Oracle® JRockit

Command-Line Reference

Release R28

E15062-19

July 2017

ORACLE®

Oracle JRockit Command-Line Reference, Release R28

E15062-19

Primary Author: Savija Vijayaraghavan

Contents

Preface

This preface describes the document accessibility features and conventions used in this guide—*Oracle JRockit Command-Line Reference*.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

About the JRockit JVM Command-Line Options

This chapter describes the command-line options, also called startup commands or startup options, available in Oracle JRockit JVM. These options are self-describing tags that you either enter at the command line or include in startup scripts for applications running on a JVM. These options are used to override the JVM default settings and otherwise define to the JVM how you want your application to run. For example, you can use the command-line option `-Xmx` to set the maximum heap size.

Command-line options can be either valid for any JVM regardless of the vendor (standard options) or specific to a JVM (nonstandard).

This chapter contains the following sections:

1.1 Standard Command-Line Options

Table 1–1 lists the standard command-line options that the Oracle JRockit JVM recognizes.

Table 1–1 Standard Command-Line Options Accepted by the Oracle JRockit JVM

Option (Alternate Usage)	Description
`-agentlib:`*`agent-lib-name`*`[=`*`options`*`]`	Loads the specified native agent library
`-agentpath:`*`path-to-agent`*`[=`*`options`*	Loads the native agent library that is located at the specified path
`-client`	Selects the JRockit client JVM
`-javaagent`	Loads a Java programming language agent (see `java.lang.instrument`)
`-jrockit`	Selects the JRockit server JVM This is equivalent to `-server` and is the default.
`-version`	Displays version information and then exits the application
`-showversion`	Displays version information and continues the application
`-verbose:`*`area`*`[,`*`options`*`]`	Displays specific information about the system For more information, see -Xverbose.

Table 1–1 (Cont.) Standard Command-Line Options Accepted by the Oracle JRockit JVM

Option (Alternate Usage)	Description
`-cp (-classpath)`	Specifies a list of directories, JAR files, and ZIP archives to search for class files. Classpath entries are separated by semicolons (`;`) in Windows and colons (`:`) in Linux and Solaris. Specifying `-classpath` or `-cp` overrides any setting of the `CLASSPATH` environment variable.
`-ea (-enableassertions)`	Enables assertions, which are disabled by default Depending on the arguments specified, this option either enables assertions, enables assertions in the specified package and any subpackages, enables assertions in the unnamed package in the current working directory, or enables assertions in the specified class.
`-da (-disableassertions)`	Disables assertions Depending on the arguments specified, this option either disables assertions, disables assertions in the specified package and any subpackages, disables assertions in the unnamed package in the current working directory, or disables assertions in the specified class
`-esa (-enablesystemassertions)`	Enables assertions in all system classes by setting the default assertion status for system classes to `true`
`-dsa (-disablesystemassertions)`	Disables assertions in all system classes

For more information about these standard command-line options, see the Java documentation at the following locations:

- Java SE 6.0

  http://java.sun.com/javase/6/docs/technotes/tools/windows/java.html#standard

- J2SE 5.0

  http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/java.html#standard

1.2 JRockit JVM-Specific Command-Line Options

The JRockit JVM uses a set of nonstandard command-line options to control JVM behavior. Because these options are nonstandard, they do not work with other JVMs. If you use the nonstandard options that are specific to a JVM with other JVMs, the results can be erroneous or an error condition might occur.

The nonstandard command-line options of JRockit JVM are divided into two groups:

- `-X` command-line options, which are the most commonly used nonstandard options
- `-XX` command-line options, which are often experimental options that have specific system requirements for their implementation

The nonstandard options described in this document (Chapter 2, "-X Command-Line Options" and Chapter 3, "-XX Command-Line Options") are subject to change or deprecation at any time.

Note: Occasionally, you might encounter JRockit JVM internal properties set with the `-D` option (for example, `-Djrockit.lockprofiling=true`). The `-D` option sets values for parameters that are used by Java programs. In the Oracle JRockit JVM, some of those parameters are read by the JVM and change how the JVM works. The -D properties are for internal use; so they are not described in this document.

1.3 About System Properties

System properties define traits or attributes of the current working environment. When the Java application starts, the system properties are initialized with information about the run-time environment, including information about the current user, the current version of the Java run time, and the product vendor's bug report URL.

For information about the system properties available with the JRockit JVM, see Chapter 4, "Oracle JRockit JVM System Properties."

2

-X Command-Line Options

This chapter is an alphabetically ordered reference for all the `-X` command-line options that you can use with the JRockit JVM.

The `-X` command-line options are exclusive to the Oracle JRockit JVM. You can use the `-X` command-line options to change the behavior of the JRockit JVM to suit the needs of different Java applications. These options do not work on other JVMs (conversely, the nonstandard options used by other JVMs do not work with the JRockit JVM).

Notes:

- The `-X` options are subject to change at any time.
- Command-line options are case sensitive unless explicitly stated. Most of the commands use the camel notation (for example, `-Xgc` and `-XlargePages`).
- If you do not add a unit with the values of options that specify memory size, you get the exact value; for example, 64 is considered as 64 bytes, not 64 megabytes or 64 kilobytes.

-Xverify

-Xbootclasspath

The `-Xbootclasspath` option specifies a list of directories, JAR files, and ZIP archives to search for bootstrap classes and resources. These are used in place of the bootstrap class files included in the Java SE JDK.

> **Note:** Applications that use this option to override a class in the `rt.jar` file should not be deployed. It violates the Java SE run-time environment binary code license.

Format

`-Xbootclasspath` *`directories and zips/jars separated by ; (Windows) or : (Linux and Solaris)`*

The `-Xbootclasspath` option name must be entered in lowercase as shown in the preceding format (not in camel notation).

Related Options

- -Xbootclasspath/a
- -Xbootclasspath/p

-Xbootclasspath/a

The `-Xbootclasspath/a` option is similar to -Xbootclasspath in that it specifies a list of directories, JAR files, and ZIP archives; however, the list is **appended** to the default bootstrap class path.

Format

`-Xbootclasspath/a` *`directories and zips/jars separated by ; (Windows) or : (Linux and Solaris)`*

The `-Xbootclasspath/a` option name must be entered in lowercase as shown in the preceding format (not in camel notation).

Related Options

- -Xbootclasspath
- -Xbootclasspath/p

-Xbootclasspath/p

The `-Xbootclasspath/p` option is similar to -Xbootclasspath in that it specifies a list of directories, JAR files, and ZIP archives; however, the list is **prepended** to the default bootstrap class path.

Format

`-Xbootclasspath/p` *`directories_and_zips/jars_separated_by ; (Windows) or : (Linux and Solaris)`*

The `-Xbootclasspath/b` option name must be entered in lowercase as shown in the preceding format (not in camel notation).

Related Options

- -Xbootclasspath
- -Xbootclasspath/a

-Xcheck:jni

The `-Xcheck:jni` option enables additional checks for JNI functions.

Note: Oracle recommends that you use `-XX:+CheckJNICalls` instead of `-Xcheck:jni`.

-Xdebug

The `-Xdebug` option enables debugging capabilities that are used by the JVM Tools Interface (JVMTI).

Note: Although `-Xdebug` works in R28, Oracle recommends that you use `-XX:+JavaDebug` instead.

For more information about `-Xdebug`, see the Oracle JRockit R27 documentation at:

http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/optionX.html

---

Caution: Do not use the `-Xdebug` option in the production environment, because, when running with the `-Xdebug` option, the JVM does not run at full speed.

-Xgc

The `-Xgc` option enables you to specify a garbage collection mode.

You can choose a garbage collector that is either generational or single spaced with a parallel or a concurrent mark and uses either a parallel sweep or a concurrent sweep.

- Generational Garbage Collection

  During a two-generational garbage collection, the heap is divided into two sections: an old generation and a young generation (nursery). Objects are allocated in the nursery and when it is full, the JVM stops all Java threads and moves the live objects from the nursery, young generation, to the old generation.

- Single-spaced Garbage Collection

  The single-spaced option of garbage collection means that all objects live out their lives in a single space on the heap, regardless of their age. In other words, a single-spaced garbage collector does not have a nursery.

- Concurrent Mark and Sweep Algorithm

  The concurrent garbage collection algorithm does its marking and sweeping concurrently with all other processing; that is, it does not stop Java threads to do the complete garbage collection.

- Parallel Garbage Collection Mark and Sweep Algorithm

  The parallel garbage collection algorithm stops Java threads when the heap is full and uses every CPU to perform a complete mark and sweep of the entire heap. A parallel garbage collector can have longer pause times than concurrent garbage collectors, but it maximizes application throughput. Even on single CPU machines, this maximized performance makes parallel the recommended garbage collector, provided that your application can tolerate the longer pause times.

Format

`-Xgc:`*mode*

Table 2–1 lists the garbage collection modes that you can specify with the `-Xgc` option.

Table 2–1 Valid Garbage Collection Modes for -Xgc

Mode	Description

`singlecon`
Alias:
`singleconcon`
	Single-space (nongenerational), concurrent garbage collection.
	In the `singlecon` garbage collection mode, most of the garbage collection tasks are performed concurrently with the Java application. All objects are maintained in a single space, or *generation*. The `singlecon` mode reduces application throughput but keeps pause times to a minimum.

`gencon` **Alias:** `genconcon`	Generational, concurrent garbage collection. In the `gencon` garbage collection mode, objects are allocated in the *young generation* (*nursery*). When the nursery is full, the JRockit JVM stops all the Java threads and moves the live objects in the young generation to the *old generation*. Most of the old collection tasks are performed concurrently with the Java application. The `gencon` mode is better than the `singlecon` mode for most applications that allocate numerous small, short-lived objects. The `gencon` mode increases heap size and reduces application throughput, but keeps pause times to a minimum.

Table 2–1 (Cont.) Valid Garbage Collection Modes for -Xgc

Mode	Description
`singlepar` **Alias:** `singleparpar`, `parallel`	Single-space, parallel garbage collection. In this mode, when the heap is full, all the Java threads are stopped and the JVM uses every CPU to perform a complete garbage collection of the entire heap. This mode increases pause times when compared with the `concurrent` mode but maximizes throughput. Even on single CPU systems, the maximized throughput makes parallel garbage collection the recommended mode, provided the application can tolerate the longer pause times.
`genpar` **Alias:** `genparpar`	Generational garbage collection. In the `genpar` mode, objects are first allocated in the young generation (nursery). When the nursery is full, the JRockit JVM stops all the Java threads and performs a parallel, young collection; that is, it uses all the available CPU resources and moves the live objects in the young generation to the old generation. When the heap is full, the JRockit JVM stops all the Java threads and performs a complete parallel collection. This collector prioritizes throughput over pause times. This mode is generally better than the `singlepar` mode for applications that allocate numerous short-lived objects. In this mode, a higher number of garbage collections are performed than in the `singlepar` mode, but the individual pause times are shorter, resulting in lower fragmentation in the old generation space.
`genconpar`	Generational garbage collection. Sets the garbage collection mode to generational (two-spaced) with a concurrent mark algorithm and a parallel sweep algorithm.
`genparcon`	Generational garbage collection. Sets the garbage collection mode to generational (two-spaced) with a parallel mark algorithm and a concurrent sweep algorithm.
`singleconpar`	Single-space garbage collection. Sets the garbage collection mode to single-spaced with a concurrent mark algorithm and a parallel sweep algorithm.
`singleparcon`	Single-space garbage collection. Sets the garbage collection mode to single-spaced with a parallel mark and a concurrent sweep algorithm.
`throughput`	The garbage collector is optimized for application throughput. This means that the garbage collector works as effectively as possible, giving as much CPU resources to the Java threads as possible. This might, however, cause nondeterministic pauses when the garbage collector stops all Java threads for garbage collection.The throughput priority should be used when non-deterministic pauses do not impact the application's behavior.
`pausetime`	The garbage collector is optimized for short pauses. This means that the garbage collection works concurrently with the Java application when necessary, in order to avoid pausing the Java threads. This inflicts a slight performance overhead to the application, as the concurrent garbage collector demands more system resources (CPU time and memory) than the parallel garbage collector that is used for optimal throughput. The target pause time is by default 500 msec. To change the default pause target, see -XpauseTarget.

Table 2–1 (Cont.) Valid Garbage Collection Modes for -Xgc

Mode	Description
`deterministic`	Optimizes the garbage collector for very short and deterministic pause times. The garbage collector tries to keep the garbage collection pauses below a given pause target. The performance depends on the application and the hardware. Running on slower hardware, with a different heap size or with a large amount of active data can break the deterministic behavior or cause performance degradation over time; faster hardware or a less amount of active data might allow you to set a lower pause target. The pause target for deterministic mode is by default 30 msec, and can be changed with the command-line option `-XpauseTarget`.

Default Value

The default garbage collection mode is the throughput mode.

Related Options

When the -XXsetGC (deprecated) or -XgcPrio (deprecated) options are specified, the `-Xgc` option is overridden, and vice versa. The option specified first on the command line is ignored.

-XgcPrio (deprecated)

The `-XgcPrio` option is deprecated in Oracle JRockit R28. The option works in R28, but Oracle recommends that you use `-Xgc` instead. For more information, see -Xgc.

For more information about the format and usage of `-XgcPrio`, see the R27 documentation at: `http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/index.html`.

-XlargePages

The `-XlargePages` option specifies to use large pages, if they are available, for the Java heap and other areas in the JVM. Large pages allow your application to more effectively use the translation look-aside buffer (TLB) in the processor.

> **Note:** Oracle recommends that you use the -XX:+|-UseLargePagesFor[Heap|Code] option for enabling large pages.

Format

`-XlargePages``:exitOnFailure=true`

Windows, Linux, and Solaris support multiple page sizes on x86 and SPARC architectures. x86 supports 4 KB and 4 MB (2 KB and 2 MB in PAE mode). SPARC supports a wider range of different sizes, from 4 KB to 256 MB, depending on the model.

By default, the JVM continues to run without large pages if large pages cannot be acquired when the `-XlargePages` option is enabled. Use the extended option (`-XlargePages:exitOnFailure`) to override this behavior and to force the JVM to exit if enough large pages cannot be acquired.

> **Note:** If you use this option, you must configure large pages on your system by following the procedures specific to your operating system.
>
> For more information about large pages on Linux, read the file `vm/hugetlbpage.txt` available in the documentation for the Linux kernel.
>
> Nothing has to be configured in the Solaris operating system to enable an application to use large pages.

If the JRockit JVM fails to acquire large pages, it prints a warning as shown in the following example and continues to work:

```
[ERROR][osal ] Unable to set Lock Page Privilege:
...
[WARN ][memory ] Could not acquire large pages for Java heap.
[WARN ][memory ] Falling back to normal page size.
```

Default

`-XlargePages` is disabled by default on Windows and Linux platforms. On Solaris SPARC, this option is enabled by default.

-Xmanagement

The `-Xmanagement` option starts the JRockit JVM concurrently with the management server and allows you to either enable and configure or explicitly disable features such as autodiscovery of the JVM instances in a network, SSL encryption, and authentication.

Format

`-Xmanagement[:`*`parameter1=value`*`[,`*`parameter2=value2`*`]]`

Table 2–6 lists the possible values for *`parametern=valuen`* pairs.

Table 2–2 -Xmanagement Parameters

Parameter	Description	Default Value
None	Enables the JMX local monitoring through a JMX connector published on a private interface used by local JMX clients that use the Attach API. JMX clients can use this connector if it is started by the same user who started the agent. No password or access files are required for requests coming through this connector.	`true`
`autodiscovery=true|false`	Enables or disables autodiscovery for the remote JMX connector, which allows Oracle JRockit Mission Control to automatically discover running JRockit JVM instances through the multicast-based JRockit Discovery Protocol (JDP). The autodiscovery enables other machines on the same subnet to automatically detect a remote management-enabled JVM. **Note:** The JVM Browser in Oracle JRockit Mission Control automatically discovers remote JVM instances only if this option is enabled. **Related -D option**: -Dcom.oracle.management.autodiscovery	`false`
`autodiscovery_name=`*`/mycluster/mymachine/Node1`*	Enables you to specify the path and name of the cluster and node from where Oracle JRockit Mission Control discover information about various JRockit JVM instances running in a network. Note: You can use this option only when `autodiscovery` is set to `true`.	-

Table 2–2 (Cont.) -Xmanagement Parameters

Parameter	Description	Default Value
`authenticate=true|false`	Enables or disables authentication. When this property is set to `false`, JMX does not use passwords or access files. All users are allowed all access. **Related -D option**: -Dcom.oracle.management.jmxremote.authenticate	`true`
`class=`*`class_name`*	Loads the class and causes its empty constructor to be called early in JVM startup. From the constructor, a new thread is then started, from which your management client is run. Further arguments cannot be given to `-Xmanagement` after the `class` argument.	-
`config_file=`*`path`*	Specifies the location of the file from which additional management configuration properties are loaded. **Related -D option**: -Dcom.oracle.management.config.file	*`JRE_HOME`*`/lib/management/management.properties`
`interface=`*`host`*`|`*`ip`*	Specifies the IP address or the host name of the remote machine. If this option is set, only then connections to a specified ip (or host) are allowed. The JMX agent still listens to and answer connections on all interfaces; however, connections to other addresses than those specified by this option are discarded. **Related -D option**: -Dcom.oracle.management.jmxremote.interface	`null`
`local=true|false`	Enables or disables the local JMX connector. **Related -D option**: -Dcom.oracle.management.jmxremote	`true`
`port=`*`portNumber`*	Identifies the port that the management server opens for remote access. When you specify a port number, the JMX remote agent is enabled and it creates a remote JMX connector to listen through the specified port. By default, the SSL, password, and access file properties are used for this connector. This option also enables local monitoring. **Related -D option**: -Dcom.oracle.management.jmxremote.port	7091, when you do not specify a value for this option.

