Oracle® JRockit

Flight Recorder Run Time Guide

Release R28

E15070-10

July 2016

This document contains background on the Oracle JRockit Flight Recorder Run-time implementation and instructions for using this tool. This document does not address the JRockit Flight Recorder Graphical User Interface.

ORACLE®

Oracle JRockit Flight Recorder Run Time Guide, Release R28

E15070-10

Primary Author: Edwin Spear, Savija Vijayaraghavan

Contents

Preface

This document contains background on the Oracle JRockit Flight Recorder Run Time implementation and instructions for using this tool.

About the Document

This document contains the following chapters:

- Chapter 1, "Introduction", which contains a description of JRockit Flight Recorder and its capabilities.
- Chapter 2, "Quick Start Procedures", which contains simple procedures for using JRockit Mission Control to create your first flight recording.
- Chapter 3, "Starting the Flight Recorder", which contains procedures for starting and controlling JRockit Flight Recorder.
- Chapter 4, "Controlling Recording Data by Using Templates", which discusses what you can control with JRockit Flight Recorder and how to use templates to do so.
- Chapter 5, "Troubleshooting and Security", which contains information about how to troubleshoot problems JRockit Flight Recorder and who can control flight recorder information.
- Appendix A, "Creating Your Own Server-side Templates", which shows you how to create a server-side flight recording template.
- Appendix B, "Command Reference", which contains a list and brief description of the command-line options and diagnostic commands you can use with JRockit Flight Recorder.
- Appendix C, "Events", which contains a list of all events you can capture with JRockit Flight Recorder.

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 that have purchased support 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 Introduction

This chapter serves as an introduction to the Oracle JRockit Flight Recorder. It contains these sections:

1.1 Overview

Have you ever wondered what really happens in a running Java program? Have you ever wanted to "go back in time" and analyze what happened right before a problem occurred in your system? Do you want an extremely detailed level of profiling without impacting performance? JRockit Flight Recorder has been engineered to meet all of these goals.

JRockit Flight Recorder does all this by being tightly integrated into the core of the JVM itself and by being very conscious of its performance overhead. JRockit Flight Recorder provides a wealth of information on the inner workings of the JVM as well as on the Java program running in the JVM. You can use this information for profiling and for root cause analysis of problems. Furthermore, JRockit Flight Recorder can be enabled at all times, without causing performance overhead—even in heavily loaded, live production environments.

While capturing details about the JVM, the JRockit Flight Recorder has also been tightly integrated into Oracle's Fusion Middleware family of products and provides a full stack view of the system. Everything from Java servlets and database execution at a high level to fine-grained information on thread synchronization and garbage collections is available at your fingertips.

JRockit Flight Recorder is comprised of a recording engine, which sits inside the JVM, and the JRockit Mission Control client. The engine produces a recording file which can later be analyzed through the client. This document primarily deals with configuration of the first part, the JVM, and only briefly mentions the JRockit Mission Control GUI (see Chapter 2, "Quick Start Procedures").

1.2 Flight Recorder Uses

The JRockit Flight Recorder has three primary uses:

- Profiling
- "Black Box" Problem Analysis
- Support and Debugging

1.2.1 Profiling

Because JRockit Flight Recorder continuously saves large amounts of data about the running system, it can operate as a profiler. Profiling information includes thread samples, which show where the program spends its time, as well as lock profiles and garbage collection details

1.2.2 "Black Box" Problem Analysis

Like its aeronautic namesake, JRockit Flight Recorder can also operate as a "black box," continuously saving information to a circular buffer. This information then can be accessed when an anomaly is detected. This information can be essential to quickly finding performance issues in a deployed system.

1.2.3 Support and Debugging

JRockit Flight Recorder provides information that can provide Oracle support personnel with important clues for diagnosing issues with the software.

1.3 Understanding Events

The basic principle underlying JRockit Flight Recorder is that every piece of data it captures is an *event*. An event is something that happens in the running application at a specific point in time. Events have a name, a time stamp, and an optional payload. The payload depends on the type of the event; for example:

- The payload for an old collection event, generated by the garbage collector, would be the heap size before and after the collection.
- The payload of an event to signal that a thread has been blocked by a lock would be the thread ID of the lock holder.

In addition to name and time stamp, most events also have information on the thread in which it occurred, the stack trace at the time the event was captured, and the duration of the event. All time stamps in JRockit Flight Recorder have nanosecond precision.

Requestable events are events that the recording engine can poll with a certain frequency. CPU Load Sample is an example for *requestable event*. You can configure the recording engine to poll for this event once every second.

By using the information available in an event, the JRockit Flight Recorder UI can reconstruct in detail what happened during program execution.

For a complete list of events, see Appendix C, "Events".

1.4 Performance Overhead

JRockit Flight Recorder is designed to keep its overhead as low as possible. When default settings are used, both internal testing and customer environments indicate that performance impact is less than one percent (< 1%). This percentage varies on the application and is sometimes significantly low.

JRockit Flight Recorder monitors the running system at an extremely high level of detail. This produces an enormous amount of data that JRockit Flight Recorder can handle. JRockit Flight Recorder filters data as early as possible to maintain low overhead. This is done in two different ways:

- By limiting the type of events that are actually captured. You can control this information when you start the recording (for more information, see Chapter 3, "Starting the Flight Recorder").
- By recording only those events with durations exceeding a certain threshold. In most cases, very short events are not of any interest and can be discarded. This effectively limits the amount of data JRockit Flight Recorder must handle. If you want to capture more data, you can change the threshold .

1.5 Memory and Disk Buffers

JRockit Flight Recorder does not write events to disk immediately, as they occur. Instead, it stores data in a hierarchy of in-memory buffers and then moves the data to the disk when the buffers are full. Initially, JRockit Flight Recorder run time puts the event data in thread-local buffers, eliminating the need to synchronize between threads for every event, which greatly improves throughput. Once a thread-local buffer has been filled, the data is transferred to a global buffer. When this happens, synchronization is necessary between threads but, because different thread-local buffers fill up at different rates, lock contention is rare. Eventually, the global buffer also runs out of space and the contents in the buffer are written to the disk. Writing to the disk is expensive and you must ensure that it happens as seldom as possible. Writing to disk files produces files in a proprietary, binary format that is extremely compact but also efficient for the applications to read and write.

