** chart shows the percentage of occupied heap over time. In this example, you can see that the JVM performs a garbage collections when the curve dips.

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

- **Used heap** shows the size of the used Java heap.
- **Free heap** shows the free Java heap size.
- **Total heap** shows the size of the entire Java heap.
- **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.
- **Total nursery** shows the amount of the physical memory that is used for the nursery (only visible when running a generational garbage collector)

Memory Tab Functionality

You can manipulate certain memory aspects of the JVM from the Memory Tab. These aspects are described in [Table 6-1](#).

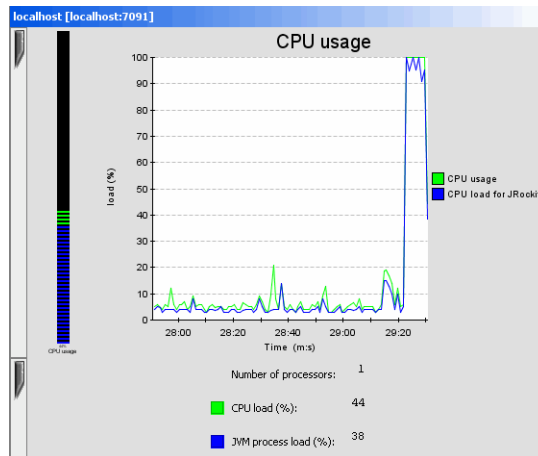
Table 6-1 Memory Tab Functionality

Function	Procedure
Manipulating the heap size	<p>You can manipulate the heap size in any of the following ways:</p> <ul style="list-style-type: none"> • Reset the size of your heap by entering a numeric value in the New heap size field and click Change heap size. The size set here, expressed in megabytes, represents the current heap size, not the maximum heap size. • Make the current size the maximum heap size by selecting lock heapsize. <p>Please refer to Setting the Heap Size for heap size requirements.</p>
Exiting on out of memory errors	<p>If you want to exit the JVM when JRockit encounters an Out of memory (OOM) error, select Exit on OOM.</p>
Changing the nursery size	<p>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. This action is not setting the nursery size to exactly the size you specify, it only tries to get as close as possible to the value.</p> <p>Note: If you are not running a generational collector, these fields are disabled.</p> <p>Please refer to Setting the Size of the Nursery for nursery size requirements.</p>

Monitoring Running Processes

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

Figure 6-6 Processor Tab

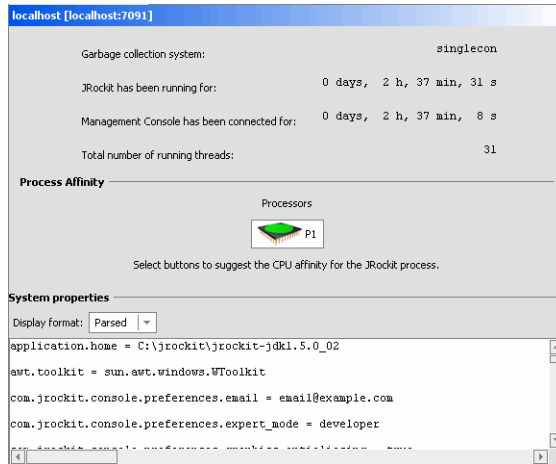


- The **CPU usage** chart shows the average processor load as a percentage over time.
- The **CPU load** bar shows the average processor load in percentages of the overall load.
- **Number of processes** shows the number of Java processes that currently are running.
- **CPU Load** shows the overall processor load in percentage.
- **JVM Process Load** shows the load of the BEA JRockit JVM process(es), expressed as a percentage.

Viewing Information about JRockit

The **System** tab (Figure 6-7) shows various information about the system status.

Figure 6-7 System Tab



- **Garbage collection system** shows which garbage collector the BEA JRockit JVM is running. If you are using the default dynamic garbage collector (`-Xgcprio`), the garbage collector system changes during the run of the JVM.
- **JRockit has been running for** shows how long BEA JRockit 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 is running on this processor and a red icon if it is not. By clicking on a process, the BEA 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, which means that the operating system might not follow the suggestion.

Note: A question mark is displayed until the processor information has had time to be displayed. If you cannot see the information right away, please check back later.

