Java Platform, Standard Edition

Java Flight Recorder Command Reference

Release 10
E92740-01
March 2018

Java Platform, Standard Edition Java Flight Recorder Command Reference, Release 10

E92740-01

Contents

Preface

This document describes command-line parameters and shell commands for Java Flight Recorder.

> **Note:**
>
> Java Flight Recorder requires a commercial license for use in production. To learn more about commercial features and how to enable them please visit `http://www.oracle.com/technetwork/java/javaseproducts/`.

Audience

This document is intended for Java developers and support engineers who use Java Flight Recorder to monitor applications and need to understand the commands and options that are available.

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 Command-Line Options

When you use the `java` command to start a Java application, you can specify options to enable Java Flight Recorder, configure its settings, and start a recording.

The following command-line options for the `java` command are specific to Java Flight Recorder:

- `-XX:+|-FlightRecorder`: Enables Java Flight Recorder (not required for JDK 8u40 and later)
- `-XX:FlightRecorderOptions`: Sets the parameters for Java Flight Recorder
- `-XX:StartFlightRecording`: Starts a recording with the provided parameters, or with the default values

The command-line options for Java Flight Recorder are described in the java chapter of the *Java Platform, Standard Edition Tools Reference*. These options are available only in the commercially licensed JDK from Oracle. To use these options, you must unlock the commercial features as described in the next section.

Unlocking Commercial Features

Use one of the following methods to access for Java Flight Recorder and related options:

- Specify the `-XX:+UnlockCommercialFeatures` option when you start your application with the `java` command.
- Use the `VM.unlock_commercial_features` parameter for the `jcmd` utility to unlock commercial features after the application is running.

Example 1-1 Unlock Commercial Features When Starting an Application

The following `java` command unlocks commercial features for an application named `MyApp` and starts a recording:

```
java -XX:+UnlockCommercialFeatures -XX:StartFlightRecording MyApp
```

Example 1-2 Unlock Commercial Features for a Running Application

The following `jcmd` command unlocks commercial features for a Java process with the process ID `1968`:

```
jcmd 1968 VM.unlock_commercial_features
```

2
Diagnostic Command Reference

Diagnostic commands provided by the `jcmd` utility are available to control Java Flight Recorder.

The `jcmd` utility is used to send diagnostic command requests to a running Java process. Information about the diagnostic commands is available by using the `help` parameter. The process ID for a running Java process is required.

To see a list of the available diagnostic commands, do not include a command name. For example, the following command lists the diagnostic commands for the process with the identifier 10568:

```
jcmd 10568 help
```

To see information about a specific diagnostic command, include the command name after the `help` parameter. For example, the following command requests information about the `JFR.start` diagnostic command for the process with the identifier 5361:

```
jcmd 5361 help JFR.start
```

The following diagnostic commands are available for Java Flight Recorder:

- JFR.configure
- JFR.start
- JFR.check
- JFR.stop
- JFR.dump
- VM.unlock_commercial_features
- VM.check_commercial_features

JFR.configure

To set the parameters for a flight recording, use the `JFR.configure` diagnostic command for the `jcmd` utility.

Table 2-1 shows the parameters for the `JFR.configure` command. All parameters are optional. If no parameters are entered, the current settings are displayed.

Table 2-1 JFR.configure Parameters

Parameter	Description	Type of Value	Default
`globalbuffercount`	Number of global buffers. Change the `memorysize` parameter to alter the number of global buffers.	Long	The default value is determined by the value for the `memorysize` parameter.

Table 2-1 (Cont.) JFR.configure Parameters

Parameter	Description	Type of Value	Default
`globalbuffersize`	Size of the global buffers, in bytes. Change the `memorysize` parameter to alter the size of the global buffers.	Long	The default value is determined by the value for the `memorysize` parameter.
`maxchunksize`	Maximum size of an individual data chunk, in bytes	Long	12582912
`memorysize`	Overall memory size, in bytes	Long	10485760
`repositorypath`	Path to the location where recordings are stored until they are written to a permanent file	String	The default location is the temporary directory for the operating system. On Oracle Solaris and Linux operating systems the temporary directory is `/tmp`. On Windows the temporary directory is specified by the `TMP` environment variable.
`stackdepth`	Stack depth for stack traces	Long	64
`thread_buffer_size`	Local buffer size for each thread, in bytes. Overriding this parameter could reduce performance and is not recommended.	Long	8192
`threadbufferstodisk`	Flag for allowing thread buffers to write directly to disk if the buffer thread is blocked	Boolean	false
`samplethreads`	Flag for activating thread sampling	Boolean	true