Table 2–2 (Cont.) -Xmanagement Parameters

Parameter	Description	Default Value
`registry_ssl=true|false`	Binds the RMI connector stub to an RMI registry protected by SSL. **Related -D option**: -Dcom.oracle.management.jmxremote.registry.ssl	`false`
`remote=true|false`	Enables or disables the remote JMX connector.	`false`
`rmiserver_port=portNumber`	Binds the RMI Server to the specified port. **Related -D option**: -Dcom.oracle.management.jmxremote.rmiserver.port	Bind to the same port as the RMI Registry. If the RMI Server is using SSL and the registry is not, a random port is selected.
`ssl=true|false`	Enables or disables SSL encryption.	`true`

Examples

```
java -Xmanagement:ssl=false,authenticate=false myApplication
```

Disables SSL encryption and authentication.

```
java -Xmanagement:autodiscovery=true myApplication
```

Enables autodiscovery.

```
java -Xmanagagement:autodiscovery=true,autodiscovery_
name=/mycluster/mymachine/Node1
```

The JRockit JVM appears under the `JDP/mycluster/mymachine` folder with the connection name as `Node1`. If you specify a forward slash (`/`) at the end of the path, the name of the resulting descriptor is determined by a reverse DNS lookup.

```
java -Xmanagagement:port=1234 myApplication
```

Directs the management server to open port 1234.

Due to the security risks and the mission-critical nature of most JRockit JVM deployments, the new default behavior of the JRockit JVM requires that you either disable security explicitly or configure and enable security. If you do not take these steps, the management server does not open a port for remote access and might cause the JVM startup to halt with an error message concerning the security configuration.

Specifying the `-Xmanagement` option also enables a local in-memory agent to improve the user experience from a developer perspective. For example, a developer running a WebLogic Server instance on JRockit JVM on a machine can specify the `-Xmanagement` option to enable the local in-memory agent to connect to it from an Oracle JRockit Mission Control Client on another machine. On the other hand, the developer would not have to specify the `-Xmanagement` option to get local access from Oracle JRockit Mission Control: the in-memory agent is always accessible locally. If you have a number of JRockit JVM instances running on your machine and you start a JRockit Mission Control Client, it automatically discovers and allows access to those JVMs. Security is enforced by allowing this type of local access only if the JRockit JVM instance and the JRockit Mission Control Client are being run by the same user.

To enable the management agent without security you must now specify that SSL and authentication should be disabled.

For maximum usability, enable the autodiscovery mechanism, which allows JRockit Mission Control to automatically discover the running JRockit JVM instances through the multicast-based JRockit Discovery Protocol. Note that this typically works only on the local subnet.

Default Values

The default behavior is as follows:

- Local agent is enabled.
- Remote management agent is enabled with security, if SSL encryption, authentication, and networking are configured. If SSL and authentication are not configured, remote management agent is enabled with security explicitly disabled.

-Xms

The -Xms option sets the initial and minimum Java heap size. The Java heap (the heap) is the part of the memory where blocks of memory are allocated to objects and freed during garbage collection.

> **Note:** The -Xms option does not limit the total amount of memory that the JVM can use.

Format

-Xms:*size*[g|G|m|M|k|K]

Combine -Xms with a memory value and add a unit.

Example

java -Xms:64m myApp

This command sets the initial and minimum java heap to 64 MB.

If you do not add a unit, you get the exact value; for example, 64 is interpreted as 64 bytes, not 64 megabytes or 64 kilobytes.

For good performance, set the -Xms option to the same size as the maximum heap size, for example:

java -Xmx:64m -Xms:64m myApp

Default Values

If you do not set this option, the minimum Java heap size defaults to the following (depending on which mode you are running):

- -server **mode:** 25% of the amount of free physical memory in the system, up to 64 MB and at least 8 MB.
- -client **mode:** 25% of the amount of free physical memory in the system, up to 16 MB and at least 8 MB.
- If the nursery size is set with the -Xns option, the default initial heap size is scaled up to at least twice the nursery size.

Exceptions and Recommendations

The initial Java heap cannot be set to a smaller value than 8 MB, which is the minimum Java heap size. If you set this option to a smaller value than 8 MB, JRockit JVM prints an error message and terminates.

The -Xms value cannot exceed the value set for -Xmx (the maximum Java heap size).

-Xmx

The -Xmx option sets the maximum Java heap size. The Java heap is the part of the memory where blocks of memory are allocated to objects and freed during garbage collection. Depending upon the kind of operating system you are running, the maximum value you can set for the Java heap can vary.

> **Note:** The -Xmx option *does not* limit the total amount of memory that the JVM can use.

Format

```
-Xmx:size[g|G|m|M|k|K]
```

Combine the -Xmx option with a memory value.

Example

```
java -Xmx:1g myApp
```

This command sets the maximum Java heap to 1 Gigabyte.

If you do not add a unit, you get the exact value; for example, 64 is interpreted as 64 bytes, not 64 megabytes or 64 kilobytes.

The -Xmx and -Xms options, in combination, are used to limit the Java heap size. The Java heap can never grow larger than -Xmx. The -Xms value can also be used as the minimum heap size to set a fixed heap size by setting -Xms = -Xmx when, for example, you want to run benchmark tests.

Default Values

If you do not set this option, the maximum Java heap size depends on the platform and the amount of memory in the system as described in Table 2–3.

Table 2–3 Default Maximum Heap Sizes

Platform	Default Maximum Heap Size
Windows on a 64 bit platform	75% of total physical memory up to 3 GB
Linux or Solaris on a 64 bit platform	75% of physical memory up to 3 GB
Windows on a 32 bit platform	75% of total physical memory up to 1 GB
Linux or Solaris on a 32 bit platform	75% of physical memory up to 1 GB

Exceptions

When using -Xmx, be aware of the following exceptions:

- If both -Xmx and -Xms are specified the value of -Xmx must be larger than or equal to that of -Xms.
- If both -Xmx and -Xns are specified the value of -Xmx must be larger than or equal to that of -Xns.
- The minimum value for -Xmx is 16 MB.

-XnoClassGC (deprecated)

The `-XnoClassGC` option is deprecated in Oracle JRockit R28. The option works in R28, but Oracle recommends that you use `-XX:-UseClassGC` instead. For more information, see -XX:+|-UseClassGC.

For more information about the format and usage of `-XnoClassGC`, see the R27 documentation at: `http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/index.html`.

-XnoOpt

The -XnoOpt option turns off adaptive optimization.

Optimized code generally runs faster than code that has not been optimized, but occasionally, the time required to optimize code results in undesirable delays in processing. -XnoOpt avoids these delays by turning off optimization. This option is also helpful when you suspect that a JVM or application problem, such as a system crash or poor startup performance, might be related to optimization. You can turn optimization off and retry your application. If it then runs successfully, you can safely assume that the problem lies with code optimization.

If -XnoOpt is not set, the JVM optimizes code as usual.

Format

```
-XnoOpt
```

or

```
-XnoOpt:delay_time_in_seconds
```

The optional argument *delay_time_in_seconds* allows users to delay the disabling of optimization until a certain amount of time has passed since the JVM startup.

If you specify -XnoOpt:600, optimization will not be disabled until the JVM has been running for 10 minutes. Specifying the delay time with -XnoOpt is often a very good compromise between completely disabling all optimization from the beginning and leaving optimization enabled for the entire run. Setting an appropriate value for -XnoOpt will allow most performance-critical methods to still get optimized during warm-up, but once the application reaches a steady state, the behavior becomes more deterministic because additional optimizations do not take place.

-Xns

The `-Xns` option sets the nursery size. The JRockit JVM uses a nursery when a generational garbage collector is being used.

Format

`-Xns:`*`size`*`[g|G|m|M|k|K]`

Combine `-Xns` with a memory value.

The nursery size value cannot exceed the maximum value set for the heap.

Example

```
java -Xns:10m myApp
```

Sets the nursery to 10 MB of the heap.

Default Value

The default value depends on the garbage collection mode, as described in Table 2–4.

Table 2–4 Default Nursery Sizes

Options used	Default value
-server (default)	50% of free heap
-client	None; nursery does not exist
-Xgc:gencon, -Xgc:pausetime	10 MB per logical processor (maximum 80 MB)
-Xgc:genpar, -Xgc:throughput	50% of free heap

Exceptions

The `-Xns` option is valid only when a generational garbage collector is used.

-XpauseTarget

The `-XpauseTarget` option sets a pause target for the garbage collection mode optimizing for short pauses (-Xgc:pausetime) and the garbage collection mode optimizing for deterministic pauses (-Xgc:deterministic). The target value is used as a pause time goal. The target helps the garbage collector to more precisely configure itself to keep pauses near the target value. Using this option allows you to specify the pause target to be between 1 millisecond and 5 seconds. If you are using the deterministic garbage collector, you can set values below 200 milliseconds.

Format

`-XpauseTarget=`*`value`*

The value set by this option is considered a soft goal; that is, if specifying the target to 100 msec, the garbage collector tries to tune itself towards a configuration that makes the pauses become as near 100 msec as possible. However, if you have an application and heap configuration that does not meet this target even after the garbage collector is tuned, the target is missed. This option specifies only the desired pause times, not the maximum allowed pause time.

When you use this option properly, it improves pause times. Otherwise, it might stress the garbage collector and affect performance.

Default Values

If you are using `-XpauseTarget` with -Xgc:pausetime, the default setting for the target is 500 msec. If you are using -Xgc:deterministic, the default value is 30 msec.

Related Options

Normally, this option requires that you use it with a pause optimizing garbage collection mode (-Xgc:pausetime or -Xgc:deterministic). If you do not specify a garbage collector, this option changes from the default garbage collector to the pause time optimizing garbage collector (the same collector used when specifying -Xgc:pausetime).

If you are using Oracle JRockit Real Time, set `-XgcPauseTarget` less than 200 msec, and do not specify a garbage collector. The garbage collector is set to -Xgc:deterministic.

Exceptions

When using `-XpauseTarget`, note the following exceptions:

- Setting `-XpauseTarget` has not effect if you are running the garbage collector in throughput mode.
- If you are using the deterministic garbage collector, you can specify pause targets below 200 msec as well.

-Xrs

> **Note:** `-Xrs` is a non-standard option in HotSpot JVM. JRockit JVM continues to support this option; however, the JRockit JVM nonstandard options `-Xnohup` and `-XX:+|-ReduceSignalUsage` provide the same functionality.

`-Xrs` reduces usage of operating-system signals by the JVM. If the JVM is run as a service (for example, the servlet engine for a web server), it can receive `CTRL_LOGOFF_EVENT` but should not initiate shutdown since the operating system does not actually terminate the process. To avoid possible interference such as this, the `-Xrs` command-line option does not install a console control handler, implying that it does not watch for or process `CTRL_C_EVENT`, `CTRL_CLOSE_EVENT`, `CTRL_LOGOFF_EVENT`, or `CTRL_SHUTDOWN_EVENT`.

Format

```
-Xrs
```

If you are running JRockit JVM as a service (for example, the servlet engine for a web server), enter the command at startup to prevent the JVM from watching for or processing `CTRL_LOGOFF_EVENT` or `SIGHUP` events.

Exceptions

The following are exceptions when you use this option:

- Pressing Ctrl-Break to create a thread dump does not work.
- User code is responsible for causing shutdown hooks to run.

-Xss

The `-Xss` option sets the thread stack size. Thread stacks are memory areas allocated for each Java thread for their internal use. This is where the thread stores its local execution state.

Format

`-Xss:`*`size`*`[g|G|m|M|k|K]`

Combine `-Xss` with a memory value.

> **Note:** JRockit does not support stack sizes above 128 MB.

Example

```
java -Xss:512k myApp
```

Sets the default stack size to 512 kilobytes.

Default Values

`-Xss` default values are specific to the JVM binary, as defined in Table 2–5.

Table 2–5 -Xss Default Values

Platform	Default
Windows IA32	64 KB
Windows x86_64	128 KB
Linux IA32	128 KB
Linux x86_64	256 KB
Solaris SPARC	512 KB

-XstrictFP

Oracle recommends that you use `-XX:+StrictFP` instead of `-XstrictFP`.

For more information about `-XstrictFP`, see the R27 documentation at http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/index.html.

-Xverbose

The `-Xverbose` option provides specific information about the system. The output is, by default, printed to the standard output for error messages (`stderr`) but you can redirect it to a file by using the -XverboseLog command-line option. The information displayed depends on the parameter specified with the option; for example, specifying the parameter `cpuinfo` displays information about your CPU and indicates whether or not the JVM can determine if hyperthreading is enabled.

Format

`-Xverbose:`*`parameter`*`[=`*`log_level`*`]`

Table 2–6 lists the parameters, and Table 2–7 lists the log levels.

> **Note:** To use more than one parameter, separate them with a comma (for example, `-Xverbose:gc,opt`).

Table 2–6 -Xverbose Parameters

Parameter	Prints to the screen
`alloc`	Information regarding allocations and out of memory.

`class`	The names of classes loaded; sample output might look like this:
	`[INFO ][class ] created: # 0 java/lang/Object (c:\jrockits\R28.0.0_ R28.0.0-617_1.6.0\jre\lib\rt.jar)`
	`[INFO ][class ] 0 java/lang/Object success (0.60 ms)`
	`[INFO ][class ] created: # 2 java/io/Serializable (c:\jrockits\R28.0.0_ R28.0.0-617_1.6.0\jre\lib\rt.jar)`
	`[INFO ][class ] 2 java/io/Serializable success (0.32 ms)`

`codegen`	The names of each method that is being compiled. Verbose output for `codegen` might look like this:
	`[INFO ][codegen][00004] #240 (Normal) java/lang/AbstractStringBuilder.<init>(I)V`
	`[INFO ][codegen][00004] #240 0.315-0.316 0x0000000100019FA0-0x0000000100019FD2 0.38 ms 128KB 31618 bc/s (138.48 ms 60442 bc/s)`
	`[INFO ][codegen][00004] #241 (Normal) java/lang/StringBuilder.append(Ljava/lang/String;)Ljava/lang/StringBuilder;`
	`[INFO ][codegen][00004] #241 0.316-0.316 0x0000000100019FE0-0x0000000100019FF5 0.37 ms 128KB 21503 bc/s (138.85 ms 60338 bc/s)`
	`[INFO ][codegen][00004] #242 (Normal) java/lang/AbstractStringBuilder.append(Ljava/lang/String;)Ljava/lang/Abstract StringBuilder;`
	`[INFO ][codegen][00004] #242 0.317-0.317 0x000000010001A000-0x000000010001A07E 0.53 ms 128KB 113605 bc/s (139.38 ms 60539 bc/s)`
	`[INFO ][codegen][00004] #243 (Normal) java/lang/StringBuilder.append(C)Ljava/lang/StringBuilder;`
	`[INFO ][codegen][00004] #243 0.318-0.318 0x000000010001A080-0x000000010001A09C 0.37 ms 128KB 21747 bc/s (139.75 ms 60437 bc/s)`

Table 2–6 (Cont.) -Xverbose Parameters

Parameter	Prints to the screen

compaction	Information related to the compaction. This information varies for different garbage collection types and also depends on the result of the compaction. Verbose output for compaction might look like this:
	`[INFO ][compact] [OC#2] Compacting 8 of 128 parts at index 0. Compaction type is internal. Exceptional: No.`
	`[INFO ][compact] [OC#2] Compaction area start: 0x2043000, end: 0x263a800.Timeout: 100.000 ms.`
	`[INFO ][compact] [OC#2] Compactset limit (per thread): 37487 (dynamic), not using matrixes.`
	`[INFO ][compact] [OC#2] Adjusted compaction area to start at 0x2043000 and end at 0x263a8d8.`
	`[DEBUG][compact] [OC#2] Internal compaction added 0x25f24f8 - 0x263a8d8 to the freelist, size: 289KB.`
	`[INFO ][compact] [OC#2] Internal compaction found 5698 objects and moved 5681 objects.`
	`[INFO ][compact] [OC#2] Compaction overhead increased to: 3.000.`
	`[INFO ][compact] [OC#2] Compaction pause: 3.677 ms (target 50.000 ms), update ref pause: 119.897 ms (target 50.000 ms).`
	`[INFO ][compact] [OC#2] Updated 518 references. Internal: 6125  External: 518.`
	`[DEBUG][compact] [OC#2] Compaction ended at index 5, object end address was 0x263a8d8.`
	`[INFO ][compact] [OC#2] Average compact time ratio: 0.022901.`
	`[INFO ][compact] [OC#2] Too few references, doubling compact ratio.`

