

# **Oracle® JRockit Real Time**

Introduction to Oracle JRockit Real Time

3.0

June 2008

**ORACLE®**

Copyright © 2007, 2008, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

# Contents

## 1. Overview

What is Oracle JRockit Real Time? . . . . .	1-1
Oracle JRockit Real Time Compatibility . . . . .	1-2
Oracle JRockit JDK Features in JRockit Real Time 3.0 . . . . .	1-2
JRockit Latency Analysis Tool . . . . .	1-3
JRockit Memory Leak Detector . . . . .	1-3
Starting JRockit Mission Control . . . . .	1-3
Example Use Cases . . . . .	1-4
Derivative Exchange Defies Arbitrage Traders . . . . .	1-4
Competition-Beating Risk Calculation Infrastructure . . . . .	1-5
Software Components . . . . .	1-5
Oracle JRockit JDK 6.0 R27.6 . . . . .	1-5
Oracle JRockit JDK 5.0 R27.6 . . . . .	1-6
Oracle JRockit JDK 1.4.2 R27.6 . . . . .	1-6
Deterministic Garbage Collection . . . . .	1-6
Enabling the Deterministic Garbage Collector . . . . .	1-7
JRockit Runtime Analyzer (JRA) . . . . .	1-7
Supported Configurations for JRockit Real Time . . . . .	1-8
Terminology . . . . .	1-8

## 2. Tuning Real Time Applications for Deterministic Garbage Collection

Basic Environment Tuning . . . . .	2-2
Basic Application Tuning . . . . .	2-2
J2EE Application Tuning . . . . .	2-3
JMS Application Tuning . . . . .	2-3
JVM Tuning for Real-Time Applications . . . . .	2-4
Allow For a Warm-up Period . . . . .	2-4
Adjust Min/Max Heap Sizes . . . . .	2-4
Increase or Decrease Pause Targets . . . . .	2-5
Set the Page Size . . . . .	2-5
Determine Optimal Load . . . . .	2-5
Analyze GC With JRockit Verbose Output . . . . .	2-5
Limit Amount of Finalizers and Reference Objects . . . . .	2-6
Adjust the Garbage Collection Trigger . . . . .	2-6
Adjust the Amount of Garbage Collection Threads for Processors . . . . .	2-6
More Tuning Information . . . . .	2-7
JRockit JVM . . . . .	2-7

## 3. Using Oracle JRockit Real Time with Other Oracle Products

Getting Started . . . . .	3-1
Oracle WebLogic Server . . . . .	3-2
WebLogic Event Server . . . . .	3-2

# Overview

This section contains information on the following subjects:

- [What is Oracle JRockit Real Time?](#)
- [Oracle JRockit Real Time Compatibility](#)
- [Oracle JRockit JDK Features in JRockit Real Time 3.0](#)
- [Example Use Cases](#)
- [Software Components](#)
- [Supported Configurations for JRockit Real Time](#)
- [Terminology](#)

## What is Oracle JRockit Real Time?

Oracle JRockit Real Time provides lightweight, front-office infrastructure for low latency, event-driven applications. For companies in highly-competitive environments where performance is key and every millisecond counts, JRockit Real Time provides the first Java-based real-time computing infrastructure.

For example, for certain types of applications, particularly in the Telecom and Finance industries, stringent requirements are placed on transaction latency. When these applications are written in Java, the unpredictable pause times caused by garbage collection can have a profound and potentially harmful affect on this latency.

For this reason, Oracle JRockit Real Time’s proprietary Oracle JRockit JVM R27.6 features *deterministic garbage collection*, a dynamic garbage collection priority that ensures extremely short pause times and limits the total number of those pauses within a prescribed window. Such short pauses can greatly lessen the impact of the deterministic garbage collection when compared to running a normal garbage collection.

## Oracle JRockit Real Time Compatibility

Oracle JRockit Real Time 3.0 is fully compatible with Oracle JRockit JDK R27.6 and all applications certified with the latter will work on the former without need for additional certification. This means that all Oracle applications supported by Oracle JRockit JDK R27.6 (whether Java 1.4.2-, 5.0- or 6-based) are also supported with Oracle JRockit Real Time 3.0.