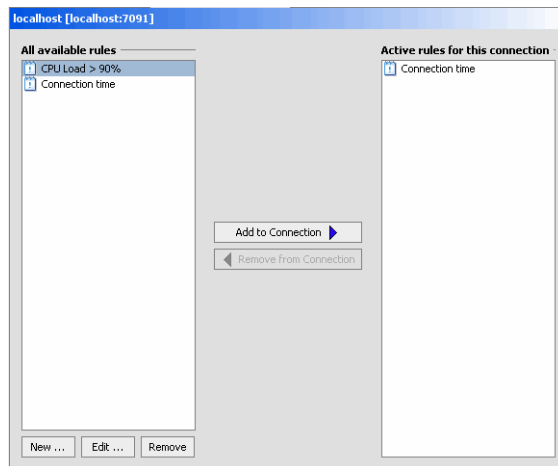
Changing the process affinity is a feature that is only available when monitoring a JRockit instance running on the Windows platform. The **Process Affinity** display is only activated when the Management Console is in the Developer mode, described in [Mode of Operation](#).

- **System properties** shows which Java system properties that are loaded in the VM.

Viewing and Creating Custom Alerts and Notifications

The **Notification tab** (Figure 6-8) lists all available rules for the alerts that you have created; the active rules for the specific connection are listed under *Active rules for this connection* (the right pane). When you start using the Management Console, the list of available rules is empty. When you exit the Management Console, the rules that you have created are saved in the console settings file (see [Console Settings File](#)). You create your own notification rules by combining an attribute subscription with a trigger, with optional constraints. The trigger alerts you with a prescribed notification (see [Creating a New Rule](#)) as soon as the rule option is met.

Figure 6-8 Notification Tab (Rules Defined)



You can programatically create custom notification actions and constraints, which are also stored in the console settings file. Once added, these actions and constraints will also appear on the **Notification** tab of the Management Console. This document does not cover how to do that.

About Notification Triggers

A notification trigger can be a certain event, for example, that the connection to BEA JRockit JVM was lost, or that an attribute reaches a certain value, for example, the used memory reaches 90%. 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:

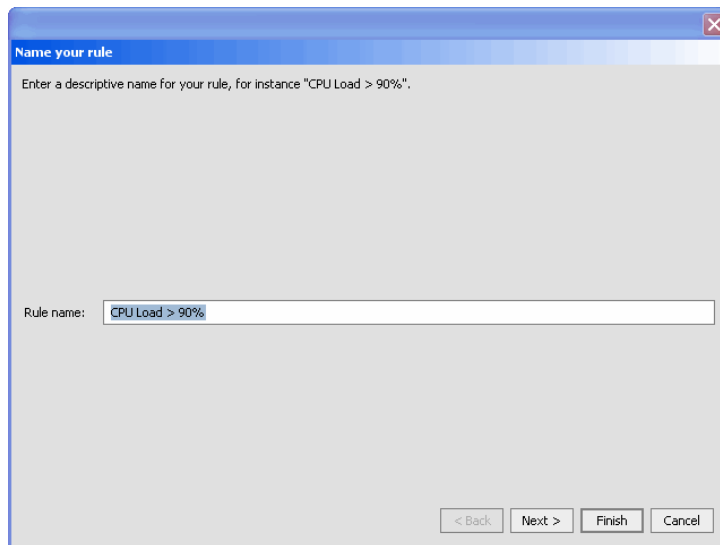
- E-mail shows an e-mail when the notification is sent to the specified address by using the specified SMTP server.
- System out action displays the notification in the command window where you started the BEA JRockit Management Console.
- Application alert displays the notification in an alert dialog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

Creating a New Rule

1. Select the **Notification** tab.
2. Click **New**.

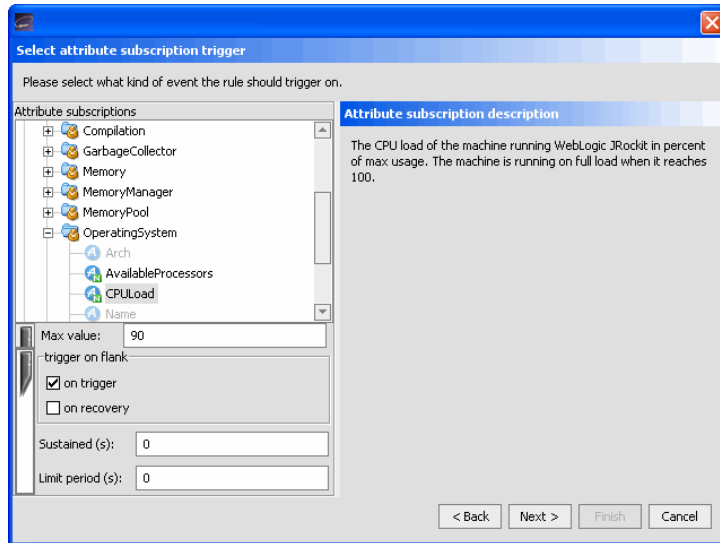
The **Name your rule** dialog appears (Figure 6-9):

Figure 6-9 Name Your Rule Dialog



3. Type in a name for the rule and click **Next**. Try to make the name as descriptive as possible, that way you will have an easier time when remembering how the rule works.

