Java Platform, Standard Edition

JDK Mission Control User Guide

Release 8
F78631-01
April 2023

ORACLE

Java Platform, Standard Edition JDK Mission Control User Guide, Release 8

F78631-01

Copyright © 2001, 2023, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

This document provides an overview of JDK Mission Control (JMC). It includes information about the features, architecture, and accessibility of the product.

Audience

This document provides Java developers and support engineers with an introduction to the functionality and architecture of JDK Mission Control. It assumes that the reader has basic knowledge of the Java programming language.

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.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documents

For information about using JMC, refer to the online help included with the product. To access the online help, open the JMC application and click **Help**, then select **JDK Mission Control Help**.

For information about Flight Recorder, see the Flight Recorder Runtime Guide Java Flight Recorder Runtime Guide.

For troubleshooting JVM issues using JMC and JFR, see JDK Troubleshooting Guide.

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

What's New in JDK Mission Control 8

JDK Mission Control (JMC) is an advanced set of tools for managing, monitoring, profiling, and troubleshooting Java applications. JMC enables efficient and detailed data analysis for areas such as code performance, memory, and latency without introducing the performance overhead normally associated with profiling and monitoring tools.

New features introduced in JMC 8 releases are listed in the following sections.

> **Note:**
>
> Stack trace graph views such as Graph, Flame, Heat map, and Dependency are currently supported on Linux and Mac.

Dependency View

Dependency view presents aggregation of events using hierarchical edge bundling. For example, it helps in visualizing the dependencies between packages.

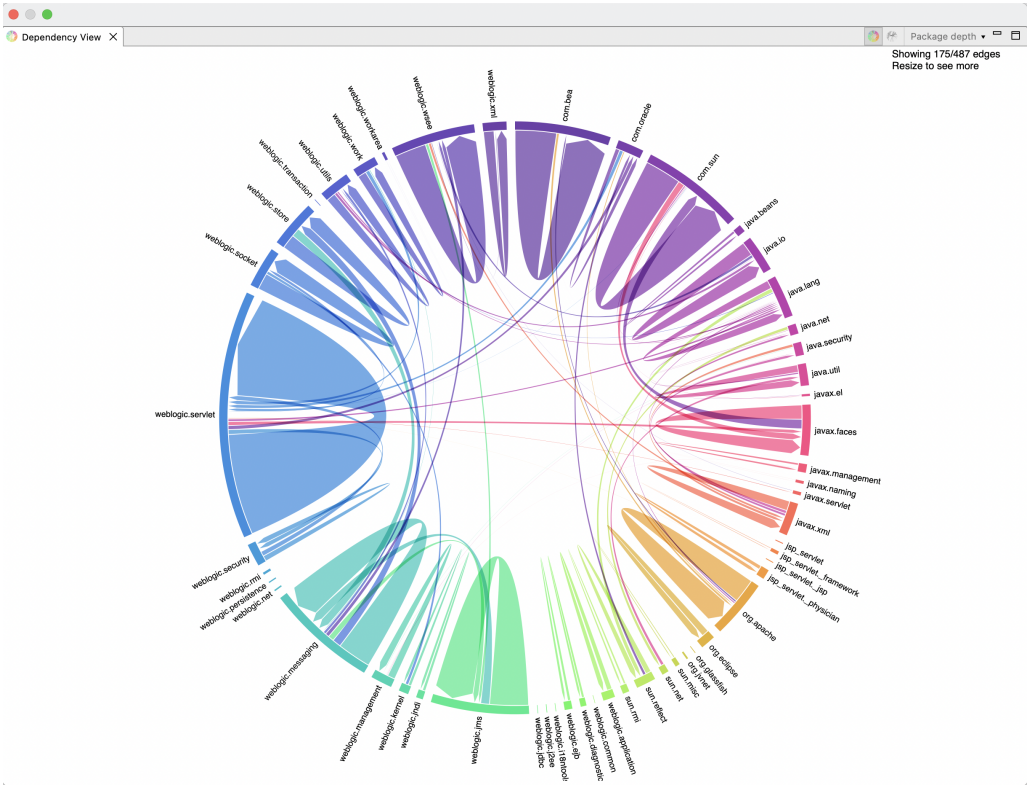
To enable dependency view in JMC, go to the menu item **Window**, **Show View**, **Other**, **Mission Control**, and then **Dependency View**. Click **Open**.

There are two modes of data representation:

Chord Diagram

Chord diagram show packages arranged radially around a circle, with arcs connecting the data points to indicate dependencies.

Figure 1-1 Chord diagram

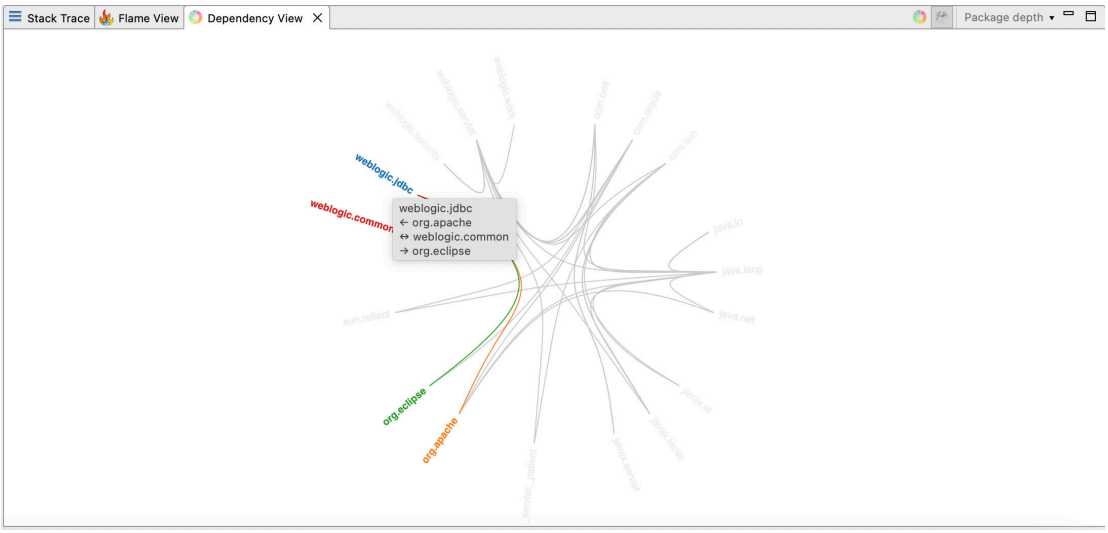


Edge Bundling

In the edge bundling visualization, hover over the packages to see dependencies highlighted in a particular color:

- Green: When you hover over a package, the methods in that package call the methods in the linked package.
- Yellow: Methods in the linked package call the methods in the hovered over package.
- Red: Methods in linked package and hovered over package call each other.