You can configure JRockit Flight Recorder so that it does not write any data to disk. In this mode, the global buffer acts as a circular buffer and the oldest data is dropped when the buffer is full. This very low-overhead operating mode still collects all the vital data necessary for root-cause problem analysis. Because the most recent data is always available in the global buffer, it can be written to disk on demand whenever operations or surveillance systems detect a problem.

1.6 Garbage Collections and the Flight Recorder

The Flight Recorder will generate special garbage collections whenever it makes a recording, which will appear as `JFR Heap Block Statistics` or `JFR Class Statistics` on the Garbage Collection Data section of the verbose output (that is, run with `-Xverbose:gc=debug`); for example:

```
[DEBUG][memory ] [OC#1] GC reason: Artificial, description: JFR Heap Block
Statistics.
```

2 Quick Start Procedures

This chapter provides instructions for quickly starting up JRockit Flight Recorder and creating your first recording.

This chapter contains the following sections:

- Section 2.1, "Using JRockit Mission Control Client"
- Section 2.2, "Step 1: Start the Flight Recorder"
- Section 2.3, "Step 2: Set Recording Parameters"
- Section 2.4, "Step 3: Start the Recording"
- Section 2.5, "Stopping a Recording"
- Section 2.6, "Additional Information on the Flight Recorder GUI"

2.1 Using JRockit Mission Control Client

The simplest way to control JRockit Flight Recorder is by using the Oracle JRockit Mission Control Client. JRockit Mission Control Client is a tools suite that you can use to monitor, manage, profile, and eliminate memory leaks in your Java application, without introducing the performance overhead normally associated with these types of tools. For more information on JRockit Mission Control, see the *Introduction to JRockit Mission Control*, available on the Oracle Technology Network.

2.2 Step 1: Start the Flight Recorder

Start JRockit Mission Control Client from the command line by entering:

```
JROCKIT_HOME/bin/jrmc (or JROCKIT_HOME\bin\jrmc.exe
```

Note: You can also launch Mission Control from the Start menu by selecting **Programs** then **Oracle JRockit JDK <*version and JDK information*>** then **Oracle JRockit Mission Control <*version*>**.

When JRockit Mission Control Client launches, a list of all the JVMs running on the system appears in the JVM Browser.

1. Choose the JVM for which you want to create a recording and right-click to open a context menu.
2. From the context menu, select **Start Flight Recording...**

The Start Flight Recording dialog box appears.

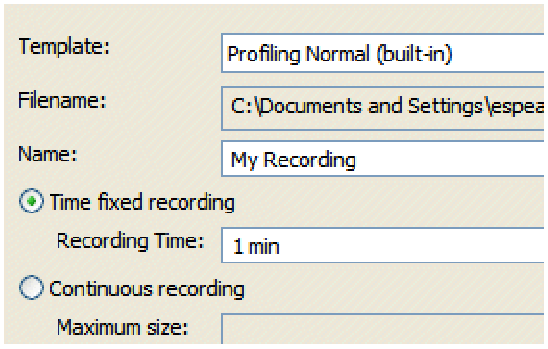
2.3 Step 2: Set Recording Parameters

Use the Start Flight Recording dialog box to specify key recording parameters. Complete this dialog box by doing the following:

- Choose the recording template (for more information on templates, see Section 4.2, "Mission Control Templates").
- Set recording time (the duration of the recording) by selecting Time fixed recording and entering a duration in Recording Time.

See Figure 2–1.

Figure 2–1 Flight Recording Parameter Fields

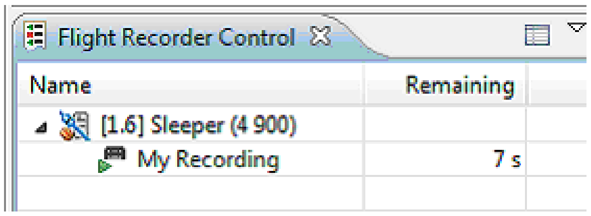


Template:	Profiling Normal (built-in)
Filename:	C:\Documents and Settings\espea
Name:	My Recording
(●) Time fixed recording	
Recording Time:	1 min
( ) Continuous recording	
Maximum size:	

2.4 Step 3: Start the Recording

Click **OK** to start the recording. You can follow the progress of the recording in the Flight Recorded Control view at the bottom of the screen (Figure 2–2). The Remaining column indicates the amount of time left before the recording terminates.

Figure 2–2 Flight Recorded Control view



Flight Recorder Control

Name	Remaining
[1.6] Sleeper (4 900)	
My Recording	7 s

Once the recording has finished it will open automatically in JRockit Mission Control Client and you can analyze the results.

2.5 Stopping a Recording

Usually, a recording will continue for the length of time specified in the Start Flight Recording dialog box. If you want to terminate before the specified time elapses, do the following:

1. On the Flight Recorder Control view, right-click the recording you want to stop to open a context menu.

2. Select **Stop**.

   The recording will stop and open automatically in JRockit Mission Control, showing all recording data up to the termination point.

2.6 Additional Information on the Flight Recorder GUI

For additional information on using the Flight Recorder GUI, refer to the online help that is installed with the product.

3 Starting the Flight Recorder

This chapter describes how to start the JRockit Flight Recorder for both default and explicit recordings from a command line.

This chapter contains these sections:

3.1 Note on Running Multiple Recordings

JRockit Flight Recorder allows many recordings to run concurrently. You can configure each recording by using different settings; in particular, you can configure different recordings to capture different sets of events. However, in order to make the internal logic of the Flight Recorder as streamlined as possible, the resulting recording always contains the union of all events for all recordings active at that time. This means that if more than one recording is running, you might end up with more information in the recording than you wanted. This can be a little bit confusing but has no other negative implications.

3.2 Running the Default Recording

The *default recording* is the recording that starts automatically, without you setting any parameters. This section shows you how to start the default recording and how to configure disk storage for that recording. It includes the following information:

- Starting the Recording
- Configuring Disk Storage
- Default Start-Up Example

3.2.1 Starting the Recording

As mentioned in Section 3.2, "Running the Default Recording", Oracle JRockit can have a default recording running in the background at all times. In the current release of Oracle JRockit, this is turned off by default but you can easily enable it by using the start-up command `-XX:FlightRecorderOptions`. In fact, because the performance impact is so low and the value of the data so high, Oracle recommends that you enable a default recording even in production environments. To enable an in-memory black box recording of the JVM, use the following command-line option:

```
-XX:FlightRecorderOptions=defaultrecording=true
```

3.2.2 Configuring Disk Storage

By adding certain parameters to the `-XX:FlightRecorderOptions` start-up command, you can configure the location of the disk repository as well as the amount of data stored in the repository.

