BEA JRockit® Mission Control™

Introduction to Oracle JRockit Mission Control

JRockit Mission Control 3.0.2
Revised: January, 2008

Contents

Introduction to Oracle JRockit Mission Control

Architectural Overview of the JRockit Mission Control Client 1-2

Starting the JRockit Mission Control Client 1-3

The JVM Browser 1-3

The Management Console. 1-4

The JRockit Runtime Analyzer (JRA) 1-5

The Memory Leak Detector 1-6

JRockit Mission Control Communications 1-7

Changing Logging Properties 1-10

Frequently Asked Questions 1-14

Is There a Forum Where I can Discuss the JRockit Mission Control Plug-ins? 1-17

Giving Feedback To the JRockit Mission Control Development Team 1-17

CHAPTER 1*

Introduction to Oracle JRockit Mission Control

Oracle JRockit Mission Control is a suite of tools you can use to monitor, manage, profile, and eliminate memory leaks in your Java application without introducing the performance overhead normally associated with these types of tools.

JRockit Mission Control's low performance overhead is a result of using data collected as part of the Oracle JRockit JVM's normal adaptive dynamic optimization. This also eliminates the problem with the Heisenberg anomaly that can occur when tools using byte code instrumentation alters the execution characteristics of the system. JRockit Mission Control functionality can always be available on-demand and the small performance overhead is only in effect while the tools are running.

This introduction contains information on the following subjects:

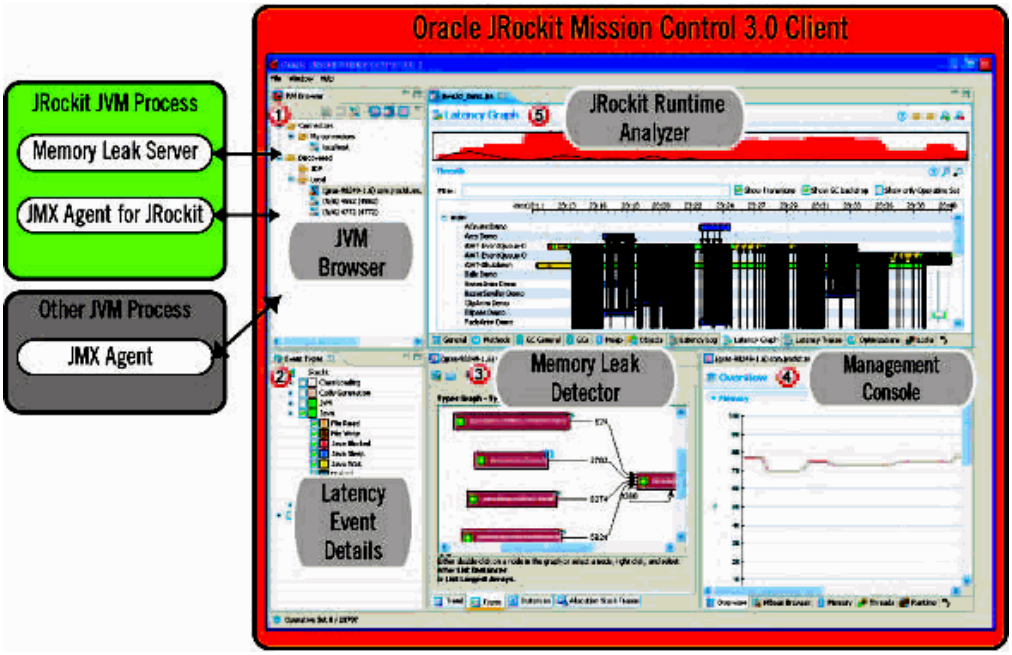
- Architectural Overview of the JRockit Mission Control Client
- Starting the JRockit Mission Control Client
- The JVM Browser
- The Management Console
- The JRockit Runtime Analyzer (JRA)
- The Memory Leak Detector
- JRockit Mission Control Communications
- Frequently Asked Questions

- Is There a Forum Where I can Discuss the JRockit Mission Control Plug-ins?
- Giving Feedback To the JRockit Mission Control Development Team

Architectural Overview of the JRockit Mission Control Client

With the Rich Client Platform (RCP) based JRockit Mission Control Client, you can launch the Memory Leak Detector, the JRockit Runtime Analyzer, and the Management Console from within the JRockit Mission Control Client. Figure 1-1 depicts how the JRockit Mission Control Client looks when all tools are loaded.