Figure 1-2 Edge bundling



Package Depth

To control the size of the graph, you can select the package hierarchy depth at a certain level. This reduces accuracy but it is easier to inspect visually.

For example, when you set the package depth as 3, the package is displayed as `java.util.zip`. However, if you set the package depth to 2, the package is displayed as `java.util`. The hierarchical depth is set according to the value selected. By default, the Package depth value is set to **2**.

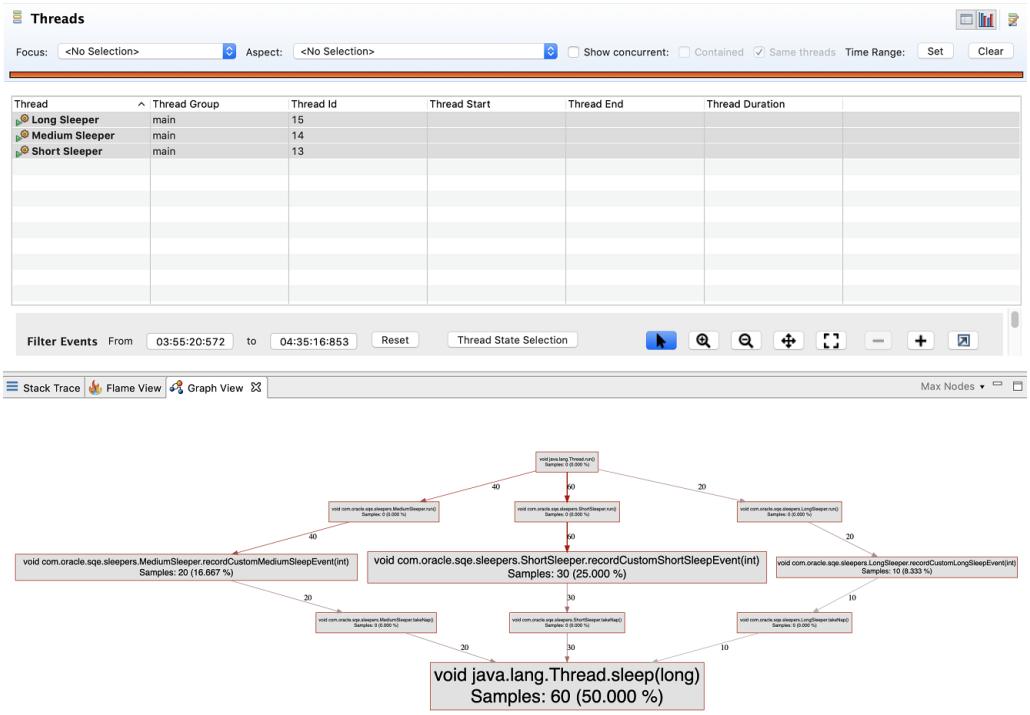
Graph View

Graph view renders aggregate stack traces with cumulative count. It presents the stack trace in a graphical format, which helps identify method path to its root.

To enable graph view in JMC, go to the menu item **Window**, **Show View**, **Other**, **Mission Control**, and then **Graph View**. Click **Open**.

Here is a sample image displaying the Threads information in the graph view:

Figure 1-3 Graph View



Rendering large graphs is also possible with smart pruning. You can focus on the most impactful nodes (nodes are data points, for example, method calls). In the **Max Nodes** drop-down, select the target number of nodes; the visualization will render maximum of that many nodes.

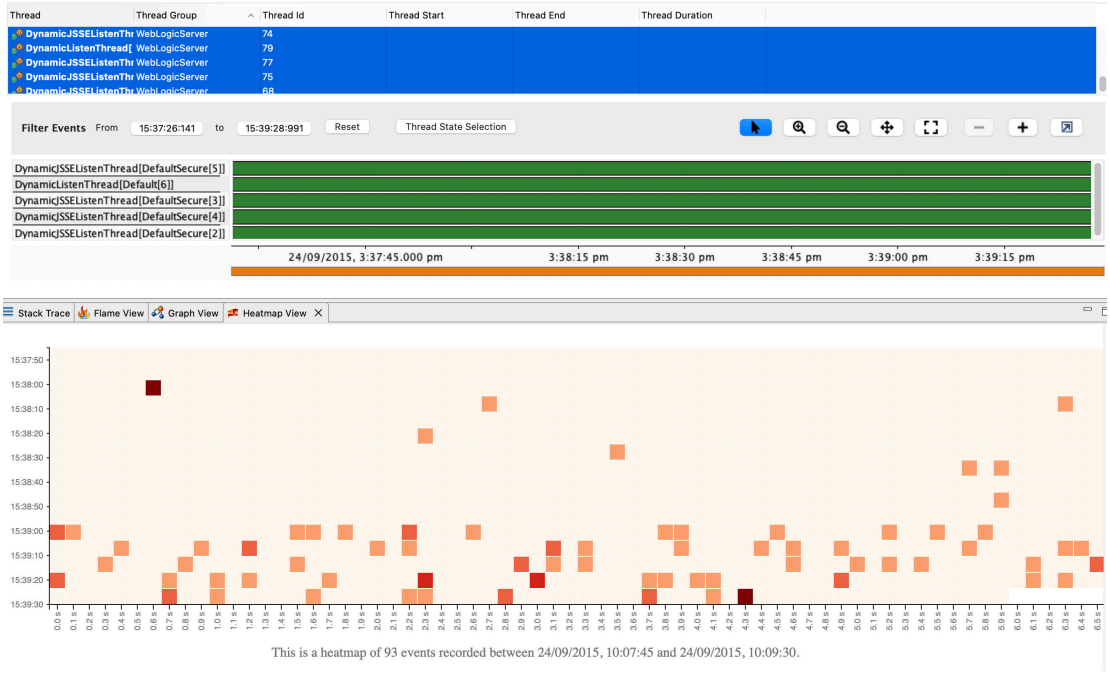
Heatmap View

Heatmap view provides a visual representation of events occurred during a specific time period within the stacktrace. The values are color coded, which helps to analyze complex data at a glance.

