BEA JRockit

Introduction to BEA JRockit SDK

Version 1.4.2
December 2004

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

Overview of BEA JRockit SDK

Understanding BEA JRockit SDK

What's in BEA JRockit SDK

New Features in BEA JRockit SDK

Using the Documentation

CHAPTER 1

Overview of BEA JRockit SDK

BEA JRockit SDK provides tools, utilities, and a complete runtime environment for developing and running applications using the Java programming language. The BEA JRockit SDK includes the BEA JRockit Java Virtual Machine (JVM), the first commercial server-side JVM. The BEA JRockit JVM is developed and optimized for Intel architectures to ensure reliability, scalability, and manageability for Java applications.

This section contains information on the following subjects:

What is BEA JRockit?

The BEA JRockit JVM is the only high performance JVM developed to ensures reliability, scalability, manageability, and flexibility for Java applications. The BEA JRockit JVM delivers a new level of performance for Java applications deployed on Intel 32-bit (Xeon) and 64-bit (Itanium) architectures at significantly lower costs to the enterprise. Furthermore, it is the only enterprise-class JVM optimized for Intel architectures, providing seamless interoperability across multiple hardware and operating system configurations. BEA JRockit JVM makes it possible to gain optimal performance for your Java applications when running it on either the Windows or Linux operating system platforms on either 32-bit and 64-bit architectures.

For more information on JVMs in general, see the Introduction to the JVM specification at:

http://java.sun.com/docs/books/vmspec/2nd-edition/html/Introduction.doc.html#3057

About the SDK

The BEA JRockit JVM is one component of the BEA JRockit software development kit (SDK). Along with the BEA JRockit JVM, this kit is comprised of the Java Runtime Environment (JRE), which contains the JVM and Java class libraries (as specified by the Java 2 Platform API Specification), along with a set of development tools, such as a compiler, a debugger, and so on. For more information on the contents of the BEA JRockit 1.4.2 SDK, please refer to "What's in BEA JRockit SDK."

What Platforms Does BEA JRockit Support?

BEA JRockit SDK is certified to be compatible with J2SE 1.4.2. For a complete list of platforms that BEA JRockit supports, see BEA JRockit 1.4.2 Supported Configurations at:

http://e-docs.bea.com/wljrockit/suppPlat/supp_142.html

Compatibility Between Releases

Continued compatibility with the Java SDK (and other listed products) is assured as defined in Compatibility Between Releases, at:

http://e-docs.bea.com/wljrockit/suppPlat/prodsupp.html#999010

BEA JRockit SDK Support

This section describes how to get support for BEA JRockit SDK.

If BEA JRockit JVM Crashes

If BEA JRockit crashes, it will dump information about the crash to `stderr` and create, in the directory where the VM was started, a file containing the same information, called `jrockit.<pid>.dump` (and, if you are using Windows, `jrockit.<pid>.mdmp)`, where `<pid>` is the id of the process that crashed.

After a crash, send a copy of `jrockit.<pid>.dump` (and, if you are using Windows, `jrockit.<pid>.mdmp)`, along with as much information as possible about your system setup and the application you were running when the VM crashed, to **`support@bea.com`**. You should provide the following information:

- Hardware
- Version of BEA JRockit JVM
- Operating system and its version
- The character of the application you ran and other information that can be of interest for trouble shooting
- Stack dumps (if any)
- If possible, a small code example that will reproduce the error.

CHAPTER 2

Understanding BEA JRockit SDK

BEA JRockit SDK has a number of important features that separate it from other JVMs on the market today. This section provides high-level descriptions of some of the more critical SDK features to help you better understand what they can do. This section includes information on the following subjects:

- The SDK
- The Java Runtime Environment
- The BEA JRockit Management Console
- Code Generation and Optimization
- Memory Management (Garbage Collection)

The SDK

BEA JRockit SDK is very similar to the Sun JDK, except that it includes a new JRE with the BEA JRockit JVM and some changes to the Java class libraries (however, all of the class libraries have the same behavior in BEA JRockit as in the Sun JDK).

The Java Runtime Environment

The BEA JRockit implementation of the Java 2 runtime environment for use by the SDK. The runtime environment includes the BEA JRockit JVM, class libraries, and other files that support the execution of programs written in the Java programming language.