Figure 1-1 Architectural Overview of JRockit Mission Control



When a JRA recording is started from within the JRockit Mission Control Client, it records the status of the Oracle JRockit JVM process for the time that you have specified and creates a ZIP file containing an XML file with the recorded data and optionally a binary file with latency data together with the corresponding data producer specification files. The ZIP file is automatically opened in JRA upon completion of the recording, valid for JDK level 1.5 and later (marked 5 in

Figure 1-1). Typical information that is recorded during a JRA recording is Java heap distribution, garbage collections, method samples, and lock profiling information (optional). New for the JRockit Mission Control 3.0 release, is that you can also record thread latency data. When viewing Latency data in JRA, the Latency Events Details become visible (marked 2 in Figure 1-1).

To view real-time behavior of your application and of the JRockit JVM, you can connect to an instance of the JRockit JVM and view real-time information through the Management Console (marked 4 in Figure 1-1). Typical data that you can view is thread usage, CPU usage, and memory usage. All graphs are configurable and you can both add your own attributes and redefine their respective labels. In the Management Console you can also create rules that trigger on certain events, for example, an mail will be sent if the CPU reaches 90% of the size.

With the JMX Agent you have access to all MBeans deployed in the platform MBean server. From these MBeans, you can read attribute information, such as garbage collection pause times.

To find memory leaks in your Java application, you connect the Memory Leak Detector to the running the JRockit JVM process. The Memory Leak Detector connects to the JMX (RMP) Agent that instructs to start a Memory Leak Server where all further communication takes place.

Starting the JRockit Mission Control Client

The JRockit Mission Control Client executable is located in `JROCKIT_HOME/bin`. If this directory is on your system path, you can start the JRockit Mission Control Client by simply typing `jrmc` in a command (shell) prompt.

Otherwise, you have to type the full path to the executable file, as shown below:

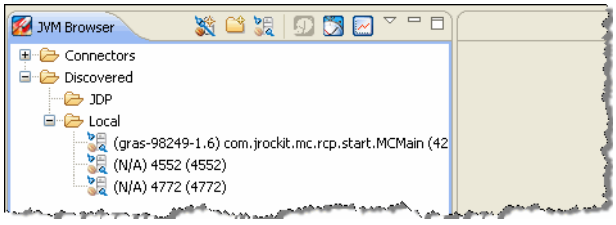
```
JROCKIT_HOME/bin/jrmc.exe (Windows)
JROCKIT_HOME\bin\jrmc (Linux)
```

On Windows installations, you can start the JRockit Mission Control Client from the **Start** menu.

The JVM Browser

The JVM Browser (see Figure 1-2) was added in the JRockit Mission Control 2.0 release. This tool allows you to set up and manage all running instances of the JRockit JVM on your system. From the JVM Browser you activate different tools, such as starting a JRA recording, connecting a Management Console, and starting memory leak detection. Each JRockit JVM instance is referred to as a *Connector*.