To enable heatmap view in JMC, go to the menu item **Window**, **Show View**, **Other**, **Mission Control**, and then **Heatmap View**. Click **Open**.

Here is a sample image of Heatmap view:

Figure 1-4 Heatmap view



Each cell represents an event that was recorded during a specific time period. Different colors indicate various events that occurred during a particular time period.

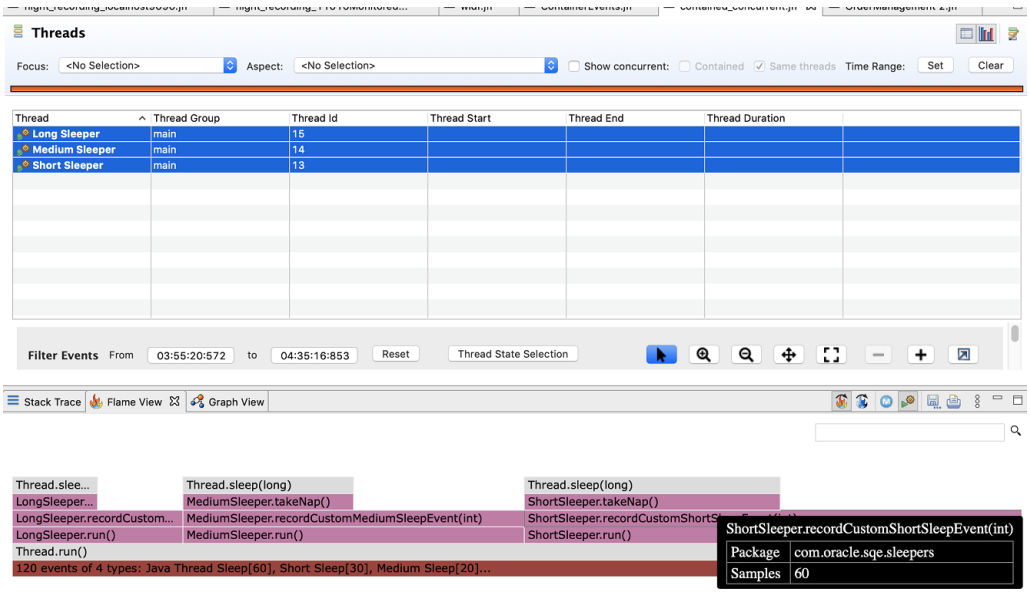
Flame Graph View

Flame Graph view renders aggregated stack trace collected by the JFR events. It helps in faster comprehension and reduced time for root cause analysis.

To enable flame graph view in JMC, go to the menu item **Window**, **Show View**, **Other**, **Mission Control**, and then **Flame View**. Click **Open**.

Here is a sample image of the flame graph view displaying the Threads information:

Figure 1-5 Flame View



A flame graph view has the following features:

- A stack trace is represented as a column of boxes, where each box represents a function (a stack frame).
- The x-axis shows the stack trace collection and the y-axis shows the stack depth.
- The width of the box indicates the frequency of function occurrences. The wider the boxes, function occurrences are more, as compared to narrow boxes.
- The search field helps you to search for a specific term, which can include regular expressions. The searched functions are highlighted. Also, the sum of matched stack traces is shown on the flame graph as a percentage of the total profile. This is useful not just for locating functions, but also for highlighting logical groups of functions.
- The orientation of the graph can be switched by selecting the Flame Graph or Icicle Graph icons. The Icicle Graph displays top-to-bottom orientation, whereas the Flame Graph displays bottom-to-top orientation.

You can aggregate the Flame graph and Stack trace by selecting the required attribute. In the **Samples** drop-down, select either **Allocation Size** or **TLAB Size**. The Flame View and/or Stack Trace view is presented according to the sample selected.

Websocket Server to Access JFR Stack Trace

When you select JFR events in JMC, you can access their associated stack trace as JSON data through a user-defined port. You can programmatically control the visualization directly in the browser. It allows you to develop or alter visualization using data visualization tools.

The websocket server port is disabled by default. To enable, click **Windows**, **Preferences**, **JDK Mission Control**, and then **Flight Recorder**. In the Flight Recorder Setting dialog, provide the port number in the **Websocket server port** field. Set **0** to disable the port.

JMC Agent

JMC agent is used to instrument or inject custom JFR events to your running JVM or applications without the need to restart the applications. You don't need to program the JFR instrumentation in the source code of the application to use JMC agent.

Here are some of the features and advantages of JMC agent:

- You can use XML configuration to define the inject events
- You can dynamically add events at runtime
- JVM agent is extremely versatile; you can dynamically load or sideload the agent anytime
- It causes minimal footprint; you can issue only event-related function calls
- It is suitable for production use when the source is not available

Using JMC agent plugin, you can manage predefined configurations and also view live information about the resulting transformations.

Click **JMC Agent** from **JVM Browser** tab to **Start JMC Agent**. The prerequisites to start the agent are:

- Build the Agent JAR: See JMC Agent Readme to to build the `agent.jar`. In the **Start JMC Agent** dialog, browse and connect to the Agent JAR file. Use the latest version of the code for building the agent.
- Define the JFR event in an xml file: You can either create an xml file manually or use **JMC Agent Preset Manager**, which is the user interface to create the XML file.

> **Note:**
>
> - If the target application that you need to instrument is running on JDK 11 or later, then run your application with the JVM argument: `--add-opens java.base/jdk.internal.misc=ALL-UNNAMED`.
> - If the target application that you need to instrument is running on JDK 8, then run your application with the JVM argument `-XX:+UnlockCommercialFeatures -XX:+FlightRecorder`.

Agent Live Config: Right-click the **JMC Agent** to open **Agent Live Config** console. You can use this page to view the Global Configurations applied to the agent, along with the Event List and Event Details.

JMC Agent Preset Manager: Preset Manager will help you to create, edit, and modify the configuration templates. Click **Windows** and select **JMC Agent Preset Manager**. You can **Add** an xml file with only required options, **Edit** the file, or **Import Files** to manage the configurations.

JMC Agent Plugin

JMC agent plugin integrates JMC agent features into JDK Mission Control. You can use the plugin to start the JMC agent and connect to a local JVM through the JMX API.