The BEA JRockit Management Console

The Management Console connects to the BEA JRockit JVM and provides real-time information about server behavior and resource availability, such as memory usage and profiling information. This gives you a powerful way of retrieving constant profile data about your application.

The Management Console provides a unique advantage when deploying a commercial Java solution because it gives you greater control of the complex set of interrelated variables that may affect your application in production. Administrators can monitor the BEA JRockit JVM operating characteristics and the Java application, and be automatically notified of changes in resource availability or operating characteristics as they occur. Based on this information, administrators can identify bottlenecks in performance and change operating and environmental parameters to optimize performance and availability.

For more information on the Management Console, please refer to Using the BEA JRockit JVM Management Console.

Code Generation and Optimization

BEA JRockit differs from most JVMs in that it compiles the code at first use by implementing a JIT ("Just In Time") compiler. This ensures desirable application performance from the outset, albeit at the cost of a slightly longer start-up time. To expedite start-up, however, BEA JRockit does not use all possible compiler optimizations. While doing so might lead to even better performance early in the application run, that would result in slower start-up.

Compilation time is part of application execution time, thus compiling all of the methods with all available optimizations also negatively impacts application performance. Therefore, BEA JRockit does not fully optimize all methods at start-up; in fact, it leaves many methods unoptimized throughout the entire application run. Instead, BEA JRockit chooses those functions whose optimization will most benefit application performance and only optimizes those methods.

BEA JRockit can thus be seen to have two distinct, but cooperating, code generators: a JIT compiler, which resolves data from bytecode, through three levels of intermediate representation, to native code (assembly language); and an optimizing compiler, which optimizes targeted methods at each level of intermediate representation.

BEA JRockit uses a "sampler thread" to identify which functions merit optimization. This thread wakes up at periodic intervals and checks the status of several application threads. It identifies what each thread is executing and notes some of the execution history. This information is tracked for all the methods and when it is perceived that a method is experiencing heavy us—-in other words, is "hot"—that method is earmarked for optimization. Usually, a flurry of such

optimization opportunities occur in the application's early run stages, with the rate slowing down as execution continues.

The optimizing compiler in BEA JRockit includes many of the best-known techniques for code generation, particularly for IA64 machines. This includes a sophisticated register allocator that takes full advantage of IA64's large register stacks.

Memory Management (Garbage Collection)

Garbage collection is the process of clearing dead objects from the Java heap to releasing that space for new objects, which, in turn, ensures efficient processing. BEA JRockit allows you to select a dynamic garbage collector that bases its collection algorithm on one of two priorities: memory throughput or duration of pause time caused by collection. The unified garbage collector is *dynamic* in that, as it runs, it uses predefined heuristics to determine which collection algorithm to use and changes that algorithm as the individual case might warrant. You do not need to specify the actual algorithm to run this garbage collector.

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

For information on selecting and using garbage collectors, see Using the BEA JRockit Memory Management System.

CHAPTER 3

What's in BEA JRockit SDK

BEA JRockit SDK is very similar to the Sun JDK, except that it includes a new JRE with the BEA JRockit JVM and some changes to the Java class libraries (however, all of the class libraries have the same behavior in BEA JRockit as in the Sun JDK). For a more detailed description of the differences between the two SDKs, please refer to File Differences Between BEA JRockit SDK and Sun HotSpot SDK.

This section describes the contents of the BEA JRockit 1.4.2 SDK and compares a BEA JRockit SDK installation to a comparable Sun SDK installation. It includes information on the following subjects:

- SDK Contents
- File Differences Between BEA JRockit SDK and Sun HotSpot SDK

SDK Contents

This section describes the various components that make up the BEA JRockit 1.4.2 SDK. It also identifies the folder in which these components reside.

Development Tools

Found in: `/bin`

Development tools and utilities help you develop, execute, debug, and document programs written in the Java programming language. The BEA JRockit 1.4.2 SDK includes the standard tools commonly distributed with the typical Java SDKs. While most of these are standard JDK tools and are proven to work well with Java development projects, you are free to use any other

third party tools, compilers, debuggers, IDEs, and so on that might work best in your situation. The tools included with BEA JRockit 1.4.2 SDK are:

- `Javac` compiler
- `Jdb` debugger
- Javadoc, which is used to create an HTML documentation site for the JVM API

For more information on these tools, please refer to Sun Microsystem's Java™ 2 SDK Tools and Utilities website at:

`http://java.sun.com/j2se/1.4/docs/tooldocs/tools.html`

Additional Libraries

Found in: `/lib`

Additional class libraries and support files required by the development tools.