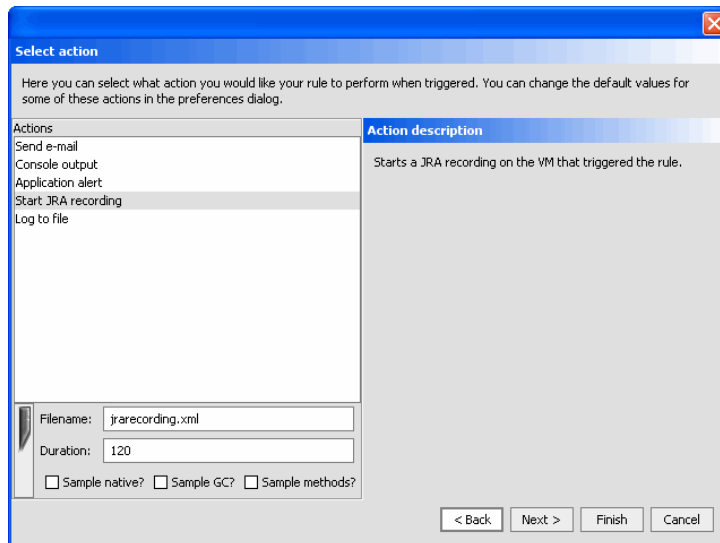
The **Select attribute subscription trigger dialog** appears (Figure 6-10):

Figure 6-10 Select Trigger Dialog

4. Select a trigger (the individual triggers are described to the right in the window).
5. Enter a threshold in **Max value** below the trigger list (not required for all triggers).
6. Set a trigger flank:
 - **on trigger** triggers the notification when the attribute reaches the trigger value from a lower value (for example, if the trigger is 80 and the attribute value goes up from a lower value, the trigger will set off).
 - **on recovery** triggers the notification when the aspect reaches the trigger value from a higher value (for example, if the trigger is 80 and the attribute value goes down from a higher value, the trigger will set off).
7. Click **Next**.

The **Select action** dialog appears (Figure 6-11):

Figure 6-11 Select Action Dialog



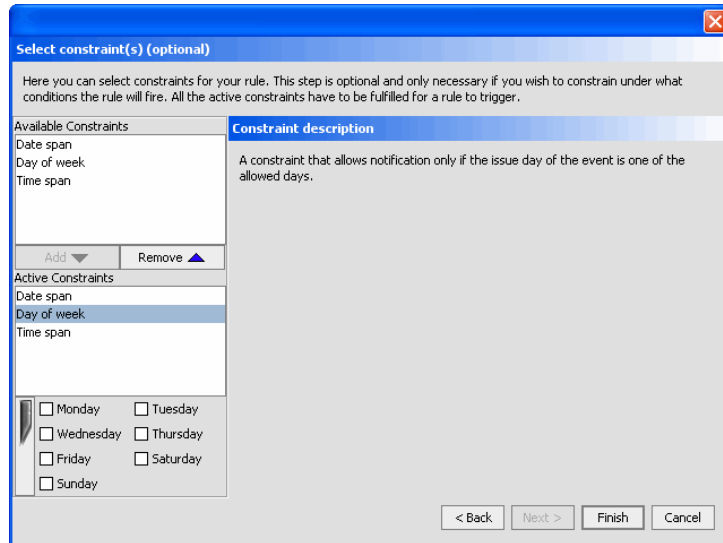
8. Select an action and enter settings data, if required.

In this example, you need to set the name of the JRA file, how long the recording should last, and the different sample statistics.

Now you have two choices:

- a. decide that your rule is done, continue with [step 12. in “Creating a New Rule”](#), or
 - b. decide that you need to add a constraint to your rule. Continue with [step 9. in “Creating a New Rule”](#).
9. Click **Next**.

The **Select constraint(s)** dialog appears ([Figure 6-12](#)):

Figure 6-12 Select Constraint Dialog

10. Select a constraint and click **Add**.

The constraint name will appear in the Active Constraints list.

11. Enter the specific constraint setting.

By selecting a date, day, or time, the event is only triggered on that specific date, day, or time.

12. Click **Finish**.

The new rule appears in the **All available rules** list on the **Notification tab**, as shown in [Figure 6-8](#). If you want to use the rule for your connection, add it as described in [Using a Rule to Monitor a Connection](#).