Example 2-1 JFR.configure Example

The following command configures Java Flight Recorder to store recordings in `D:\jfr\recordings`. The value for *pid* is the process ID of the Java process to record.

```
jcmd pid JFR.configure repositorypath=D:\jfr\recordings
```

JFR.start

To start a flight recording, use the `JFR.start` diagnostic command for the `jcmd` utility.

Table 2-2 shows the parameters for the `JFR.start` command. All parameters are optional. If no parameters are entered, a recording is started by using default values.

Table 2-2 JFR.start Parameters

Parameter	Description	Type of Value	Default
`delay`	Length of time to wait before starting to record	Integer followed by `s` for seconds, `m` for minutes, or `h` for hours	0s

Table 2-2 (Cont.) JFR.start Parameters

Parameter	Description	Type of Value	Default
`disk`	Flag for writing the data to disk while recording	Boolean	true
`dumponexit`	Flag for writing the recording to disk when the Java Virtual Machine (JVM) shuts down. If set to `true` and no value is entered for `filename`, the recording is written to a file in the directory where the process was started. The file name is a system-generated name that contains the process ID, recording ID, and current time stamp (for example,`hotspot-pid-47496-id-1-2018_01_25_19_10_41.jfr`).	Boolean	false
`duration`	Length of time to record	Integer followed by `s` for seconds, `m` for minutes, or `h` for hours	0s (forever)
`filename`	Name of the file to which the recording is written when the recording is stopped. If no path is provided, the file is in the directory where the process was started. Examples of filenames: • `recording.jfr` • `/home/user/recordings/recording.jfr` • `c:\recordings\recording.jfr`	String	No default value
`maxage`	Maximum time to keep the recorded data on disk. This parameter is valid only when the `disk` parameter is set to `true`.	Integer followed by `s` for seconds, `m` for minutes, or `h` for hours	0s (forever)
`maxsize`	Maximum size of the data to keep on disk, in bytes if one of the following suffixes is not used: • `m` or `M` for megabytes • `g` or `G` for gigabytes 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 the `JFR.configure` command.	Long	0 (no maximum size)
`name`	Name of the recording. If no name is provided, a name is generated. Make note of the generated name that is shown in the response to the command so that you can use it with other commands.	String	The default is a system-generated name.

Table 2-2 (Cont.) JFR.start Parameters

Parameter	Description	Type of Value	Default
`path-to-gc-roots`	Flag for collecting the path to garbage collection (GC) roots at the end of a recording. This flag was introduced in JDK 10. The path information is useful for finding memory leaks, but collecting it is time-consuming. Turn this flag on only when you start a recording for an application that you suspect has a memory leak. If the `settings` parameter is set to `profile`, then the information collected includes the stack trace from where the potential leaking object was allocated.	Boolean	false
`settings`	Name of the settings file that identifies the events to record. To specify more than one file, separate the names with a comma (,). Include the path if the file is not in `JRE_HOME/lib/jfr`. The following profiles are included with the JDK in the `JRE_HOME/lib/jfr` directory: • `default.jfc`: 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. • `profile.jfc`: Provides more data than the `default.jfc` profile, but with more overhead and impact on performance. Use this configuration for short periods of time when more information is needed. Examples of settings file names: • `profile` • `default` • `default.jfc` • `/home/user/settings/my-events.jfc,profile` • `c:\settings\custom.jfc`	String	`JRE_HOME/lib/jfr/default.jfc`

Example 2-2 JFR.start Examples

The following command starts a recording that runs for 1 minute, uses the settings file named `profile` to identify the events to record, and writes the recording to a file named `d:\recordings\page-load.jfr`. Because no value is provided for the `name` parameter, a system-generated name is used. The value for *pid* is the process ID of the Java process to record.

```
jcmd pid JFR.start duration=1m settings=profile filename=d:\recordings\page-load.jfr
```

The following command starts a recording named `monitor1hour` that keeps data for a maximum of 1 hour, and limits the size of the recording to 500 megabytes. The value for *pid* is the process ID of the Java process to record.

```
jcmd pid JFR.start name=monitor1hour maxage=1h maxsize=500M
```

JFR.check

To show information about a flight recording that is running, use the `JFR.check` diagnostic command for the `jcmd` utility.