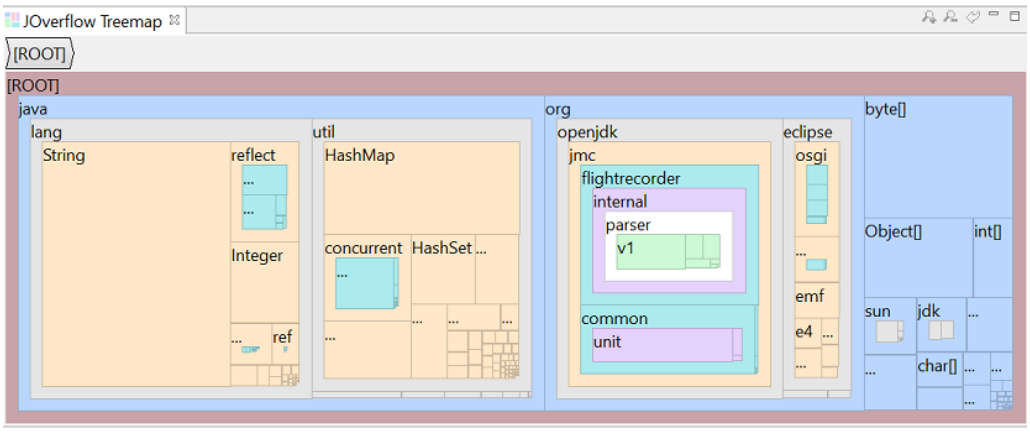
Treemap Viewer in JOverflow

Treemaps are used to visualize the memory usage by classes. It helps to identify the resources consuming most RAM or memory. Like JOverflow instance viewer, the treemap viewer utilizes the filter feature provided by JOverflow.

To enable JOverflow Treemap view in JMC, go to the menu item **Window**, **Show View**, **Other**, **JOverflow**, and then **JOverflow Treemap**. Click **Open**.

Here is a sample image of JOverflow Treemap:

Figure 1-6 JOverflow Treemap



The squarified treemap is displayed where memory usage is aggregated by class and package names. The larger area depicts more memory consumed by the class or package. The background colors indicate the treemap nodes at different levels.

Coherence Tab Pack Plug-in Converted to SWT

Coherence Tab Pack plug-in is used for viewing cluster information as well as aggregated Coherence MBeans. Coherence plug-in is now converted to Standard Widget Toolkit (SWT), which removes dependency on JDK 11 and JavaFX.

See Install Plug-ins for JMC Standalone Application for the process to install the plug-in.

The Coherence Tab Pack plug-in consists of the following tabs:

- **Cluster Overview**: Provides various metrics regarding the overall health of the cluster
- **Machines**: Provides information about all the cluster machines, including cores, load, and memory pressure
- **Members**: Provides details about the individual members
- **Services**: Provides information regarding the running services and various performance metrics

- **Caches**: Lists the individual cache and their size, memory footprint, and individual node details
- **Proxy Servers**: Displays proxy servers and connections
- **Persistence**: Provides information about active or on-demand persistence

Support for LZ4 Compressed Recordings

JMC 8 now supports working with JFR recordings that are compressed using LZ4 format. LZ4 is an extremely fast decoder, which provides compression speed of more than 500 MB/s per core and is scalable with multi-cores CPU.

You can compress the JFR recordings that are in `zip` or `gzipped` format. Use the `file` utility to know the file type. Compress the data (.jfr) file using LZ4 utility or record using `compress=true`.

Example code to compress JRF recording:

```
$ lz4 <un_compressed_filename>.jfr <compressed_filename>.jfr

Output: Compressed 8182601 bytes into 3626010 bytes ==> 44.31%
```

Improved Thread Graphs

Thread graph is enhanced with the following:

- Time filter that display the start and end time of the visible range
- Zoom in and zoom out functionality
- Zoom to automatically select the highlighted area on the chart
- Vertical and horizontal scrolling of chart canvas

Allocation Pressure Column Added to Memory and TLAB Pages

A new column is added to the **Memory** page to display the Total Allocation as a percentage value - **Total Allocation (%)**.

A new tab **By Top Methods** is added to the **TLAB Allocations** (Thread Local Allocation Buffers) page in addition to the existing **By Threads** tab to classify the Item Histograms against Top Methods. Both tabs now have **Alloc in TLABs (%)** and **Alloc Outside TLABs (%)** columns, which provides estimated allocation size of TLAB as percentage. These updates will make it easier to view relevant areas of allocation pressure.

Enabling Non-execute JDBC Events of WebLogic Plugin

The default filter for JDBC events of JDBC Operations and SQL Statements pages have been modified. By default, you can view the events generated by `JDBC Statement Execute` event type. Click **Show page filter** if you wish to view all events.

Reintroduction of Percentage Column

Percentage column is reintroduced in all histogram tables.

2
JDK Mission Control

JDK Mission Control (JMC) is a production-time profiling and diagnostics tool. It includes tools to monitor and manage your Java application with a very small performance overhead, and is suitable for monitoring applications running in production.

JMC is not part of the regular JDK installation. For more information on JMC downloads and documentation, see JDK Mission Control Page.

JMC consists of :

- *JVM Browser* shows running Java applications and their JVMs.
- *JMX Console* is a mechanism for monitoring and managing JVMs. It connects to a running JVM, collects, displays its characteristics in real time, and enables you to change some of its runtime properties through *Managed Beans* (MBeans). You can also create rules that trigger on certain events (for example, send an e-mail if the CPU usage by the application reaches 90 percent).
- *Flight Recorder* (JFR) is a tool for collecting diagnostic and profiling data about a running Java application. It is integrated into the JVM and causes very small performance overhead, so it can be used in production environments. JFR continuously saves large amounts of data about the running applications. This profiling information includes thread samples, lock profiles, and garbage collection details. JFR presents diagnostic information in logically grouped tables and charts. It enables you to select the range of time and level of detail necessary to focus on the problem. Data collected by JFR can be essential when contacting Oracle support to help diagnose issues with your Java application.
- Plug-ins help in heap dump analysis and DTrace recording. See Plug-in Details. JMC plug-ins connect to a JVM using the *Java Management Extensions* (JMX) agent. For more information about JMX, see the *Java Platform, Standard Edition Java Management Extensions Guide* .

3

Install JDK Mission Control and Supported Plug-ins

Review the prerequisites and system requirements before you install JMC. Apart from the base JMC application, there are many plug-ins available that you can install depending on your requirements.