Creating Your Own Notification Actions and Constraints

You can create your own notification actions and constraints that you can use for your rules, see [Adding Custom Notification Actions and Constraints](#) for more information on how to do that.

Editing a Rule

1. In the **All available rules** list, select the rule to be edited and click **Edit**.

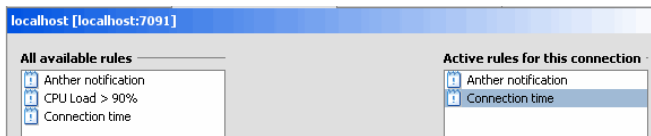
2. Follow [step 3. in “Creating a New Rule”](#) through [step 12. in “Creating a New Rule”](#) to edit your rule.

Using a Rule to Monitor a Connection

1. Select the rule, in the **Available rules** list.
2. Click **Add to Connection**.

The rule appears in the **Active rules for this connection** list, as shown in [Figure 6-13](#).

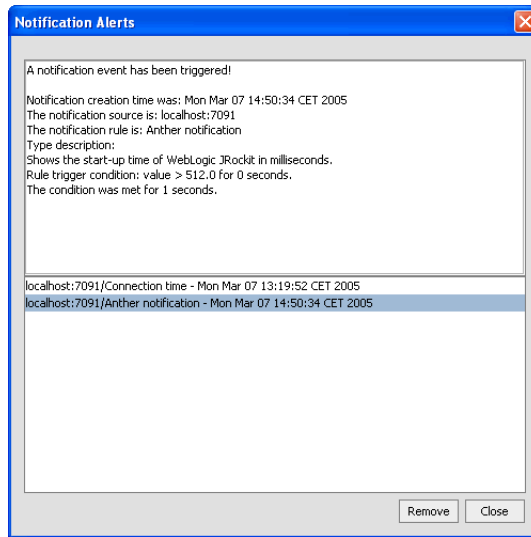
Figure 6-13 Rule Added to Active rules for This Connection List



Viewing a Notification Alert

1. Click **View > Notification Alert**.

A **Notification Alerts** window appears (see [Figure 6-14](#) for an example of an alert that was sent to the application):

Figure 6-14 Notification Alert Sent to Application

2. Select the notification, from the bottom half of the **Notification Alerts** window, you want to view.

In the **Notification Alerts** window you can view how the notification was triggered, when, for how long the condition was met, etc.

3. Click **Close** to close the window or **Remove** if you want to delete the selected alert.

Note: The application alerts are not saved when you exit the Management Console.

Removing a Rule from a Connection

1. Select the rule, in the **Active rules for this connection** list, you want to stop using.
2. Click **Remove from Connection**.

The rule now only appears in the **Active rules for this connection** list.

Removing a Rule from the Available Rules List

1. Select the rule to be removed from the **All available rules** list.
2. Click **Remove**.

A removal confirmation dialog box appears.

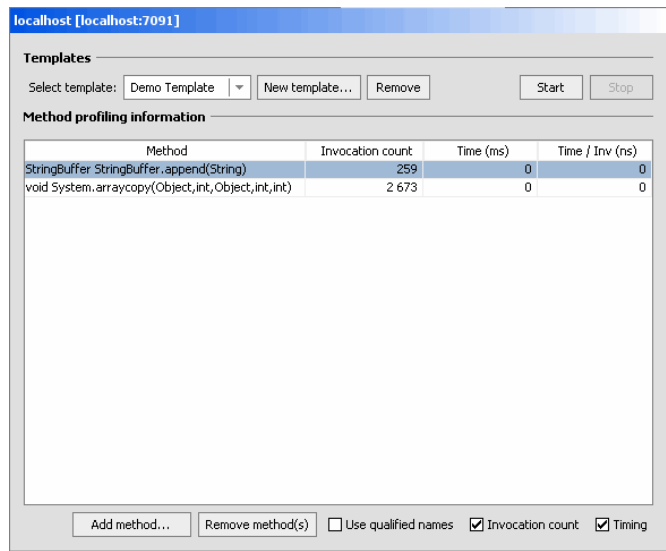
3. Click **Yes**.

The rule disappears from the **All available rules** list.

Getting Method Profiling Data

The **Method Profiler** tab (see [Figure 6-15](#)) contains templates that allows you to monitor method execution in a non-intrusive way. Method profiling can provide information about the average time spent in selected methods and the number of times methods are invoked. A template is a collection of different monitoring methods (the Management Console is shipped with a **Demo Template**). The template settings are stored in the console settings file (see [Console Settings File](#)) and can be used with all connections.