Oracle JRockit Real Time 3.0 also supports standalone Java applications running on Java SE 6 and J2SE 1.4.2 and 5.0 runtime environments, as well as Spring Framework-based applications, as described in [“Software Components” on page 1-5](#).

Oracle JRockit Real Time 3.0 performance still depends upon application type and size, so you will need to verify that your applications meet the base requirements on hardware, heap size and other metrics for optimal performance.

## Oracle JRockit JDK Features in JRockit Real Time 3.0

JRockit Real Time 3.0 is bundled with the following versions of the Oracle JRockit JDK:

- Oracle JRockit JDK 6 Update 5
- Oracle JRockit JDK 5.0 Update 15
- Oracle JRockit JDK 1.4.2\_17

These versions of the Oracle JRockit JDK are included in Oracle JRockit Mission Control, which is a suite of tools designed to monitor, manage, profile, and gain insight into problems occurring in your Java application without requiring the performance overhead normally associated with these types of tools.

JRockit Mission Control includes the following two tools that are of particular interest to Oracle JRockit Real Time 3.0 users:

- [JRockit Latency Analysis Tool](#)
- [JRockit Memory Leak Detector](#)

## JRockit Latency Analysis Tool

The Latency Analysis Tool (LAT), part of the JRockit Runtime Analyzer (JRA) helps you work your way down to a Java application latency. You can use the **Latency Graph** to visually see how a Java application that contains latencies looks like. This tool gives you great flexibility to pinpoint where in the code waits and other latencies occur.

To record latency data, you need to create a JRA recording. Before you start the JRA recording, you must select one of the Latency Recording profiles in order to record latency data.

See [Oracle JRockit Runtime Analyzer](#) for additional information about using the latency analyzer and JRA recordings to record latency data. After you launch Oracle JRockit Mission Control, you can also access additional documentation about this feature using online help.

## JRockit Memory Leak Detector

The JRockit Memory Leak Detector is a tool for discovering and finding the cause for memory leaks in a Java application. The JRockit 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.

See [Introduction to JRockit Memory Leak Detector](#) for additional information about using the memory leak detector. After you launch JRockit Mission Control, you can also access additional documentation about this feature using online help.

## Starting JRockit Mission Control

To start JRockit Mission Control, follow these steps:

1. Ensure that your `JAVA_HOME` environment variable points to the root folder of the Oracle JRockit JDK included in JRockit Real Time 3.0.

Use this command for Windows platforms:

**Note:** This example assumes that you are using the Windows Command Prompt (DOS) or compatible command shell and have selected the default product installation directory.

```
set JAVA_HOME=%ProgramFiles%\JRockit Real Time
\wlr<wlr_version>-<java_version>\bin
```

Use this command for Linux and Solaris platforms:

**Note:** This example assumes that you are using the UNIX bash shell or compatible command shell and have selected the default product installation directory.

```
export JAVA_HOME=$HOME/wlrt<wlrt_version>-<java_version>/bin
```

2. Open up a command window.
3. Run the `jrmc` executable file, located in the `%JAVA_HOME%\bin` directory:

```
(Windows) prompt> %JAVA_HOME%\bin\jrmc
```

```
(Linux) prompt> ${JAVA_HOME}/bin/jrmc
```

## Example Use Cases

These use cases provide examples of how JRockit Real Time can provide solutions for high-performance environments with response-time sensitive applications.

### Derivative Exchange Defies Arbitrage Traders

An investment arm of a large retail bank provides an exchange for derivatives of European securities. It is an over-the-counter (OTC) request-for-quote and execution system (but provides no settlement and clearing services). A broker submits a request for a quotation and includes the investment identifier and quantity. The system accepts the quotation and applies certain business rules. Depending upon the investment identifier and market conditions, the request is routed to a particular third-party market-maker who then calculates and provides the bid and ask price for the derivative. The response is returned to the broker via the OTC exchange. The broker can then execute the trade of the derivative through a subsequent request, which is routed via the OTC exchange to the appropriate market maker.

