OpenJDK 8

Using JDK Flight Recorder for JDK Mission Control

Last Updated: 2021-04-22

Legal Notice

Copyright © 2021 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

OpenJDK 8 is a Red Hat offering on the Red Hat Enterprise Linux platform. The Using Java Flight Recorder for Java Mission Control guide provides an overview of Java Flight Recorder (JFR) and Java Mission Control (JMC), and explains how to start the JFR.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE 3

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION 4

CHAPTER 1. INTRODUCTION TO JDK FLIGHT RECORDER 5
- 1.1. JFR COMPONENTS 5

CHAPTER 2. BENEFITS OF USING JDK FLIGHT RECORDER 6

CHAPTER 3. INTRODUCTION TO JDK MISSION CONTROL 7

CHAPTER 4. STARTING JDK FLIGHT RECORDER 8
- 4.1. STARTING JDK FLIGHT RECORDER WHEN JVM STARTS 8
- 4.2. STARTING JDK FLIGHT RECORDER ON A RUNNING JVM 8
- 4.3. STARTING THE JDK FLIGHT RECORDER ON JVM BY USING THE JDK MISSION CONTROL 9

CHAPTER 5. CONFIGURATION OPTIONS FOR JDK FLIGHT RECORDER 11
- 5.1. CONFIGURE JDK FLIGHT RECORDER USING THE COMMAND LINE 11
  - 5.1.1. Start JFR 11
  - 5.1.2. Control behavior of JFR 12
- 5.2. CONFIGURING JDK FLIGHT RECORDER USING DIAGNOSTIC COMMAND (JMCD) 13
  - 5.2.1. Start JFR 14
  - 5.2.2. Stop JFR 14
  - 5.2.3. Check JFR 14
  - 5.2.4. Dump JFR 15
  - 5.2.5. Configure JFR 16

CHAPTER 6. HISTORICAL ANALYSIS 17

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. To provide feedback, you can highlight the text in a document and add comments.

This section explains how to submit feedback.

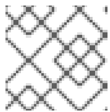
Prerequisites

- You are logged in to the Red Hat Customer Portal.
- In the Red Hat Customer Portal, view the document in **Multi-page HTML** format.

Procedure

To provide your feedback, perform the following steps:

1. Click the **Feedback** button in the top-right corner of the document to see existing feedback.



NOTE

The feedback feature is enabled only in the **Multi-page HTML** format.

2. Highlight the section of the document where you want to provide feedback.
3. Click the **Add Feedback** pop-up that appears near the highlighted text.
   A text box appears in the feedback section on the right side of the page.
4. Enter your feedback in the text box and click **Submit**.
   A documentation issue is created.
5. To view the issue, click the issue tracker link in the feedback view.

CHAPTER 1. INTRODUCTION TO JDK FLIGHT RECORDER

JDK Flight Recorder (JFR) is a low overhead framework for monitoring and profiling: - Java applications - Java Virtual Machine (JVM) - Runtimes For more information, see https://openjdk.java.net/jeps/328

You can collect data from events originating within the JVM and the application code. Data is then written in memory. At first, to thread-local buffer and then promoted to fixed-size global ring buffer before being flushed to **JFR** files (*.jfr) on the disk. Other applications can consume these files for analysis. For example, by JFR tool included with OpenJDK or by **JDK Mission Control** (JMC).

1.1. JFR COMPONENTS

Recordings

You can manage system recordings. Each recording has a unique configuration. You can start or stop the recording, or dump it to disk on demand.

Events

Events provide the tracing of data as well as metadata that forms a JFR file. The JVM has many pre-existing events that are continuously added. An API is available for users to inject custom events into their applications.
You can enable or disable any event when recording to minimize overhead by supplying event configurations. These configurations take the form of **xml** documents and are called JFR profiles (*.jfc). The OpenJDK comes with two profiles for the most common set of use cases:

- **default**: The *default* profile is a low-overhead configuration that is safe for continuous use in production environments. Typically, overhead is less than 1%.
- **profile**: The *profile* profile is a low-overhead configuration that is ideal for profiling. Typically, overhead is less than 2%.

CHAPTER 2. BENEFITS OF USING JDK FLIGHT RECORDER

Some of the key benefits of using JDK Flight Recorder (JFR) are:

- JFR allows recording on a running JVM. It is ideal to use JFR in production environments where it is difficult to restart or rebuild the application.
- JFR allows for the definition of custom events and metrics to monitor.
- JFR is built into the JVM to achieve the minimum performance overhead (around 1%).
- JFR uses coherent data modeling to provide better cross-referencing of events and filtering of data.
- JFR allows for monitoring of third-party applications using APIs.
- JFR helps in reducing the cost of ownership by:
    - Spending less time diagnosing.
    - Aiding in troubleshooting problems.
- JFR reduces operating costs and business interruptions by:
    - Providing faster resolution time.
    - Identifying the performance issues which helps in improving system efficiency.