Figure 6-15 Method Profiler tab



By creating your own method templates, you get a powerful tool to monitor you running application’s methods and find out where in the code you might have glitches.

Starting and Stopping a Method Profiling Template

1. From the **Select template** list, select the template you want to start.
If you have not created a template yet, you can start the template called **Demo Template**.
2. Click **Start** or **Stop**.

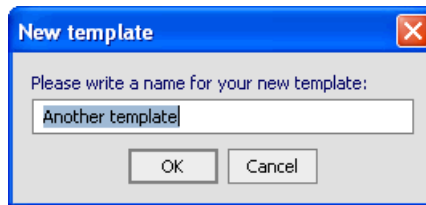
The numbers in the **Invocation count** cells for each method begin to increment as method calls are made. If you select **Stop**, this activity will cease.

Creating a New Method Profiling Template

1. Click **New template**.

The **New template** dialog box appears (Figure 6-16).

Figure 6-16 New Template Dialog Box



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

You can now add methods to your own template.

Removing a Method Profiling Template

1. Select the template you wish to remove.
2. Click **Remove**.

A confirmation dialog box appears.

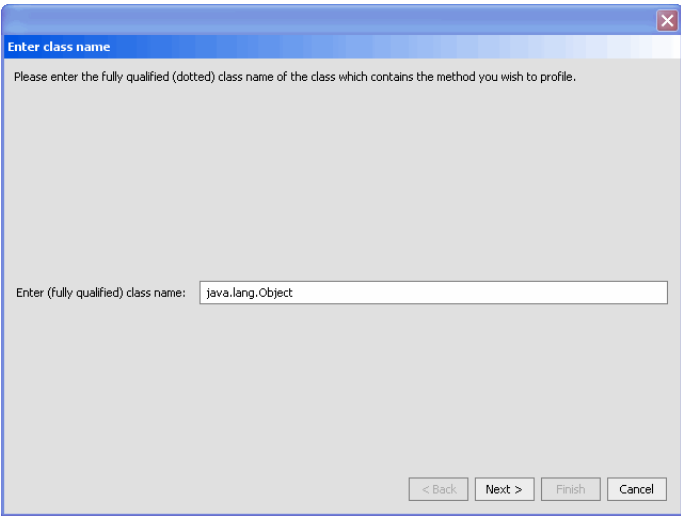
3. Click **Yes**.

Adding a Method to a Template

1. Select the template you wish to add methods to from the **Select template** list.
2. Click **Add Method**.

The **Enter class name** dialog appears (Figure 6-17).

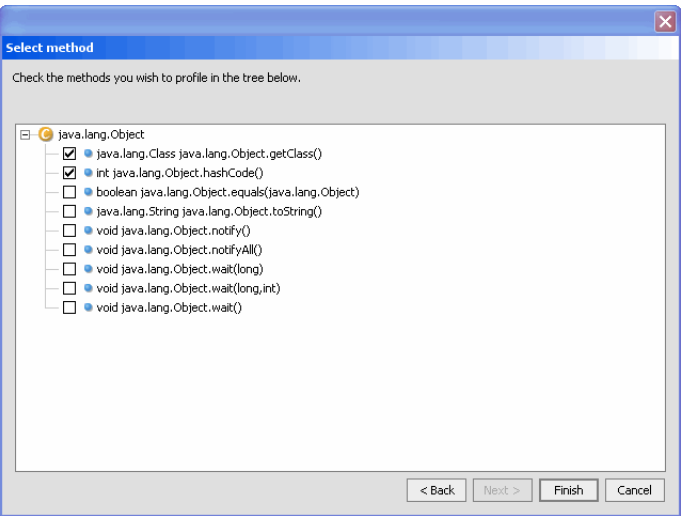
Figure 6-17 Enter Class Name Dialog



- 3. Enter a fully qualified class name, for example, `java.lang.Object` in the text field and click **Next**.

The **Select method** dialog appears (Figure 6-18):

Figure 6-18 Select Method Dialog



- 4. Select the methods to be added to the template and click **Finish**.

The method name will appear in the **Method** list for that template, as shown in [Figure 6-19](#).