The complication with this arrangement is that arbitrage traders can take advantage of the latency delay in the bank's OTC exchange infrastructure because the arbitrage trader can measure the latency that occurs during the small period in which the request for quotation is handled. In a fast moving market, price changes of the derivative may occur within this latency period. This presents an opportunity for an arbitrage trader to take advantage of inefficiency in the marketplace and expose the investment bank to intolerable risk.

The investment bank requires a very high performance-driven software infrastructure, such as JRockit Real Time. It requires that the latency of the OTC exchange be extremely low. Specifically, to combat arbitrage traders, the latency of the exchange's infrastructure must be less than the latency of the arbitrage traders' infrastructure. In this way, the arbitrage traders' data becomes stale before the exchange's, and therefore is not actionable.



## Competition-Beating Risk Calculation Infrastructure

A large investment bank is a market-maker for fixed income securities. A request-for-quote (RFQ) is received from an inter-dealer market electronic communication network (ECN), such as TradeWeb. This RFQ would have been submitted to a number of entities. To be competitive, it is vital that the quotation is returned as quickly as possible with the best possible price.

Therefore, a minimum amount of latency is necessary to ensure that the investment bank wins customers, or at least, the latency is less than that of the organization's competitors.

During the quotation process, a risk and pricing model is executed to determine the quote price to provide to the customer. Because of the complexity of these calculations, they are currently performed overnight. The result is a stratum of at least four grades of risk advisories that govern fixed rate securities prices. Note that there is at least a twelve-hour lag in these risk calculations. This leads to a risk window since the calculations are stale even at the start of next-day business. To lower this risk, and potentially provide better rates to customers, a real-time risk and pricing calculator would be required. JRockit Real Time provides a latency-adverse infrastructure to make this feasible.

## Software Components

JRockit Real Time supports Java applications running on such Oracle products as (but not limited to) Oracle WebLogic Event Server 2.0, Oracle WebLogic Server 10.0 (or higher), and Oracle WebLogic Server 8.1. It also supports standalone Java applications running on Java SE 6 and J2SE 5.0 and 1.4.2 runtime environments.

JRockit Real Time includes the following software components:

- [Oracle JRockit JDK 6.0 R27.6](#)
- [Oracle JRockit JDK 5.0 R27.6](#)
- [Oracle JRockit JDK 1.4.2 R27.6](#)
- [Deterministic Garbage Collection](#)
- [JRockit Runtime Analyzer \(JRA\)](#)

## Oracle JRockit JDK 6.0 R27.6

The Oracle JRockit JDK 6.0 R27.6 is certified to be compatible with Java SE 6 (update 3). This version includes the Deterministic Garbage Collector for dynamic garbage collection priority that ensures extremely short pause times and limits the total number of those pauses within a

prescribed window, as described in [“Deterministic Garbage Collection” on page 1-6](#). It also installs the JRockit Runtime Analyzer (JRA), Latency Analysis Tool (LAT), and Memory Leak Detector, which provide internal metrics that are useful for profiling the Oracle JRockit JVM, as described in [“JRockit Runtime Analyzer \(JRA\)” on page 1-7](#).

For a listing of the hardware and software configurations supported by JRockit Real Time, see [“Supported Configurations for JRockit Real Time” on page 1-8](#).

## Oracle JRockit JDK 5.0 R27.6

The Oracle JRockit JDK 5.0 R27.6 is certified to be compatible with J2SE 5.0 (update 14). The 5.0 R27.3 JVM includes the Deterministic Garbage Collector for dynamic garbage collection priority that ensures extremely short pause times and limits the total number of those pauses within a prescribed window, as described in [“Deterministic Garbage Collection” on page 1-6](#). It also installs the JRockit Runtime Analyzer (JRA), Latency Analysis Tool (LAT), and Memory Leak Detector, which provide internal metrics that are useful for profiling the JRockit JVM, as described in [“JRockit Runtime Analyzer \(JRA\)” on page 1-7](#).