You can install JMC as a standalone application or as an Eclipse plug-in.

System Requirements and Supported Platforms

JMC system requirements and supported platforms.

See Oracle JDK Mission Control Certified System Configurations.

Download and License Information

JDK Mission Control homepage provides access to JMC and related plug-in downloads.

Product Download

Download JMC or Eclipse IDE as per your requirement.

Standalone JMC and Plug-ins Download

Download JMC from: JDK Mission Control (JMC) 8 Downloads; ensure that you accept the license agreement.

Download Plug-ins from:

- Oracle plug-ins: https://download.oracle.com/technology/products/missioncontrol/updatesites/oracle/8.3.1/rcp/
- OpenJDK plug-ins: https://download.oracle.com/technology/products/missioncontrol/updatesites/openjdk/8.3.1/rcp/

JMC Plug-ins for the Eclipse IDE

Download Eclipse IDE from Eclipse IDE for Eclipse Committers

Download Plug-ins from:

- Oracle plug-ins: https://download.oracle.com/technology/products/missioncontrol/updatesites/oracle/8.3.1/ide/
- OpenJDK plug-ins: https://download.oracle.com/technology/products/missioncontrol/updatesites/openjdk/8.3.1/ide/

JMC Core APIs

Access JMC APIs from Documentation for JMC Core API.

You can also download the Core APIs from Maven Central. See MVN Repository to view the list of JMC bundles. You can download the JMC Core APIs by adding the dependency in the `pom.xml` file. For example, to download `org.openjdk.jmc.common` API, provide the following dependency values in `pom.xml` file:

```
<dependency>
    <groupId>org.openjdk.jmc</groupId>
    <artifactId>common</artifactId>
    <version>8.3.1</version>
</dependency>
```

License Information

See UPL License and Java SE Licensing Information User Manual for licensing information.

Installation Instructions

You can install JMC as a standalone application or as a plug-in within the Eclipse IDE.

> **Note:**
>
> Apply the latest Critical Patch Update (CPU) or Security Alert; older versions are not updated with the latest security patches.

Install JMC Standalone Application

Prerequisites

- Install JDK 11 64-bit (or later); see Download and License Information.
- Ensure that you set the *<jdk_installation_path>*`/bin` PATH environment variable and the `$TEMP` directory is accessible.
- For configuration details, see JDK Mission Control 8 Installation Instructions.

Follow these steps to install the JMC standalone application and relevant plug-ins.

1. Download the latest JMC relevant to your platform. For instance, `jmc-8.3.1_windows-x64.zip` for Windows.

   > **Note:**
   >
   > If you have multiple JDK versions installed and you want to configure a specific JDK version, see the *Specify the JDK Version to be Used by JMC* section in JDK Mission Control 8 Installation Instructions.

2. Extract the downloaded compressed file, access the `/bin` folder, and open `jmc.exe`.

   You can also start JMC from command prompt using the command:

   - Windows: *<JMC_ROOT>*`\jmc-`*<version>*`_windows-x64\bin\jmc.exe`

- Linux: `<JMC_ROOT>/jmc-<version>_linux-x64_bin/bin/jmc`
- macOS: `open <JMC_ROOT>/jmc-<version>_macos-x64/JDK\ Mission\ Control.app`

Install Plug-ins for JMC Standalone Application

The plug-ins for JMC are a set of artifacts designed to extend its functionality.

There are several plug-ins that are built for JMC. These plug-ins are not distributed with the base JMC application. They are hosted on remote locations (update sites) from where they can be download. Some plug-ins are open source while some are provided by Oracle.

The URL of the remote locations (update site) is updated in the JMC application.

Follow these steps to install the JMC plug-ins:

1. Open JMC. Go to **Help** and **Install New Software**.

   The available plug-ins are categorized into Mission Control, Mission Control (Oracle), and Mission Control (Experimental).

   > **Note:**
   >
   > - If JMC is unable to contact the update sites, then check the JMC application proxy.
   > - Go to **Windows** and **Preferences** (in macOS, go to **JDK Mission Control** and **Preferences**). In **General** and **Network Connections** window, verify the **Host** proxy setting for **HTTP**, **HTTPs**, and **Port** value.

2. In the **Available Software** window, select the required plug-ins, click **Next**. See Plug-in Details to know more about the plug-ins available for the JMC standalone application.
3. In the **Install Details** window, view the details of the selected plug-ins and click **Next**.
4. Accept the license agreements and click **Finish**.

   > **Note:**
   >
   > To use the installed plug-ins, restart JMC.

Install JMC and Associated Eclipse Plug-ins

The JDK Mission Control plug-ins for Eclipse are a set of artifacts for the Eclipse IDE designed to help develop, profile, and diagnose applications running on supported JDKs.

Prerequisites

- Install Eclipse 4.24 and later
- Uninstall previous versions of JMC, if any
- Ensure that Eclipse is running on a supported JDK version (not JRE)

Follow these steps to install the JMC as an Eclipse plug-in:

1. Open Eclipse IDE. Go to **Help** and **Install New Software**.

2. In the **Work with** field, enter the URL `https://download.oracle.com/technology/products/missioncontrol/updatesites/openjdk/8.3.1/ide/`.

   The available plug-ins are categorized into Mission Control, Mission Control (Oracle), and Mission Control (Experimental).
3. Expand all the categories, select **Base (Required)** and other required plug-ins, and click **Next**. See Plug-in Details.
4. In the **Install Details** window, view the details of the selected plug-ins and click **Next**.
5. Select **I accept the terms of the license agreement** and click **Finish**.
6. Restart Eclipse.

The JMC icon should appear in Eclipse toolbar.

Plug-in Details

You can install Oracle, Open source, and Experimental plug-ins according to your requirement.

> **Note:**
>
> Install the latest version of the plug-ins to get the most recent security patches.

- Oracle plug-ins: Developed and supported by Oracle, which extends the base JMC functionality.
- Open source plug-ins: Developed by open source community, which provides additional functionality.
- Experimental plug-ins: Developed by Oracle as a technology preview that may later be incorporated into the main distribution, depending on the feedback. These plug-ins are currently not supported and are provided only for evaluation purposes. The experimental plug-ins are not included in the JDK Mission Control product distribution.