C Header Files

Found in: `/include`

Header files that support native-code programming using the Java Native Interface, the Java Virtual Machine Debugger Interface, the Java Virtual Machine Profiler Interface and other functionality of the Java 2 Platform.

The Management Console

Found in: `/console`

The BEA JRockit Management Console is used to monitor and control running instances of BEA JRockit JVM. It provides real-time information about the running application's characteristics, which can be used both during development-for example, to find where in an application's life cycle it consumes more memory-and in a deployed environment-for example, to monitor the system health of a running application server.

Java Runtime Environment (JRE)

The BEA JRockit implementation of the Java 2 runtime environment for use by the SDK. The runtime environment includes the BEA JRockit JVM, class libraries, and other files that support the execution of programs written in Java.

Java Virtual Machine

By definition, the JVM is BEA JRockit JVM, as described in this documentation set.

Standard J2SE JRE Features

In addition to JRE components specific to BEA JRockit SDK (found in `/jre`), the JRE also contains components found in the Sun implementation of the JRE. Table 3-1 lists these components.

Table 3-1 Standard J2SE Components

Component	Description	For Details
Core APIs		
XML	APIs supporting the use of XML-formatted data.	XML in the Java™ 2 Platform
Logging	APIs that facilitate software service and maintenance at customer sites by producing log reports suitable for analysis by end users, system administrators, field service engineers, and software development teams.	Java™ Logging APIs
JavaBeans	An API that extends the Java platform's Write Once, Run Anywhere™ capability to reusable component development.	JavaBeans™ Component API
Locale Support	APIs that enable application internationalization, the process of designing an application so that it can be adapted to various languages and regions without engineering changes.	Internationalization
Preferences	An API that provides a way for applications to store and retrieve user and system preference and configuration data.	Preferences API
Collections	A unified architecture for representing and manipulating collections, allowing them to be manipulated independently of the details of their representation.	The Collections Framework
JNI	The interface for writing Java native methods and embedding the JVM into native applications.	Java Native Interface
Security	APIs for program security, cryptography, signing, and so on.	Security

Table 3-1 Standard J2SE Components

Component	Description	For Details
Lang	APIs providing fundamental classes for the designing the Java™ programming language.	The `java.lang.*` and `java.util.*` Packages Documentation Contents
Util	APIs providing support for the event model, collections framework, date and time facilities.	The `java.lang.*` and `java.util.*` Packages Documentation Contents
New I/O	New I/O (NIO) APIs introduced in J2SE 1.4 to provide new features and improved performance in the areas of buffer management, scalable network and file I/O, character-set support, and regular-expression matching.	New I/O APIs
Networking	APIs that provide support for sockets, URL authentication, and so on.	Networking Features
Integration APIs		
RMI	Interfaces and classes responsible for specifying the remote behavior of the Remote Method Invocation (RMI) system, as defined in the `java.rmi` package hierarchy.	Java™ Remote Method Invocation (RMI)
JDBC	An API that provides universal data access from the JavaTM programming language. Using the JDBC 3.0 API, you can access virtually any data source, from relational databases to spreadsheets and flat files.	JDBC™ API Documentation
JNDI Toolkit (AWT) supports Graphical User Interface (GUI) programming.	An interface that provides naming and directory functionality to applications written in Java. This API is independent of any naming or directory service implementation	Java Naming and Directory Interface™
CORBA	An Object Request Broker (ORB) and two CORBA programming models that use the Java CORBA ORB and Internet InterORB Protocol (IIOP).	CORBA Technology and the Java™ 2 Platform, Standard Edition
User Interface Tools		
Swing	One of the JavaTM Foundation Classes (JFC) that implements a set of GUI components with a pluggable look and feel.	Project Swing (Java™ Foundation Classes) Software