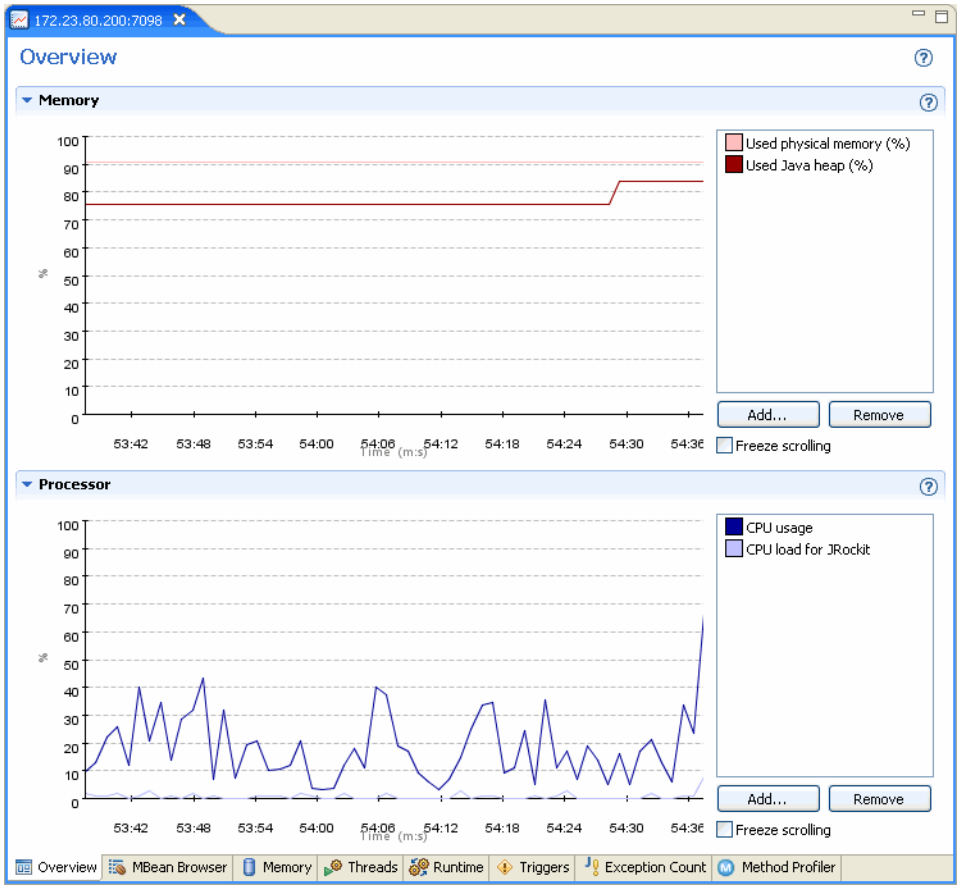
Figure 1-2 The JVM Browser



The Management Console

The Management Console (see Figure 1-3) is used to monitor a JRockit JVM instance. Several Management Consoles can be running concurrently side by side. The tool captures and presents live data about memory, CPU usage, and other runtime metrics. For the Management Console that is connected to the JRockit JVM 5.0, information from any JMX MBean deployed in the JRockit JVM internal MBean server can be displayed as well. For a Management Console connected to JRockit JVM 1.4, RMP capabilities are exposed by a JMX proxy. JVM management includes dynamic control over CPU affinity, garbage collection strategy, memory pool sizes, and more.