Table 2–6 (Cont.) -Xverbose Parameters

Parameter	Prints to the screen

cpuinfo	Technical information about your CPUs. Verbose output for `cpuinfo` might look like this:
	`[INFO ][cpuinfo] HT is: not supported by the CPU, not enabled by the OS, not enabled in JRockit.`
	`[INFO ][cpuinfo] CPU: Intel Pentium M model D SSE SSE2`
	`[INFO ][cpuinfo] Vendor: GenuineIntel`
	`[INFO ][cpuinfo] Family: Pentium M model D`
	`[INFO ][cpuinfo] Model: Pentium M model D`
	`[INFO ][cpuinfo] Name: Intel(R) Pentium(R) M processor 2.00GHz`
	`[INFO ][cpuinfo] Sockets: 1`
	`[INFO ][cpuinfo] Cores: 1`
	`[INFO ][cpuinfo] HWThreads: 1`
	`[INFO ][cpuinfo] Supports: On-Chip FPU`
	`[INFO ][cpuinfo] Supports: Virtual Mode Extensions`
	`[INFO ][cpuinfo] Supports: Debugging Extensions`
	`[INFO ][cpuinfo] Supports: Page Size Extensions`
	`[INFO ][cpuinfo] Supports: Time Stamp Counter`
	`[INFO ][cpuinfo] Supports: Model Specific Registers`
	`[INFO ][cpuinfo] Supports: Machine Check Exceptions`
	`[INFO ][cpuinfo] Supports: CMPXCHG8B Instruction`
	`[INFO ][cpuinfo] Supports: Fast System Call`
	`[INFO ][cpuinfo] Supports: Memory Type Range Registers`
	`[INFO ][cpuinfo] Supports: Page Global Enable`
	`[INFO ][cpuinfo] Supports: Machine Check Architecture`
	`[INFO ][cpuinfo] Supports: Conditional Mov Instruction`
	`[INFO ][cpuinfo] Supports: Page Attribute Table`
	`[INFO ][cpuinfo] Supports: the CLFLUSH Instruction`
	`[INFO ][cpuinfo] Supports: the Debug Trace Store feature`
	`[INFO ][cpuinfo] Supports: ACPI registers in MSR space`
	`[INFO ][cpuinfo] Supports: Intel Architecture MMX Technology`
	`[INFO ][cpuinfo] Supports: Fast Float Point Save and Restore`
	`[INFO ][cpuinfo] Supports: Streaming SIMD extensions`
	`[INFO ][cpuinfo] Supports: Streaming SIMD extensions 2`
	`[INFO ][cpuinfo] Supports: Self-Snoop`
	`[INFO ][cpuinfo] Supports: Thermal Monitor`

exceptions	Displays exception types and messages (excluding the common types of exceptions). Verbose output for `exceptions` might look like this: `[excepti][00002] java/lang/NumberFormatException: null`

Table 2–6 (Cont.) -Xverbose Parameters

Parameter	Prints to the screen

exceptions=debug	Displays exception types and messages (excluding the common types of exceptions). It also displays stacktraces; Verbose output for exceptions=debug might look like this:
	`[excepti][00002] java/lang/NumberFormatException: null`
	`at java/lang/Integer.parseInt(Ljava/lang/String;I)I(Integer.java:415)`
	`at java/lang/Integer.<init>(Ljava/lang/String;)V(Integer.java:620)`
	`at sun/net/InetAddressCachePolicy.<clinit>()V (InetAddressCachePolicy.java:77)`
	`at jrockit/vm/RNI.c2java(IIII)V(Native Method)`
	`at jrockit/vm/RNI.generateFixedCode(I)I(Native Method)`
	`at java/net/InetAddress.<clinit>()V(InetAddress.java:640)`
	`at jrockit/vm/RNI.c2java(IIII)V(Native Method)`
	`at jrockit/vm/RNI.generateFixedCode(I)I(Native Method)`
	`at java/net/InetSocketAddress.<init>(Ljava/lang/String;I)V (InetSocketAddress.java:124)`
	`at java/net/Socket.<init>(Ljava/lang/String;I)V (Socket.java:178)`
	`at Ex.main([Ljava/lang/String;)V(Ex.java:5)`
	`at jrockit/vm/RNI.c2java(IIII)V(Native Method)`
	`--- End of stack trace`

exceptions=trace	The same information as debug, but includes the common types of exceptions. Verbose output for exceptions=trace looks the same as -Xverbose:exceptions=debug but also prints exceptions of types: ■ java.util.EmptyStackException ■ java.lang.ClassNotFoundException ■ java.security.PrivilegedActionException

gc	Displays information about the memory system in the following format:
	`[INFO][memory ] <start>-<end>: <type> <before>KB-><after>KB (<heap>KB), <time> ms, sum of pauses <pause> ms.`
	`[INFO][memory ] <start> - start time of collection (seconds since jvm start).`
	`[INFO][memory ] <type> - OC (old collection) or YC (young collection).`
	`[INFO][memory ] <end> - end time of collection (seconds since jvm start).`
	`[INFO][memory ] <before> - memory used by objects before collection (KB).`
	`[INFO][memory ] <after> - memory used by objects after collection (KB).`
	`[INFO][memory ] <heap> - size of heap after collection (KB).`
	`[INFO][memory ] <time> - total time of collection (milliseconds).`
	`[INFO][memory ] <pause> - total sum of pauses during collection (milliseconds).`
	
	For example:
	`[INFO][memory] [YC#1] 0.749-0.992: YC 32768KB->32770KB (65536KB), 0.242 s, sum of pauses 242.279 ms, longest pause 242.279 ms.`
	`[INFO][memory] [YC#2] 1.029-1.221: YC 57344KB->65536KB (65536KB), 0.191 s, sum of pauses 191.391 ms, longest pause 191.391 ms.`
	`[INFO][memory] [OC#1] 1.221-1.305: OC 65536KB->57169KB (102512KB), 0.085 s, sum of pauses 83.469 ms, longest pause 83.469 ms.`
	`[INFO][memory] [YC#3] 1.334-1.343: YC 65118KB->60949KB (102512KB), 0.009 s, sum of pauses 8.534 ms, longest pause 8.534 ms.`

Table 2–6 (Cont.) -Xverbose Parameters

Parameter	Prints to the screen
`gcheuristic`	Information about the decisions that the garbage collection heuristics make and also the heap size changes.

`gcpause`	Prints the pause times caused by the garbage collector during a run. The pause times are shown during run time on your screen during the running of the application. Use this option at startup. As pauses are encountered, a report is printed.
	Output from `-Xverbose:gcpause` used with `-Xgc:gencon`:
	`[INFO ][gcpause] [YC#92] [---] 4.145 ms (26.121000-26.125000) YC`
	`[INFO ][gcpause] [YC#92] [con] 0.003 ms (26.121000-26.121000) YC:PreGC`
	`[INFO ][gcpause] [YC#92] [pau] 4.095 ms (26.121000-26.125000) YC:Main`
	`[INFO ][gcpause] [YC#92] [con] 0.008 ms (26.125000-26.125000) YC:PostGC`
	`[INFO ][gcpause] [YC#93] [---] 4.710 ms (26.352000-26.356000) YC`
	`[INFO ][gcpause] [YC#93] [con] 0.002 ms (26.352000-26.352000) YC:PreGC`
	`[INFO ][gcpause] [YC#93] [pau] 4.644 ms (26.352000-26.356000) YC:Main`
	`[INFO ][gcpause] [YC#93] [con] 0.009 ms (26.356000-26.356000) YC:PostGC`
	`[INFO ][gcpause] [OC#1] [---] 63.312 ms (26.362000-26.425000) OC`
	`[INFO ][gcpause] [OC#1] [con] 0.006 ms (26.362000-26.362000) OC:PreGC`
	`[INFO ][gcpause] [OC#1] [pau] 4.979 ms (26.362000-26.367000) OC:Initial`
	`[INFO ][gcpause] [OC#1] [con] 50.462 ms (26.367000-26.417000)`
	`OC:ConcurrentMark`
	`[INFO ][gcpause] [OC#1] [pau] 4.427 ms (26.417000-26.422000) OC:Main`
	`[INFO ][gcpause] [OC#1] [con] 0.829 ms (26.422000-26.422000)`
	`OC:ConcurrentSweep1`
	`[INFO ][gcpause] [OC#1] [pau] 0.011 ms (26.422000-26.422000) OC:SweepSwitch`
	`[INFO ][gcpause] [OC#1] [con] 1.137 ms (26.422000-26.424000)`
	`OC:ConcurrentSweep2`
	`[INFO ][gcpause] [OC#1] [pau] 0.224 ms (26.424000-26.424000) OC:Cleanup`
	`[INFO ][gcpause] [OC#1] [con] 0.982 ms (26.424000-26.425000) OC:PostGC`
	`[INFO ][gcpause] [YC#94] [---] 4.738 ms (26.720000-26.725000) YC`
	`[INFO ][gcpause] [YC#94] [con] 0.003 ms (26.720000-26.720000) YC:PreGC`
	`[INFO ][gcpause] [YC#94] [pau] 4.692 ms (26.720000-26.725000) YC:Main`
	`[INFO ][gcpause] [YC#94] [con] 0.007 ms (26.725000-26.725000) YC:PostGC`

`gcpausetree`	Prints the same information as `gcpause`, but with indentation instead of full level information, therefore makes the output more readable. But this output is less parsable.

Table 2–6 (Cont.) -Xverbose Parameters

Parameter	Prints to the screen

`gcreport`	Generates a report that shows garbage collection statistics for your application. You can use this report to determine if you are using the most effective garbage collector. The report divides the statistics into young collections and old collections, and for each type the following information gathered during the run time is displayed
	■ Number of collections: The total number of garbage collections of this type.
	■ Total promoted: The total number of objects and bytes promoted from young space to old space by this type of garbage collections.
	■ Max promoted: The maximum number of objects and bytes promoted by any single garbage collection of this type.
	■ Total GC time: The total time spent in this type of garbage collections. For concurrent garbage collections, the total garbage collection time and the total garbage collection pause time differs.
	■ Mean GC time: The average time spent in a single garbage collection of this type. For concurrent garbage collections, the garbage collection time and the garbage collection pause time differs.
	■ Maximum GC pauses: The three longest garbage collection pauses caused by this type of garbage collection.
	Output from `-Xverbose:gcreport` used with `-Xgcprio:pausetime`:
	`[INFO ][gcrepor]`
	`[INFO ][gcrepor] Memory usage report:`
	`[INFO ][gcrepor]`
	`[INFO ][gcrepor] Young Collections:`
	`[INFO ][gcrepor] number of collections = 1674.`
	`[INFO ][gcrepor] total promoted = 125334074 (size 4556008384).`
	`[INFO ][gcrepor] max promoted = 672140 (size 29971808).`
	`[INFO ][gcrepor] total YC time = 12.959 s (total paused 12.810 s).`
	`[INFO ][gcrepor] mean YC time = 7.741 ms (mean total paused 7.652 ms).`
	`[INFO ][gcrepor] maximum YC Pauses = 38.941 , 40.473, 63.004 ms.`
	`[INFO ][gcrepor]`
	`[INFO ][gcrepor] Old Collections:`
	`[INFO ][gcrepor] number of collections = 828.`
	`[INFO ][gcrepor] total promoted = 37563437 (size 1393626896).`
	`[INFO ][gcrepor] max promoted = 328350 (size 14209984).`
	`[INFO ][gcrepor] total OC time = 349.090 s (total paused 83.380 s).`
	`[INFO ][gcrepor] mean OC time = 421.606 ms (mean total paused 100.701 ms).`
	`[INFO ][gcrepor] maximum OC Pauses = 445.945 , 446.394, 3096.186 ms.`
	`[INFO ][gcrepor]`
	`[INFO ][gcrepor] number of emergency parallel sweeps = 361.`
	`[INFO ][gcrepor]`
	`[INFO ][gcrepor] number of internal compactions = 582.`
	`[INFO ][gcrepor] number of internal compactions skipped because pointer storage overflowed = 8.`
	`[INFO ][gcrepor] number of external compactions = 234.`
	`[INFO ][gcrepor] 177 of these were aborted because they timed out.`
	`[INFO ][gcrepor] number of external compactions skipped because pointer storage overflowed = 4.`
	`[INFO ][gcrepor]`

Table 2–6 (Cont.) -Xverbose Parameters

Parameter	Prints to the screen

`load`	The name of each loaded Java or native library:
	`[INFO ][load ] opened zip c:\jrockits\R28.0.0_R28.0.0-617_1.6.0\jre\lib\rt.jar`
	`[INFO ][load ] opened zip c:\jrockits\R28.0.0_R28.0.0-617_1.6.0\jre\lib\resources.jar`
	`[INFO ][load ] opened zip c:\jrockits\R28.0.0_R28.0.0-617_1.6.0\jre\lib\jsse.jar`
	`[INFO ][load ] opened zip c:\jrockits\R28.0.0_R28.0.0-617_1.6.0\jre\lib\jce.jar`
	`[INFO ][load ] opened zip c:\jrockits\R28.0.0_R28.0.0-617_1.6.0\jre\lib\charsets.jar`

`memory`	Prints information about the memory management system, including: ■ Start time of collection (seconds since JVM start) ■ End time of collection (seconds since JVM start) ■ Memory used by objects before collection (KB) ■ Memory used by objects after collection (KB) ■ Size of heap after collection (KB) ■ Total time of collection (seconds or milliseconds) ■ Total pause time during collection (milliseconds) The information printed by `-Xverbose:memory` varies depending on the type of garbage collector you are using.

Table 2–6 (Cont.) -Xverbose Parameters

Parameter	Prints to the screen

memdbg	Enables debug level for verbose modules that are useful when you debug memory-related issues. Verbose output for memdbg might look like this:
	`[DEBUG][memory ] [YC#3526] GC reason: Allocation request failed.`
	`[DEBUG][memory ] [YC#3526] 455.018: YC started.`
	`[INFO ][alloc ] [YC#3526] Pending requests at 'Before YC' - Total: 8, TLAs: 8 (approx 262144 bytes), objects: 0 (0 bytes). Max age: 0.`
	`[INFO ][nursery] [YC#3526] Young collection 3526 started. This YC is running while the OC is in phase: not running.`
	`[DEBUG][memory ] [YC#3526] Promotion failed: not enough free memory for java/lang/String, 32 B.`
	`[DEBUG][memory ] [YC#3526] Returning duplicate cardTablePart entry 2895 (0x00000000155EDE00-0x000000001562DE00)`
	`[DEBUG][memory ] [YC#3526] Returning duplicate cardTablePart entry 2895 (0x00000000155EDE00-0x000000001562DE00)`
	`[DEBUG][memory ] [YC#3526] SemiRef phase Finalizers run in single threaded mode.`
	`[DEBUG][memory ] [YC#3526] SemiRef phase WeakJNIHandles run in single threaded mode.`
	`[DEBUG][memory ] [YC#3526] SemiRef phase ClassConstraints run in single threaded mode.`
	`[INFO ][nursery] [YC#3526] Setting forbidden area for keeparea: 0x00000000133479A0-0x000000001386DE20.`
	`[INFO ][nursery] [YC#3526] Next keeparea will start at 0x0000000012F34E48 and end at 0x00000000133479A0.`
	`[INFO ][alloc ] [YC#3526] Pending requests at 'After YC' - Total: 8, TLAs: 8 (approx 262144 bytes), objects: 0 (0 bytes). Max age: 0.`
	`[DEBUG][memory ] [YC#3526] YC promoted 10964 objects (384KB).`
	`[DEBUG][memory ] [YC#3526] Page faults before YC: 283554, page faults after YC: 283554, pages in heap: 65536.`
	`[DEBUG][memory ] [YC#3526] Nursery size after YC: 0KB. (Free: 0KB Parts: 0)`
	`[INFO ][memory ] [YC#3526] 455.018-455.033: YC 261708KB->262144KB (262144KB), 0.015 s, sum of pauses 14.090 ms, longest pause 14.090 ms.`

opt	Information about all methods that get optimized. Verbose output for opt might look like this:
	`[INFO ][opt ][00036] #1 (Opt) ObjAlloc.main([Ljava/lang/String;)V`
	`[INFO ][opt ][00036] #1 3.756-3.758 0x0000000100060000-0x000000010006004E 2.10 ms 128KB 7633 bc/s (2.10 ms 7633 bc/s)`

refobj	Information on reference objects and handles at each garbage collection. The output is a summary of reference objects of different types and how many of them are activated. A reference object is activated when the requirements for the reference object type are fulfilled. Upon activation, the memory management system can clear the reference, enqueue it in a reference queue or enqueue it for finalization, depending on the type of reference. The performance overhead of this log module is low on info level. On debug level, the performance overhead is high.

Table 2–6 (Cont.) -Xverbose Parameters

Parameter	Prints to the screen
starttime	The values of `System.currentTimeMillis()` and `System.nanoTime()` at the time JRockit JVM started. These can be used to correlate log output between different processes. Verbose output for `starttime` might look like this: `[INFO ][startti] VM start time: 1260962573921 millis 6922526 nanos 18442244770397334 ticks` Where: ■ `millis` is the number of milliseconds elapsed since midnight, January 1, 1970 UTC. This is same value that `System.currentTimeMills`() would render. ■ `nanos` measures time to the resolution of one-billionth of a second (a nanosecond); however, the time from which `nanoTime()` is measured (the start time) is unspecified so that the most efficient method of measurement for different operating systems can be used. It is the same value that `System.nanoTime()` would render.

shutdown	Information about any event that has triggered a normal shutdown of the JVM (not guaranteed to work under all platforms or conditions). This parameter is available since R28.3.2:
	The following output is an example of a verbose shutdown report when a Signal 15 (SIGTERM) is received on a Solaris system:
	`[INFO ][shutdow] JVM_Halt() called in response to:`
	`[INFO ][shutdow] Signal 15 received from PID:21152 UID:142`

systemgc	Notifies of garbage collections started by a call to `System.gc()` and `-Xverbose:memdbg` outputs, for example a call to a JMAPI function that implicitly starts a garbage collection or to the diagnostics command `runsystemgc`.
	A garbage collection started by a direct call to `System.gc()` results in a verbose output similar to:
	`[INFO ][sysgc  ] GC requested by thread 1`
	The thread number in this output is the thread ID of the thread that requests the garbage collection.
	The output for a garbage collection started by other means display the reason for the garbage collection, for example:
	`[INFO ][sysgc ] GC triggered for reason: Set Nursery Size`

timing	The timer resolution and the method used to get a time value. This is the resolution of the timer used by the `System.nanoTime()` method.
	The following output is an example of a verbose timing report on Windows:
	`[INFO ][timing ] Fast time frequency is 1995000000hz`
	`[INFO ][timing ] Drift is 0.00000021 = per day 0.018secs (max 300.000)`
	`[INFO ][timing ] Hardware fast time enabled`
	`[INFO ][timing ] Counter timer using resolution of 1995MHz`

Table 2–7 -Xverbose Log Levels

Log Level	Description
quiet	No logging. No messages or errors are generated.
error	Only error messages are logged.
warn	Warning messages are logged along with errors. Still a low logging level, `warn` is usually used to warn about events that could possibly lead to an error later on.
info	At the `info` level, not only are errors and warnings logged, but also informational messages about the current state of JRockit JVM and various JVM events. This is the default logging level if `-Xverbose` is used without arguments.