3.2.2.1 Setting the Repository Location

By default, JRockit Flight Recorder stores the temporary recording files in the path specified by the `java.io.tmpdir` system property, but you can change this by adding this parameter:

```
repository=<path>
```

`<path>` is the preferred repository location; for example, `/var/log/jfr`.

3.2.2.2 Setting the Amount of Data Stored

You can configure the amount of data stored by the default recording in the repository as an absolute amount of bytes by using the `maxsize` option. For example, you can set this option to use 100 MB of disk space for storing the recording data.

You can also mention that you always want to store certain minutes or hours worth of data by using the `maxage` option. In this case, JRockit Flight Recorder only discards data when it is older than the specified age. This is a very powerful way of ensuring that data always exists for at least some time leading up to a problem.

If you do not specify any values for `maxsize` and `maxage`, the default values are taken. For more information about these options, see the description of `-XX:FlightRecorderOptions` in *Oracle JRockit Command Line Reference*.

- To set the maximum size, use the `maxsize` option as follows:

  ```
  maxsize=<size>
  ```

  `size` can be specified with `k` (kilobytes), `m` (megabytes) and `g` (gigabytes) suffixes; for example, `5m`.

- To set the maximum age, use the `maxage` option as follows:

  ```
  maxage=<age>
  ```

  `age` can be specified by `s` (seconds), `m` (minutes), `h` (hours), or `d` (days); for example, `10s`. The default value is 15 minutes.

3.2.3 Default Start-Up Example

To enable a default recording that stores temporary data in the `/var/log/jfr` directory and that covers at least the last five minutes of an application run, use the following command-line option:

```
-XX:FlightRecorderOptions=defaultrecording=true,disk=true,repository=/var/log/jfr,
maxage=5m
```

By default, the recording will be saved in the current working directory.

3.3 Starting an Explicit Recording

In addition to the default recording, you can create an *explicit recording*; that is, one you start explicitly and let run for some predetermined length of time or until you manually stop it. This section describes how to do this. It includes the following information:

- Controlling the Flight Recorder for Explicit Recordings
- Starting a Recording
- Checking Recording Status
- Stopping a Recording

3.3.1 Controlling the Flight Recorder for Explicit Recordings

Regardless of the method you use to start a recording, the same set of parameters are available. You can use any of the following tools to control explicit recordings:

- JRockit Mission Control Client
- Command-Line Option
- Diagnostic Commands

3.3.1.1 JRockit Mission Control Client

The simplest way to control JRockit Flight Recorder is by using the JRockit Mission Control client. For more information, see Section 2.1, "Using JRockit Mission Control Client".

3.3.1.2 Command-Line Option

You can start and configure a recording from the command-line by using the `-XX:StartFlightRecording` start-up option:

```
-XX:StartFlightRecording=duration=<duration>,filename=<filename>
```

This command will start a recording immediately when the JVM starts. The recording then runs for the specified *`duration`* and will be saved to the given *`filename`*; for example, to starts a 60-second recording and to save the result in `myrecording.jfr` in the current directory, use this command:

```
-XX:StartFlightRecording=duration=60s,filename=myrecording.jfr
```

You can use several other options to further configure an explicit recording. For more information, see Section B.1, "Start-up Commands" or the *Oracle JRockit Command Line Reference*, available on the Oracle Technology Network.

3.3.1.3 Diagnostic Commands

You can also control recordings by using JRockit-specific diagnostic commands. For a more detailed description of Diagnostic Commands, see Section B.2, "Diagnostic Command Reference". The simplest way to execute a diagnostic command is to use the `JROCKIT_HOME/bin/jrcmd` (or `JROCKIT_HOME\bin\jrcmd.exe`) executable in the JRockit installation. Issue a diagnostic command with jrcmd in the following format:

```
jrcmd <pid> <command>
```

Where *`<pid>`* is the PID for the JVM to which to send the command and *`<command>`* is the diagnostic command itself.

Note: If you execute jrcmd without any parameters, Oracle JRockit will return a list the running Java processes and their PIDs.

You can see a list of available commands by typing:

```
jrcmd <pid> help
```

In this list, you will most likely find the relevant commands `start_flightrecording`, `check_flightrecording`, and `stop_flightrecording`.

To get detailed help for a commend, type:

```
jrcmd <pid> help <command>
```

For more information on using jrcmd, see "Using jrcmd" in the *Oracle JRockit JDK Tools*.

3.3.2 Starting a Recording

Diagnostic commands allow you to start a flight recording in a running JRockit instance. For example, to initiate a 60 second recording and save it to `myrecording.jfr` in the current directory, you would enter this jrcmd command:

```
jrcmd <pid> start_flightrecording duration=60s filename=myrecording.jfr
```

3.3.3 Checking Recording Status

To see which recordings are currently running and the status of each, use the `check_flightrecording` command. For example, if you type:

```
jrcmd <pid> check_flightrecording
```

and execute it before a recording ends, information similar to the following is displayed:

```
recording : id=1 name="myrecording.jfr" duration=60s dest="myrecording.jfr"
compress=false (running)
```

This indicates that the recording is still running.

3.3.4 Stopping a Recording

If a recording was started with the duration option, it will automatically stop after that time. You can also start a recording without specifying a duration, in which case it will run until explicitly stopped. Do this by using the `stop_flightrecording` diagnostic command:

```
jrcmd <pid> stop_flightrecording recording=1
```

The `recording` parameter indicates which recording to stop and its value is the `id`, as seen in the output of `check_flightrecording`.