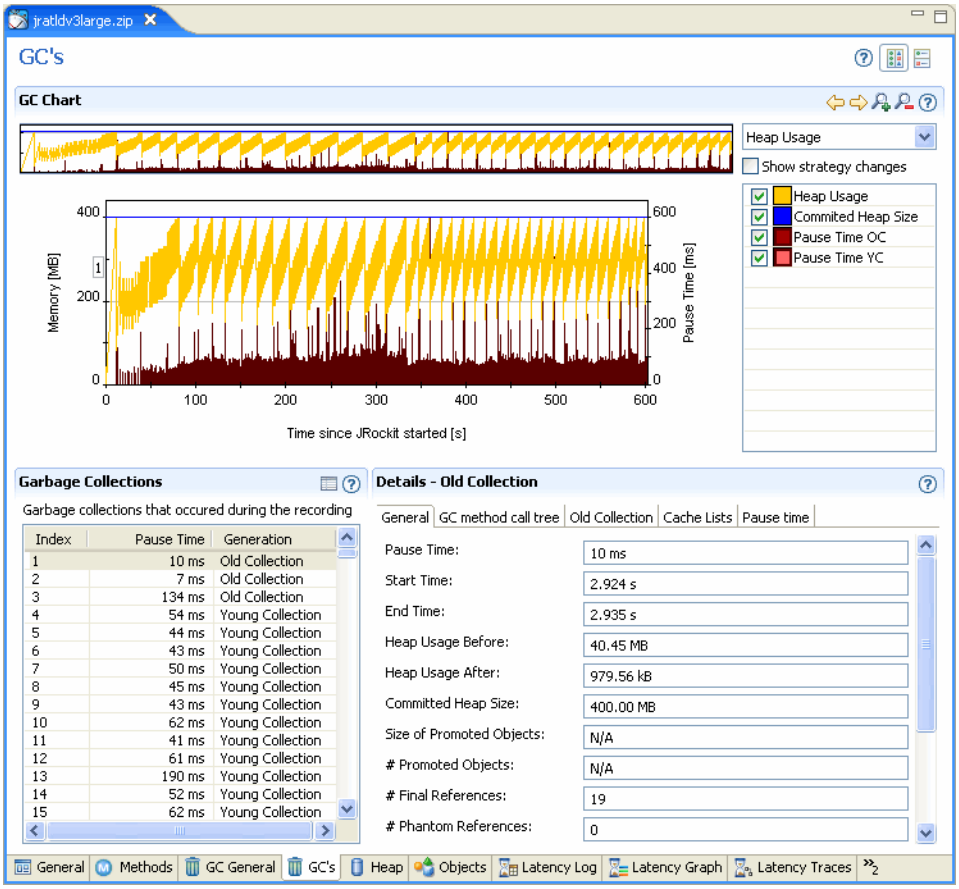
Figure 1-3 The Management Console



The JRockit Runtime Analyzer (JRA)

The JRockit Runtime Analyzer (see Figure 1-4) is an on-demand "flight recorder" that produces detailed recordings about the JRockit JVM and the application it is running. The recorded profile can later be analyzed off line, using JRA. Recorded data includes profiling of methods and locks, as well as garbage collection statistics, optimization decisions, and event latencies.