CHAPTER 3. INTRODUCTION TO JDK MISSION CONTROL

JDK Mission Control (JMC) is a collection of tools to read and analyze JFR files. It includes detailed views and graphs that plot JFR events. With JFR analysis, JMC also consists of:

- JMX Console MBean
- Historical analysis via flight recordings and **hprof** files (as of JMC 7.1.0)
- HPROF-dump analyzer

JMC is based on the Eclipse platform. You can extend JMC by adding plugins using the Eclipse RCP API and other specific APIs.

CHAPTER 4. STARTING JDK FLIGHT RECORDER

4.1. STARTING JDK FLIGHT RECORDER WHEN JVM STARTS

You can start the JDK Flight Recorder (JFR) when a Java process starts. You can modify the behavior of the JFR by adding optional parameters.

Procedure

- Run the **java** command using the **--XX** option.
  $ java --XX:StartFlightRecording ***Demo***

  where *Demo* is the name of the Java application.

  The JFR starts with the Java application.

Example

The following command starts a Java process (*Demo*) and with it initiates an hour-long flight recording which is saved to a file called **demorecording.jfr**:

$ java --XX:StartFlightRecording=duration=1h,filename=demorecording.jfr ***Demo***

Additional resources

- For a detailed list of JFR options, see Java tools reference.

4.2. STARTING JDK FLIGHT RECORDER ON A RUNNING JVM

You can use the **jcmd** utility to send diagnostic command requests to a running JVM. **jcmd** includes commands for interacting with JFR, with the most basic commands being **start**, **dump**, and **stop**.

To interact with a JVM, **jcmd** requires the process id (pid) of the JVM. You can retrieve the by using the **jcmd -l** command which displays a list of the running JVM process ids, as well as other information such as the main class and command-line arguments that were used to launch the processes.

The **jcmd** utility is located under **$JAVA_HOME/bin**.

Procedure

- Start a flight recording using the following command:
  $ jcmd ***<pid>*** **JFR.start** ***<options>***

  For example, the following command starts a recording named **demorecording**, which keeps data from the last four hours, and has size limit of 400 MB:

  $ jcmd ***<pid>*** **JFR.start name=demorecording maxage=4h maxsize=400MB**

Additional resources

- For a detailed list of **jcmd** options, see jcmd Tools Reference.

4.3. STARTING THE JDK FLIGHT RECORDER ON JVM BY USING THE JDK MISSION CONTROL

The **JDK Mission Control** (JMC) application has a Flight Recording Wizard that allows for a streamlined experience of starting and configuring flight recordings.

Procedure

1. Open the JVM Browser.
   $ JAVA_HOME/bin/jmc

2. Right-click a JVM in JVM Browser view and select **Start Flight Recording**.
   The Flight Recording Wizard opens.

   Figure 4.1. JMC JFR Wizard

   The JDK Flight Recording Wizard has three pages:

   - The first page of the wizard contains general settings for the flight recording including:
     - Name of the recording
     - Path and filename to which the recording is saved
     - Whether the recording is a fixed-time or continuous recording, which event template will be used
     - Description of the recording
   - The second page contains event options for the flight recording. You can configure the level of detail that Garbage Collections, Memory Profiling, and Method Sampling and other events record.
   - The third page contains settings for the event details. You can turn events on or off, enable the recording of stack traces, and alter the time threshold required to record an event.

3. Edit the settings for the recording.

4. Click **Finish**.
   The wizard exits and the flight recording starts.

CHAPTER 5. CONFIGURATION OPTIONS FOR JDK FLIGHT RECORDER

You can configure JDK Flight Recorder (JFR) to capture various sets of events using the command line or diagnostic commands.

5.1. CONFIGURE JDK FLIGHT RECORDER USING THE COMMAND LINE

You can configure JDK Flight Recorder (JFR) from the command line using the following options:

5.1.1. Start JFR

Use **-XX:StartFlightRecording** option to starts a JFR recording for the Java application. For example:

```
java -
XX:StartFlightRecording=delay=5s,disk=false,dumponexit=true,duration=60s,filename=myrecording.jfr
<<YOUR_JAVA_APPLICATION>>
```

You can set the following parameter=value entries when starting a JFR recording:

delay=time

Use this parameter to specify the delay between the Java application launch time and the start of the recording. Append s to specify the time in seconds, m for minutes, h for hours, or d for days. For example, specifying 10m means 10 minutes. By default, there is no delay, and this parameter is set to 0.

disk={true|false}

Use this parameter to specify whether to write data to disk while recording. By default, this parameter is **true**.