Table 2–7 (Cont.) -Xverbose Log Levels

Log Level	Description
`debug`	`debug` logs messages with detailed information of JRockit JVM's behavior. Usually, `debug` provides too much information for day to day logging, but useful for debugging.
`trace`	`trace` provides very verbose logging. This level is used by modules where even the debug level would be cluttered by the amount of information generated. Typically, `trace` is used when up to ten or one hundred pages of text per minute needs to be logged.

Example

```
java -Xverbose:gcpause=debug myClass
```

Enables pause time sampling and information during a run and logs messages with detailed information of the JRockit JVM.

Related Options

`-Xverbose` must be set for the following options to work:

- -Xverbosedecorations
- -XverboseLog
- -XverboseTimeStamp

-Xverbosedecorations

Decorations are additional information, usually system-related, that are used to enhance the meaningfulness of verbose output; for example, the name of the module in which the message originated or number of milliseconds elapsed since the current JRockit JVM session started. The `-Xverbosedecorations` option adds this information to the verbose output.

Format

```
-Xverbosedecorations=decoration names
```

> **Note:** You can also use the diagnostic command `verbosity` with the argument `decorations`.

Table 2–8 lists the possible decorations.

Table 2–8 Verbose Output Decorations

Decoration	Description
`level`	Prints the logging level for the message.
`millis`	Prints the number of milliseconds elapsed since midnight, January 1, 1970 UTC. This is same value that would be generated by `System.currentTimeMills`().
`millisstart`	Prints the number of milliseconds elapsed since JRockit JVM started.
`module`	Prints the module in which the message originated, same as the arguments to `-Xverbose`.
`nanos`	Prints the same value that `System.nanoTime()` would render. "nanoTime" measures time to the resolution of one-billionth of a second (a nanosecond); however, the time from which `nanoTime()` is measured (the start time) is unspecified so that the most efficient method of measurement for different operating systems can be used.
`nanosstart`	Prints the number of nanoseconds since JRockit JVM started.
`pid`	Prints the process ID.
`threadid`	Prints the index of the thread. This is the same value provided by `idx` in thread dumps.
`timestamp`	Prints a human readable timestamp. This is the same value you would receive if you used the -XverboseTimeStamp option.

Example

```
java -Xverbose:gcpause -Xverbosedecorations=timestamp,module myApp
```

The output includes the following decorations:

- A readable *timestamp*.
- The name of the *module* in which the message originated.

Default Values

If you use `-XverboseDecorations` without specifying a decoration, the verbose output displays the module, timestamp, and pid (in that order); for example:

```
D:\jrockits\R28.0.0-617_1.6.0\bin>java -Xverbose:load -Xverbosedecorations -cp
L:\src\ HelloWorld
[load   ][Wed Sep 13 19:43:14 2006][00728] opened zip D:\jrockits\R28.0.0-617_
1.6.0\jre\lib\rt.jar
```

Related Options

`-Xverbosedecorations` can be used only if -Xverbose is also set.

-XverboseLog

The `-XverboseLog` option sends messages (such as verbose output and error messages) from the Oracle JRockit JVM to the specified file instead of `stderr`.

Format

```
-Xverboselog:myFile.txt
```

When this command is used with a filename and extension (for example, `myFile.txt`), the JVM writes any logging information to the specified file.

Example

```
java -Xverbose:gcpause -Xverboselog:verboseText.txt myApp
```

Writes verbose logging information for the application with the main class `myApp` to a file named `verboseText.txt`.

Related Options

The `-XverboseLog` option works only when -Xverbose is set.

Exceptions

The `-XverboseLog` option does not print to the screen.

-XverboseTimeStamp

The `-XverboseTimeStamp` option adds a timestamp to the verbose printout, which can be useful when logging events.

Format

```
-XverboseTimeStamp
```

You can force a timestamp to print out with other information generated by `-Xverbose` if you combine it with the command `-XverboseTimeStamp`.

Example

```
java -Xverbose -XverboseTimeStamp myApp
```

The printout generated by `-XverboseTimeStamp` precedes the information printed by -Xverbose, as follows:

```
L:\src>D:\jrockits\R28.0.0-617_1.6.0\bin\java -Xverbose
-XverboseTimeStamp HelloWorld
[load   ][Mon Sep 25 09:57:56 2006][00624] opened zip
D:\jrockits\R28.0.0-617_1.6.0\jre\lib\rt.jar
```

Related Options

The `-XverboseTimeStamp` option is only effective if verbose logging is enabled either by using -Xverbose or by enabling it at run time.

-Xverify

The `-Xverify` option sets the mode of the bytecode verifier. Bytecode verification ensures that class files are properly formed and meet the constraints listed in Verification of class Files in the The Java Virtual Machine Specification. Do not turn off verification as this reduces the protection provided by Java and could cause problems due to ill-formed class files.

Format

`-Xverify:`*`parameter`*

Combine this option with one of the parameters described in Table 2–9.

Table 2–9 -Xverify Parameters

Parameter	Description
`none`	Does not verify the bytecode. **Note:** Use of `-Xverify:none` is not supported.
`remote`	Verifies those classes that are not loaded by the bootstrap class loader. This is the default behavior if you do not specify the `-Xverify` option.
`all`	Verifies all classes.

Default Value

If you do not use `-Xverify`, by default, the JVM verifies only those classes that are not loaded by the bootstrap class loader (`-Xverify:remote`).

Related Options

-XX:+|-FailOverToOldVerifier

3

-XX Command-Line Options

This chapter describes the `-XX` command-line options of Oracle JRockit JVM; these options are all prefixed by `-XX`.

To implement some of the options, specific system requirements must be met, otherwise, the particular option does not work. Oracle recommends that you use these options only if you have a thorough understanding of your system. Improper usage of these options can affect the stability or performance of your system.

> **Note:** The `-XX` options are subject to change at any time.

In this chapter, all the `-XX` command-line options that you can use with the JRockit JVM are listed in the alphabetical order.

> **Notes:**
>
> - Command-line options are case sensitive unless explicitly stated. Most of the commands use the camel notation (for example, `-XXgcThreads` and `-XXcompaction`).
> - Some command-line options use the HotSpot implementation format; that is, you must place the colon (:) between the `-XX` and the option name followed by a the necessary operator to indicate enabling (+) or disabling (-) the new hash function.
> - If you do not add a unit with the values of options that specify memory size, you get the exact value; for example, 64 is considered as 64 bytes, not 64 megabytes or 64 kilobytes.

-XX:+|-CrashOnOutOfMemoryError

-XX:+|-DisableAttachMechanism

-XXdumpFullState

-XXdumpSize

-XX:ExceptionTraceFilter

-XX:+|-ExitOnOutOfMemoryError

-XX:ExitOnOutOfMemoryErrorExitCode

-XXexternalCompactRatio (deprecated)

-XX:+|-FailOverToOldVerifier

-XX:+|-FlightRecorder

-XX:FlightRecorderOptions

-XX:+|-FlightRecordingDumpOnUnhandledException

-XX:FlightRecordingDumpPath

-XXfullSystemGC

-XXgcThreads

-XX:GCTimeRatio

-XX:GCTimePercentage

-XXgcTrigger

-XX:+|-HeapDiagnosticsOnOutOfMemoryError

-XX:HeapDiagnosticsPath

-XX:+|-HeapDumpOnCtrlBreak

-XX:+|-HeapDumpOnOutOfMemoryError

-XX:HeapDumpPath

-XX:HeapDumpSegmentSize

-XXheapParts (deprecated)

-XXinternalCompactRatio (deprecated)

-XX:+|-JavaDebug

-XXkeepAreaRatio

-XXlargeObjectLimit (deprecated)

-XX:MaxCodeMemory

-XX:MaxDirectMemorySize

-XX:MaximumNurseryPercentage

-XX:MaxLargePageSize

-XX:MaxRecvBufferSize

-XXminBlockSize (deprecated)

-XXnoSystemGC

-XX:OptThreads

-XX:+|-RedoAllocPrefetch

-XX:+|-ReserveCodeMemory

-XX:SegmentedHeapDumpThreshold

-XXsetGC (deprecated)

-XX:StartFlightRecording

-XX:+|-StrictFP

-XXtlaSize

-XX:TreeMapNodeSize

-XX:+|-UseAdaptiveFatSpin

-XX:+|-UseAllocPrefetch

-XX:+|-UseCallProfiling

-XX:+|-UseCfsAdaptedYield

-XX:+|-UseClassGC

-XX:+|-UseCPoolGC

-XX:+|-UseFastTime

-XX:+|-UseFatSpin

-XX:+|-UseLargePagesFor[Heap|Code]

-XX:+|-UseLazyUnlocking

-XX:+|-UseLockProfiling

-XX:+|-UseLowAddressForHeap

-XX:+|-UseNewHashFunction

-XX:+|-UseThreadPriorities

-XXaggressive

The `-XXaggressive` option is a collection of configurations that make the JVM perform at a high speed and reach a stable state as soon as possible. To achieve this goal, the JVM uses more internal resources at startup; however, it requires less adaptive optimization once the goal is reached.

What this option configures is subject to change between releases.

Format

```
-XXaggressive
```

Example

```
java -XXaggressive myApp
```

Related Options

The `-XXaggressive` option sets several things, which can be reset or changed by adding the explicit options on the command line after the `-XXaggressive` option.

-XX:AllocChunkSize

When you combine this option with `-XX:+UseAllocPrefetch`, the `-XX:AllocChunkSize` option sets the size of the chunks to be cleared.

Format

`-XX:+UseAllocPrefetch -XX:AllocChunkSize=`*`size`*`[k|K][m|M][g|G]`

Example

```
java -XX:+UseAllocPrefetch -XX:AllocChunkSize=1K myApp
```

Default Value

The default value varies depending on the platform.

Related Options

-XX:+ | -UseAllocPrefetch

-XX:+|-CheckJNICalls

If you enable this option at startup, the JRockit JVM verifies all arguments to JNI calls, and when it detects an illegal JNI call, the JVM terminates with an error message describing the transgression.

Format

```
-XX:+|-CheckJNICalls
```

Example

```
java -XX:+CheckJNICalls myApp
```

Default

Disabled

Related Options

`-XX:+CheckJNICalls` is equivalent to -Xcheck:jni.

-XX:+|-CheckStacks

This option specifies whether the JVM should explicitly check for stack overflows on a JNI method entry.

Format

```
-XX:+|-CheckStacks
```

Example

```
java -XX:+CheckStacks myApp
```

Default

In JRockit versions R28.0.0 to R28.3.1, this option is disabled by default.

In JRockit R28.3.2 and later versions, this option is enabled by default.

-XXcompaction

Compaction moves live objects closer together in the Java heap to create larger, contiguous free areas that can be used for allocation of large objects. When the JVM compacts the heap, all threads should be paused because the objects are being moved around. To reduce pause time, only a part of the heap is compacted.

Format

```
-XXcompaction:parameter1=value1[,parameter2=value2]
```

Table 3–1 lists the parameters that you can specify for the `-XXcompaction` option.

Table 3–1 Parameters for -XXcompaction

Parameter	Description	Default Value
`abortable`	Specifies that the compactions are possible to abort.	If the garbage collection type is deterministic or pausetime, the default value is `true`. Otherwise, the default value is `false`. **Note:** You cannot set this option to false.
`enable`	Enables compaction. When set to `false`, this option disables compaction during garbage collection. Disabling compaction can reduce garbage collection pause times, but might also lead to fragmentation in the Java heap and lower the application throughput or cause an out-of-memory error. During every garbage collection, at least a partial compaction is done. If you prefer no compaction, you must use this command at startup to disable compaction, but compaction still occurs in the following cases: ■ If you shrink the heap, compaction is performed to move objects that reside on the top of the heap. ■ When object allocations fail several times due to fragmentation, compaction occurs. ■ During a full garbage collection triggered by an external API (such as SystemGC, a diagnostic command, and `jrockit.vm.GC.runFullGc`), compaction occurs.	`true` **Note:** The valid value for this option is `true`. When this option is set to `false`, a warning is printed.
`externalPercentage`	Sets the percent of the heap to compact during external compaction. **Note:** The `-XXcompaction:percentage` option sets values for both `-XXcompcation:internalPercentage` and `-XXcompaction:externalPercentage`.	If the compaction is abortable, the default value is 0.78 percent of the heap. Otherwise, the default value is 6.25 percent of the heap.

Table 3–1 (Cont.) Parameters for -XXcompaction

Parameter	Description	Default Value
`full`	Causes full compaction at all times, compacting the entire heap at each old collection. Full compaction can increase the application throughput by minimizing the fragmentation of the heap but can also cause extremely long garbage collection pauses during the compaction. **Example:** `java` **`-XXcompaction:full`** *`myApp`* Enter this command at startup to force full compaction. This is the only way to ensure that full compaction occurs.	false
`heapParts`	Sets the number of heap parts for compaction.	4096
`initialPercentage`	Sets the percent of the heap to compact during internal compaction.	If the compaction is abortable, the default value is 0.78 percent of the heap. Otherwise, the default value is 6.25 percent of the heap.
`internalPercentage`	Sets the number of heap parts to compact during internal compaction. **Note**: The `-XXcompaction:percentage` option sets values for both `-XXcompcation:internalPercentage` and `-XXcompaction:externalPercentage`.	If the compaction is abortable, the default value is 0.78 percent of the heap. Otherwise, the default value is 6.25 percent of the heap.
`maxReferences`	Sets the maximum number of references to objects in the compaction area. The JRockit JVM compacts a small part of the Java heap at each garbage collection. The references to the objects in the compacted area are stored in a compact set. When running non-deterministic garbage collection, the number of references to the compaction area affects the compaction pause. This option can be used to limit the compaction pauses. **Example:** `java` **`-XXcompaction:maxReferences=10000`** *`myApp`* This command sets the compaction limit to 10,000 references to objects in the compaction area.	If the garbage collection type is pausetime or deterministic, the default value is 10200. Otherwise, the value is 299900.

Table 3–1 (Cont.) Parameters for -XXcompaction

Parameter	Description	Default Value
`maxReferencesPerObject`	Sets the maximum number of references to any single object in the compaction area. If the number of references to an object exceeds this value, the object is not moved during the compaction. When an object is moved during compaction, the references to that object must be updated. Moving an object with a lot of references to it is more costly than moving an object with only a few references to it. **Example:** `java` **`-XXcompaction:maxReferencesPerObject=500`** *`myApp`* This command sets the compaction limit per object to 500 references.	100
`percentage`	Specifies the percent of the heap that the garbage collector compacts at each garbage collection. While the JVM is compacting the heap, all threads that want to access objects need to wait because the JVM is moving the objects around. Consequently, only a part of the heap is compacted to reduce pause time. In some cases, when the garbage collection time is too long, consider reducing the compaction area to reduce the pause times. In some other cases, especially when you are allocating very large arrays, consider increasing the compaction area to reduce the fragmentation on the heap and make the allocation faster. **Example:** `java` **`-XXcompaction:percentage=10`** *`myApp`* When this command is set in a 500 MB heap, the garbage collector compacts 50 MB of the heap at each old collection. **Note:** The `-XXcompaction:percentage` options sets values for both `-XXcompcation:internalPercentage` and `-XXcompaction:externalPercentage`.	If the compaction is abortable, the default value is 0.78 percent of the heap. Otherwise, the default value is 6.25 percent of the heap.

Example

```
java -XXcompaction:heapParts=8000,internalPercentage=0.5 myApp
```

When you set this option on a 2 GB heap, it compacts 10 MB of the heap at each internal compaction.

Default

Enabled

Exceptions

When using the `-XXcompaction` option, consider the following:

- Do not use parameters that conflict each other in the same `-XXcompaction` option:
  - The `percentage` parameter conflicts with the `internalPercentage` and `externalPercentage` parameters.
  - The `full` parameter conflicts with other percentage parameters such as `percentage`, `initialPercentage`, `internalPercentage`, and `externalPercentage`
  - You cannot set the `abortable` parameter to `false`.
  - When you set the `enable` parameter to `false`, a warning is printed
- Parameters that are specified later in the command line override parameters specified earlier.

-XXcompactRatio (deprecated)

The `-XXcompactRatio` option is deprecated in Oracle JRockit R28. The option works in R28, but Oracle recommends that you use `-XXcompaction:percentage` instead. For more information, see -XXcompaction.

For more information about the format and usage of `-XXcompactRatio`, see the R27 documentation at: `http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/index.html`.