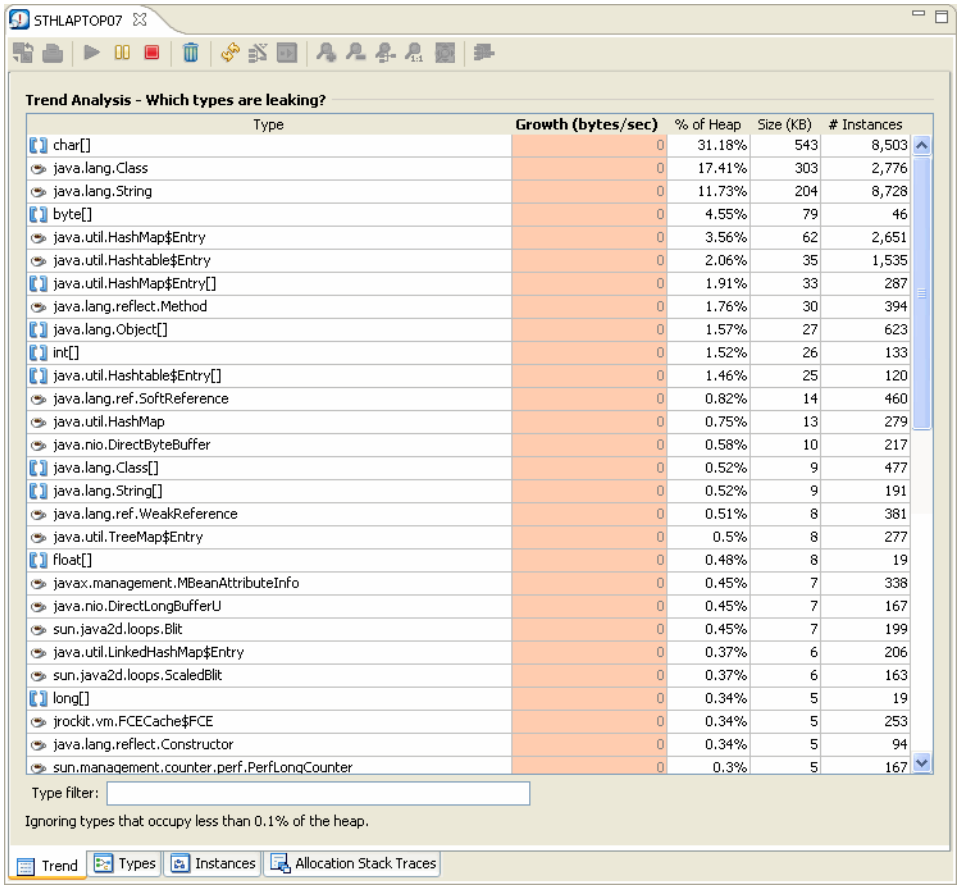
Figure 1-4 The JRockit Runtime Analyzer



The Memory Leak Detector

The Memory Leak Detector (see Figure 1-5) is a tool for discovering and finding the cause for memory leaks in a Java application. The Memory Leak Detector's trend analyzer discovers slow leaks, it shows detailed heap statistics (including referring types and instances to leaking objects), allocation sites, and it provides a quick drill down to the cause of the memory leak. The Memory Leak Detector uses advanced graphical presentation techniques to make it easier to navigate and understand the sometimes complex information.

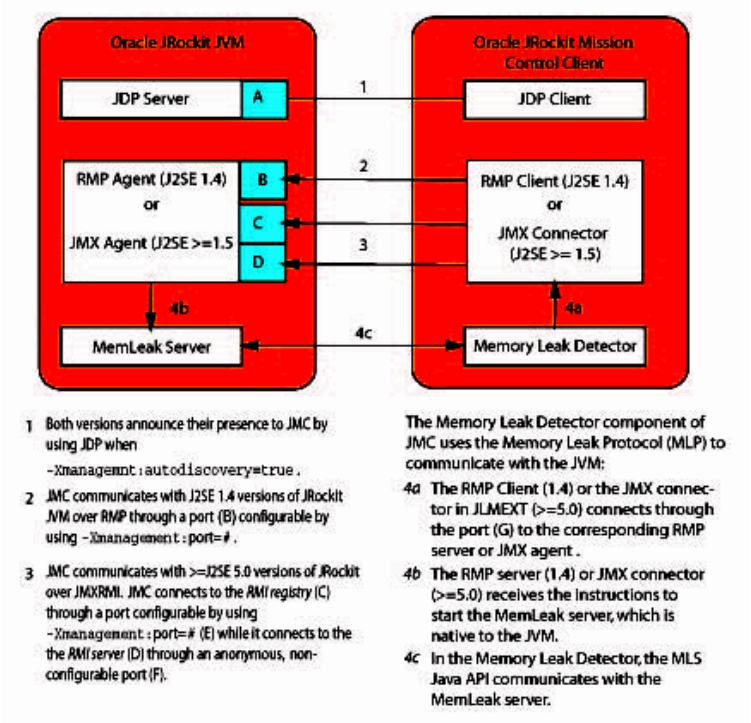
Figure 1-5 The Memory Leak Detector



JRockit Mission Control Communications

This topic describes the communication protocols for JRockit Mission Control.

Figure 1-6 JRockit Mission Control Communications Overview



J2SE 1.4

J2SE 1.4 versions of JRockit Mission Control uses *RMP* (Rockit Management Protocol.), an older legacy protocol that has existed since the 1.3 versions of the JRockit JVM. RMP uses a single socket. You can specify the port of the listening socket by using the `-Xmanagement:port` option; for example `-Xmanagement:port=7090`. Table 1-1 lists additional system properties you can use to further configure the agent.