For a listing of the hardware and software configurations supported by JRockit Real Time, see [“Supported Configurations for JRockit Real Time” on page 1-8](#).

## Oracle JRockit JDK 1.4.2 R27.6

The Oracle JRockit JDK 1.4.2 R27.6 is certified to be compatible with J2SE 1.4.2\_16. The 1.4.2 R27.6 JVM includes the Deterministic Garbage Collector for dynamic garbage collection priority that ensures extremely short pause times and limits the total number of those pauses within a prescribed window, as described in [“Deterministic Garbage Collection” on page 1-6](#). It also installs the JRockit Runtime Analyzer (JRA), which provides internal metrics for Java developers using the JRockit JVM as their runtime JVM, as described in [“JRockit Runtime Analyzer \(JRA\)” on page 1-7](#).

For a listing of the hardware and software configurations supported by Oracle JRockit Real Time, see [“Supported Configurations for JRockit Real Time” on page 1-8](#).

## Deterministic Garbage Collection

Memory management relies on effective *garbage collection*, which is the process of clearing dead objects from the heap, thus releasing that space for new objects. JRockit Real Time uses a dynamic “deterministic” garbage collection priority (`-Xgcprio:deterministic`) that is

optimized to ensure extremely short pause times and limit the total number of those pauses within a prescribed window.

For certain types of applications, particularly in the Telecom and Finance industries, stringent requirements are placed on transaction latency. When these applications are written in Java, the unpredictable pause times caused by garbage collection can have a profound and potentially harmful affect on this latency.

However, shorter deterministic pause times do not necessarily equal higher throughput. Instead the goal of the deterministic garbage collection is to lower the *maximum* latency for applications that are running when garbage collection occurs. Such shorter pause times should lessen the impact of the deterministic garbage collection compared to running a normal garbage collection.

For more information on the deterministic garbage collector, see the Oracle JRockit [Diagnostics Guide](#).

## Enabling the Deterministic Garbage Collector

For standalone or Spring-Based Java applications, enable the Deterministic Garbage Collector by doing one of the following:

- Enter the `-Xgcprio:deterministic` option from a Java command line.
- Use the sample startup scripts, `startRealTime (.cmd/.sh)`, that demonstrate how to start the Oracle JRockit JVM with deterministic garbage collection enabled.

## JRockit Runtime Analyzer (JRA)

The JRockit Runtime Analyzer (JRA) is an application that helps you profile your application and the Java runtime. It provides a wealth of useful metrics that are useful when using the JRockit JVM as your runtime VM.

The JRockit Runtime Analyzer consists of two parts. One part runs inside the JVM and records information about the currently running JVM and the Java application and saves this information to a JRA recording file (`filename.jra`). This file is opened in the other part of the tool, the analyzer, which is a regular Java application used to visualize the information in the file.

The JRocking Runtime Analyzer is packaged as part of the JRockit Mission Control 3.0 tool suite. Documentation for Mission Control 3.0 is bundled with the tools as online documentation. For general information about Mission Control 3.0, see [Introduction to Oracle JRockit Mission Control](#).

# Supported Configurations for JRockit Real Time

Oracle JRockit Real Time is supported on the same configurations (hardware and platform) as Oracle JRockit JVM R27.6, with the exception of Windows Itanium and Linux Itanium configurations: it *is not* supported on those configurations. For a complete list of supported configurations, please refer to Oracle JRockit [Supported Configurations](#).

## Terminology

[Table 1-1](#) defines the terms and acronyms used this document:

**Table 1-1 Terminology**

Terms	Definition
Real-time	A level of computer responsiveness that a user senses as sufficiently immediate or that enables the computer to keep up with some external process (for example, to present visualizations of the weather as it constantly changes).
Latency	An expression of how much time it takes for data to get from one designated point to another.
Throughput	The amount of work that a computer can do in a given time period.
Deterministic garbage collection	Short, predictable pause times for memory heap garbage collection, which is the process of clearing dead objects from the heap, thus releasing that space for new objects.