Plug-ins for JMC Standalone Application

- Oracle plug-ins:
  - Coherence Tab Pack
  - WebLogic Pages
  - Threadlogic

  > **Note:**
  >
  > The Threadlogic plug-in is deprecated in JMC 8 and will be removed in JMC 9.
- Open source plug-ins:
  - JDK Mission Control RCP Updates

  - Flight Recorder Metadata Page
  - JavaFX Page
- Experimental plug-ins:
  - Subscription Tab
  - Twitter Action
  - G1 Page

JMC Plug-ins for Eclipse IDE

- Open Source plug-ins:
  - JDK Mission Control IDE Integration
  - Flight Recorder Metadata Page
  - JavaFX Page
- Experimental plug-ins:
  - Subscription Tab
  - Twitter Action
  - G1 Page
  - Flight Recorder Launch Configuration Tab
  - JDK Mission Control PDE Integration
- Mission Control Localization plug-ins:
  - Japanese Language Support (for IDE Integration)
  - Simplified Chinese Language Support (for IDE Integration)

Start JDK Mission Control

See JDK Mission Control 8 Installation Instructions.

4

JDK Mission Control Application GUI

When you first start the JMC application, a number of views open by default.

You can open the **Window** menu, select **Show View** and then **Other** to open a window that lists all available views. Select the necessary views and click **OK** to open them. The following views are available:

- **JVM Browser**: Lists all the JVM instances running locally (on the host) and JVMs discovered on the network. The **JVM Browser** can be viewed in two different modes: as a flat list, and as a tree (visible by default).
- **Outline**: Shows the data collected in a Flight Recording. It organizes and presents flight recording data as pages in a tree for easy navigation (visible by default).
- **Progress View**: Displays the progress of running operations, for example, a flight recording.
- **Properties**: Lists the properties of items that you select in tables, including hidden properties that are not displayed in the tables (visible by default).
- **Results**: Displays a list of rules with their corresponding scores (visible by default). It also shows the results from the automated analysis relevant to the currently opened page in the editor.
- **Stack Trace**: Displays stack traces for the recorded events (visible by default).

You can rearrange these views by dragging and dropping them to the desired location. You can also maximize and minimize the views.

5
Working with Remote JVMs

JMC allows you to connect to a remotely running Java application or JVM using:

- Java Management Extension
- Java Discovery Protocol (JDP)

Java Management Extension

Java virtual machine (JVM) uses Java Management Extensions (JMX) technology over a Remote Method Invocation (RMI) connector, known as JMXRMI to enable monitoring and management features.

Prerequisites:

- To connect to the Java application (remote JVM) from JMC, you must start the remote JVM using JMX JVM arguments such as:
  - `-Dcom.sun.management.jmxremote`
  - `-Dcom.sun.management.jmxremote.authenticate`
  - `-Dcom.sun.management.jmxremote.port`
  - `-Dcom.sun.management.jmxremote.ssl`

  See Monitoring and Management Using JMX Technology for complete list of arguments and their values.
- Ensure that there are no firewall rules on the server or client (the machine where JMC is running), which block connectivity to the JMX remote port.
- Obtain the configuration details such as port number, host name, authentication credentials, and digital certificates from your remote JVM, if any.

SSL is enabled by default when you enable remote monitoring and management. To use SSL, you need to set up a digital certificate on the system and configure SSL on the server application or JVM.
Once SSL is enabled on the remote JVM, configure JMC using the same set of digital certificates used in the remote JVM.

To set digital certificates in JMC:

1. Go to **Window**, **Preferences**, **JDK Mission Control**, and click **JMXRMI**.

   > **Note:**
   >
   > On macOS, preferences are at **JDK Mission Control**, **Settings**, **JDK Mission Control**, **JMXRMI**.

2. Browse and select the **Key Store** location.
3. Enter the **Key Store Password**.

4. Browse and select the **Trust Store** location.
5. Enter the **Trust Store Password**.
6. Click **Unlock** to provide the master password.
7. In the **Set Master Password** screen, type the password twice and click **OK**.
8. Click **Apply**.
9. Restart JMC.

Creating New Custom Connection

You can add connections to favorite JVMs so that they are always available in the JVM Browser.

To add a new custom connection:

1. Right click in the JVM Browser tab and select **New Connection**.

   Alternatively, go to **File**, **Connect**, select **Create a new connection**, and click **Next**.
2. Enter the **Host** name and **Port** number of the JVM to connect to.

   Alternatively, you can provide the Custom JMX service URL in the format:

   ```
   service:jmx:rmi:///jndi/rmi://<<remote-host>>:7091/jmxrmi
   ```
3. If client authentication is enabled in the remote JVM, specify the **User** name and **Password** for authentication. Select **Store credentials in settings file** to encrypt them using a master password.
4. Enter the **Connection name**.
5. Click **Finish** to create the connection.

   Alternatively, click **Next** and select either to **Start JMX Console** or **Start Flight Recording**.

> **Note:**
>
> If SSL is not enabled, a warning message is displayed recommending you to secure your connection using SSL configuration.

Java Discovery Protocol (JDP)

The Java Discovery Protocol (JDP) enables the JVM Browser to list JVM instances across the same network subnet.

Prerequisites:

- Connect to the Java application (remote JVM) from JMC using JDP arguments. See Java Discovery Protocol (JDP) documentation for more details.
- Ensure that there are no firewall rules on the server or client (the machine where JMC is running), which block connectivity.

To set up JDP preferences:

1. Go to **Window**, **Preferences**, **JDK Mission Control**, **JVM Browser**, and click **JDP**.
2. Select the **auto-discover** checkbox.

   Click **OK** when prompted to enable auto discovery in safe environment.
3. Enter the IP address of **Multicast group to join**. Default value is `224.0.23.178`.
4. Enter the **Multicast port** number. Default value is `7095`.
5. Enter the **Heart Beat Timeout** in seconds. Default is `12`.
6. Restart JMC for the changes to take effect.

> **Note:**
>
> You can modify the default values as required.

JMC will identify the JDP enabled JVMs, running on the configured network, and let you monitor and record JFRs.
See Troubleshooting Tricks for solutions to commonly occurring issues.

6
Real-time JMX Monitoring

Java Management console (JMX) connects to a running JVM and collects and displays key characteristics in real time. It is a tool for monitoring and managing a running JVM instance. The tool presents live data about memory and CPU usage, garbage collections, thread activity, and so on. It also includes a fully featured JMX MBean browser that you can use to monitor and manage MBeans in the JVM and in your Java application.