3.4 Configuring Explicit Recordings

You can configure an explicit recording in a number of other ways. These techniques work the same regardless of how you started the recording; that is, either by using the command-line approach or by using diagnostic commands. This section contains the following information:

- Setting Maximum Size and Age
- Setting the Delay
- Setting Compression

3.4.1 Setting Maximum Size and Age

Like a default recording, you can configure an explicit recording to have a maximum size or age. For a more in-depth discussion of these concepts, see Section 3.2.2, "Configuring Disk Storage". You can configure size and age at startup by using the following parameters:

- To set the maximum size:

  ```
  maxsize=<size>
  ```

  `size` can be specified with the `k` (kilobytes), `m` (megabytes) and `g` (gigabytes) suffixes; for example, `10m`.

- To set the maximum age:

  ```
  maxage=<age>
  ```

  `age` can be specified by `s` (seconds), `m` (minutes), `h` (hours), or `d` (days); for example, `10s`.

If both a size limit and an age are specified, the data is deleted when it is older than the age or when the size limit is exceeded.

3.4.2 Setting the Delay

When scheduling a recording. you might want to add a delay before the recording is actually started; for example, when running from the command line, you might want the application to boot or reach a steady state before starting the recording. To achieve this, use the `delay` parameter:

```
delay=<delay>
```

Specify the delay period with `s` (seconds), `m` (minutes), `h` (hours), or `d` (days); for example, `10s`.

3.4.3 Setting Compression

Although the recording file format is very compact, you can compress it further by zipping the recording. To cause this to happen automatically, use the following parameter:

```
compress=true
```

Note that quite a bit of CPU power is required to do the compression which means that compressing recordings can negatively impact performance.

3.5 Creating Recordings Automatically

When running with a default recording you can configure JRockit Flight Recorder to automatically save the current in-memory recording data to a file whenever certain conditions occur. If a disk repository is used, the current information in the disk repository will also be included. This section includes the following information:

- Creating a Recording On Exit

- Creating a Recording On an Unhandled Exception
- Creating a Recording by Using Triggers
- Manually Dumping Recording Data

3.5.1 Creating a Recording On Exit

To save the recording data every time the JVM exits, use this command:

```
-XX:FlightRecorderOptions=dumponexit=true,dumponexitpath=<path>
```

Set *<path>* to the location where the recording should be saved. If you specify a directory, a file with a unique name is created in that directory. If you specify a file name, that name is used. If you do not specify a path, the recording will be saved in the current directory.

3.5.2 Creating a Recording On an Unhandled Exception

To create a recording when an unhandled exception occurs (that is, an exception that is not caught by any exception handlers in a thread, resulting in the thread terminating) use the `-XX:+FlightRecordingDumpOnUnhandledException` start-up command.

Specify the location for the recording dump by using the `-XX:FlightRecordingDumpPath=<path>` start-up command:

The same rules for *<path>* apply as in Section 3.5.1, "Creating a Recording On Exit".

3.5.3 Creating a Recording by Using Triggers

You can use the Console in JRockit Mission Control to set *triggers*. A trigger is a rule that executes an action whenever a condition specified by the rule is true. For example, you can create a rule that triggers a flight recording to commence whenever the heap size exceeds 100 MB. Triggers in JRockit Mission Control can use any property exposed through a JMX MBean as the input to the rule. They can launch many other actions than just Flight Recorder dumps.

Define triggers on the Triggers tab of the JRockit Mission Control Console's MBean page. For more information on how to create triggers, see the online help in JRockit Mission Control.

3.5.4 Manually Dumping Recording Data

In addition to the automatically dumping recording data from a running default recording, you can also do it manually by using the following diagnostic command:

```
jrcmd <jrockit pid> dump_flightrecording id=<id> copy_to_file=<path>
```

This dumps the currently available data for the recording identified by the given *<id>* to the file given in *<path>*. You can automatically compress the dumped recording by adding:

```
compress_copy=true
```

For more information about the diagnostic command, see Section B.2, "Diagnostic Command Reference".

Additionally, you can use JRockit Mission Control Client to dump recording data. Do the following:

1. Right-click a JVM in the JVM Browser on the Mission Control Console.

2. Select **Dump Default Recording…**

4

Controlling Recording Data by Using Templates

This chapter describes how to use templates for controlling the information that the flight recordings capture.

This chapter contains the following sections:

- Section 4.1, "What You Can Control"
- Section 4.2, "Mission Control Templates"

4.1 What You Can Control

Almost everything about JRockit Flight Recorder can be controlled through different kinds of settings. To control the amount of data that is recorded, you can configure the following information for each type of event:

- *Enabled*; you can enable or disable each event type for a particular recording. Disabling events that occur often (such as synchronization events) helps reduce the size of the recording.
- *Threshold*; you can filter events that have duration by setting a threshold. Events with shorter duration than the threshold are not saved.
- *Stack trace*; you can enable or disable stack trace information for each event for the point from which the event is generated. Creating stack traces can be costly but can be extremely helpful to understanding why the event happened.
- *Request periods*; you can configure the frequency with which a *requestable* events is requested by the JRockit Flight Recorder run time.

JRockit Flight Recorder uses templates so that you do not need to modify all these configuration options every time you start a recording. Sets of best-known options for different tasks have been stored in both Mission Control templates and server-side templates. You can use these templates as is or you can modify them to suit your needs.

WARNING: Even though the data format for a flight recording (`.jfr`) file is extremely compact, enabling too many event types in the recording, especially resource heavy ones—such as those that frequently collect stack traces—might produce large amounts of data.

4.2 Mission Control Templates

When you start a recording in JRockit Mission Control Client, you will be prompted for a recording template (see Figure 2–1). The template controls which events will be enabled during the recording. The following templates are available by default:

- Profiling Normal
- Profiling with Locks
- Profiling with Exceptions
- Real Time

4.2.1 Profiling Normal

This template includes most of the profiling events of interest. Some very low level events and very resource hungry events have been left out to ensure that the overhead remains low.

Overhead: Since this template's recording overhead is very low, you can use it in a production environment. See the JRockit Flight Recorder online help for more information.

4.2.2 Profiling with Locks

This template is very similar to the normal profiling template, except that it also includes very low level locking events. This profile is useful when you are hunting down lock-related issues; however, it will incur more overhead than the normal template. You must start the JVM with `-XX:+UseLockProfiling` for this template to work

Overhead: Since the JVM must be started with a flag that incurs overhead even when not profiling, you should not use this template in a production environment.