dumponexit={true|false}

Use this parameter to specify if the running recording is dumped when the JVM shuts down. If the parameter is enabled and a file name is not set, the recording is written to a file in the directory where the recording progress has started. The file name is a system-generated name that contains the process ID, recording ID, and current timestamp. For example, hotspot-pid-47496-id-1-2018_01_25_19_10_41.jfr. By default, this parameter is **false**.

duration=time

Use this parameter to specify the duration of the recording. Append s to specify the time in seconds, m for minutes, h for hours, or d for days. For example, if you specify duration as 5h, it indicates 5 hours. By default, this parameter is set to 0, which means there is no limit set on the recording duration.

filename=path

Use this parameter to specify the path and name of the recording file. The recording is written to this file when stopped. For example:

- recording.jfr
- /home/user/recordings/recording.jfr

name=identifier

Use this parameter to specify both the name and the identifier of a recording.

maxage=time

Use this parameter to specify the maximum number of days the recording should be available on the disk. This parameter is valid only when the disk parameter is set to true. Append s to specify the time in seconds, m for minutes, h for hours, or d for days. For example, when you specify 30s, it indicates 30 seconds. By default, this parameter is set to 0, which means there is no limit set.

maxsize=size

Use this parameter to specify the maximum size of disk data to keep for the recording. This parameter is valid only when the disk parameter is set to true. The value must not be less than the value for the **maxchunksize** parameter set with **-XX:FlightRecorderOptions**. Append m or M to specify the size in megabytes, or g or G to specify the size in gigabytes. By default, the maximum size of disk data isn't limited, and this parameter is set to 0.

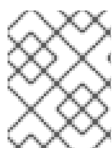
path-to-gc-roots={true|false}

Use this parameter to specify whether to collect the path to garbage collection (GC) roots at the end of a recording. By default, this parameter is set to false.

The path to GC roots is useful for finding memory leaks. For OpenJDK 8, you can enable **OldObjectSample** event which is a more efficient alternative than using heap dumps. You can use **OldObjectSample** event in the production as well. Collecting memory leaks is still time-consuming and incurs extra overhead. You should enable this parameter only when you start recording an application that you suspect has memory leaks. If the JFR profile parameter is set to profile, you can trace the stack from where the object is leaking. It is included in the information collected.

settings=path

Use this parameter to specify the path and name of the event settings file (of type JFC). By default, the default.jfc file is used, which is located in JAVA_HOME/lib/jfr. This default settings file collects a predefined set of information with low overhead, so it has minimal impact on performance and can be used with recordings that run continuously. The second settings file is also provided, profile.jfc, which provides more data than the default configuration, but can have more overhead and impact performance. Use this configuration for short periods of time when more information is needed.



NOTE

You can specify values for multiple parameters by separating them with a comma. For example, **-XX:StartFlightRecording=disk=false**,**name=example-recording**.

5.1.2. Control behavior of JFR

Use **-XX:FlightRecorderOptions** option to sets the parameters that control the behavior of JFR. For example:

```
java -XX:FlightRecorderOptions=duration=60s,filename=myrecording.jfr -
XX:FlightRecorderOptions=stackdepth=128,maxchunksize=2M <<YOUR_JAVA_APPLICATION>>
```

You can set the following parameter=value entries to control the behavior of JFR:

globalbuffersize=size

Use this parameter to specify the total amount of primary memory used for data retention. The default value is based on the value specified for **memorysize**. You can change the **memorysize** parameter to alter the size of global buffers.

maxchunksize=size

Use this parameter to specify the maximum size of the data chunks in a recording. Append m or M to specify the size in megabytes (MB), or g or G to specify the size in gigabytes (GB). By default, the maximum size of data chunks is set to 12 MB. The minimum size allowed is 1 MB.

memorysize=size

Use this parameter to determine how much buffer memory should be used. The parameter sets the **globalbuffersize** and **numglobalbuffers** parameters based on the size specified. Append m or M to specify the size in megabytes (MB), or g or G to specify the size in gigabytes (GB). By default, the memory size is set to 10 MB.

numglobalbuffers=number

Use this parameter to specify the number of global buffers used. The default value is based on the size specified in the **memorysize** parameter. You can change the **memorysize** parameter to alter the number of global buffers.

old-object-queue-size=number-of-objects

Use this parameter to track the maximum number of old objects. By default, the number of objects is set to 256.

repository=path

Use this parameter to specify the repository for temporary disk storage. By default, it uses system temporary directory.

retransform={true|false}

Use this parameter to specify if event classes should be retransformed using JVMTI. If set to **false**, instrumentation is added to loaded event classes. By default, this parameter is set to **true** for enabling class retransformation.

samplethreads={true|false}

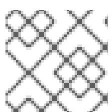
Use this parameter to specify whether thread sampling is enabled.Thread sampling only occurs when the sampling event is enabled and this parameter is set to **true**. By default, this parameter is set to **true**.

stackdepth=depth

Use this parameter to set the stack depth for stack traces. By default, the stack depth is set to 64 method calls. You can set the maximum stack depth to 2048. Values greater than 64 could create significant overhead and reduce performance.

threadbuffersize=size

Use this parameter to specify the local buffer size for a thread. By default, the local buffer size is set to 8 kilobytes, with a minimum value of 4 kilobytes. Overriding this parameter could reduce performance and is not recommended.