Table 1-1 Additional Communication Settings for JRockit Mission Control on J2SE 1.4

System property	Description	Default
`jrockit.managementserver.address`	Bind to a specific interface	Not enabled, listens on all interfaces)

Table 1-1 Additional Communication Settings for JRockit Mission Control on J2SE 1.4

System property	Description	Default
`jrockit.managementserver.timeout`	Socket timeout in MS	4000
`jrockit.managementserver.maxconnect`	Maximum number of connections	4

J2SE 5.0 and Later

J2SE 5.0 and later versions of the JRockit JVM use *JMXRMI* (JMX over RMI). This protocol uses one port for the RMI registry, which is configured with the `-Xmanagement:port` option, and a second port (on an anonymous port) for communication with the RMI server. Note that you cannot configure the port for the RMI server; however, you can write your own agent that defines a fixed port for the RMI server. Please see the following link for further information:

`http://java.sun.com/javase/6/docs/technotes/guides/management/agent.html#gdfvv.`

Table 1-2 lists the options available for the `-Xmanagement` flag:

Table 1-2 -Xmanagement Option

Option	Description	Default
`authenticate`	Use password authentication	True
`ssl`	Use secure sockets layer	True
`port`	What port to use for the RMI registry	7091

For a more comprehensive discussion on what these options mean, please see:

`http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html.`

All Versions

For all J2SE versions, you can use the `-Xmanagement` option `autodiscovery` to make the JRockit JVM use the JRockit Discovery Protocol (JDP) to announce its presence; for example `-Xmanagement:autodiscovery=true`.

Table 1-3 lists additional system properties you can use to control the behavior of the JDP server:

Table 1-3 System properties used to control the JDP server

System property	Description	Default
`jrockit.managementserver.discovery.period`	The time to wait between multicasting the presence in ms	5000
`jrockit.managementserver.discovery.ttl`	The number of router hops the packets being multicasted should survive	1
`jrockit.managementserver.discovery.address`	The multicast group/address to use	232.192.1.212
`jrockit.managementserver.discovery.targetport`	The target port to broadcast	7090(1.4)/7091(1.5)

All versions of JRockit Mission Control also employ an additional protocol when using the Memory Leak Detector. The memleak server is not written in Java; rather it is an integral part of the JRockit JVM. This is because a potential use case for the memleak server is to optionally be able to start it when an out of memory condition occurs in the JVM. When such a condition occurs, it is impossible to execute Java code because no heap would be available.

MLP (MemLeak Protocol) is used by the native memleak server during a memleak session. JRockit Mission Control communicates over RMP (1.4) or JMXRMI (5.0 and higher) to ask the JRockit JVM to start up the server. You can configure the port on which you want to start the memleak server on, and to use for the session, by using JRockit Mission Control preferences.

Changing Logging Properties

You can change the way the JRockit Mission Control Client internally logs such items as error messages and warning messages from the JRockit Mission Control Client system by updating the logging.properties file. By default, you JRockit Mission Control Client implementation will use the logging.properties file that is stored in com.jrockit.mc.core. You can either create a new version of this file or change the settings in the original. Whichever option you choose, you should move the file to a common location, such as your root directory.

To modify and move the logging.properties file:

1. Go to your file system and locate the logging.properties file at com.jrockit.mc.core.
2. Make a copy of this file to serve as a baseline and save it to a common location, such as your root directory.
3. Open the original version of logging.properties in a text or code editor. Listing 1-1 shows an example of what you should see.

   Note: The first time you open logging.properties from some versions of Windows Explorer, you might have to select an program in which to open the file. Follow the instructions on the Windows dialog box that appears when you try to open the file. Check **Select the program from a list** and click **OK**. On the Open With dialog box, select the desired program. When you attempt to open logging.properties in the future, Windows will automatically use the application selected here.