-XXcompactSetLimit (deprecated)

The `-XXcompactSetLimit` option is deprecated in Oracle JRockit R28. The option works in R28, but Oracle recommends that you use `-XXcompaction:maxReferences` instead. For more information, see -XXcompaction.

For more information about the format and usage of `-XXcompactSetLimit`, see the R27 documentation at: `http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/index.html`.

-XXcompactSetLimitPerObject (deprecated)

The `-XXcompactSetLimitPerObject` option is deprecated in Oracle JRockit R28. The option works in R28, but Oracle recommends that you use `-XXcompaction:maxReferencesPerObject` instead. For more information, see -XXcompaction.

For more information about the format and usage of `-XXcompactSetLimitPerObject`, see the R27 documentation at: `http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/index.html`.

-XXcompressedRefs

The `-XXcompressedRefs` option governs the use of compressed references, limiting all pointers stored on the heap to 32 bits. Compressed references use fewer Java heap resources and transport less data on the memory bus and improves the performance. This option also frees space on the heap.

> **Note:** `-XXcompressedRefs` is equivalent to `-XX:CompressedRefs`.

Format

```
-XXcompressedRefs:parameter=value
```

Use the command with one of the parameters listed in Table 3–2 to specify the behavior of compressed references.

Table 3–2 Parameters for -XXcompressedRefs

Parameter	Description
`enable=true|false`	Enables or disables compressed references.
`size=4GB|32GB|64GB`	Specifies the size of compressed references. The default size varies for different heap sizes as listed in Table 3–3.

Examples

```
java -Xgc:pausetime -XXcompressedRefs:enable=true myApp

java -Xgc:pausetime -XXcompressedRefs:size=32GB myApp
```

Default Values

If `-XXcompressedRefs` is not specified, compressed references are enabled on all 64-bit machines as long as the heap size is less than 4 GB. When you use the -Xmx option, the default values vary on the heap size as listed in Table 3–3.

Table 3–3 Default Size of Compressed References

Heap Size	Default Size of Compressed References
-Xmx:3g or lower	4 GB
-Xmx:25g or lower	32 GB
-Xmx:57g or lower	64 GB

Related Options

Other command-line options are affected by `-XXcompressedRefs` as described here:

- If you use this option with an initial heap size (-Xmx) that is too large, execution will stop and an error message are generated.
- If you do not specify compressed references explicitly by using the -XXcompressedRefs option and you specify either an initial heap (`-Xms`) or a maximum heap (`-Xmx`) that is too large (more than 64 GB) for compressed references, compressed references will not be used. The JVM will not stop; it will run normally.

Exceptions

If compressed references are not available on a given hardware platform or operating system, a warning is printed and execution stops.

-XX:+|-CrashOnOutOfMemoryError

If this option is enabled, when an out-of-memory error occurs, the JRockit JVM crashes and produces text and binary crash files.

> **Note:** This option is new in R28.1. It does not work in R28.0.

Note that generation of crash files when the JVM crashes is enabled by default. For more information, see "About Crash Files" in the *Oracle JRockit Diagnostics and Troubleshooting Guide*.

Format

```
-XX:+|-CrashOnOutOfMemoryError
```

Example

```
java -XX:+CrashOnOutOfMemoryError
```

Default

Disabled

Related Options

`-XX:+ExitOnOutOfMemoryError` takes precedence over this option.

-XX:+|-DisableAttachMechanism

This option specifies whether tools (such as `jrcmd` and `jconsole`) are allowed to attach to the JRockit JVM.

Format

```
-XX:+|-DisableAttachMechanism
```

Example

```
java -XX:+DisableAttachMechanism
```

Default

Enabled

-XXdumpFullState

Usually when the JRockit JVM crashes, it saves out the state of the process (called a core dump). When you use `-XXdumpFullState`, the JVM saves all the process state including the heap. More disk space is used, but it makes much easier for you to use the core dump to find out what the problem was that caused the crash. This option saves a significant amount of information to disk.

Format

```
-XXdumpFullState
```

Related Options

`-XXdumpFullState` is equivalent to `-XXdumpsize:large`. For more information, see -XXdumpSize.

-XXdumpSize

The `-XXdumpSize` option causes a dump file to be generated and allows you to specify the relative size of that file.

Format

`-XXdumpsize:`*`size`*

Use the command with one of the parameters listed in Table 3–4 to specify the relative size of the dump file.

Table 3–4 Parameters for -XXdumpsize

FIle Size	Description
`none`	Does not generate a dump file.
`small`	On Windows, a small dump file is generated (on Linux a full core dump is generated). A small dump only include the thread stacks including their traces and very little else.).
`normal`	Causes a normal dump to be generated on all platforms. This dump file includes all memory except the java heap.
`large`	Includes everything that is in memory, including the Java heap. This option makes `-XXdumpSize` equivalent to -XXdumpFullState.

-XX:ExceptionTraceFilter

Specify this option at startup to filter exception logging. JRockit JVM logs only those exceptions that match a type specified by this option.

Format

```
-XX:ExceptionTraceFilter=java.lang.Exception
```

Enter this command at startup to log exceptions of type `java.lang.Exception` class.

Default Values

Disabled

Related Options

None

-XX:+|-ExitOnOutOfMemoryError

When you enable this option, JRockit JVM exits on the first occurrence of an out-of-memory error. It can be used if you prefer restarting an instance of JRockit JVM rather than handling out of memory errors.

Format

```
-XX:+|-ExitOnOutOfMemoryError
```

Enter this command at startup to force JRockit JVM to exit on the first occurrence of an out of memory error.

Default Values

Disabled

Related Options

-XX:ExitOnOutOfMemoryErrorExitCode

-XX:ExitOnOutOfMemoryErrorExitCode

This option specifies the exit code for termination of the JVM process when an out-of-memory error occurs.

Format

`-XX:ExitOnOutOfMemoryErrorExitCode=`*`value`*

Default Value

51

Related Options

This option works with the -XX:+|-ExitOnOutOfMemoryError option

-XXexternalCompactRatio (deprecated)

The `-XXexternalCompactRatio` option is deprecated in Oracle JRockit R28. The option works in R28, but Oracle recommends that you use `-XXcompaction:externalPercentage` instead. For more information, see -XXcompaction.

For more information about the format and usage of `-XXexternalCompactRatio`, see the R27 documentation at: `http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/index.html`.

-XX:+|-FailOverToOldVerifier

This option specifies whether a failover happens to the old verifier when the new type checker fails.

Format

```
-XX:+|-FailOverToOldVerifier
```

Default

Enabled

-XX:+|-FlightRecorder

This option enables or disables the JRockit Flight Recorder. If JRockit Flight Recorder was disabled at application startup, you cannot enable it at run time.

Format

```
-XX:-FlightRecorder
```

Default

This option is enabled, but there are no recordings running by default. Use the -XX:FlightRecorderOptions to specify options and to start a recording.

-XX:FlightRecorderOptions

This option enables or disables the JRockit Flight Recorder arguments. It is used only when the JRockit Flight Recorder is enabled.

Format

-XX:FlightRecorderOptions=*parameter1=value[,parameter2=value]*

Table 3–5 lists the parameters for `-XX:FlightRecorderOptions`.

Table 3–5 Parameters for -XX:FlightRecorderOptions

Parameter	Description	Default Value
`defaultrecording=true|false`	Specifies whether the background recording is enabled or disabled.	false
`disk=true|false`	Specifies whether JRockit Flight Recorder should write the continuous recording to a disk.	false
`dumponexit=true|false`	This parameter is new in R28.1. It does not work in R28.0. Specifies whether a dump of Flight Recording data should be generated when the JVM terminates in a controlled manner. The dump file is written to the location defined by the `dumponexitpath` parameter.	false
`dumponexitpath=`*`path`*	This parameter is new in R28.1. It does not work in R28.0. Specifies the path and name of the dump file (containing Flight Recorder data), which is created (if `dumponexit=true`) when the JVM exits in a controlled manner. If the specified path is a directory, the JVM assigns a filename that shows the creation date and time. If the specified path includes a filename and if that file that already exists, the JVM creates a new file by appending the date- and time-stamp to the specified filename.	
`globalbuffersize=`*`size`*	Specifies the total amount of primary memory used for data retention.	10 MB
`maxage=`*`time`*	Specifies the maximum age of disk data for default recording. This option is valid only when the parameter `disk` is set as `true`.	15 minutes
`maxchunksize=`*`size`*	Specifies the maximum size, in megabytes, of the data chunks in a recording.	12 MB

Table 3–5 (Cont.) Parameters for -XX:FlightRecorderOptions

Parameter	Description	Default Value
`maxsize=`*`size`*	Specifies the maximum size of disk data for default recording. This option is valid only when the parameter `disk` is set as `true`.	Unbound
`repository=`*`file_location`*	Specifies the repository (a directory) for temporary disk storage.	The default location is the system temporary directory.
`settings=`*`file_location`*	Specifies the name and location for the event settings (`.jfs`) file. You can set this option multiple times. For more information about the event settings file, see "Events" in *Oracle JRockit Flight Recorder Run Time Guide*.	`jre/lib/jfr/default.jfs`
`threadbuffersize=`*`size`*	Specifies the per-thread local buffer size. Higher values for this parameter allow more data gathering without contention to flush it to the global storage. It can increase application footprint in a thread-rich environment. For more information about buffers, see "JFR Buffers" in Oracle JRockit Flight Recorder Run Time Guide.	5 KB

Example

```
java -XX:+FlightRecorder -XX:FlightRecorderOptions=disk=true,maxchunksize=10M
MyApp
```

Related Options

This option works only when the -XX:+|-FlightRecorder option is enabled.

-XX:+|-FlightRecordingDumpOnUnhandledException

This option, when enabled, generates a Flight Recording dump when a thread is terminated due to an unhandled exception. The dump file is written to the location defined by the -XX:FlightRecordingDumpPath option.

Format

-XX:+|-FlightRecordingDumpOnUnhandledException

Example

```
java -XX:+FlightRecordingDumpOnUnhandledException myApp
```

Default

Disabled

Related Option

-XX:FlightRecordingDumpPath

-XX:FlightRecordingDumpPath

This option specifies the path and name of the dump file (containing Flight Recorder data), which is created (if the -XX:+|-FlightRecordingDumpOnUnhandledException is enabled) when a thread is terminated due to an unhandled exception.

Format

-XX:FlightRecordingDumpPath=*path*

Example

```
java -XX:+FlightRecordingDumpOnUnhandledException
-XX:FlightRecordingDumpPath=D:\myapp\jfr
```

Default Value

The default Flight Recording dump file is present in the working directory as `jrockit_pid_thread_id.jfr` (where *pid* represents the JRockit process identifier and *thread_id* represents the thread for which the unhandled exception is thrown).

Related Options

-XX:+|-FlightRecordingDumpOnUnhandledException

-XXfullSystemGC

The `-XXfullSystemGC` option causes the garbage collector to do a full garbage collection every time `System.gc()` is called. Full garbage collection includes old space collection and the elimination of soft references. Use this option when you want the garbage collector to do maximum garbage collecting every time you explicitly invoke a garbage collection from Java.

This option is useful when the default garbage collector does not free enough memory; however, using it can cause longer garbage collection pauses.

Format

`-XXfullSystemGC`

When you use this option, if an old space collection is already running when `System.gc()` is called, it will first wait for it to finish and then trigger a new old space collection. The `-XXfullSystemGC` option frees all softly referenced objects.

Exceptions

You cannot use `-XXfullSystemGC` together with -XXnoSystemGC.

-XXgcThreads

This option specifies how many garbage collection threads the garbage collector will use. This applies both to parallel nursery and parallel old space collectors as well as the concurrent and deterministic collector.

Format

-XXgcthreads:*parameter1=value1,parameter2=value2,...*

Use the command with one of the parameters listed in Table 3–6 to specify the number of threads used by the garbage collector.

Note: If you specify a number instead of the *parameter*, that number determines the number of garbage collection threads used for garbage collection during parallel phases. This is equivalent to using the parameter `all=`*number*.

Example:

```
java -XXgcThreads:4 myApp
```

Table 3–6 Parameters for -XXgcThreads

Parameter	Description
`all=`*number of threads*	Specifies the same number of garbage collection threads for all garbage collectors (young, concurrent, and parallel).
`yc=`*number of threads*	Specifies the number of garbage collection threads that the young collector uses in parallel.
`con=`*number of threads*	Specifies the number of garbage collection threads that the old collector uses in parallel during concurrent phases.
`par=`*number of threads*	Specifies the number of garbage collection threads that the old collector uses in parallel during the stopped (paused) phases.

Example

```
java -XXgcThreads:yc=4,con=2,par=4
```

Default Values

By default, these values are based on the number of cores and hardware threads on the machine.

-XX:GCTimePercentage

The `-XX:GCTimePercentage` option determines the percent of time spent in garbage collection of total run time.

Format

```
-XX:GCTimePercentage=nn
```

nn is the time spent in garbage collection.

`-XX:GCTimePercentage` can be used as an alternative to `-XX:GCTimeRatio`.

Example

```
java -XX:GCTimePercentage=5 myApp
```

When you set `-XX:GCTimePercentage` to 5, five percent of the total time is spent in garbage collection; it is equivalent to `-XX:GCTimeRatio=19`.

Default Value

5 percent

Related Options

-XX:GCTimeRatio

-XX:GCTimeRatio

The `-XX:GCTimeRatio` option specifies the ratio of the time spent outside the garbage collection (for example, the time spent for application execution) to the time spent in the garbage collection.

Format

```
-XX:GCTimeRatio=nn
```

nn is a number that specifies the ratio of the time spent (as number of times).

Example

```
java -XX:GCTimeRatio=50 myApp
```

Default Value

The default value is 19, which means that the time outside the garbage collection is 19 times the time spent in garbage collection.

Related Options

-XX:GCTimePercentage

-XXgcTrigger

This option determines how much free memory should remain on the heap when a concurrent garbage collection starts. If the heap becomes full during the concurrent garbage collection, the Java application cannot allocate more memory until the garbage collection frees some heap space, which might cause the application to pause. While the trigger value will tune itself in run time to prevent the heap from becoming too full, this automatic tuning might take too long. Instead, you can use `-XXgcTrigger` to set from the start a garbage collection trigger value more appropriate to your application.

If the heap becomes full during the concurrent mark phase, the sweep phase will revert to parallel sweep. If this happens frequently and the garbage collection trigger does not increase automatically to prevent this, use `-XXgcTrigger` to manually increase the garbage collection trigger.

Format

```
-XXgcTrigger=nn
```

where *nn* is the amount of free heap, as a percent of the heap, available when a garbage collections triggered. Note that the young space is not considered as part of the free heap.

Example

```
java -XXgcTrigger=50 myApp
```

With this option set, JRockit JVM will trigger a garbage collection when 50% of the heap. For example, about 512 MB on a 1 GB heap or less remains free. The current value of the garbage collection trigger will appear in the `-Xverbose:memdbg` outputs whenever the trigger changes.

Default Values

If `-XXgcTrigger` is not specified, the system tries to automatically find a good percentage value. If `-XXgcTrigger=`*nn* is specified, it is used instead and no automatic process is involved.

Exceptions

The garbage collector ignores the `-XXgcTrigger` value when it runs both parallel mark and parallel sweep, for example if you specify `-Xgc:singlepar` or `-Xgc:genpar` on the command line.

-XX:+|-HeapDiagnosticsOnOutOfMemoryError

This option specifies whether the JVM should print Java heap diagnostics when an out-of-memory error occurs. The dump file is written to the location defined by the -XX:HeapDumpPath option.

Format

-XX:+|-HeapDiagnosticsOnOutOfMemoryError

Example

```
java -XX:+HeapDiagnosticsOnOutOfMemoryError -XX:HeapDiagnosticsPath=D:\myapp\diag_
dumps myApp
```

-XX:HeapDiagnosticsPath

This option specifies the path and file name of the dump file if you have enabled the -XX:+|-HeapDiagnosticsOnOutOfMemoryError option.

Format

-XX:HeapDiagnosticsPath=*path*

Example

```
java -XX:+HeapDiagnosticsOnOutOfMemoryError -XX:HeapDumpPath=D:\myApp\diag_dumps
myApp
```

Default Value

The default name of the dump file is `jrockit_pid.oomdiag`, and the default path is the working directory.

Related Options

This option works with the -XX:+|-HeapDiagnosticsOnOutOfMemoryError option.

-XX:+|-HeapDumpOnCtrlBreak

The `-XX:HeapDumpOnCtrlBreak` option adds the `hprofdump` diagnostic command to the list of commands that run automatically when the `Ctrl-break` keys are pressed (similar to the `print_threads` diagnostic command). The HPROF dump file is written to the location defined by the -XX:HeapDumpPath option.

Format

`-XX:+HeapDumpOnCtrlBreak`

Example

```
java -XX:+HeapDumpOnCtrlBreak myApp
```

-XX:+|-HeapDumpOnOutOfMemoryError

If you enable the `-XX:+|-HeapDumpOnOutOfMemoryError` option, the JRockit JVM dumps the Java heap in HPROF binary format (*.hprof* file) when an out-of-memory error occurs.

Format

```
-XX+|-HeapDumpOnOutOfMemoryError
```

Example

```
java -XX:+HeapDumpOnOutOfMemoryError myApp
```

Default

Disabled

Related Options

-XX:HeapDumpPath

-XX:HeapDumpPath

The `-XX:HeapDumpPath` option specifies the path and file name of the HPROF dump file if you have used `-XX:+HeapDumpOnOutOfMemoryError` option.