Figure 6-19 Method Profiling Information List with Method Added

Method	Invocation count
Class Object.getClass()	?
StringBuffer StringBuffer.append(String)	259
int Object.hashCode()	?
void System.arraycopy(Object,int,Object,int,int)	2 673

Removing a Method from a Template

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

Method Profiling Settings

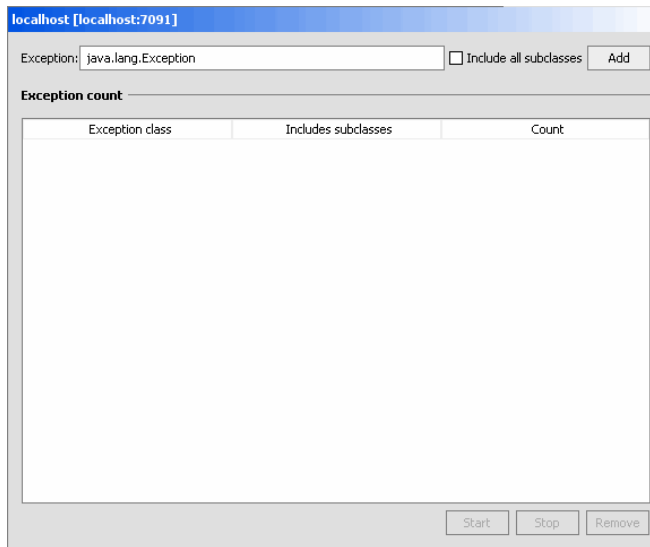
You can switch between using qualified method names or short names in the method profiling table.

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

Counting Exceptions in BEA JRockit

The **Exception Count** tab ([Figure 6-20](#)) shows exceptions thrown in BEA JRockit JVM. It counts the number of exceptions of a certain type. This information is helpful when you are troubleshooting your Java application.

Figure 6-20 Exception Counting Tab



Adding an Exception

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 all 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**.

Using Advanced Features of the Management 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 [Mode of Operation](#).

This section is divided into the following topics:

- [Historical Data](#)
- [Thread Stack Dump](#)
- [Starting and Running the Console in Headless Mode](#)
- [JRA Recordings](#)

Historical Data

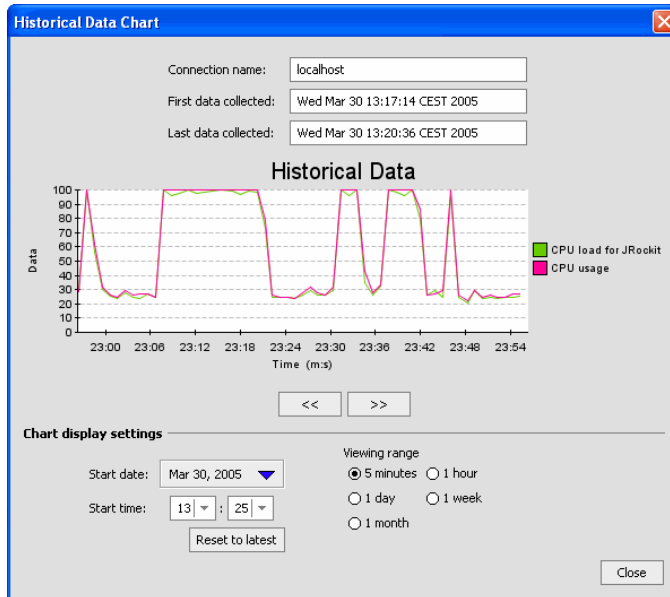
The historical data window displays a chart where historical data for an attribute subscription can be viewed. This is useful for observing trends over time and, for example, finding when a server running with BEA JRockit JVM has its peak loads.

Viewing Historical Data

1. Select the connection for which you want to view historical data.
2. Click **View > View Historical Data**.

The **Historical Data Chart** appears (see [Figure 7-1](#)).

Figure 7-1 Historical Data Chart



3. Right-click on the Historical Data graph.
The **Edit Graph Attributes** window appears.
4. Add an attribute subscription to the Historical Data Graph (see [Customizing Graphs](#) for information on how to change the graph).
5. Set a start date and start time for when you want to view the data.
6. Select a viewing range.
7. Scroll, using the arrow buttons below the graph, to find the time period of interest.