Overview Tab

To start the JMX Console, right-click the required JVM in **JVM Browser** and select **Start JMX Console**.

By default, the JMX Console displays the **Overview** tab with the **JMX Data Persistence Settings** and **Dashboard** panels. It also displays the **Processor** and **Memory** charts.

The default dials on the **Dashboard** show information about memory utilization, CPU usage in the JVM, and live set and fragmentation.

The **Overview** tab allows you to add, remove, and edit dials and charts.

Adding or Editing a Dial
Click **Add attributes**, the plus (+) button in the top right corner of the **Dashboard** panel. For example, when the dialog opens, in the **Filter:** text field, enter `FreePhysicalMemorySize` and click **Finish**. A Free Physical Memory dial is added to the **Dashboard**. To edit a dial's name and its properties, right-click the dial. You can set values for the dial gradients, set the colors, and select whether you want to show the watermark.

Adding or Editing a Chart
Click **Add Chart**, the plus (+) button next to the **Overview** tab. Once the chart is added, click the **Add attributes**, (+) button in the top right corner of the charts panel to add the desired attribute. For example, when the dialog opens, in the **Filter** text field, enter `ThreadCount` and click **Finish**. A Threadcount chart is added to the page. You can right-click the attribute list and select **Edit Color** to change the color of the Threadcount chart. To perform other actions on the chart, such as changing its titles or exporting it as an image, use the context menu of the chart.

> **Note:**
>
> To reset the **Overview** tab, click the **Reset to default controls** button in the top right corner of the page.

JMX Data Persistence

You can use JMX attributes to gather persistent data and store it for analysis. JMC persists JMX data to files that you can open and view in the GUI.

Enable data persistence

The JMX Data Persistence Settings panel is collapsed at the top of the **Overview** tab. To enable data persistence for the attributes in the list, click the **Activate JMX Data Persistence** button in the top right corner of the **Overview** tab.

Add more attributes to the settings
To add an attribute to the list, click the **Add attributes** button in the top right corner of the JMX Data Persistence Settings panel. Select an attribute from the **Select Attributes to Add** dialog and click **Finish**.

> **Note:**
>
> To remove an attribute from the list, right-click the list and select **Delete**.

View the JMX data persistence file
The JMX Data Persistence data is stored in binary files in the persistence directory. By default, this directory is located in *`USER-HOME`*`/.jmc/7.x.x/persisted_jmx_data`. The persistence directory contains folders corresponding to the names of the JVM connections for which data persistence was enabled. Each JVM connection folder contains subfolders with names of attributes for which data persistence was enabled. These subfolders contain the log files named `*.persisted_jmx_data` and additional metadata files named `series.info`. You can drag and drop the log files to the JMX Data Persistence window to view them.

> **Note:**
>
> You can change the persistence directory.
>
> - For Windows OS, open the JMC application, select **Windows**, then **Preferences**, then **JDK Mission Control**, then **JMX Console**, and then select **JMX Data Persistence**. Browse to select the desired directory and apply the changes.
> - For macOS, open the JMC application, select **JDK Mission Control**, then **Preferences**, then **JDK Mission Control**, then **JMX Console**, and then select **JMX Data Persistence**. Browse to select the desired directory and apply the changes.

MBean Browser Tab

The **MBean Browser** tab allows you to monitor and manage MBeans deployed in the JMX server inside the JVM and in your Java application.

The browser provides access to all registered MBeans. An MBean can represent a device, an application, or any resource that needs to be managed. To make it easier for you to view large collection of attributes, the browser will automatically group values into subgroups. You can control this grouping by changing the preferences. For more information on MBean Preferences, see JDK Mission Control Help.

You can use the MBean browser to look at specific values of attributes, change the update intervals for attributes, add attributes to charts, and view notifications.

Create and Register a New Mbean

To create and register a new MBean, click the plus **(+)** button at the top of the **MBean Tree** panel and specify a valid object name and a class name for the new MBean in the **Dynamically Create and Register a New MBean** dialog. To unregister a particular MBean, right-click and select **Unregister** from the **MBean Tree** panel context menu.

Update Interval for an Attribute

To update the interval of an attribute in the **Attributes** table, right-click an attribute and select **Change Update Interval**. The update interval can be set to one of these following values:

- Default: The default update interval setting.
- Once: The attribute will be fetched only once. For example, you can select this interval for name of the operating system.
- Custom: A custom update intervals specified in milliseconds (ms). For example, you can change the **CpuLoad** attribute to 2000 ms. In this case, the CPU load will be fetched once every two seconds.

Change the Value of an Attribute

You can update the value of an attribute only if it is rendered in *bold font*. For example, select **MemoryPool** and then **Compresses Class Space** MBean and go to the **Attributes** subtab. From the list, right-click the **UsageThreshold** attribute (note that this attribute is in bold font). You can either double-click the **Value** field or right-click and select **Change Value** to change the threshold value.

Visualize an Attribute

To visualize an attribute as a chart in the **Overview** tab follow these steps:

1. Select any attribute from the **MBean Tree** panel. For example, from **java.lang** select **Threading**.
2. In the **MBean Features** panel, right-click **ThreadCount** and select **Visualize**.
3. In the **Create Chart** dialog, click **Add Chart** and enter a new name for the chart.
4. Click **OK**.
   The ThreadCount chart is added to the **Overview** tab.

> **Note:**
>
> If an attribute has a boolean value, you cannot visualize it as a chart.

View Notifications

The **Notifications** subtab enables you to view JMX notifications available on a selected MBean. Not all MBeans provide such information, and you have to subscribe to the notifications (if they are available). For example, select the **GarbageCollector** MBean under **java.lang** and then select **G1 Old Generation**. Go to the **Notifications** subtab and select the **Subscribe** check box. Notifications are added to the **Log** panel as a separate entry to the table with the name, which comprises of the date and time of the notification. Expand the entries to view the details of the notifications.

Invoke Diagnostic Commands

You can use the **MBean Browser** tab to invoke diagnostic commands. For example, click the **Operations** subtab and then select **Diagnostic Command** from the **com.sun.management**

domain. From the list of commands, select the **vmInfo : String** command and click **Execute**. The result displays in an output panel.

Triggers Tab

The **Triggers** tab enables you to define and activate rules that trigger events when a certain condition is met.