# Tuning Real Time Applications for Deterministic Garbage Collection

This section contains the following guidelines for tuning your applications for the Oracle JRockit JVM deterministic garbage collector that is included with Oracle JRockit Real Time.

**Note:** For more information on adjusting other non-standard start-up commands available with JRockit, see the JRockit [Configuration and Tuning Guide](#).

- “Basic Environment Tuning” on page 2-2
- “Basic Application Tuning” on page 2-2
- “J2EE Application Tuning” on page 2-3
- “JMS Application Tuning” on page 2-3
- “JVM Tuning for Real-Time Applications” on page 2-4
- “More Tuning Information” on page 2-7

# Basic Environment Tuning

Use these guidelines for configuring your environment to use Oracle JRockit Real Time.

- *Ensure that CPUs are not at maximum capacity out on servers or clients*  
If an application takes a majority of the CPU, then the deterministic GC performance may actually degrade the average latency. The reason is that deterministic GC will do continuous GC and the GC will be competing with the application for CPU cycles. It is best that the CPU is not fully utilized to get the best latency. A best practice is to run your benchmarks at various loads (with and without deterministic GC) to determine the optimal load.
- *Too many active threads can cause increased latency due to context switching*  
The “sweet-spot” number is generally one thread per virtual CPU (i.e., counting dual-core and HyperTransport as separate CPUs), but leaving one CPU free for background GC work. However, if you make external calls (e.g., to a database), then it does make sense to allocating a few extra threads to utilize idle cycles.

For information on tuning JRockit garbage collection threads, see [“Adjust the Amount of Garbage Collection Threads for Processors” on page 2-6](#).

# Basic Application Tuning

Use these guidelines when designing your applications for Oracle JRockit Real Time.

- Understand your application code and how to measure latency.
- Avoid making synchronous calls to slow back-office systems as part of a transaction as this defeats the purpose of real-time. Conversely, make sure any non-critical calls are handled asynchronously through work thread pools, or by using JMS.
- Minimize memory allocation. If possible, allocate and free memory for a single transaction in a *chunk* as this helps avoid fragmentation of the Java heap. Also, minimize the amount and size of your objects.
- Control memory utilization by avoiding rampant memory allocation and allocating many large arrays.
- Free all objects as soon as possible; otherwise, objects that become unreferenced during a garbage collection might still be marked alive if they were referenced when the DetGC marked all live objects.

- Avoid long critical sections in your code, as synchronized blocks of Java code may cause a transaction to block.
- Avoid long linked structures; the deterministic GC needs to iterate through these objects.
- If transactions span more than one highly-active JVM, each such JVM may need to run Deterministic GC. For example, if a transaction is initiated by a Java client JVM, and the transaction includes both JMS server and J2EE server operations, all three JVMs may require Deterministic GC to reliably meet maximum latency criteria.

## J2EE Application Tuning

Use these guidelines when tuning your J2EE applications for Oracle JRockit Real Time.

- For server-side EJBs, MDBs, and Servlets ensure that there are enough concurrent instances configured to respond immediately to client requests (if all instances are active, this is a sign that client requests are queuing up behind each-other on the server).
- Make sure that resource pools contain enough instances so that threads are not forced to wait for resources. In J2EE for example, tune the EJB `max-beans-in-free-pool` property and tune thread pool sizes

## JMS Application Tuning

Use these guidelines when using Oracle WebLogic JMS applications with JRockit Real Time.

- Consider using asynchronous consumers rather than synchronous consumers.

For more information on JMS consumers, see [Best Practices for Application Design](#) in *Programming WebLogic JMS*.

- Tune all JMS connection factory Messages Maximum settings to 1. This can potentially provide better latency at the expense of possibly lowering throughput. Similarly, configure your MDBs to refer to a custom connection factory with the following settings:
  - Messages Maximum = 1
  - XA Connection Factory Enabled = `enabled`
  - Client Acknowledge Policy = `ACKNOWLEDGE_PREVIOUS`

For more information on configuring JMS connection factories, see [Configure connection factories](#) in the *Administration Console Online Help*.