Listing 1-1 logging.properties Default Example

```
############################################################
#      Default Logging Configuration File
#
# You can use a different file by specifying a filename
# with the java.util.logging.config.file system property.
# For example java -Djava.util.logging.config.file=myfile
############################################################

############################################################
#      Global properties
############################################################

# "handlers" specifies a comma separated list of log Handler
# classes.  These handlers will be installed during CorePlugin
# startup.
# Note that these classes must be on the system classpath.
handlers= java.util.logging.FileHandler, java.util.logging.ConsoleHandler

# Default global logging level.
# This specifies which kinds of events are logged across
# all loggers.  For any given facility this global level
```

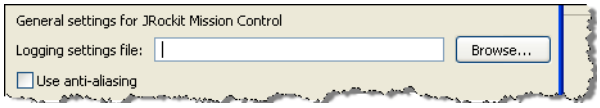
```
# can be overriden by a facility specific level
# Note that the ConsoleHandler also has a separate level
# setting to limit messages printed to the console.
.level= WARNING

############################################################
# Handler specific properties.
# Describes specific configuration info for Handlers.
############################################################

# default file output is in user's home directory.
java.util.logging.FileHandler.pattern = %h/mc_%u.log
java.util.logging.FileHandler.limit = 50000
java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.formatter =
java.util.logging.SimpleFormatter

# Limit the message that are printed on the console to INFO and above.
java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter =
java.util.logging.SimpleFormatter

############################################################
# Facility specific properties.
# Provides extra control for each logger.
############################################################
#com.jrockit.mc.browser.level = WARNING
```

4. Change any property you want to To see guidelines and additional information on setting logging properties, please refer to the documentation for java.util.logging.
5. Save and close logging.properties.
6. In Eclipse, open **Windows>Preferences**.

   The Preferences dialog box appears.
7. Select **JRockit Mission Control**.

The right pane changes to show the location of the logging settings file for the JRockit Mission Control Client (Figure 1-7).

Figure 1-7 Logging Settings File Location



8. Do one of the following:
   - Type the path to logging.properties; for example, C:/logging.properties.

   OR

   a. Click **Browse** to open the Open dialog box.

   b. Navigate to modified version of logging.properties and, once you've selected logging.properties, click **Open**.

9. In the Preferences dialog box, click **OK**.
10. Close and restart the JRockit Mission Control Client.

To create a new logging.properties file

1. In the folder where you want to store the logging.properties file, create a new file and save it as logging.properties.
2. At [MISSION_CONTOL_HOME]/com.jrockit.mc.core, open the logging.properties file and copy the contents (as shown in Listing 1-1) to you clipboard.
3. Paste the copied contents from your clipboard into the logging.properties file you created in step 1.
4. Change any property you want to To see guidelines and additional information on setting logging properties, please refer to the documentation for java.util.logging.
5. Save and close logging.properties.
6. In Eclipse, open **Windows>Preferences**.

   The Preferences dialog box appears.

7. Select **JRockit Mission Control**.

The right pane changes to show the location of the logging settings file for the JRockit Mission Control Client (Figure 1-7).

Figure 1-8 Logging Settings File Location

8. Do one of the following:
   - Type the path to logging.properties; for example, C:/logging.properties.

   OR

   a. Click **Browse** to open the Open dialog box.

   b. Navigate to modified version of logging.properties and, once you've selected logging.properties, click **Open**.

9. In the Preferences dialog box, click **OK**.

10. Close and restart the JRockit Mission Control Client.

Frequently Asked Questions

This topic lists and provides answers to questions frequently asked about the JRockit Mission Control Client.

- I cannot connect the JRockit Mission Control Client. What could be the problem?
- When attempting to connect the JRockit Mission Control Client I get a stack trace indicating that JRockit Mission Control attempts to communicate with a strange IP or host name.
- I'm getting exceptions during startup about classes not being found
- JRockit Mission Control can't find any local JVMs
- Why can't I see any Method Profiling information in my JRA recording?
- When using the Memory Leak Detector, nothing happens in the growth column of the trend table.

I cannot connect the JRockit Mission Control Client. What could be the problem?