Thread Stack Dump

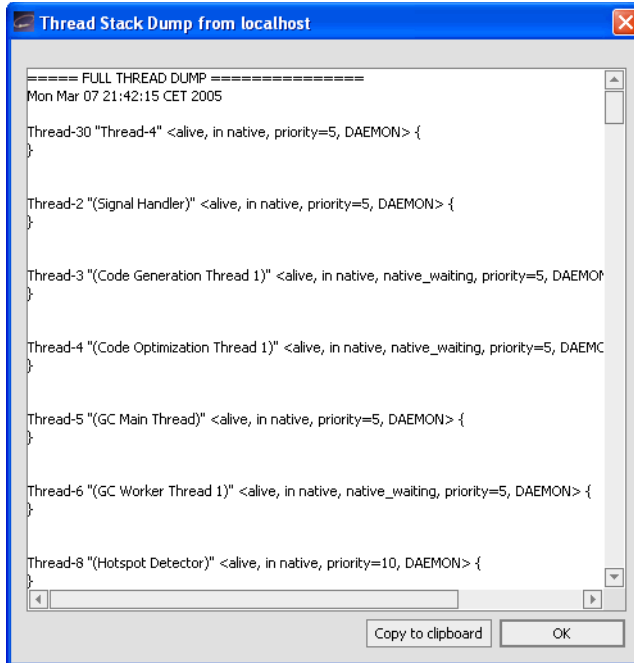
The thread stack dump contains a list of all running threads in BEA JRockit JVM with a method call stack trace for each thread.

Viewing the Thread Stack Dump

1. Click **Tools > View Thread Stack Dump**.

The **Thread Stack Dump** window appears (see [Figure 7-2](#)).

Figure 7-2 Thread Stack Dump



2. Click **Copy to Clipboard** if you want to paste the stack dump info into; for example, MSNotepad.
3. Click **OK** to close the file.

Starting and Running the Console in 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.

Look into the information on:

- [Running a Headless Management Console](#)
- [Controlling the Console with Command Line Options](#)

Running a Headless Management Console

To run the Console in the headless mode, start the console as you normally would (see “[Start the BEA 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 7-1](#).

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 7-1](#) to 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 7-1 Headless Management Console Command-line Options

Option	Description
<code>-headless</code>	Starts the console in the headless mode (won't load GUI related classes).
<code>-settings <settings file></code>	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.
<code>-connectall</code>	Makes all connections available in the settings file (that is, previously added by using the GUI).
<code>-connect <connection 1> <connection 2> <...></code>	Connects to the named connections available in the settings file previously added by using the GUI.

Table 7-1 Headless Management Console Command-line Options

<code>-autoconnect</code>	Automatically connects to any JVM running the management server with JDP turned on.
<code>-uptime <time in seconds></code>	Runs the Console for the specified amount of time, and then automatically shut it down.
<code>-useraction <name> <delay in seconds> <period (optional)></code>	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.
<code>-version</code>	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 an hour it will automatically shut down (`-uptime 3600`).
- After running 30 seconds, it will start issuing control breaks to all connected JVMs every minute (`-useraction ctrlbreak 30 60`).

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

JRA Recordings

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.

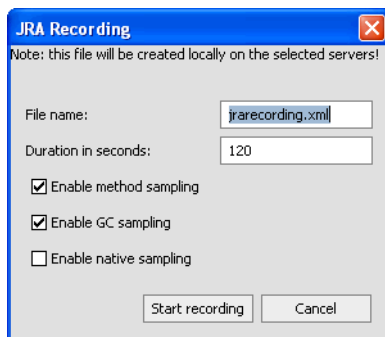
One part of the JRA runs inside the JVM, recording information about it and the Java application currently running. You start the recording in the Management Console and view the recorded result in the JRA. For information on how to use the JRA, see [Using the BEA JRockit Runtime Analyzer](#).

Creating a JRA Recording

1. Click **Plugins > Start JRA Recording**.

The **JRA Recording** dialog appears ([Figure 7-3](#)).

Figure 7-3 JRA Recording Dialog



2. Type a descriptive name for the recording in the **File name** field. This is the name by which the file is saved.
3. Type in the duration of the recording.
4. Select the type(s) of recording you want.
 - Enable method sampling
 - Enable GC sampling
 - Enable native sampling
5. Click **Start recording**.

The **JRA Recording Progress** box appears ([Figure 7-4](#)).

Figure 7-4 JRA Recording Progress Box



6. When the recording is complete, click **Done**.
You can cancel the recording at any time by clicking **Abort**.

Adding Custom Notification Actions and Constraints

After starting the Management Console for the first time, a file named `consolesettings.xml` is 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:

- [Creating a Custom Action](#)
- [Creating and Implementing a Notification Action—an Example](#)
- [Creating a Custom Constraint](#)

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 [Creating a New Rule](#)).

Creating and Implementing a Notification Action—an 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, 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`, you implemented the abstract method `handleNotificationEvent()`. This method acts on the incoming event and this is where you put the code representing the action you wish to take. From the event, you can access the `RJMXConnectorModule` via `getSource()` from which you can access the functionality available from the JVM.

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-2](#) shows how this is done.

Listing A-2 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-3](#).