NOTE

You can specify values for multiple parameters by separating them with a comma.

5.2. CONFIGURING JDK FLIGHT RECORDER USING DIAGNOSTIC COMMAND (JMCD)

You can configure JDK Flight Recorder (JFR) using Java diagnostic command. The simplest way to execute a diagnostic command is to use the jcmd tool which is located in the Java installation directory. To use a command, you have to pass the process identifier of the JVM or the name of the main class, and the actual command as arguments to jcmd. You can retrieve the JVM or the name of the main class by running jcmd without arguments or by using jps.

To see a list of all running Java processes, use the jcmd command without any arguments. To see a complete list of commands available for a running Java application, specify help as the diagnostic command after the process identifier or the name of the main class.

Use the following diagnostic commands for JFR:

5.2.1. Start JFR

Use **JFR.start** diagnostic command to start a flight recording. For example:

```
jcmd <PID> JFR.start delay=10s duration=10m filename=recording.jfr
```

Table 5.1. The following table lists the parameters you can use with this command:

Parameter	Description	Data type	Default value
name	Name of the recording	String	-
settings	Server-side template	String	-
duration	Duration of recording	Time	0s
filename	Resulting recording file name	String	-
maxage	Maximum age of buffer data	Time	0s
maxsize	Maximum size of buffers in bytes	Long	0
dumponexit	Dump running recording when JVM shuts down	Boolean	-
path-to-gc-roots	Collect path to garbage collector roots	Boolean	False

5.2.2. Stop JFR

Use **JFR.stop** diagnostic command to stop running flight recordings. For example:

```
jcmd <PID> JFR.stop name=output_file
```

Table 5.2. The following table lists the parameters you can use with this command.

Parameter	Description	Data type	Default value
name	Name of the recording	String	-
filename	Copy recording data to the file	String	-

5.2.3. Check JFR

Use **JFR.check** command to show information about the recordings which are in progress. For example:

```
jcmd <PID> JFR.check
```

Table 5.3. The following table lists the parameters you can use with this command.

Parameter	Description	Data type	Default value
name	Name of the recording	String	-
filename	Copy recording data to the file	String	-
maxage	Maximum duration to dump file	Time	0s
maxsize	Maximum amount of bytes to dump	Long	0
begin	Starting time to dump data	String	-
end	Ending time to dump data	String	-
path-to-gc-roots	Collect path to garbage collector roots	Boolean	false

5.2.4. Dump JFR

Use **JFR.dump** diagnostic command to copy the content of a flight recording to a file. For example:

```
jcmd <PID> JFR.dump name=output_file filename=output.jfr
```

Table 5.4. The following table lists the parameters you can use with this command.

Parameter	Description	Data type	Default value
name	Name of the recording	String	-
filename	Copy recording data to the file	String	-
maxage	Maximum duration to dump file	Time	0s
maxsize	Maximum amount of bytes to dump	Long	0
begin	Starting time to dump data	String	-

Parameter	Description	Data type	Default value
end	Ending time to dump data	String	-
path-to-gc-roots	Collect path to garbage collector roots	Boolean	false

5.2.5. Configure JFR

Use **JFR.configure** diagnostic command to configure the flight recordings. For example:

```
jcmd <PID> JFR.configure repositorypath=/home/jfr/recordings
```

Table 5.5. The following table lists the parameters you can use with this command.

Parameter	Description	Data type	Default value
repositorypath	Path to repository	String	-
dumppath	Path to dump	String	-
stackdepth	Stack depth	Jlong	64
globalbuffercount	Number of global buffers	Jlong	32
globalbuffersize	Size of a global buffer	Jlong	524288
thread_buffer_size	Size of a thread buffer	Jlong	8192
memorysize	Overall memory size	Jlong	16777216
maxchunksize	Size of an individual disk chunk	Jlong	12582912
Samplethreads	Activate thread sampling	Boolean	true

>>>>>>> using_jfr/master.adoc

CHAPTER 6. HISTORICAL ANALYSIS

Java Mission Control (JMC) has a suite of pages and tools for displaying flight recording information. Flight recording files can be opened via the **File** menu, or automatically when a flight recording is dumped using JMC.

As of JMC 7.1.0, the **JOverflow** plugin has been refactored and included by default with JMC. This allows you to see a visual representation of heap dump **hprof** files.

Revised on 2021-04-22 13:59:42 UTC