4.2.3 Profiling with Exceptions

This template is very similar to the normal profiling template but it also includes exception events with stack traces. This profile can be quite expensive if the application throws a lot of exceptions. Of course, if that is happening, the application is very likely running slower than it should. This profile is useful when you are hunting down exception related issues but it, too, incurs more overhead than the normal template.

Overhead: You can use this template in a production environment but be aware it incurs more overhead than the default. This overhead is proportional to the number of exceptions being thrown

4.2.4 Real Time

This template focuses on memory system related information, such as garbage collection information. It uses the same event types as the built-in default recording.

Overhead: You can use this template in a production environment. It is enabled by default and no overhead is introduced. See the JRockit Flight Recorder online help for more information

4.2.5 Modifying a Template

You can modify a template to change exactly which events are recorded and how. Click on "Advanced…" to do this. Templates can be saved to disk and shared with others.

4.2.6 Server-side Templates

When starting a recording from the command line or by using a diagnostic command, a number of different templates are available ready-to-use. Table 4–1 describes this templates.

Table 4–1 Server-side Templates

Template Name	Description
code	Additional settings for enabling more verbose compiler logging.
default	Default settings tuned for a very low performance overhead and recommended for always-on production use.
freemem	Additional settings for debugging out-of-memory and fragmentation problems.
full	Enables collection of all available events for all subsystems. Warning: This has a very high performance overhead.
io	Additional settings for enabling more verbose Java I/O logging.
leak	Additional settings for debugging memory leaks.
locks	Additional settings for enabling more verbose synchronization logging.
memory	Additional settings for enabling more verbose GC/memory management logging.
off	Disables all events for all subsystems.
profile	Recommended settings for creating a profiling recording. They provide a good balance between the amount of information available and the performance overhead introduced.
sample	Additional settings for enabling hotspot sampling of code.
semirefs	Additional settings for debugging problems with `java.lang.ref.Reference objects` and its subclasses.

Specify templates with the `settings` parameter when staring a recording; for example:

```
jrcmd <pid> start_flightrecording duration=5min settings=io
```

or

```
-XX:FlightRecorderOptions=defaultrecording=true,settings=default,settings=freemem
```

The preceding example shows how you can combine templates by specifying several `settings` parameters.

You can also design custom templates by creating your own template file. The templates are stored in `.jfs` files in the directory `JROCKIT_HOME/jre/lib/jfr`. The easiest way to create your own template is to make a copy of one of the existing files and modify it. For a detailed description of the file format, see Section A.3, "File Format".

5

Troubleshooting and Security

This chapter describes JRockit Flight Recorder troubleshooting and security measures.

It includes these sections:

- Section 5.1, "Troubleshooting"
- Section 5.2, "Security"

5.1 Troubleshooting

You can enable a significant amount of diagnostic information from JRockit Flight Recorder by starting the Oracle JRockit JVM with the command-line option `-Xverbose`:jfr. To include more information, use `-Xverbose:jfr=debug` and `-Xverbose:jfr=trace`. For more information on `-Xverbose`, see the *Oracle JRockit Command Line Reference* on the Oracle Technology Network.

5.2 Security

The recording file can potentially contain security information (such as user names and passwords, if they are specified on the command line). You should treat them with care.

You can only start a recording by using one of the following means: the command line, diagnostic commands, or JRockit Mission Control. These methods provide security as described in Table 5–1:

Table 5–1 Security Permissions

Method	Security
Command line	Anyone with access to the command line of the JRockit process must be trusted.
Diagnostic Command	Only the owner of the JRockit process can use jrcmd to control the process.
JRockit Mission Control Client	JRockit Mission Control Client uses JMX to access JRockit. .

A Creating Your Own Server-side Templates

This chapter describes how you can modify or write your own server-side templates.

It contains these sections:

- Section A.1, "Event Types and Relational Keys"
- Section A.2, "Server-side Templates"
- Section A.3, "File Format"
- Section A.4, "Concatenation Tool"

A.1 Event Types and Relational Keys

Event types are referenced by relational keys, describing what type of subsystem in the JVM or application they belong to. The relational key for event types for all JVM internal systems start with `http://www.oracle.com/jrockit/jvm`. The JVM has different subsystems, such as `vm` which refers to the runtime, `os` that refers to the operating system it runs on, `java` which refers to the executing Java program, and so on; for example, the key for the event that is triggered upon the JVM entering a lock at the native level is called:

```
http://www.oracle.com/jrockit/jvm/vm/sync/mutex_enter
```

The only other master relational key except for the `http://www.oracle.com/jrockit/jvm` key you might encounter is `http://www.oracle.com/jrockit/jfr-info`, which is the "meta producer" for Flight Recorder; that is, event types internal to JRockit Flight Recorder.

A.2 Server-side Templates

Many event types are enabled in the default flight recording, however through server-side templates you can customize this.

A server-side template is a file with the suffix `.jfs`. It contains data in JSON format, which is used to modify or extend the settings of a flight recording. Normally, you do not need to create your own server-side template but, should you have to do this, you can modify one of the sample templates in the `JROCKIT_HOME/jre/lib/jfr` folder.

You can pass server-side templates that customize a recording to the Oracle JRockit JVM by using the `-XX:FlightRecorderOptions` with the `settings` subflag or by using the `start_flightrecording` diagnostic command. `settings` can either be the name of a predefined template (located in `JROCKIT_HOME/jre/lib/jfr`) or the path to a completely custom template.

A.3 File Format

Each server-side template consists of a single section, containing mappings of relational keys representing event types to their customized properties. For each event type, you can set the properties `enable`, `stacktrace`, `threshold`, and `period`:

- Set `enable` to either `true` or `false,` depending on whether or not the recording should contain this event.
- Set `stacktrace` to either `true` or `false`, depending on if a stack trace should be collected from the point that triggered the event.
- Set a `threshold` for the minimum duration of the event (where applicable) that you want logged.
- Set `period` to how often you want the event triggered (for requestable events); for example how often exception statistics should be gathered for events in the `/java/statistics/exceptions` event under the key `http://www.oracle.com/jrockit/jvm`. If `period` is set to `0` this specifies a *constant event*; that is, one that is only generated once per recording (such as logging the system properties). Depending on event types, the number of applicable modifiers might vary.