Table 3-1 Standard J2SE Components

Component	Description	For Details
AWT	A toolkit that supports Graphical User Interface (GUI) programming.	Abstract Window Toolkit (AWT)
Sound	An API for capturing, processing, and playing back audio and MIDI (Musical Instrument Digital Interface) data.	Sound
Input Methods	An input method framework that enables the collaboration between text editing components and input methods in entering text.	Input Method Framework
Java 2D	An API comprised of classes for advanced 2D graphics and imaging. It encompasses line art, text, and images in a single comprehensive model.	Java 2D™ Technology
Accessibility	APIs, utilities, and other components for making applications accessible to disabled persons.	Java™ Accessibility
Deployment		
Java Plug-in	A component that enables applets written to the Java 2 Platform 1.4 specification to be run in Netscape Navigator and Microsoft Internet Explorer web browsers.	The Java™ Plug-in Component

File Differences Between BEA JRockit SDK and Sun HotSpot SDK

This section describes how BEA JRockit SDK differs from Sun Microsystems' HotSpot SDK. Each table below lists, by component and operating system (O/S), files that either exist in HotSpot SDK 1.4.2 or BEA JRockit SDK.

Be aware of these variables:

- `$ARCH` = i386 on linux32, but `$ARCH` = ia64 on linux64
- `mydir[/*]` means mydir and mydir/*

Table 3-1 Files Contained Only in Sun SDK; Not Supported by BEA JRockit

Component	O/S	File
Sun JCOV	win32 win64	`jre/bin/jcov.dll` `jre/lib/jvm.jcov.txt`
	linux32 linux64	`jre/lib/$ARCH/libjcov.so` `jre/lib/jvm.jcov.txt`

Table 3-2 Files Used Only in Sun SDK; Not used by BEA JRockit

Component	O/S	Files

Sun Hotspot VM support files	win32
	win64
		`jre/bin/msvcrt.dll`
		`jre/bin/nio.dll`
		`jre/bin/zip.dll`
		`jre/bin/hpi.dll`
		`jre/bin/server[/*]`

	win32	`jre/bin/client[/*]`
	win64	`jre/bin/msvcrtd.dll`

	linux32
	linux64
		`jre/lib/$ARCH/libnio.so`
		`jre/lib/$ARCH/libzip.so`
		`jre/lib/$ARCH/libjsig.so`
		`jre/lib/$ARCH/server[/*]`

	linux32	`jre/lib/$ARCH/client[/*]`

Table 3-3 Files Contained Only in Sun SDK; Removed from BEA JRockit to Reduce Package Size

Component	O/S	Files	Notes
Sun Demo sources	win32 win64 linux32 linux64	`demo[/*]`	See BEA JRockit 1.4.2 SDK Sources and Demos Download Page

Table 3-3 Files Contained Only in Sun SDK; Removed from BEA JRockit to Reduce Package Size

Sun Java API sources	win32 win64 linux32 linux64	`src.zip`	See BEA JRockit 1.4.2 SDK Sources and Demos Download Page

Table 3-4 Files Only in BEA JRockit SDK; Not Used or Supported by Sun

Component	O/S	Files

BEA JRockit Management Console	linux32
	linux64
	win32
	win64
		`bin/console[.exe]`
		`bin/jrcc[.exe]`
		`console[/*]`

BEA JRockit Licensing	linux32 linux64 win32 win64	`jre/jrockit.license`
BEA JRockit Management API	win32 win64 linux32 linux64	`jre/lib/managementapi.jar`

BEA JRockit JVM support files	win32
	win64
		`jre/bin/dbghelp.dll`
		`jre/bin/jrockit[/*]`
		`jre/lib/managementserver.jar`

	win32	`jre/bin/msvcr71.dll`
	win64	`jre/bin/msvcrt.dll`
	linux32 linux64	`jre/lib/$ARCH/jrockit[/*]` `jre/lib/managementserver.jar`

CHAPTER 4

New Features in BEA JRockit SDK

BEA JRockit SDK represents a incremental step up from previous version of this SDK. Now compatible with JDK version 1.4.2, BEA JRockit contains many new features designed to improve performance and streamline operation. Note that some features provided in this version of BEA JRockit SDK are non-standard or "experimental." These features, while tested to work with this version of the SDK, are provided as-is and might change at any time.

This section includes information on the following subjects:

Standard Features

This section describes standard features introduced in BEA JRockit SDK. "Standard features" are fully-supported by BEA Systems and should not change without appropriate notice. These features include:

- Headless Management Console
- New JRE Packaging
- Support for Browser Plugin on IA32 Machines
- Java.* Source Files and Demos Available
- Java Web Start
- GC Support for Profiling Interface
- man Pages Available for Linux Tools

Improved IPF Performance

BEA JRockit supports Intel Corp's. Itanium Platform Family (IPF) of 64-bit processors. This version on BEA JRockit has been enhanced to improve its performance with these processors.