Format

`-XX:HeapDumpPath`=*`path_of_the_dump_file`*

Example

```
java -XX:+HeapDumpOnOutOfMemory -XX:HeapDumpPath=D:\myApp\hprof-dumps myApp
```

Default Value

The default name of the binary dump file is `jrockit_`*`pid`*`.hprof`, where *`pid`* represents the JRockit process identifier, and the default path is the working directory.

Related Options

This option works with the -XX:+|-HeapDumpOnOutOfMemoryError and -XX:+|-HeapDumpOnCtrlBreak options.

-XX:HeapDumpSegmentSize

The `-XX:HeapDumpSegmentSize` option specifies an appropriate segment size when generating a segmented HPROF heap dump.

Format

`-XX:HeapDumpSegmentSize=`*`size[k|K][m|M][g|G]`*

Example

```
java -XX:+HeapDumpOnOutOfMemory -XX:HeapDumpSegmentSize=512M myApp
```

Default Values

1 GB

Related Options

No segments are generated unless heap size is larger than the value specified by the -XX:SegmentedHeapDumpThreshold option.

-XXheapParts (deprecated)

The `-XXheapParts` option is deprecated in Oracle JRockit R28. The option works in R28, but Oracle recommends that you use `-XXcompaction:heapParts` instead. For more information, see -XXcompaction.

For more information about the format and usage of `-XXheapParts`, see the R27 documentation at: `http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/index.html`.

-XXinternalCompactRatio (deprecated)

The `-XXinternalCompactRatio` option is deprecated in Oracle JRockit R28. The option works in R28, but Oracle recommends that you use `-XXcompaction:internalPercentage` instead. For more information, see -XXcompaction.

For more information about the format and usage of `-XXinternalCompactRatio`, see the R27 documentation at: `http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/index.html`.

-XX:+|-JavaDebug

The `-XX:+|-JavaDebug` option enables or disables the debugging capabilities that the JVM Tools Interface (JVMTI) uses. JVMTI is a low-level debugging interface used by debuggers and profiling tools, to inspect the state and control the execution of applications running in the JVM.

Note that the subset of JVMTI that is most typically used by profilers is always enabled. However, the functionality used by debuggers to be able to step through the code and set break points has some overhead associated with it and is not always enabled. To enable this functionality, use the `-XX:+JavaDebug` option.

> **Caution:** Do not use the `-XX:+JavaDebug` option in the production environment, because, when running with the `-XX:+JavaDebug` option, the JVM does not run at full speed.

Format

```
-XX:+|-JavaDebug
```

Example

```
java -XX:+JavaDebug -agentlib:jdwp=transport=dt_socket,server=y,suspend=n myApp
```

For more information about the `-agentlib` option, see the Java documentation at the following locations:

- Java SE 6.0

  http://java.sun.com/javase/6/docs/technotes/guides/jpda/conninv.html#Invocation

- J2SE 5.0

  http://java.sun.com/j2se/1.5.0/docs/guide/jpda/conninv.html#Invocation

Default

Disabled

Related Options

This option is equivalent to the -Xdebug option.

-XXkeepAreaRatio

The `-XXkeepAreaRatio` option sets the size of the keep area within the nursery as a percent of the nursery. The keep area prevents newly allocated objects from being promoted to old space too early.

Format

`-XXkeepAreaRatio:<percentage>`

Example

`java` **`-XXkeepAreaRatio:10`** *`myApp`*

Sets the keep area size to 10% of the nursery size.

Default Value

By default, the keep area is 25% of the nursery size. The keep area should not exceed 50% of the nursery size.

Exceptions

The keep area ratio is only valid when the garbage collector is generational.

-XXlargeObjectLimit (deprecated)

The `-XXlargeObjectLimit` option is deprecated in Oracle JRockit R28. The option works in R28, but Oracle recommends that you use `-XXtlaSize:wasteLimit` instead. For more information, see -XXtlaSize.

For more information about the format and usage of `-XXlargeObjectLimit`, see the R27 documentation at: `http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/index.html`.

-XX:MaxCodeMemory

The `-XX:MaxCodeMemory` option specifies the maximum memory used for generated code.

Format

`-XX:MaxCodeMemory=`*`size`*`[g|G|m|M|k|K]`

Example

```
java -XX:MaxCodeMemory=2g myApp
```

Default Values

On Windows IA32, Linux IA32, Windows x86_64, and Linux x86_64, the default value is unbounded. If the `-XX:+ReserveCodeMemory` option is specified on these platforms, the default maximum code memory is as follows:

- When you use `-XX:+UseLargePagesForCode`: 64 MB
- When you use `-XX:-UseLargePagesForCode`: 1024 MB

On Solaris SPARC the default maximum code memory is 256 MB.

Related Options

-XX:+|-ReserveCodeMemory

-XX:+|-UseLargePagesFor[Heap|Code]

-XX:MaxDirectMemorySize

This option specifies the maximum total size of java.nio (New I/O package) direct buffer allocations.

Format

-XX:MaxDirectMemorySize=*size*[g|G|m|M|k|K]

Example

```
java -XX:MaxDirectMemorySize=2g myApp
```

Default Value

The default value is zero, which means the maximum direct memory is unbounded.

-XX:MaximumNurseryPercentage

The `-XX:MaximumNurseryPercentage` option allows you to set an upper nursery size limit that is relative to the free heap space available after the latest old collection. You must specify the limit as a percentage value of the available free heap size.

Format

`-XX:MaximumNurseryPercentage=<value> [1-95]`

If you try to set the upper nursery size limit to a value lower than 1 or higher than 95, an error message is displayed.

Example

`java` **`-XX:MaximumNurseryPercentage=80`** *`myApp`*

Default Value

95 percent

Exceptions

This option has no effect unless a generational garbage collector is being used.

-XX:MaxLargePageSize

This option specifies the maximum size for large pages.

Format

-XX:MaxLargePageSize=*size*[g|G|m|M|k|K]

Example

```
java -XX:MaxLargePageSize=2g myApp
```

Default Value

The default value is zero in JRockit versions R28.0.0 to R28.2.2. This means that this value is specified by the operating system.

For JRockit versions R28.2.3 and later, the default value is 256 MB.

Related Options

The option is valid only if -XlargePages or -XX:+|-UseLargePagesFor[Heap|Code] is specified.

-XX:MaxRecvBufferSize

This option specifies the maximum size of the receive buffer when reading from network sockets.

This option is applicable only to Windows.

> **Note:** This option is new in R28.1. It does not work in R28.0.
>
> If you set this option to 0, the receive buffer size is unlimited, which is the behavior in R28.0.

Format

-XX:MaxRecvBufferSize=*size*[g|G|m|M|k|K]

Example

```
java -XX:MaxRecvBufferSize=32k myApp
```

Default Value

64k

-XXminBlockSize (deprecated)

The `-XXminBlockSize` option is deprecated in Oracle JRockit R28. The option works in R28, but Oracle recommends that you use `-XXtlaSize:min` instead. For more information, see -XXtlaSize.

For more information about the format and usage of `-XXminBlockSize`, see the R27 documentation at: `http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/index.html`.

-XXnoSystemGC

The `-XXnoSystemGC` option prevents a call to the `System.gc()` method from starting a garbage collection. For more information about the System.gc() method, see the specification for the `java.lang.System` at the following locations:

- Java SE 6.0

  `http://java.sun.com/javase/6/docs/api/java/lang/System.html`

- J2SE 5.0

  `http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html`

If your application uses `System.gc()` and you want to the garbage collector itself to decide when to run the collection, you should use this option. This option is useful in some debugging situations and can also enhance performance as it prevents unnecessary garbage collection from happening.

Format

```
-XXnoSystemGC
```

Enter the command in the above format at startup. This option can cause longer garbage collection pauses in some cases, but generally, it makes the application perform better.

Exceptions

You cannot use `-XXnoSystemGC` together with -XXfullSystemGC.

-XX:OptThreads

The `-XX:OptThreads` option specifies the number of threads the JVM uses for the optimization of Java methods. The optimization of Java methods runs in the background.

Format

```
-XX:OptThreads=nn
```

Example

```
java -Xgc:pausetime -XX:OptThreads=3 myApp
```

Directs the compiler to use three threads for optimization tasks.

Default Value

By default, one thread is used for the optimization of Java methods.

-XX:+|-RedoAllocPrefetch

With this option, an additional chunk (that is, two chunks subsequent) is fetched whenever a new chunk is used.

> **Note:** To get the full benefit from this feature on Intel Xeon servers, Oracle recommends that you disable hardware prefetching in the BIOS of the computer.

Format

```
-XX:+RedoAllocPrefetch
```

Example

```
java -XX:+UseAllocPrefetch -XX:+RedoAllocPrefetch myApp
```

Default

Enabled

Exceptions

This option will not work unless -XX:+|-UseAllocPrefetch is set.

-XX:+|-ReserveCodeMemory

When this option is enabled, the JVM reserves the memory for generated code at startup.

Format

```
-XX:+ReserveCodeMemory
```

Example

```
java -XX:+ReserveCodeMemory myApp
```

Default

This option is enabled by default on Solaris SPARC, Windows x86_64, and Linux x86_64 platforms. The default value depends on the `-XX:+|-UseLargePagesForCode` option as follows:

- When you use `-XX:+UseLargePagesForCode`: 64 MB
- When you use `-XX:-UseLargePagesForCode`: 1024 MB

On Solaris SPARC the default code memory is 256 MB.

Related Option

-XX:+|-UseLargePagesFor[Heap|Code]

-XX:SegmentedHeapDumpThreshold

The `-XX:SegmentedHeapDumpThreshold` option generates a segmented heap dump (.hprof file, 1.0.2 format) when the heap usage is larger than the specified size.

The segmented hprof dump format is required to correctly generate heap dumps containing more than 4 GB of data. If the value of `-XX:SegmentedHeapDumpThreshold` option is set more than 4 GB, heap dumps may not be generated correctly.

Format

```
-XX:SegmentedHeapDumpThreshold=size
```

Example

```
java -XX:SegmentedHeapDumpThreshold=512M myApp
```

Default Value

2 GB

Related Options

This option can be used with -XX:+|-HeapDumpOnOutOfMemoryError.

-XXsetGC (deprecated)

The `-XXsetGC` option is deprecated in Oracle JRockit R28. The option works in R28, but Oracle recommends that you use `-Xgc` instead. For more information, see -Xgc.

For more information about the format and usage of `-XXsetGC`, see the R27 release documentation at: `http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/index.html`.

-XX:+|-StrictFP

When you enable this option, JRockit JVM enables strict floating point arithmetics globally for all methods in all classes. The JVM also calculates with more precision, and with a greater range of values than the Java specification requires. When you use this option, the compiler generates code that adheres strictly to the Java specification to ensure identical results on all platforms. When you do not use this option, the JVM does not enforce floating point values.

This option is similar to the Java keyword `strictfp`; however, that keyword applies at the class level whereas `-XX:+|-StrictFP` applies globally. For more details about the `strictfp` keyword, see the Java Language Specification at:

`http://java.sun.com/docs/books/jls/`

Format

```
-XX:+StrictFP
```

Default Values

Disabled

Related Options

This option is equivalent to -XstrictFP.

-XX:StartFlightRecording

Specify this option at startup to start a Flight Recorder recording for an application that runs on JRockit JVM. This option is equivalent to the `start_flightrecording` diagnostic command that starts the Flight Recorder at run time. For more information, see *Oracle JRockit JDK Tools*.

Format

`-XX:StartFlightRecording=`*`parameter1=value1[,parameter2=value2]`*

Use `-XX:StartFlightRecording` with the parameters listed in Table 3–7.

Table 3–7 Parameters for -XX:StartFlightRecording

Parameter	Description	Default Value
`compress=true|false`	Specifies whether to compress the Flight Recorder recording log file (the .jfr file) on the disk using the gzip file compression utility. This parameter is valid only if the `filename` option is specified.	false
`defaultrecording=true|false`	Specifies whether the recording is continuous or it runs for a limited time.	false
`delay=`*`time`*	Specifies the time to elapse during run time before starting the recording.	0
`duration=`*`time`*	Specifies the time for the recording.	Unlimited
`filename=`*`name`*	Specifies the name of the Flight Recorder recording log file (the .jfr file).	None
`name=`*`identifier`*	Specifies an identifier for the Flight Recorder recording.	Recording *x* (for example, Recording 1, Recording 2)
`maxage=`*`time`*	Specifies the maximum age of disk data for default recording.	15 minutes
`maxsize=`*`size`*	Specifies the maximum disk space for a recording. This parameter is valid only for the size-bound recordings.	Unbound
`settings=`*`eventfile`*	Specifies the event setting file to use for the recording. You can use this parameter multiple times in one command.	`jre/lib/jfr/default.jfs`

Example

```
java -XX:+FlightRecorder -XX:FlightRecorderOptions=disk=true,maxchunksize=10M
-XX:StartFlightRecording=filename=test.jfr myApp
```

-XXtlaSize

The `-XXtlaSize` option sets the thread-local area size.

To increase performance JRockit JVM uses thread-local areas (TLA) for object allocation. This option can be used to tune the size of the thread-local areas, which can affect performance.

Format

`-XXtlaSize:`*`parameter=size`*`[k|K][m|M][g|G]`

Table 3–8 lists the *`parameter`*s.

Table 3–8 -XXtlaSize Parameters

Parameter	Description
`min=`*`size`*	Sets the minimum size of a TLA.
`preferred=`*`size`*	Sets the preferred size of a TLA. The system will try to get TLAs of this size if possible, but will accept TLAs down to the minimum size, if that's what's available. Occasionally, a TLA can get larger than the preferred size, too. The preferred size must not be lower than the minimum size.
`wasteLimit=`*`size`*	Sets the waste limit for TLAs. This is the maximum amount of free memory that a TLA is allowed to have when a thread requires a new TLA.

There are no upper or lower limits to `-XXtlaSize`.

Use this option with caution, as changing the thread-local area size can have severe impact on the performance.

Specify *`size`* in bytes, using the normal `k`,`M`,`G` suffixes.

> **Note:** The old style for setting TLA size (that is, `-XXtlasize=256k`) is supported but has been deprecated. If you use the old style, JRockit JVM interprets the option as if the `preferred` parameter was used; so `-XXtlasize=256k` would be interpreted as `-XXtlasize:preferred=256k`.

Example

```
-XXtlasize:min=2k,preferred=16k
```

Default Value

The default value for both the minimum size and the waste limit is 2 KB. The waste limit cannot be set to a larger value than the minimum size.

The default value for the preferred size depends on the heap size or the nursery size and the garbage collector selected at startup.Table 3–9 lists the default sizes for the different configurations.

Table 3–9 Default Preferred TLA Sizes

Garbage Collectors	Default Preferred TLA Size
`-Xgc:deterministic`, `-Xgc:singlecon`	16 KB
`-Xgc:pausetime`, `-Xgc:gencon`	16 KB - 256 KB depending on the nursery size
`-Xgc:throughput`, `-Xgc:genpar`	16 KB - 64 KB depending on the heap size
`-Xgc:parallel`	16 KB - 256 KB depending on the heap size
Nursery size set with `-Xns`	16 KB - 256 KB depending on the nursery size

Related Options

The following relation is true for the TLA size parameters:

`-XXtlaSize:wasteLimit` <= `-XXtlaSize:min` <= `-XXtlaSize:preferred`

If you set two or more of the options, then you must make sure that the values you use fulfil these criteria. By default, the waste limit is set to whichever is the lower value of the minimum TLA size and the preferred TLA size divided by 2.

-XX:TreeMapNodeSize

This option specifies the size of the entry array in each `java.util.TreeMap` node. This option affects the JVM performance and the default value is appropriate for most of the applications.

Format

`-XX:TreeMapNodeSize=`*`array_size`*

Example

```
java -XX:TreeMapNodeSize=128 myApp
```

Default Value

The default size of the key-value array in each TreeMap node is 64 entries.

-XX:+|-UseAdaptiveFatSpin

This option specifies whether threads should spin against a fat lock or not (and directly go to sleep state when failed to acquire the fat lock).

Format

```
-XX:+|-UseAdaptiveFatSpin
```

Default Value

By default, adaptive spinning against fat locks is disabled. This behavior cannot be changed during run time.

-XX:+|-UseAllocPrefetch

With this option a Thread Local Area is split into chunks and, when a new chunk is reached, the subsequent chunk is prefetched.

> **Note:** To fully benefit from this feature on Intel Xeon servers, it is recommended that you disable hardware prefetching in the BIOS of the computer.

Format

```
-XX:+|-UseAllocPrefetch
```

Example

```
java -Xgc:pausetime -XX:+UseAllocPrefetch myApp
```

Default

Enabled

Related Options

This option must be set if you want to use -XX:+|-RedoAllocPrefetch. This option is also used with the -XX:AllocChunkSize option.

-XX:+|-UseCallProfiling

This option enables the use of call profiling for code optimizations. Profiling records useful run-time statistics specific to the application and can increase performance because the JVM can then act on those statistics.

Format

```
-XX:+|-UseCallProfiling
```

Example

```
java -XX:+UseCallProfiling myApp
```

Default

Disabled

-XX:+|-UseCfsAdaptedYield

When enabled, this option uses a version of yield adapted for the Completely Fair Scheduler (CFS). This option should be used only on CFS and only if you are experiencing performance issues with the JVM.

> **Note:** This option is available only on Linux.

Format

```
-XX:+|-UseCfsAdaptedYield
```

Example

```
java -XX:+UseCfsAdaptedYield myApp
```

Default

Disabled

-XX:+|-UseClassGC

This option enables or disables garbage collection of classes.