The typical format for a server-side template (`.jfs`) file is shown in Example A–1:

Example A–1   Typical Format for a Server-side Template

```
{
    <relational-key> : {
             <sub-key 1> : {
                      <attribute> : <value>
                      <attribute> : <value>
                      ...
                      <attribute> : <value>
             },
             ...
             <sub-key 2> : {
                      <attribute> : <value>
                      <attribute> : <value>
                      ...
                      <attribute> : <value>
            },
    },
    ...
}
```

For readability, the relational key is usually split into several levels; for example, a server-side template enabling more verbose Java I/O information in the recording might look like Example A–2:

Example A–2   Server-side Template Enabling More Verbose Java I/O Information in the Recording

```
{
   "http://www.oracle.com/jrockit/jvm/" : {

           // Socket/SocketChannel read/write
           "java/socket_*" : {
                "enable" : true,
                "stacktrace" : true
           },
            // FileInputStream/RandomAccessFile/FileChannel read/write
            "java/file_*" : {
                 "enable" : true,
                 "stacktrace" : true
            }
    }
}
```

You can use wildcard for attributes and relational keys. In Example A–1, all event types with descriptors starting with `http://www.oracle.com/jrockit/jvm/java/socket_` and `http://www.oracle.com/jrockit/jvm/java/file_` are enabled (with stack trace recording).

Wildcards can be arbitrarily powerful. For example, in Example A–3, brute force enables collections of all event types, with a minimum period of 1,000 ms to avoid extreme data bloat:

Example A–3 Brute Force Enabling the Collecting of All Event Types

```
// Settings file for JRockit Flight Recorder enabling collection of all events
{
    "*" : {
       "enable" : true,
       "stacktrace" : true,
       "threshold" : 0,
       "period" : 1000ms
    }
}
```

You should copy and play around with the pre-installed custom templates available in the `JROCKIT_HOME/jre/lib/jfr` to get a better understanding of how this works.

A.4 Concatenation Tool

The JRockit Flight Recorder repository is made up of multiple files which you might find tedious to open, one after the other, in JRockit Mission Control. You can, however, use a Flight Recorder tool that concatenates all of the recording files in a repository into a single file that you can then open in JRockit Mission Control.

Use the concatenation tool by entering this command:

```
java oracle.jrockit.jfr.tools.ConCatRepository [directory] [-o output_filename]
[-f]
```

- If no arguments are given, the tool creates a file based on the timestamps from the chunk files in the current directory.
- If `directory` is specified, that directory is used as repository.
- If `-o` *`output_filename`* is specified, the resulting file is named *`output_filename`*.
- If `-f` is specified, any existing file with the same name is overwritten.

B Command Reference

This appendix serves as a basic reference to the commands you can use with the JRockit Flight Recorder. It contains these sections:

- Section B.1, "Start-up Commands"
- Section B.2, "Diagnostic Command Reference"

B.1 Start-up Commands

Start-up Commands are the `-X` and `-XX` command-line options that you enter when you start a Java program. The specific JRockit Flight Recorder command-line options are:

- `-XX:+|-FlightRecorder`
- `-XX:FlightRecorderOptions`
- `-XX:+|-FlightRecordingDumpOnUnhandledException`
- `-XX:FlightRecordingDumpPath`
- `-XX:StartFlightRecording`

These commands are described in the *Oracle JRockit Command Line Reference*, available on the Oracle Technology Network.

Note: You should use `-XX` options only if you have a thorough understanding of your system. If you use these commands improperly, you might affect the stability or performance of your system. `-XX` options are subject to change at any time.

B.2 Diagnostic Command Reference

This is a description of the Diagnostic Commands available to control JRockit Flight Recorder and the parameters available for each command. This information is also available by typing `jrcmd <pid> help <command>`. The diagnostic commands associated with the JRockit Flight Recorder are:

- start_flightrecording
- check_flightrecording
- stop_flightrecording
- dump_flightrecording

These commands are described in the *Oracle JRockit Command Line Reference*, available on the Oracle Technology Network.

For more information on using jrcmd, see "Using jrmcd" in the *Oracle JRockit JDK Tools*.

B.2.1 start_flightrecording

The `start_flightrecording` diagnostic command starts a flight recording. Table B–1 lists the parameters you can use with this command.

Table B–1 start_flightrecording

Parameter	Description	Type of value	Default
name	Name of recording	String	
settings	Server-side template	String	
defaultrecording	Starts default recording	Boolean	False
delay	Delay start of recording	Time	0s
duration	Duration of recording	Time	0s (means "forever")
filename	Resulting recording filename	String	
compress	GZip compress the resulting recording file	Boolean	False
maxage	Maximum age of buffer data	Time	0s (means "no age limit")
maxsize	Maximum size of buffers in bytes	Long	0 (means "no max size")

B.2.2 check_flightrecording

The `check_flightrecording` diagnostic command checks running flight recordings. Table B–2 lists the parameters you can use with this command.

Table B–2 check_flightrecording

Parameter	Description	Type of value	Default
name	Recording name	String	
recording	Recording id	Long	1
verbose	Print verbose data about the recording(s)	Boolean	False

B.2.3 stop_flightrecording

The `stop_flightrecording` diagnostic command stops running flight recordings. Table B–3 lists the parameters you can use with this command.

Table B–3 stop_flightrecording

Parameter	Description	Type of value	Default
name	Recording name	String	
recording	Recording id	Long	1
discard	Discards the recording data	Boolean	
copy_to_file	Copy recording data to file	String	

Table B–3 (Cont.) stop_flightrecording

Parameter	Description	Type of value	Default
compress_copy	GZip compress "copy_to_file" destination	Boolean	False

B.2.4 dump_flightrecording

The `dump_flightrecording` diagnostic command dumps flight recordings. Table B–4 lists the parameters you can use with this command.

Table B–4 dump_flightrecording

Parameter	Description	Type of value	Default
name	Recording name	String	
recording	Recording id	Long	1
copy_to_file	Copy recording data to file	String	
compress_copy	GZip compress "copy_to_file" destination	Boolean	False

C Events

The Flight Recorder records "events" that occur during run time. An event is a distinct data point with associated data; that is, it is any occurrence during run time that can be recorded, such as the CPU load at a certain time or a thread waiting for a lock. These events are then reported on the Flight Recorder GUI to provide insight into system health and behavior.