- For consumers of non-persistent messages from queues, consider using the WebLogic JMS `WLSession NO_ACKNOWLEDGE` extension.
  - Ensure that your Spring JMS Templates leverage resource reference pooling (otherwise, they negatively impact response times as they implicitly create and close JMS connections, sessions, and producers once per message).
- Note:** Resource reference pooling is not suitable if the target destination changes with each call, in which case change application code to use *regular* JMS and cache the JMS connections, sessions, producers, and consumers.

## JVM Tuning for Real-Time Applications

These tuning suggestions can further improve performance and decrease pause times when using the JRockit JVM deterministic garbage collector. For more information on the deterministic garbage collector, see the Oracle JRockit *Diagnostics Guide*.

### Allow For a Warm-up Period

There may be a *warm-up period* before response times achieve desired levels. During this warm-up, JRockit JVM will optimize the critical code paths. The warm-up period is application and hardware dependent, as follows:

- For smaller applications (in terms of amount of Java code) with high loads that are running on fast hardware, there may be a warm-up period of one-to-three minutes.
- For large applications (in terms of amount of Java code) with low loads that are running on slow hardware (in particular, most SPARC hardware), there may be a warm-up period of approximately thirty minutes.

### Adjust Min/Max Heap Sizes

Setting the minimum heap size (`-Xms`) smaller or the maximum heap size (`-Xmx`) larger affects how often garbage collection will occur and determines the approximate amount of live data an application can have. To begin with, try using the following heap sizes:

```
java -Xms1024m -Xmx1024m -XgcPrio:deterministic -XpauseTarget=30
```

For more information, see [-X Command-line Options](#) in the Oracle JRockit *JVM Command-Line Reference*.



## Increase or Decrease Pause Targets

- If you specify `-Xgcprio:deterministic` without the `pauseTarget` option, it will be set to a default value, which in this release is 30 milliseconds.
- Running on slower hardware with a different heap size and/or with more live data may break the deterministic behavior. In these cases, you might need to increase the default pause time target (30 milliseconds) by using the `-XpauseTarget` option. The maximum allowable value for the `pauseTarget` option is currently 5000 milliseconds.
- Conversely, if you want to test your application for the lowest possible pause time, you can lower the default `-XpauseTarget` value down to a minimum value. In this release, the minimum value is 10 milliseconds.

For more information, see [-X Command-line Options](#) in the Oracle JRockit *Command-Line Reference*.

## Set the Page Size

Increasing the page size (`-XXlargePages`) can increase performance and lower pause times by limiting cache misses in the translation look-aside buffer (TLB). See [-XX Command-line Options](#) in the Oracle JRockit *JVM Command-Line Reference*.

## Determine Optimal Load

Do not be overcautious in terms of load. The deterministic garbage collector can handle a fair amount of load without breaking its determinism guarantees. Too little load means the JVM's optimizer and GC heuristics have too little information to work with, resulting in sub-par performance. A best practice is to run your benchmarks at various loads (with and without deterministic GC) to determine the optimal load.

## Analyze GC With JRockit Verbose Output

JRockit JVM verbose output normally doesn't incur a measurable performance impact, and is quite useful for analyzing JVM memory and GC activity. [Table 2-1](#) defines recommended verbose options for analyzing JVM memory and GC activity.

**Table 2-1 JRockit JVM Verbose Output Options**

Option	What it does...
<code>-Xverbose:opt,memory,memdbg,gcpause,compact,license</code>	For GC and memory analysis.
<code>-Xverboselog:verbose-jrockit.log</code>	Redirects verbose output to the designated file.
<code>-Xverbosetimestamp</code>	Prints a formatted date before each verbose line.

## Limit Amount of Finalizers and Reference Objects

Try to limit the amount of Finalizers and reference objects that are used, such as `Soft-`, `Weak-`, and `Phantom-` references. These types require special handling, and if they occur in large numbers then pause times can become longer than 30ms.