New Default Parameter Settings

BEA JRockit supports the `-server` and `-client` startup parameters.

- `-server` starts BEA JRockit JVM as a server-side JVM. This value is the default.
- `-client` starts BEA JRockit JVM as a client-side JVM. This option is helpful if you have a smaller heap and are anticipating shorter runtimes for your application.

Setting the JVM type (or accepting the default) will also set the garbage collection algorithm that will be used during runtime. For more information on garbage collection and the `-server` and `-client` options, please refer to "Setting the Default Garbage Collector" in Using BEA JRockit Memory System.

Native POSIX Thread Library Available with Linux

The Native POSIX Thread Library (NPTL) is a thread library option available for use instead of LinuxThreads with Red Hat Enterprise Linux 3.0. NPTL provides significant performance improvement for multi-threading Linux applications. For more information on NPTL, please see the white paper "The Native POSIX Thread Library for Linux", by Ulrich Drepper (this file will open as a `.pdf` file; you will need Adobe Acrobat or comparable PDF reader to view this file).

Warning: While this NPTL is fully-supported by BEA JRockit 1.4.2_04, it is not fully supported by BEA JRockit 1.4.2_03. If you are using that version of the JVM and

your system has NPTL enabled, BEA JRockit might throw an error at runtime as it will try to use this library automatically. To avoid the error, either disable NPTL by setting `LD_ASSUME_KERNEL=2.4.1` in your environment or upgrade your version of BEA JRockit to 1.4.2_04.

New Heap Sizing Behavior

In BEA JRockit, automatic heap resizing behavior has been enhanced to allow more dynamic and adaptive heap usage. These enhancements are particularly evident when minimum and maximum heap sizes are not specified (the default case) but they also affect the behavior when these values are set.

Shrinking the Heap

The heap can now shrink when the system runs low on memory and BEA JRockit is using less than 50% of its heap (that is, the amount of live objects after a garbage collection is less than 50% of the heap size). In BEA JRockit 8.1 and earlier, the heap could only shrink when running with the generational copy garbage collector, which has been deprecated in BEA JRockit 1.4.2. Now the heap can shrink when running any garbage collector; however, the heap will not shrink below the initial specified size if that value (specified by `-Xms:`*`nn`*`MB|GB`) is set.

Growing the Heap

The ability of the heap to grow is now more restricted. If the maximum heap size (specified by `-Xmx:`*`nn`*`MB|GB`) is *not* set, the heap will not grow when the system runs out of memory if such growth would cause paging. Instead, assuming object allocation fails and the heap cannot be compacted more, an OutOfMemoryError will be thrown. If `-Xmx` is set, the heap can grow up to the `-Xmx` value, but BEA JRockit will try full compaction before allowing the heap to grow to as size that causes paging.

For more information on heap sizing, please refer to "Setting the Heap Size" in *Tuning the BEA JRockit 1.4.2 JVM*.

Headless Management Console

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

For more information on this feature, please refer to Starting and Running the Console in the Headless Mode in *Using BEA JRockit*.

New JRE Packaging

The Java Runtime Environment, which is comprised of the BEA JRockit JVM and the components described in "Standard JRE Features," has been repackaged so that it can now be installed separately from the SDK by using the JRE installer program shipped with this product. In previous versions of BEA JRockit, installing just the JRE required manipulating files within the various directory structures and posed a risk of error. The new installation instructions can be found in "Installing the BEA JRockit 1.4.2 JRE."

Support for Browser Plugin on IA32 Machines

The Java Plugin extends the functionality of a web browser, allowing applets or Java Beans to be run under BEA JRockit JRE rather than the Java runtime environment that comes with the web browser. The Java Plugin is part of the BEA JRockit JRE and is installed when the JRE is installed on a computer. It works with both Netscape and Internet Explorer. This feature is available only on IA32 implementations of BEA JRockit.

For information on installing and using the Java plug-in, please refer to "Using the Java Plugin."

Java.* Source Files and Demos Available

On edocs you can now find Java demos and sources for BEA JRockit 1.4.2.

- The `Java.*` sources files contain the source code to the Java classes in the Sun SDK. These sources are the reference implementation provided by Sun. Be aware that the source to some of the class files might be different.
- The demos include applets, applications, and sample code that demonstrate various features of the Java 2 Platform.