This appendix describes the structure of events and how these events are used. This list describes the events reported by the JVM and class libraries only. Additional events are available from applications running on the JVM (such as WebLogic Server).

Note: This appendix contains a number of Oracle JRockit- and JVM-specific terms that might be unfamiliar to you. If you encounter any unfamiliar terminology, we recommend you refer to the other documentation in the Oracle JRockit library.

Table C–1 lists the events you can capture in a flight recording.

Table C–1 JRockit Flight Recorder Events

Name	Description	Path
Exception Thrown		`java/exception_throw`
File Read	Reading from Java FileInputStream/RandomAccessFile/FileChannel	`java/file_read`
File Write	Writing to Java FileInputStream/RandomAccessFile/FileChannel	`java/file_write`
Java Monitor Enter	Entering Java monitor	`java/monitor_enter`
Java Monitor Wait	Waiting for Java monitor	`java/monitor_wait`
Object Allocated in New TLA	Object was allocated, which required a new thread local area (TLA) to be retrieved	`java/object_alloc_in_new_tla`
Object Allocated outside TLA	Object was allocated outside a TLA, directly on the heap	`java/object_alloc_outside_tla`
Socket Read	Reading from Java Socket/SocketChannel	`java/socket_read`
Socket Write	Writing to Java Socket/SocketChannel	`java/socket_write`

Table C–1 (Cont.) JRockit Flight Recorder Events

Name	Description	Path
Allocated by Thread	Total number of bytes and TLAs that have been allocated by the thread	`java/statistics/alloc_thread`
Allocated by All Threads	Summary of the total number of bytes and TLAs that have been allocated, for all threads	`java/statistics/alloc_total`
Exception Count	Accumulated number of thrown exceptions	`java/statistics/exceptions`
Java Lock Profiling Snapshot	Detailed profiling information on Java locking for class	`java/statistics/lock_profile`
Java Thread Statistics		`java/statistics/threads`
Java Thread End		`java/thread_end`
Java Thread Park	Waiting in `LockSupport.park()`	`java/thread_park`
Java Thread Sleep		`java/thread_sleep`
Java Thread Start		`java/thread_start`
Thread Context Switch Rate		`os/context_switch_rate`
CPU Load Sample		
Environment Variables		`os/environment`
Physical Memory Statistics		`os/physical_memory`
Active System Processes		`os/processes`
List of Active Recordings		`recordings/active`
Event Settings Changed		`recordings/settingsChanged`
Class GC Free Data		`vm/class/free`
Class Load		`vm/class/load`
Class GC Unlink	Removal of unreachable classes	`vm/class/unlink`
Class Unload		`vm/class/unload`
Code GC Call Prune	Removal of calls to obsolete code	`vm/codegc/prune_calls`
Code GC Code Prune	Cleanup of global lookup tables	`vm/codegc/prune_code`
Code GC Free Data		`vm/codegc/release_code`
Compiler Statistics		`vm/compiler/compiler_statistics`
Compilation Abort	Aborted compilation due to exception or error	`vm/compiler/fail`
JIT Code Compilation		`vm/compiler/jit_compile`
Method Inline		`vm/compiler/method_inline`

Table C–1 (Cont.) JRockit Flight Recorder Events

Name	Description	Path
Optimized Code Compilation		`vm/compiler/opt_compile`
Code Performance Warning	Performance warning for compiled method	`vm/compiler/performance_log`
JIT Compiler Phase Level 1	Statistics for specific JIT compilation phase	`vm/compiler/phases/jit_phase_level_1`
JIT Compiler Phase Level 2	Statistics for specific JIT compilation phase	`vm/compiler/phases/jit_phase_level_2`
JIT Compiler Phase Level 3	Statistics for specific JIT compilation phase	`vm/compiler/phases/jit_phase_level_3`
JIT Compiler Phase Trace Level 1	Detailed statistics for specific JIT compilation phase	`vm/compiler/phases/jit_phase_trace_level_1`
JIT Compiler Phase Trace Level 2	Detailed statistics for specific JIT compilation phase	`vm/compiler/phases/jit_phase_trace_level_2`
JIT Compiler Phase Trace Level 3	Detailed statistics for specific JIT compilation phase	`vm/compiler/phases/jit_phase_trace_level_3`
Optimizer Phase Level 1	Statistics for specific code optimization phase	`vm/compiler/phases/opt_phase_level_1`
Optimizer Phase Level 2	Statistics for specific code optimization phase	`vm/compiler/phases/opt_phase_level_2`
Optimizer Phase Level 3	Statistics for specific code optimization phase	`vm/compiler/phases/opt_phase_level_3`
Optimizer Phase Trace Level 1	Detailed statistics for specific code optimization phase	`vm/compiler/phases/opt_phase_trace_level_1`
Optimizer Phase Trace Level 2	Detailed statistics for specific code optimization phase	vm/compiler/phases/opt_phase_trace_level_2
Optimizer Phase Trace Level 3	Detailed statistics for specific code optimization phase	`vm/compiler/phases/opt_phase_trace_level_3`
CPU Information	Detailed description of the CPU(s) in the system	`vm/cpu_info`
Free Memory Cache Bucket Element	Individual free memory cache bucket element	`vm/freemem/cache_bucket_elem`
Free Memory Cache Bucket Summary	Summary of an individual bucket in the free memory cache	`vm/freemem/cache_bucket_total`
Free Memory Cache Summary	Summary of free memory in the free memory cache, excluding the free memory list and the free memory TLA cache	`vm/freemem/cache_total`
Free Memory List Element	Individual free memory list element	`vm/freemem/list_elem`
Free Memory List Summary	Summary of free memory in the free memory list, excluding the free memory cache and the free memory TLA cache	`vm/freemem/list_total`
Pending Allocation Request Created	Pending allocation request is created by the thread	`vm/gc/alloc_pending/request_created`