## Adjust the Garbage Collection Trigger

Try adjusting the garbage collection trigger (`-XXgctrigger`) to limit the amount of heap space used. This way, you can force the garbage collection to trigger more frequent garbage collections without modifying your applications. The garbage collection trigger is somewhat deterministic, since garbage collection starts each time the trigger limit is hit. See the [Oracle JRockit JVM Diagnostics Guide](#).

**Note:** If the trigger value is set to low, the heap might get full before the garbage collection is finished, causing even longer pauses for threads since they have to wait for the garbage collection to complete before getting new memory. Typically, memory is always available since a portion of the heap is free and any pauses are just the small pauses when the garbage collection stops the Java application.

## Adjust the Amount of Garbage Collection Threads for Processors

With the variety of sophisticated processing hardware currently available (HyperTransport, Strands, Dual Core, etc.), the JRockit JVM may not be able to determine the appropriate number of GC threads it should start. The current recommendation is to start one thread per physical CPU; that is, one thread per chip not per core. However, having too many garbage collection threads could affect the latency of applications since more threads will be running on the system, which might saturate the CPUs, and thus affect the Java application. Conversely, setting them too low

could increase the mark phase of the GC, since less parallelism is possible. For example, on a dual core Intel Woodcrest machine with four cores the recommended number of GC threads is two, which is the same as the number of processors in the machine.

To see how many garbage collection threads that the JRockit JVM uses on your machine, start the JRockit JVM with `-verbose:memdbg` and then check for the following lines that are printed during startup:

```
[memdbg ] number of oc threads: <num>
[memdbg ] number of yc threads: <num>
```

If necessary, adjust the number of GC threads using the `-XXgcthreads:<# threads>` parameter.

For more information, see [-XX Command-line Options](#) in the Oracle JRockit *JVM Command-Line Reference*.

## More Tuning Information

This section contains pointers to additional performance and tuning information.

### JRockit JVM

- [Memory Management Basics](#) contains information on all of the garbage collection options.
- [About Profiling and Performance Tuning](#) provides information on tuning the JRockit JVM.

See [Oracle JRockit JVM Diagnostics Guide](#) for additional diagnostic information about the Oracle JRockit JVM.



# Using Oracle JRockit Real Time with Other Oracle Products

Oracle JRockit Real Time is fully compatible with Oracle JRockit JVM R27.6 and is therefore supported by Oracle when used with Oracle products that are supported with Oracle JRockit JVM R27.6. This chapter describes the tasks necessary for using JRockit Real Time with other Oracle products. It includes information on the following subjects:

- [Getting Started](#)
- [Oracle WebLogic Server](#)
- [WebLogic Event Server](#)

## Getting Started

To use JRockit Real Time with an Oracle product, you must download and install the Oracle product and JRockit Real Time separately and then make the following configuration changes:

- Update the Oracle product start script to use the `WLRT JAVA_HOME`.
- Remove any `-Xgc:` and `-Xgcprio:` parameters from the Java command line.
- Add `-Xgcprio:deterministic` and `-Xpausetarget=nnms` (where *nn* is the desired pause target) to the Java command line

Selecting a lower pause target will result in shorter garbage collection pauses, with the caveats outlined in [Tuning the Pause Target](#) in the Oracle JRockit *JVM diagnostics Guide*. Here are some recommendations for the most common combinations:

## Oracle WebLogic Server

Oracle WebLogic Server can be enabled to use JRockit Real Time either by following the generic instructions above or by using a preconfigured domain template. Oracle has prepared templates for the following Oracle WebLogic Server versions:

- Oracle WebLogic Server 8.1 domain template.
- Oracle WebLogic Server 10.0 domain template.

Oracle has verified that larger J2EE applications (including SPECjAppServer2004) works well with a 30ms pause targets. Lower pause targets might be possible for smaller J2EE applications or faster hardware.

## WebLogic Event Server

The Oracle WebLogic Server has been extensively optimized for use with Oracle JRockit Real Time, so a good starting point for the pausetarget is 10 ms. Oracle has verified that small WLEvS applications on recent x86 hardware work with pausetargets down to 3 ms. For more information on tuning Oracle WebLogic Server with JRockit Real Time, see the [Oracle WebLogic Server documentation](#).