Table 2-3 lists the parameters for the `JFR.check` command. All parameters are optional. If no parameters are entered, information for all active recordings is shown.

Table 2-3 JFR.check Parameters

Parameter	Description	Type of Value	Default
`name`	Name of the recording	String	No default value
`verbose`	Flag for printing the event settings for the recording	Boolean	false

Example 2-3 JFR.check Example

The following command shows the information for a recording named `Recording-1`. The value for *pid* is the process ID of the Java process being recorded.

```
jcmd pid JFR.check name=Recording-1
```

JFR.stop

To stop a flight recording, use the `JFR.stop` diagnostic command for the `jcmd` utility.

Table 2-4 shows the parameters for the `JFR.stop` command. All parameters are optional. However, if no name is entered, then no recording is stopped.

Table 2-4 JFR.stop Parameters

Parameter	Description	Type of Value	Default
`filename`	Name of the file to which the recording is written when the recording is stopped. If no path is provided, the file is in the directory where the process was started. Examples of filenames: • `recording.jfr` • `/home/user/recordings/recording.jfr` • `c:\recordings\recording.jfr`	String	No default value

Table 2-4 (Cont.) JFR.stop Parameters

Parameter	Description	Type of Value	Default
`name`	Name of the recording	String	No default value

Example 2-4 JFR.stop Example

The following command stops the recording named `debugrun1` and writes it to `debugrun1.jfr` in the directory where the process was started. The value for *pid* is the process ID of the Java process being recorded.

```
jcmd pid JFR.stop name=debugrun1 filename=debugrun1.jfr
```

JFR.dump

To write data to a file while a flight recording is running, use the `JFR.dump` diagnostic command for the `jcmd` utility.

Table 2-5 shows the parameters for the `JFR.dump` command. The `name` and `filename` parameters are required. The recording continues to run after the data is written.

Table 2-5 JFR.dump Parameters

Parameter	Description	Type of Value	Default
`filename` (required)	Name of the file to which the recording is written. If no path is provided, the file is in the directory where the process was started. Examples of filenames: • `recording.jfr` • `/home/user/recordings/recording.jfr` • `c:\recordings\recording.jfr`	String	No default value
`name` (required)	Name for the recording	String	No default value

Table 2-5 (Cont.) JFR.dump Parameters

Parameter	Description	Type of Value	Default
`path-to-gc-roots`	Flag for collecting the path to garbage collection (GC) roots at the time that the recording data is dumped. This flag was introduced in JDK 10. This information is useful for finding memory leaks, but collecting it can cause the application to halt for a short period of time. If you suspect a memory leak in an application that is running, use this flag to collect the information without having to start another recording.	Boolean	false

Example 2-5 JFR.dump Example

The following command writes a recording named `monitor1hour` to a file named `/usr/testsamples/recordings/monitor1hour-halfway.jfr`. The value for *pid* is the process ID of the Java process being recorded.

```
jcmd pid JFR.dump name=monitor1hour filename=/usr/testsamples/recordings/
monitor1hour-halfway.jfr
```

VM.unlock_commercial_features

To unlock commercial features in a Java process that is already running, use the `VM.unlock_commercial_features` diagnostic command for the `jcmd` utility.

The `VM.unlock_commercial_features` command has no parameters. The following example shows the command for unlocking commercial features. The value for *pid* is the process ID of the Java process:

```
jcmd pid VM.unlock_commercial_features
```

VM.check_commercial_features

To check if commercial features are locked or unlocked in a Java process that is already running, use the `VM.check_commercial_features` diagnostic command for the `jcmd` utility.

The `VM.check_commercial_features` command has no parameters. The following example shows the command for checking the status of commercial features. The value for *pid* is the process ID of the Java process:

```
jcmd pid VM.check_commercial_features
```