Table C–1 (Cont.) JRockit Flight Recorder Events

Name	Description	Path
Pending Allocation Request Got OOM	Pending allocation request was denied and turned into an OutOfMemoryException by the garbage collector	`vm/gc/alloc_pending/request_got_oom`
Pending Allocation Request Satisfied	Pending allocation request is satisfied by the garbage collector	`vm/gc/alloc_pending/request_satisfied`
Pending Allocations at GC End	Summary of the pending allocation requests at the end of the garbage collection	`vm/gc/alloc_pending/summary_gc_end`
Pending Allocations at GC Start	Summary of the pending allocation requests at the start of the garbage collection	`vm/gc/alloc_pending/summary_gc_start`
Compaction	Compaction of the live data on the heap, done as part of an old collection	`vm/gc/compaction/compaction`
Compaction Heap Shrink Preparation	Compaction prepared for shrinking the heap	`vm/gc/compaction/heap_shrink_preparation`
Compaction Move Phase	Move phase of a dual-phased compaction	`vm/gc/compaction/phases/move`
Compaction Update Phase	Update phase of a dual-phased compaction	`vm/gc/compaction/phases/update`
GC Configuration	Configuration of parameters for the garbage collector	`vm/gc/configuration`
Emergency Parallel Sweep Requested	Garbage collector changed sweep from concurrent to parallel due to special circumstances	`vm/gc/emergency_parallel_sweep_requested`
Garbage Collection	Garbage collection performed by the JVM (old collection or young collection)	`vm/gc/garbage_collection`
Heap Size Changed	Change of the heap size (expansion or contraction)	`vm/gc/heap_size_changed`
Heap Blocks Snapshot	Statistics for contiguous blocks of heap memory (used, free, dark matter)	`vm/gc/heap_statistics/blocks`
Heap Usage Snapshot	Statistics for classes that take up more than 0.5% of the heap	`vm/gc/heap_statistics/class`
GC History	Summary of previously finished garbage collections	`vm/gc/history`
GC Mode Changed	Garbage collector changed mode	`vm/gc/mode_changed`
Old Collection	Old collection performed by the JVM (collecting the whole heap)	`vm/gc/oc/old_collection`
OC Mark Phase	Mark phase of an old collection	`vm/gc/oc/phases/mark`
OC Sweep Phase	Sweep phase of an old collection	`vm/gc/oc/phases/sweep`

Table C–1 (Cont.) JRockit Flight Recorder Events

Name	Description	Path
GC Concurrent Phase	Top-level phase of the garbage collection, during which the threads are running	`vm/gc/phases/concurrent`
GC Stopped Phase	Top-level phase of the garbage collection, during which the threads are stopped	`vm/gc/phases/stopped`
GC Transition Phase from Stopped	Top-level phase of the garbage collection, during which the threads are transitioning from stopped to running	`vm/gc/phases/transition_from_stopped`
GC Transition Phase to Stopped	Top-level phase of the garbage collection, during which the threads are transitioning from running to stopped	`vm/gc/phases/transition_to_stopped`
GC Concurrent Sub-Level 1 Phase	Sub-level phase of the garbage collection, during which the threads are running	`vm/gc/phases_sublevels/level_1/concurrent`
GC Stopped Sub-Level 1 Phase	Sub-level phase of the garbage collection, during which the threads are stopped	`vm/gc/phases_sublevels/level_1/stopped`
GC Concurrent Sub-Level 2 Phase	Sub-level phase of the garbage collection, during which the threads are running	`vm/gc/phases_sublevels/level_2/concurrent`
GC Stopped Sub-Level 2 Phase	Sub-level phase of the garbage collection, during which the threads are stopped	`vm/gc/phases_sublevels/level_2/stopped`
GC Concurrent Sub-Level 3 Phase	Sub-level phase of the garbage collection, during which the threads are running	`vm/gc/phases_sublevels/level_3/concurrent`
GC Stopped Sub-Level 3 Phase	Sub-level phase of the garbage collection, during which the threads are stopped	`vm/gc/phases_sublevels/level_3/stopped`
GC Concurrent Sub-Level 4 Phase	Sub-level phase of the garbage collection, during which the threads are running	`vm/gc/phases_sublevels/level_4/concurrent`
GC Stopped Sub-Level 4 Phase	Sub-level phase of the garbage collection, during which the threads are stopped	`vm/gc/phases_sublevels/level_4/stopped`
GC Requested	Garbage collection request generated by the requesting thread, including the reason for the garbage collection	`vm/gc/request`
Semiref Processing Phase Snapshot	Number of semirefs (reference objects and handles) that were processed during different garbage collection phases	`vm/gc/semiref/counts_phase`
Semiref State Snapshot	Number of semirefs (reference objects and handles) in different states	`vm/gc/semiref/counts_state`

Table C–1 (Cont.) JRockit Flight Recorder Events

Name	Description	Path
Semiref Soft Alive Snapshot	Number of soft references that were soft alive (i.e. not eligible for garbage collection due to too recent access)	`vm/gc/semiref/counts_state_softalive`
Semiref Class Snapshot	Detailed information on all pairs of semiref and referent classes	`vm/gc/semiref/details_class`
Total Semiref Count	Total number of semirefs (reference objects and handles)	`vm/gc/semiref/total`
GC Strategy Changed	Garbage collector changed strategy	`vm/gc/strategy_changed`
Young Collector Nursery Snapshot	Updated status of the nursery after a young collection	`vm/gc/yc/nursery`
YC Promotion Failed	Promotion of an object failed during a young collection, since the old space of the heap is full	`vm/gc/yc/promotion_failed`
Young Collection	Young collection performed by the JVM (collecting the nursery only)	`vm/gc/yc/young_collection`
JVM Information	Description of JVM, Java application and Operating System	`vm/info`
Method Profiling Sample	Snapshot of the state of a thread	`vm/prof/execution_sample`
Method Hotspot Sample	A more lightweight sample of the state of a thread state. This redundant and not generated if 'execution_sample' is enabled.	`vm/prof/hotspotsample`
Memory Usage	Snapshot of JVM virtual memory footprint	`vm/prof/memory_usage`
JVM Event Wait		`vm/sync/event_wait`
JVM Lock Profiling Sample	Detailed profiling information on a JVM lock object	`vm/sync/lock_profile`
JVM Monitor Wait		`vm/sync/monitor_wait`
JVM Mutex Enter		`vm/sync/mutex_enter`
System Properties	System properties set at command line	`vm/system_properties`
JVM Thread Sleep		`vm/thread/sleep`
JVM Thread Suspend	Suspension of JVM Thread	`vm/thread/suspend`
JVM Thread Suspended		`vm/thread/suspended`