When you disable garbage collection of classes, you can save some garbage collection time, which minimizes interruptions during the application run. Disabling garbage collection of classes can result in more memory being permanently occupied; so if the option is not used carefully, the JVM throws out-of-memory error.

> **Note:** Oracle recommends that you do not disable this option unless required because it can lead to memory leak when the application is running.

Format

```
-XX:-UseClassGC
```

Example

```
java -XX:-UseClassGC myApp
```

The class objects in the application specified by *myApp* are left untouched during garbage collection and are always considered active.

-XX:+|-UseCPoolGC

This option enables or disables garbage collection of constant pool strings.

When you disable garbage collection of constant pool strings, you may be able to reduce some garbage collection overhead associated with removal of strings from the runtime shared pool. Disabling garbage collection of constant pool strings can result in more memory being permanently occupied. Therefore, if the option is not used carefully, the JVM may throw an out-of-memory error.

> **Note:** This option is available since JRockit R28.3.2.
>
> Oracle strongly recommends that you do not disable this option unless requested by Oracle Support because it can lead to a memory leak when the application is running.

Even with constant pool garbage collection disabled, there are still cases where the JVM can determine that certain strings are no longer needed and are removed from the constant pool without the help of the garbage collection system.

Format

```
-XX:+|-UseCPoolGC
```

Example

```
java -XX:-UseCPoolGC myApp
```

The constant pool strings in the application specified by `myApp` are left untouched during garbage collection and may never be removed.

Default

This option is enabled by default.

Since

JRockit version R28.3.2.

-XX:+|-UseFastTime

This option specifies the use of hardware support for low-latency timestamps.

> **Note:** If you enable this option on a platform that does not support hardware timestamps, it might cause JRockit JVM to use invalid timestamps and it can result in fatal errors.

Format

```
-XX:+|-UseFastTime
```

Default

This option is enabled by default on latest AMD and Intel XEON platforms.

-XX:+|-UseFatSpin

The `-XX:-UseFatSpin` option disables the fat lock spin code in Java, allowing threads that block trying to acquire a fat lock go to sleep directly.

Objects in Java become a lock as soon as any thread enters a synchronized block on that object. All locks are held (that is, stay locked) until released by the locking thread. If the lock is not going to be released very fast, it can be inflated to a fat lock. Spinning occurs when a thread that wants a specific lock continuously checks that lock to see if it is still taken, spinning in a tight loop as it makes the check. Spinning against a fat lock is generally beneficial although, in some instances, it can be expensive and might affect performance. The `-XX:-UseFatSpin` allows you to turn off spinning against a fat lock and eliminate the potential performance hit.

Format

```
-XX:+|-UseFatSpin
```

Default

Enabled

-XX:+|-UseLargePagesFor[Heap|Code]

This option enables the use of large pages, if they are available, for the Java heap and code in the JVM. Large pages allow your application to more effectively use the translation look-aside buffer (TLB) in the processor.

> **Note:** This option duplicates the functionality of the -XlargePages option. Oracle recommends that you use the `-XX:+|-UseLargePagesFor[Heap|Code]` option instead of using the `-XlargePages` option.
>
> Use the extended option (`-XX:+ForceLargePagesForHeap`) to force the JVM to exit if enough large pages cannot be acquired and you have used the `-XX:+UseLargePagesForHeap` option.

Format

```
-XX:+|-UseLargePagesFor[Heap|Code]
```

Example

```
-XX:+UseLargePagesForHeap
```

Enables the use of large pages for the Java heap.

-XX:+|-UseLazyUnlocking

When `-XX:+|-UseLazyUnlocking` is enabled, locks are not released when a critical section is exited. Instead, once a lock is acquired, the next thread that tries to acquire such a lock will have to ensure that the lock is or can be released. It does this by determining if the initial thread still uses the lock. A truly shared lock is eventually converted to a normal lock and this improves the performance of locking operations on shared locks.

Format

```
-XX:+|-UseLazyUnlocking
```

Example

```
java -XX:-UseLazyUnlocking myApp
```

Disables lazy unlocking.

Default

Lazy unlocking is enabled by default on all platforms except when the `-Xdebug` option is used.

Exceptions

This option is intended for applications with many unshared locks. This option can introduce performance penalties with applications that have many short-lived but shared locks.

-XX:+|-UseLockProfiling

The `-XX:+|-UseLockProfiling` option enables or disables profiling of Java locks in JRockit Flight Recorder.

To get the Java lock profiling data, the lock events in the JRockit Flight Recorder must also be enabled. For more information, see *Oracle JRockit Flight Recorder Run Time Guide*.

Format

```
-XX:+|-UseLockProfiling
```

Example

```
java -XX:+UseLockProfiling myApp
```

Enables Java lock profiling.

Default

Disabled

-XX:+|-UseLowAddressForHeap

This option directs the JVM to use the low 4 GB address space for Java heap, if available.

Format

```
-X:+|-UseLowAddressForHeap
```

Example

```
-XX:+UseLowAddressForHeap
```

Enables use of the low address space for the Java heap.

Default

Enabled

-XX:+|-UseNewHashFunction

This option specifies whether a new, faster hash function is enabled for java.util.HashMap. This hash function can improve performance through improved hash spread, but changes the order in which elements are stored in the HashMap.

Format

```
-X:+|-UseNewHashFunction
```

Example

```
-XX:+UseNewHashFunction
```

Enables the new hash function.

Default

The new hash function is disabled by default in the JRockit JVM that bundles J2SE 5.0.

Related Options

`-XXaggressive` enables use of the new hash function unless it is explicitly disabled using `-XX:-UseNewHashFunction`.

-XX:+|-UseThreadPriorities

This option enables you to control the priority of Java threads using `java.lang.Thread.setPriority()` and related APIs. If this feature is disabled, using these APIs has no effect.

> **Caution:** This feature is experimental and not supported by Oracle at this time. Improper use can cause serious performance issues.

Format

```
-XX:+|-UseThreadPriorities
```

Example

```
-XX:+UseThreadPriorities
```

Enables use of the `java.lang.Thread.setPriority()` and related APIs.

Default

Disabled

Exceptions

Availability of this option is determined by the platform.

- **Windows**: Available.
- **Linux**: Available; you must have root privileges to use thread priorities on most Linux versions.
- **Solaris**: Not available.

4

Oracle JRockit JVM System Properties

This chapter describes the key system properties available with the Oracle JRockit JVM.

The `System` class maintains properties (key and value pairs) that define traits or attributes of the current working environment. When the Java application starts, the system properties are initialized with information about the run-time environment, including information about the current user, the current version of the Java run time, and so on.

This chapter describes the following properties:

- java.vendor
- java.vendor.url
- java.vendor.url.bug
- java.version
- java.runtime.version
- java.vm.name
- java.vm.vendor
- java.vendor.url
- java.vm.version
- java.vm.specification.version
- java.vm.specification.vendor
- java.vm.specification.name
- os.name
- os.arch
- os.version

System properties are part of the `java.lang.System` class as defined by the Java specifications. For more information, see the specification for the `java.lang.System` class at the following locations:

- Java SE 6.0

  `http://java.sun.com/javase/6/docs/api/java/lang/System.html`

- J2SE 5.0

  `http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html`

You can obtain the properties in a Java class by calling the `getProperty()` method as shown in Example 4–1.

Example 4–1 Obtaining System Properties

```
String os_name        =   System.getProperty("os.name");
String os_arch        =   System.getProperty("os.arch");
String java_home      =   System.getProperty("java.home");
String java_vm_name   =   System.getProperty("java.vm.name");
```

4.1 java.vendor

The `java.vendor` property identifies the JDK/JRE vendor.

4.2 java.vendor.url

The `java.vendor.url` property identifies the URL of the JDK/JRE vendor.

- HotSpot: `http://java.sun.com/`
- JRockit: `http://www.oracle.com/`

4.3 java.vendor.url.bug

The `java.vendor.url.bug` property identifies the bug report URL of the JDK or JRE vendor.

- HotSpot: `http://java.sun.com/cgi-bin/bugreport.cgi`
- JRockit: `http://download.oracle.com/docs/cd/E15289_01/go2troubleshooting.html`

4.4 java.version

The `java.version` property identifies the version of the JDK or JRE that you are running.

The value of the property is common to the HotSpot JRE and the JRockit JRE. It is displayed on the first line of the output of the `java -version` command in the following format:

major_version.minor_version.micro_version[*_update_version*][*-milestone*]

The following example shows the output of the `java -version` command. The `java.version` information is highlighted.

```
java version "1.6.0_17"
Java(TM) SE Runtime Environment (build 1.6.0_17-b04)
Oracle JRockit(R) (build R28.0.0-617-125986-1.6.0_17-20091215-2120-windows-x86_64,
compiled mode)
```

For more information about the `java.version` property, see the *J2SE SDK/JRE Version String Naming Convention* at:

http://java.sun.com/j2se/versioning_naming.html

4.5 java.runtime.version

The `java.runtime.version` property identifies the Java SE JDK/JRE version and build.

The value of the property is common to the HotSpot JRE and the JRockit JRE. It is displayed on the second line of the output of the `java -version` command in the following format:

major_version.minor_version.micro_version[*_update_version*][*-milestone*]*-build*

The following example shows the output of the `java -version` command. The `java.runtime.version` information is highlighted.

```
java version "1.6.0_17"
Java(TM) SE Runtime Environment (build 1.6.0_17-b04)
Oracle JRockit(R) (build R28.0.0-617-125986-1.6.0_17-20091215-2120-windows-x86_64,
compiled mode)
```

For more information about the `java.runtime.version` property, see the *J2SE SDK/JRE Version String Naming Convention* at:

http://java.sun.com/j2se/versioning_naming.html

4.6 java.vm.name

The `java.vm.name` property identifies the JVM implementation. The value depends on the JVM you use:

- HotSpot: `Java HotSpot(TM) Client VM` or `Java HotSpot(TM) Server VM`
- JRockit: `Oracle JRockit(R)`

The `java.vm.name` information is displayed on the third line of the output of the `java -version` command as highlighted in the following example:

```
java version "1.6.0_17"
Java(TM) SE Runtime Environment (build 1.6.0_17-b04)
Oracle JRockit(R) (build R28.0.0-617-125986-1.6.0_17-20091215-2120-windows-x86_64,
compiled mode)
```

4.7 java.vm.vendor

The `java.vm.vendor` property identifies the JVM implementation vendor.

4.8 java.vm.vendor.url

The `java.vm.vendor.url` property identifies the URL of the JVM implementation vendor.

- HotSpot: `http://java.sun.com/`
- JRockit: `http://www.oracle.com/`

4.9 java.vm.version

The `java.vm.version` property identifies the JVM implementation version. The version is displayed on the third line of the output of the `java -version` command.

The `java.vm.version` information is displayed on the third line of the output of the `java -version` command, as highlighted in the following example:

```
java version "1.6.0_17"
Java(TM) SE Runtime Environment (build 1.6.0_17-b08)
Oracle JRockit(R) (build R28.0.0-617-125986-1.6.0_17-20091215-2120-windows-x86_64,
compiled mode)
```

The `java.vm.version` property is the main means to distinguish between JRockit JVM releases. The following are examples of JVM implementation versions from a few different JRockit JVM releases:

- R24.5.0: `ari-49095-20050826-1856-win-ia32`
- 5.0 SP2: `dra-45238-20050523-2021-win-ia32`
- R26.4.0: `R26.4.0-63-63688-1.5.0_06-20060626-2259-win-ia32`
- R27.3.1: `R27.3.1-1-85830-1.6.0_01-20070716-1248-windows-ia32`
- R28.0.0: `R28.0.0-615-125739-1.6.0_17-20091210-2122-windows-ia32`

4.10 java.vm.specification.version

The `java.vm.specification.version` property identifies the version of the JVM specification on which the JRockit JVM instance is based (example: `1.0`).

4.11 java.vm.specification.vendor

The `java.vm.specification.vendor` property identifies the vendor of the JVM specification on which the JRockit JVM instance is based.

4.12 java.vm.specification.name

The `java.vm.specification.name` property identifies the name of the specification on which the JRockit JVM instance is based (example: `Java Virtual Machine Specifications`).

4.13 os.name

The `os.name` property identifies the operating system. For the JRockit JVM, this includes Windows, Linux, and Solaris versions (example: `Windows XP`).

For information about supported hardware and software configurations, see the *Oracle JRockit JDK Supported Configurations* at http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html.

4.14 os.arch

The `os.arch` property identifies the operating system architecture. For the JRockit JVM, this includes the following:

- x86 on IA32 systems
- amd64 on x86_64 systems
- sparcv9 on SPARC systems

For more information, see *Oracle JRockit JDK Supported Configurations* on the Oracle Technology Network (OTN).

4.15 os.version

The `os.version` property identifies the operating system version.

For more information about operating system support, see *Oracle JRockit JDK Supported Configurations* on the Oracle Technology Network (OTN).

5
Diagnostic Commands

This chapter is an alphabetically ordered reference for all the diagnostic commands that you can send to a running Oracle JRockit JVM process.

For more information about sending diagnostic commands to a JVM process, see "Running Diagnostic Commands" in *Oracle JRockit JDK Tools Guide*.

This chapter describes the following diagnostic commands and their attributes:

5.1 check_flightrecording

This command is associated with Oracle JRockit Flight Recorder. It prints information about the status of Flight Recording. This command accepts the following arguments:

Argument	Description
`name`	The recording identifier as a string. If you use this parameter, you do not have to specify a value for `recording`.
`recording`	The recording identifier as a number. If you use this parameter, you do not have to specify a value for `name`.
`verbose`	Specifies whether or not to enable verbose output. The default value is `false`.

5.2 command_line

This command prints the command-line options used to start the JRockit JVM.

5.3 dump_flightrecording

This command is associated with Oracle JRockit Flight Recorder. It dumps the running Flight Recorder recordings. This command accepts the following arguments:

Argument	Description
`name`	The recording identifier as a string. If you use this parameter, you do not have to specify a value for `recording`.
`recording`	The recording identifier as a number. If you use this parameter, you do not have to specify a value for `name`.
`copy_to_file`	Specify the name of the file to which you want the recording data to be copied.
`compress_copy`	Compress the file at the `copy_to_file` location using GZip. The default value is `false`.

5.4 exception_trace_filter

This command filters exception logging and JRockit Flight Recorder exception events based on the exception type specified. This command accepts the following argument:

Argument	Description
`pattern`	Specify the exception type as a string. For example: `jrcmd <pid> exception_trace_filter pattern=java.lang.Exception`

5.5 force_crash

Forces the Oracle JRockit JVM to execute code that will make the process fail and create a core file.

This command is not enabled by default because of the serious implications of using it. Use this command only for diagnosing problems that require getting the full state of the JVM. To be able to run the command, start JRockit with `-Djrockit.ctrlbreak.enableforce_crash=true`.

5.6 fork_and_abort

This command causes JRockit to fork a child process that will immediately abort. If the environment has been properly configured to generate a core file when a process crashes, a core file will be created. This command is intended for generating a core file when tools such as `gdb` or `gcore` cannot be used. Since the core file generated using this command will not include context information for each thread, generating a core file using `gdb` or `gcore` is always preferred when possible.

Note: This command is available since R28.3.9. This command is not available on Windows environments.

5.7 heap_diagnostics

This command causes a heap diagnostic report to be printed (also, see `-XX:HeapDiagnosticsOnOutOfMemoryError` in the *Oracle JRockit Command Line Reference*). This command sends output to the Ctrl-Break Handler output stream and not to the path specified by the `-XX:HeapDiagnosticsPath` option.

5.8 help

This command displays additional information about a specific command or all commands. This command accepts the following arguments:

Argument	Description
`all`	Shows help for all commands. The default is `false`; this argument is optional.
`command`	Shows help for the specified command. This argument is optional. If you omit it, the command will show a list of available commands.

5.9 hprofdump

This command generates an HPROF format dump of the Java heap.

Format:

```
hprofdump [filename=<file>] [segment_threshold] [segment_size]
```

This command accepts the following arguments:

Argument	Description
`filename`	Sets the name of the file to which the dump should be written. If you do not specify a filename, the command will use the value specified with the `-XX:HeapDumpPath` command-line option.
`segment_threshold`	Sets the amount of heap usage above which the JVM should generate a segmented heap dump (JAVA PROFILE 1.0.2 format). If you do not specify a segment threshold, the command will use the default value of `-XX:SegmentedHeapDumpThreshold`, as described in the *Oracle JRockit Command-Line Reference*.
`segment_size`	Sets the approximate segment size to which a generated segmented heap dump should be limited. If you do not specify `segment_size`, this command uses the default value set for `-XX:HeapDumpSegmentSize`, as described in the *Oracle JRockit Command-Line Reference*.

5.10 kill_management_server

This command stops the management server by shutting down the listening socket. The `managementserver.jar` has to be in the boot classpath for this command to work.

5.11 list_vmflags

This command lists the flag options in the Oracle JRockit JVM and their current values:

- `flag`: Lists only this flag.
- `describe`: Shows the description for flags.
- `alias`: Prints the flag alias if one is available.
- `setonly`: Lists only flags that are explicitly or implicitly set.

5.12 lockprofile_print

This command prints the current values of the lock profile counters. You can enable lock profiling by using the `-XX+UseLockProfiling` option.

5.13 lockprofile_reset

This command resets the current values of the lock profile counters. You can enable lock profiling by using the `-XX+UseLockProfiling` option.