Normally, these files are only available from Sun, but they can now be downloaded directly from:

`http://edocs.bea.com/wljrockit/docs142/demo_src.html`

You can download the source and demo files as `.zip` files and install them wherever you determine appropriate.

Java Web Start

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

Please refer to Using Web Start with BEA JRockit.

GC Support for Profiling Interface

The JVM Profiling Interface (JVMPI) is now supported on all garbage collectors. For more information on JVMPI, please refer to "Profiling and Debugging with BEA JRockit 1.4.2."

man Pages Available for Linux Tools

For more information on the Linux man pages, please refer to "Using the Linux man Pages."

Non-standard Features for 1.4.2_05

This section describes new, non-standard BEA JRockit features. Since these features are non-standard, they are provided "as-is" and are subject to change at any time. The feature described in this section is:

- BEA JRockit Memory Leak Detector

BEA JRockit Memory Leak Detector

The BEA JRockit Memory Leak Detector is a tool that detects memory leaks within Java applications running on BEA JRockit. A memory leak means application code holding on to memory which is not actually used by the application any more. The BEA JRockit Memory Leak Detector is a real-time profiling tool that gives information about what type of objects are allocated, how many, of what size and how they relate to each other.

For more information about the Memory Leak Detector, please refer to Using the BEA JRockit Memory Leak Detector.

Note: This product is provided "as-is," without any expressed or implied warranties or support by BEA Systems, Inc. This product, which may or may not become an officially supported product from BEA Systems, may contain errors and/or inaccuracies. Use of this product is left solely upon the discretion of the user without any endorsement from BEA Systems. The Memory Leak Detector functionality may or may not be available in

future BEA JRockit versions. Questions and problems may be reported via online BEA JRockit newsgroups at http://newsgroups.bea.com.

Non-standard Features for 1.4.2_04

This section describes new, non-standard BEA JRockit features. Since these features are non-standard, they are provided "as-is" and are subject to change at any time. The features described in this section are:

- Setting a Dynamic Garbage Collector with -Xgcprio
- Manually Configuring the Garbage Collector
- JRockit Runtime Analyzer

Setting a Dynamic Garbage Collector with -Xgcprio

This version of BEA JRockit is released with a unified garbage collector that allows you to select a dynamic garbage collector based solely upon one of two priorities: memory throughput or duration of pause time caused by collection. The unified garbage collector is *dynamic* in that, as it runs, it uses predefined heuristics to determine which collection algorithm to use and changing that algorithm as the individual case might warrant. You do not need to specify the actual algorithm to run this garbage collector.

For more information on the unified garbage collector, please refer to "Running the Dynamic Garbage Collector" in Using BEA JRockit Memory System.

Manually Configuring the Garbage Collector

Three fixed garbage collectors originally available in earlier versions of BEA JRockit SDK are still available if you want to manually configure the garbage collector or override the default garbage collection settings available with the `-server` or `-client` flags.

The available garbage collectors are:

- Single-spaced Concurrent (`-Xgc:singlecon`)
- Generational Concurrent (`-Xgc:gencon`)
- Parallel (`-Xgc:parallel`)

These options are useful if you think they meet your needs better than the unified collector or the default collectors. Additionally, if you want to continue to use scripts written for the earlier

versions of BEA JRockit that implement these collectors, those scripts will continue to work without requiring any modification.

Note: The generational copy garbage collector, provided with BEA JRockit 8.1 SDK, is not provided with BEA JRockit 1.4.2 SDK.

JRockit Runtime Analyzer

The BEA JRockit Runtime Analyzer (JRA) is an internal experimental tool used by the BEA JRockit development team to analyze runtime performance of BEA JRockit and Java applications running on BEA JRockit. The tool provides a wealth of information on internals in BEA JRockit that are interesting to the development team. Some of these metrics are interesting to Java developers using BEA JRockit as their runtime as well.

Warning: JRA is provided "as-is," without any expressed or implied warranties or support by BEA Systems, Inc. This product might contain errors and/or inaccuracies. Its use is left solely upon the discretion of the user without any endorsement from BEA Systems.

Note: JRA supports only J2SE version 1.4 and higher.

New Features for JRA for 1.4.2_04

New features have been added expressly for the version of JRA available with BEA JRockit 1.4.2_04. They are not available with the version of JRA available with BEA JRockit 1.4.2_03.