Consider the following:

- Have you started the management server? You must do so if you want to enable your application for remote monitoring or if you want to monitor an instance of a JRockit JVM running with JDK 1.4. You can start the management server by adding the -Xmanagement option to your Java command line. SSL and authentication are available in JDK 1.5/1.6 and will be enabled by default. If you do not want to set up certificates, SSL and authentication can be disabled by providing `ssl=false` and `authenticate=false`. Also, if you want to use the remote discovery feature of JRockit, you can enable it by setting `autodiscovery=true`; for example:

  ```
  java -Xmanagement:ssl=false,authenticate=false,autodiscovery=true
  ```

  You can also start the management server on an already running JRockit JVM by using the jrcmd utility available in the `JROCKIT_HOME/bin` directory.

- Are you using the correct protocol?

  The easiest way is to ensure that you are using the same version of the JRockit JVM you want to monitor as the JRockit JVM running the JRockit Mission Control Client. If that is not an option, you can use the radio buttons in the connection dialog box in JRockit Mission Control to select which protocol to use: 1.4 will select RMP and 1.5 and later will select JMXRMI.

  For earlier versions of the JRockit Mission Control Client these radio buttons don't exist and, to make a 1.5 JRockit JVM instance connect to a 1.4 version, you must explicitly specify the JMX Service URL. The format of the service URL is:

  ```
  service:jmx:rmp://<hostname>:<port>
  ```

  for example:

  ```
  service:jmx:rmp://localhost:7091
  ```

- Are the correct ports opened?

  Note that JMX over RMI uses two ports and that one of the ports will not be known beforehand.

- Maybe the communication is caught in the firewall?

  Please see JRockit Mission Control Communications for more information.

When attempting to connect the JRockit Mission Control Client I get a stack trace indicating that JRockit Mission Control attempts to communicate with a strange IP or host name.

Sometimes RMI can have a problem determining which address to use. This can happen because of

- Access restrictions in the Security manager,
- The machine being multihomed and RMI picking the wrong interface
- A misconfigured hosts file or a number of different network related configuration problems.

If all else fails you can try specifying the `java.rmi.server.hostname` system property. Please note that this can affect applications running in the JRockit JVM.

I'm getting exceptions during startup about classes not being found

Make sure you are using the proper launcher to start up the JRockit Mission Control Client. You must **only** use `JROCKIT_HOME/bin/jrmc`.

JRockit Mission Control can't find any local JVMs

Make sure you are using the proper launcher to start up the JRockit Mission Control Client. You must **only** use `JROCKIT_HOME/bin/jrmc`.

Why can't I see any Method Profiling information in my JRA recording?

By default, the JRockit Mission Control Client doesn't show tabs if no data has been recorded for them. Ensure that method profiling was enabled for your JRA recording and that the application was under load. If the JRockit JVM is spending most of the time with none of the threads doing any work, no samples will be recorded. If you still want to create a JRA recording with method sampling and a low load, try increasing the sampling frequency.

When using the Memory Leak Detector, nothing happens in the growth column of the trend table.

The algorithm needs at least three data points to kick in and the data is collected as part of the old space mark phase of the garbage collection. If you see no data, possibly not enough garbage has been collected for these collections to occur. To speed up the process, try clicking the garbage can in the tool bar of the Memory Leak Detector to force three successive garbage collections, with a brief pause in between each collection.

Is There a Forum Where I can Discuss the JRockit Mission Control Plug-ins?

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

http://newsgroups.bea.com

Giving Feedback To the JRockit Mission Control Development Team

If you have any suggestions about how to improve the JRockit Mission Control plug-ins or information on how it is most commonly used in your development environments, we would be grateful to receive your input. This information would contribute to our understanding on how to best further improve these tools in the future.

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

`mailto:jrockit-improve@oracle.com`

The feedback will be considered by the development team designing the JRockit Mission Control plug-ins. We will look at collected ideas and improve the plug-ins to make them even easier to use. our goal with these plug-ins is to simplify the tasks in getting your applications to run as smoothly as possible on JRockit JVM.