5.14 memleakserver

This command starts or shuts down the memory leak server. This command accepts the following arguments:

Argument	Description
`port`	Identifies the port to which to bind. The default value is `7095`.

Argument	Description
version	Identifies the required protocol version. The default value is 3.
action	Starts or stops the server. The default is to toggle state.
force	Forces an action. The default value is false.

5.15 print_class_summary

This command prints all loaded classes.

5.16 print_exceptions

This command enables or disables the printing of exceptions (see -Xverbose in the *Oracle JRockit Command-Line Reference*).

Format:

```
print_exceptions stacktraces= all|true|false][exceptions= all|true|false]
```

Note: To turn exception printing off completely, set exceptions=false even if it was turned on by stacktraces=true.

5.17 print_memusage

This command prints all memory used by the JRockit JVM process, per OS data, plus any memory usage perceived by each subsystem.

To get the most detailed information out of this command, set USE_OS_MALLOC to 0 in your environment variables. Also, if you enable the variable TRACE_ALLOC_SITES (that is, set it to 1) to enable allocation site tracking by default, you might add some overhead, but you will also receive information about the location of every allocation.

Format:

```
print_memusage [baseline] [test] [level=<1 | 2 | 3>] [reset] [displayMap]
```

Use print_memusage with any of the following arguments:

Argument	Description
baseline	Stores a snapshot of the memory usage.
test	Prints the difference between this stored baseline and the previously stored baseline.
level	Specifies the level of detail to be printed. Level 1 gives memory usage for the source files; level 2 gives more detail, providing memory usage for the function name; level 3 provides the most detail, providing memory usage statistics at the source code level.
reset	Prints the memory usage statistics and removes the stored baseline. Further print_memusage results will not show any difference until a new baseline is set.
displayMap	Sets the memory usage baseline on the virtual memory.

5.18 print_object_summary

See the Memory Leak Detector online help that ships with Oracle JRockit Mission Control.

5.19 print_threads

Prints a normal thread dump.

Format:

```
print_threads [nativestack= true] [jvmmonitors=true]
```

This command accepts these arguments:

Argument	Description
`compact`	Prints all threads with the same stacktrace together (will not print nativestack or monitors). The default value is `false`.
`concurrentlocks`	Prints `java.util.concurrent` locks. The default value is `false`.
`internal`	Prints Oracle JRockit internal threads. The default value is `true`.
`javastack`	Prints Java stack frames. The default value is `false`.
`jvmmonitors: true`	Prints the JRockit JVM's internal native locks (those that are registered): status and wait queue, and, with `XX:+UseNativeLockProfiling`, their profile statistics (acquired/contended/tryfailed).
`monitors`	Prints lock information. The default value is `true`.
`nativestack: true`	Includes native frames in the stack trace. The default value is `false`.

5.20 print_utf8pool

This command prints all UTF8 strings.

5.21 print_vm_state

This command prints information about the internal state of the JVM. This information can be used for troubleshooting by Oracle support and is the same information that is included in Oracle JRockit failure reports.

5.22 runsystemgc

This command calls the `java.lang.System.gc()` method and runs the garbage collector.

Format:

```
runsystemgc [full=false] [fullcompact=true]
```

The command accepts these arguments:

Argument	Description
full	Does a full garbage collection. It inherits the system default value (see -XXfullSystemGC).
fullcompact	Forces full compaction for each full garbage collection event. The default value is true.

5.23 set_filename

This command sets the file that all commands following this command will use for printing. You can have several set_filename commands in a file. By default, for SIGQUIT or ctrl+break invocations, the commands print to the stderr output stream of the JVM. For jrcmd invocations, the print goes to the stdout output stream of the jrcmd process by default. The append argument defaults to overwrite.

Format:

```
set_filename [filename=<file>] [append=true]
```

Argument	Description
filename	(Optional) Specifies the name of the file for printing. If not specified, JVM will reset to the default behavior.
append	(Optional) Specifies whether you want to append to the file or overwrite it.

5.24 start_flightrecording

Starts a flight recording. This command accepts the following arguments:

Argument	Description
compress: true	Directs the Flight Recorder to gzip the .jfr file on the disk.The default value is false.
defaultrecording: true	Enables a default recording. The default value is false.
delay	Sets the amount of time to elapse during run time before starting the recording.
duration	Sets the amount of time for the recording to run.
filename	Sets the name of the flight recording log file. This file will have the extension .jfr.
maxage	For time-bound recordings, sets the maximum amount of time a recording can last before that recording is flushed from the thread buffer to the global buffer.
maxsize	For size-bound recordings, sets the maximum size of a recording before that recording is flushed from the thread buffer to the global buffer.
name	Sets a recording identifier.
settings	Identifies the event settings file to use for the recording.

5.25 start_management_server

This command starts the management server by starting the listening socket that, in turn, starts servers whenever a connection is established. This command accepts these arguments:

Argument	Description
`none`	Enables the JMX local monitoring through a JMX connector published on a private interface used by local JMX clients that use the Attach API.
`autodiscovery`	Enables or disables autodiscovery for the remote JMX connector, which enables Oracle JRockit Mission Control to automatically discover running JRockit JVM instances through the multicast-based JRockit Discovery Protocol (JDP).
`autodiscovery_name`	Enables you to specify the path and name of the cluster and node from where Oracle JRockit Mission Control discover information about various JRockit JVM instances running in a network.
`authenticate`	Enables or disables authentication. When this property is set to false, JMX does not use passwords or access files. All users are allowed all access.
`class`	Loads the class and causes its empty constructor to be called early in the JVM startup. From the constructor, a new thread is then started, from which your management client is run. Further arguments cannot be given to `-Xmanagement` after the class argument.
`config_file`	Specifies the location of the file from which additional management configuration properties are loaded.
`interface`	Specifies the local address (on the management server side) on which to listen for connections. This applies to machines with several addresses (network cards).
`local`	Enables or disables the local JMX connector.
`port`	Identifies the port that the management server opens for remote access.
`registry_ssl`	Binds the RMI connector stub to an RMI registry protected by SSL.
`remote`	Enables or disables the remote JMX connector.
`rmiserver_port`	Binds the RMI server to the specified port.
`ssl`	Enables or disables SSL encryption.

This command serves the same purpose as the `-Xmanagement` command-line option. For more information, see the entry for `-Xmanagement` in the *Oracle JRockit Command Line Reference*.

5.26 stop_flightrecording

This command stops an in-process flight recording. This command accepts the following arguments:

Format:

```
stop_flightrecording [name=<string>][recording=<s8>] [discard=<true | false>]
[copy_to_file=<string>] [compress_copy=<true | false>]
```

Argument	Description
name	Recording identifier as a string.
recording	Recording identifier as a number.
discard	Discards the recording; the default is false.
copy_to_file	Transfers the recording to a .jfr file.
compress_copy	Gzip the .jfr file on disk.

5.27 stop_management_server

This command stops the management server.

5.28 timestamp

This command prints a timestamp, including the uptime of the queried JVM..

5.29 verbosity

This command changes the verbosity level usually specified with -Xverbose.

Format:

```
verbosity [args=<components>] [filename=<file>]
```

5.30 version

This command prints the JRockit JVM version.

A

Changes in Command-Line Options

This appendix lists the commands that are introduced or deprecated in Oracle JRockit R28.x releases.

This appendix includes the following sections:

A.1 Command-Line Options Introduced in Oracle JRockit R28.0

The following are the command-line options introduced in Oracle JRockit R28.0:

- -XX:+|-CheckStacks
- -XXcompaction
- -XX:+|-DisableAttachMechanism
- -XX:ExitOnOutOfMemoryErrorExitCode
- -XX:+|-FailOverToOldVerifier
- -XX:+|-FlightRecorder
- -XX:FlightRecorderOptions
- -XX:+|-FlightRecordingDumpOnUnhandledException
- -XX:FlightRecordingDumpPath
- -XX:GCTimeRatio
- -XX:GCTimePercentage
- -XX:+|-HeapDumpOnCtrlBreak
- -XX:+|-HeapDumpOnOutOfMemoryError
- -XX:HeapDumpPath
- -XX:HeapDumpSegmentSize
- -XX:MaxCodeMemory
- -XX:MaxLargePageSize
- -XX:MaximumNurseryPercentage

- -XX:SegmentedHeapDumpThreshold
- -XX:StartFlightRecording
- -XX:+|-JavaDebug
- -XX:+|-UseCfsAdaptedYield
- -XX:+|-UseClassGC
- -XX:+|-UseLockProfiling
- -XX:+|-UseLowAddressForHeap

A.2 Command-Line Options and Parameters Introduced in Oracle JRockit R28.1

The following are the command-line options and parameters introduced in Oracle JRockit R28.1:

> **Note:** These options and parameters do not work in R28.0.

Command-Line Options Introduced in Oracle JRockit R28.1

- -XX:+|-CrashOnOutOfMemoryError
- -XX:MaxRecvBufferSize

Command-Line-Option Parameters Introduced in Oracle JRockit R28.1

The following new parameters are available in R28.1 for the -XX:FlightRecorderOptions option:

- `dumponexit`
- `dumponexitpath`

A.3 Command-Line Options Deprecated and Removed in Oracle JRockit R28.0

The deprecated options continue to work in Oracle JRockit R28, but Oracle does not support any of these options. Options that are removed do not work in JRockit R28. Oracle recommends that you use the alternate options provided for each deprecated or removed option as listed below:

- -XclearType (removed)
- -XgcPause (removed, use -Xverbose:gcpause instead)
- -XgcPrio (use -Xgc instead)
- -XgcReport (removed, use -Xverbose:gcreport instead)
- -XnoClassGC (use -XX:+|-UseClassGC instead)
- -XXcompactRatio (use -XXcompaction:percentage instead)
- -XXcompactSetLimit (use -XXcompaction:maxReferences instead)
- -XXcompactSetLimitPerObject (use -XXcompaction:maxReferencesPerObject instead)
- -XXdisableGCHeuristics

- -XXexternalCompactRatio (use -XXcompaction:externalPercentage instead)
- -XXfullCompaction (use -XXcompaction:full instead)
- -XXheapParts (use -XXcompaction:heapParts instead)
- -XXhpm (removed)
- -XXinitialPointerVectorSize (removed)
- -XXinternalCompactRatio (use -XXcompaction:internalPercentage instead)
- -XXjra (removed)
- -XXlargeObjectLimit
- -XXmaxPooledPointerVectorSize (removed)
- -XXminBlockSize (use -XXtlaSize:min instead)
- -XXmme (removed)
- -XXnoCompaction (use -XXcompaction:enable=false instead)
- -XXnoJITInline (removed)
- -XXpointerMatrixLinearSeekDistance (removed)
- -XXprintSystemGC (removed, use -Xverbose:systemgc instead)
- -XXsetGC (use -Xgc instead)
- -XXstaticCompaction (removed)
- -XXthroughputCompaction (removed)
- -XXtsf (removed)
- -XXusePointerMatrix (removed, use -XXcompaction:abortable instead)

For more information about the deprecated and removed options in Oracle JRockit R28.0, see the R27 command-line reference guide at:

`http://download.oracle.com/docs/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/index.html`.

A.4 Command-Line Options Converted to HotSpot Format in Oracle JRockit R28.0

In Oracle JRockit R28.0, the format of several command-line options has been changed to the HotSpot format as listed in Table A–1.

Table A–1 Command-Line Options Converted to the HotSpot Format

Old Option	New Format
-Xcheck:jni	-XX:+|-CheckJNICalls
-XXallocClearChunkSize	-XX:AllocChunkSize
-XXallocPrefetch	-XX:+|-UseAllocPrefetch
-XXallocRedoPrefetch	-XX:+|-RedoAllocPrefetch
-XXcallProfiling	-XX:+|-UseCallProfiling
-XXdisableFatSpin	-XX:+|-UseFatSpin
-XXexitOnOutOfMemory	-XX:+|-ExitOnOutOfMemoryError

Table A–1 (Cont.) Command-Line Options Converted to the HotSpot Format

Old Option	New Format
-XXoptThreads	-XX:OptThreads
-XXlargePages	-XX:+|-UseLargePagesFor[Heap|Code]
-XXlazyUnlocking	-XX:+|-UseLazyUnlocking
-XXoptThreads	-XX:OptThreads
-XXMaxDirectMemorySize	-XX:MaxDirectMemorySize
-XXuseAdaptiveFatSpin	-XX:+|-UseAdaptiveFatSpin

B

JMX Agent-Related –D Options

Table B–1 lists the -D options that enable you to access remote JRockit JVM instances in Oracle JRockit Mission Control using a Java Management Extensions (JMX) connector.

Note: Unless explicitly stated, the options listed in Table B–1 are also available in the `com.sun.management` namespace.

Oracle recommends that you use the equivalent -Xmanagement command-line options for -D options (if available).

Table B–1 JMX Agent-Related -D Options in R28

-D Option	Description	Default Value
-Dcom.oracle.management.jmxremote	See -Xmanagement.	`true` When you set this option to `false`, no local connector is started even if you specify a port number for jmxremote.port.
-Dcom.oracle.management.jmxremote.port	See -Xmanagement:port	7091
-Dcom.oracle.management.jmxremote.interface	See -Xmanagement:interface	null
-Dcom.oracle.management.jmxremote.rmiserver.port	See -Xmanagement:rmiserver_port	None
-Dcom.oracle.management.jmxremote.registry.ssl	See -Xmanagement:registry_ssl	false
-Dcom.oracle.management.jmxremote.ssl	See -Xmanagement:ssl	true
-Dcom.oracle.management.jmxremote.ssl.enabled.protocols	A comma-delimited list of SSL or TLS Protocol versions to be enabled. This option is used with com.oracle.management.jmxremote.ssl.	Default SSL or TLS Protocol

Table B–1 (Cont.) JMX Agent-Related -D Options in R28

-D Option	Description	Default Value
-Dcom.oracle.management.jmxremote.ssl.enabled.cipher.suites	A comma-delimited list of SSL or TLS cipher suites to be enabled. This option is used with com.oracle.management.jmxremote.ssl.	Default SSL or TLS cipher suites
-Dcom.oracle.management.jmxremote.ssl.need.client.auth	When this property is set to `true` and the property com.oracle.management.jmxremote.ssl is also `true`, the client authentication is performed.	false
-Dcom.oracle.management.jmxremote.authenticate	See -Xmanagement:authenticate	`true`
-Dcom.oracle.management.jmxremote.password.file	Specifies the location for the password file. If the com.oracle.management.jmxremote.authenticate property is false then this property and the password and access files are ignored. Otherwise, the password file must exist and it should be in the valid format. If the password file is empty or does not exist, you cannot access the remote JVM.	*JRE_HOME*`/lib/management/jmxremote.password`
-Dcom.oracle.management.jmxremote.access.file	Specifies the location for the access file. If the com.oracle.management.jmxremote.authenticate property is `false` then this property, the password file, and access file are ignored. Otherwise, the access file must exist and it should be in the valid format. If the access file is empty or does not exist, you cannot access the remote JVM.	*JRE_HOME*`/lib/management/jmxremote.access`

Table B–1 (Cont.) JMX Agent-Related -D Options in R28

-D Option	Description	Default Value
-Dcom.oracle.management.jmxremote.login.config	Specifies the name of a Java Authentication and Authorization Service (JAAS) login configuration entry to use when the JMX agent authenticates users. When using this property to override the default login configuration, the named configuration entry must be in a file that is loaded by JAAS. In addition, the login modules specified in the configuration should use the name and password callbacks to acquire the user credentials. For more information, see the API documentation for javax.security.auth.callback.NameCallback and javax.security.auth.callback.PasswordCallback.	Default login configuration is a file-based password authentication.
-Dcom.oracle.management.config.file	See -Xmanagement:config.file	*JRE_HOME*/lib/management/management.properties
-Dcom.oracle.management.snmp.port	Enables the SNMP agent on a specified port.	None
-Dcom.oracle.management.snmp.trap	Specifies the remote port to which the SNMP agent sends traps.	162
-Dcom.oracle.management.snmp.acl	Enables Access Control List for the SNMP agent.	true
-Dcom.oracle.management.snmp.acl.file	Specifies the location of valid ACL file. Once the SNMP agent is started, you cannot modify the ACL file.	*JRE_HOME*/lib/management/snmp.acl
-Dcom.oracle.management.snmp.interface	Specifies the local host InetAddress. The SNMP agent will then bind to the specified InetAddress. This is used when you have multihome hosts and you want the port to listen only to a specific subnet. This property is optional.	None
-Dcom.oracle.management.autodiscovery	See -Xmanagement:autodiscovery	false
-Dcom.oracle.management.autodiscovery.period (not available in the com.sun.management namespace)	Specifies the time interval between the broadcast messages during autodicovery in milliseconds.	5000 milliseconds

Table B–1 (Cont.) JMX Agent-Related -D Options in R28

-D Option	Description	Default Value
-Dcom.oracle.management.autodiscovery.ttl (not available in the com.sun.management namespace)	Time-to-live for autodiscovery packets.	1
-Dcom.oracle.management.autodiscovery.address (not available in the com.sun.management namespace)	Multicast address to send autodiscovery packets.	232.192.1.212
-Dcom.oracle.management.autodiscovery.port (not available in the com.sun.management namespace)	Multicast port to send autodiscovery packets	7095
-Dcom.oracle.management.autodiscovery.name (not available in the com.sun.management namespace)	Broadcast name of the JVM	
-Dcom.oracle.management.autodiscovery.property.prefix (not available in the com.sun.management namespace)	Copies all system properties starting with a specified prefix to the autodiscovery packets and make them available to remote agent.	