These features are:

- New Command-line Start-up Options
- Lock Profiling Information
- New Usability Features

Note: If you are currently using the JRA tool included with BEA JRockit 1.4.2_03 and want access to these features, please download the latest version of the JRA at:

http:edocs.bea.com/wljrockit/docs142/JRA.zip

New Command-line Start-up Options

You can now start a JRA recording by using the command `-XXjra` and any of the following parameters:

`delay`	Amount of time, in seconds, to wait before recording starts.
`recordingtime`	Duration, in seconds, for the recording.
`filename`	The name of recording file.
`sampletime`	The time, in milliseconds, between samples.
`nativesample`	Displays method samples in native code; that is, you will see the names of functions written in C-code

For example:

```
-XXjra:delay=10,recordingtime=100,filename=jrarecording2.xml,sampletime=10000
```

Lock Profiling Information

You can now see comprehensive information about lock activity for the application JRA is monitoring on the Lock Profiling tab. A lock profile can only be generated when the `-Djrockit.lockprofiling` command is issued at the command line. For more information, please refer to Lock Profiling Tab.

New Usability Features

The following usability improvements have been made to the version of the JRA available with BEA JRockit 1.4.2_04:

- All columns on the Ojbect Statistics are now sortable.
- Tooltips have been added to the JRA console.
- The Method Tree toggle button has been relocated to the Methods page.

For More Information

For more information on JRA, please refer to Using the BEA JRockit Runtime Analyzer

Code Caching

Code caching—or code persistence—is the process of storing generated code to disk for retrieval when that code is required later. Since cached code is already generated, the time that code generation would require on subsequent startups is no longer an issue and—usually—execution time is reduced. For more information on Code Caching, please refer to Code Caching with BEA JRockit.

Note: This feature is available only in BEA JRockit 1.4.2_04.

CHAPTER 5

Using the Documentation

This section provides hints for using the BEA JRockit SDK documentation set. It includes information on the following subjects:

- BEA JRockit SDK Documentation Set
- Printing These Documents
- Understanding Documentation Conventions

BEA JRockit SDK Documentation Set

The documentation set is comprised of these documents:

- *Release Notes* and assorted web documents
- *Introduction to BEA JRockit 1.4.2 SDK*

  This document serves as an overview of BEA JRockit. It allows anyone, from managers to any personnel down through the organization to gather all essential background about the product without having to read through procedural information normally included in a user guide or development guide.

- *Installing BEA JRockit 1.4.2 SDK*

  This web document contains information for installing BEA JRockit SDK as a standalone application.

- *Using BEA JRockit 1.4.2 SDK*

This is the BEA JRockit SDK user guide. It describes such functions as selecting and running a memory management system, configuring the JVM, and using the BEA JRockit Management Console.

- *Tuning BEA JRockit 1.4.2 JVM*

  This is the BEA JRockit JVM tuning guide. It includes tuning guidelines for heap and thread management along with tips and techniques for tuning BEA JRockit JVM to ensure optimal performance with your Java applications.

- *Developing Applications*

  The guide contains tips and techniques for developing applications to run successfully on a JVM, particularly BEA JRockit. It includes a detailed troubleshooting guide, information on implementing profiling and debugging interfaces, and instructions for migrating applications developed on a third-party JVM (for example, Sun Microsystem's HotSpot JVM) to BEA JRockit JVM.

These documents can be found at:

`http://edocs.bea.com/wljrockit/docs142/index.html`

Printing These Documents

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

PDF versions of all BEA JRockit documents are available on the BEA JRockit documentation pages on the e-docs Web site. You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access and print the PDFs, do the following:

1. Open the web page for the BEA JRockit document you want to print and click the view as PDF icon.

   A new browser launches, running the Adobe Acrobat Reader, which contains the PDF version of the document you selected.

2. Click the print button on the Adobe Acrobat Reader toolbar.

   The Print dialog box appears.

3. Select the Print range (All, Current page, or Pages from) and click OK to print the document.

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

Understanding Documentation Conventions

The following documentation conventions are used throughout this document.

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

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

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR

Convention	Item
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
[ ]	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
. . .	Indicates one of the following in a command line: • That an argument can be repeated several times in a command line • That the statement omits additional optional arguments • That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. *Example*: `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.