Listing A-3 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 `<notification_actions>` element, as shown in [Listing A-4](#).

Listing A-4 Adding the New Action to the Deployment Entries

```

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

Index

Symbols

.csv 2-5

A

adding an exception 6-20

Administrator mode 5-5

anti-aliasing 5-5

aspect value change

 add constraint 6-13

 on recovery 6-11

 on trigger 6-11

asynchronos 2-5

attribute logs 5-6

attribute subscription 2-4, 2-5

 browser 2-5

attribute subscription browser 2-5

attribute subscriptions

 viewing 2-6

C

change scale of graph 6-3

changing graph attributes 6-4

command line options 7-4

 autoconnect 7-5

 connect 7-4

 connectall 7-4

 -Djrockit.managementserver.port 3-3

 headless 7-4

 settings 7-4

 uptime 7-5

 useraction 7-5

 version 7-5

 -Xmanagement 3-2

composite attribute 2-5

connecting a connection to JRockit 5-3

connection browser 4-2, 5-1

connection node 5-1

connection time 6-8

console overhead 1-1

consolesettings 6-16

consolesettings.xml 5-7, A-1

core update interval 5-7

CPU Load 6-7

CPU load bar 6-7

CPU usage 6-2

CPU usage chart 6-2, 6-7

customizing

 display 6-2

 gauges and bars 6-3

 graphs 6-3

D

dash board 6-1

default port 3-3

Developer mode 5-5

developer mode 4-3

disconnecting a connection from JRockit 5-3

E

e-mail settings 5-5

Enable Persistence 2-5

Enable Persistence 2-6

enabling console settings 5-4

Exception Count 5-5
Exception Count tab 6-19

F

free heap 6-6
free memory 6-6

G

garbage collection system 6-8
gauges 6-3
GC sampling 7-6
graphics setting 5-5
GUI slow 5-5

H

headless mode 7-3
heap usage chart 6-2, 6-5
hide a graph 6-3
hiding a disconnected connection 5-4
highest level measured 6-3
historical data 7-1

I

information tabs
 Memory tab 6-5
 Notification tab 6-9, 6-13
 Overview tab 6-1
 Processor tab 6-7
 System tab 6-7

J

Java heap memory 6-2
java.lang.management.ThreadMXBean 2-3
JRA 7-5
JRA recording 7-6
JRockit uptime 6-8
jrockit.management.ThreadMXBean 2-3

JVM Process Load 6-7

M

MBean 2-5
Memory tab 6-5
Method Profiler 5-5
Method Profiler tab 6-16
Method Profiling
 Method Profiling Information List 6-19
 settings 6-19
 starting and stopping 6-16
method sampling 7-6
Method Templates 6-16
 adding a method 6-17
 creating a new template 6-17
 Method Profiling Information List 6-19
 removing 6-17
 removing a method 6-19
method templates 6-16
mode of operation 5-5

N

native sampling 7-6
new connection 5-1
notification action 6-9
 Application alert 6-10
 creating a new rule 6-10
 editing a rule 6-13
 E-mail 6-10
 Log to file 6-10
 System out 6-10
notification constraint 6-9
Notification tab 6-9
notification trigger 6-9
Notifications tab A-1
number of processes 6-7
numerical attribute 2-5
nursery size 6-6

O

- operation mode 5-5
 - Administrator mode 5-5
 - Developer mode 5-5
- out of memory errors 6-6
- overhead 1-1
- Overview tab 6-1

P

- persistence directory 5-6
- persistence logs 5-6
- platform beans 2-2
- Process Affinity 6-8
- Processor tab 6-7
 - CPU load bar 6-7
 - CPU Usage 6-7
- profiling overhead 1-1

R

- remote X connection 5-5
- removing a connection 5-3
- removing a folder 5-3
- renaming a connection 5-3
- renaming a folder 5-3
- reply timeout 5-7
- running threads 6-8

S

- setting the port 3-3
- settings file 5-7
- slow GUI 5-5
- socket timeout 5-7
- starting, stopping, and removing an exception count 6-20
- status bar 4-2
- synthetic numerical attribute 2-5
- System Properties 6-8
- System tab 6-7

T

- tabbed interface 4-2
- thread stack dump 7-2
 - viewing 7-2
- timeout
 - reply 5-7
 - socket 5-7
- total heap 6-6
- total memory 6-6
- total nursery 6-6

U

- used heap 6-6
- used heap gauge 6-5
- used Java heap gauge 6-2
- used memory 6-6
- used memory gauge 6-5
- used physical memory gauge 6-2

W

- watermark 6-3