The rules consists of the following components:

- Condition: This specifies when to activate a trigger. For example, activate a trigger when the CPU Load exceeds 90 percent.
- Action: This specifies what action to do when the condition is met. For example, send an email with the details or start a flight recording.
- Constraints: Additional constraints to the trigger condition. For example, send trigger alerts only during weekdays.

The **Triggers** tab allows you to add, remove, rename, activate, and deactivate rules. You can export and import the rules. The **Trigger Rules** panel contains some predefined rules. You can make modification to these existing rules in the **Rules Details** panel.

Set a Trigger

Follow these instructions to set up JMC to automatically start a flight recording if a condition is met. This is useful for tracking specific JVM runtime issues.

1. In the **Triggers** tab, click the **Add** button. You can choose any MBean in the application, including your own application-specific ones.
2. In the **Add New Rule** dialog that opens, Select an attribute for which the rule should trigger and click **Next**. For example, select **java.lang**, then **OperatingSystem**, and then **ProcessCpuLoad**.
3. Set the condition on which the rule should trigger and click **Next**. For example, set a value for the **Maximum trigger value**, **Sustained period**, and **Limit period**.

> **Note:**
>
> You can select **Trigger when condition is met** and **Trigger when recovering from condition** check boxes.

4. Select what action you would like your rule to perform when triggered and click **Next**. For example, choose **Start Time Limited Flight Recording** and browse the file destination and recording time. Select the **Open automatically** checkbox if you wish to open the flight recording automatically when it is triggered.
5. Select constraints for your rule and click **Next**. For example, select particular dates, days of the week, or the time of day when the rule should be active.
6. Enter a name for your rule and click **Finish**.
   The rule is added to the **My Rules** list in the **Triggers Rules** panel.

When you select your rule from the **Trigger Rules** list, the **Rule Details** pane displays its components in the **Conditions**, **Attributes**, and **Constraints** tabs. You can edit them if you wish.

System Tab

The **System** tab provides you with information about the resources of the system on which the JVM is running, performance attributes of the JVM, and a list of system properties.

The **Server Information** panel contains a list of categories with values for the server on which the JVM is running. This information is useful for debugging development and runtime problems of the application, and for filing support requests. You cannot change this general system information.

The **JVM Statistics** panel contains the current values for key performance attributes of the JVM. By default, the following attributes are displayed in the table:

- **Currently Loaded Class Count**
- **Uptime**

To add an attribute to the **JVM Statistics** table, click the **Add attributes** button in the top right corner of the **JVM Statistics** panel. To modify an attribute, right-click an attribute and perform operations such as, delete, update interval, and for some attributes, change the value.

The **System Properties** pane contains a table with keys and system properties of the JVM. This panel has a search feature, which enables you to filter system properties either by key or value. For example, to show the properties that start with *java.vm*, enter `java.vm` in the filter text box.

Memory Tab

The **Memory** tab enables you to monitor how efficiently your application is using memory resources. This tab focuses on heap usage, garbage collection, and active memory pools. The information provided on this tab helps you determine whether you have configured the JVM to provide optimal application performance.

The **Memory** tab helps you to quickly narrow down a memory leak. A memory leak occurs when an application unintentionally holds references to objects in the heap, preventing them from being garbage collected. These unintentionally held objects can grow in the heap over time, eventually filling up the entire Java heap space, causing frequent garbage collections and ultimately the program termination with `OutOfMemoryError`.

You can manually initiate a full garbage collection using the **Run a full garbage collection** button in the top right corner of the tab and then analyze the heap dump.

Heap Histogram

To display a snapshot of memory allocation on the heap per class, click the **Refresh Heap Histogram** button. Note that a warning message appears to let you know that this can cause some overhead. The output shows the instance count, total size, and delta for each class type in the heap. When the first heap histogram is captured, that will be the baseline, and the delta will be set to zero. Every subsequent histogram captured will show the delta to the previous histogram. If the **Reset delta** calculation is pressed, the currently captured snapshot will be used as baseline. If a sequence of histograms is obtained (for example, every two minutes), then you might be able to observe a trend that can lead to further analysis.

When an application experiences a `java.lang.OutOfMemoryError`, analyze the heap histogram to diagnose the problem. It will indicate what objects were in the memory and what size of memory they were occupying when the `java.lang.OutOfMemoryError` occurred.

GC Tables
The **GC Tables** panel contains the current values for key performance attributes of available garbage collectors, such as G1 Young Generation and G1 Old Generation. The collection count can help you to analyze various issues. For example, a high number of young collections could be the cause of a response-time problem. If the utilization of the old generation fluctuates greatly without rising after garbage collection (GC), then objects are being copied unnecessarily from the young generation to the old generation.

Right-click an attribute to update interval, units, and for some attributes, change value.

Active Memory Pools
Memory shortage is the main reason for increased GC activity. Therefore, it is important to monitor utilization of the different memory pools (Eden, Survivor, and Old). You can achieve this by analyzing the **Active Memory Pools** panel, which displays information about memory pools available to the JVM.

Threads Tab

The **Threads** tab contains information about running threads in your application.

Live Thread Graphs

Click **Live Thread Graph** to view the number of threads started by the Java application. The graph contains the following attributes, by default:

- Daemon Live Thread Count: Displays the number of live daemon threads.
- Peak Live Thread Count: Displays the number of peak live threads.
- Total Live Thread Count: Displays the total number of live threads.

Add More Attributes to the Graph.

To add more attributes to the **Live Thread** Graph, click the **Add attributes** button. From the **Select attributes to add** dialog, add the required attributes. For example, add **DeadlockedThreadCount** and click **Finish**.

Live Threads Table
All available threads are listed in the **Live Threads** table, with information such as thread name, thread state, and blocked count. To filter the threads in the table, enter the filter string in the filter text box. For example, enter `AWT` to view all the AWT threads.

Monitor Threads in real time
To monitor the threads in real time, select the **CPU Profiling**, **Deadlock Detection**, and **Allocation** check boxes. All these check boxes are disabled by default as enabling them consumes a lot of system resources.

Show or Hide Columns in the Table
To add more columns, right-click a thread name and select **Visible Columns**. From the list, select the columns you want to show or hide.

Stack Trace

If a thread is selected, its stack trace will appear in the **Stack Traces for Selected Threads** panel. The stack trace contains the call path for all methods up to the one that is currently executed to help you detect the method that caused an issue.

Diagnostic Commands

The **Diagnostic Commands** tab lists the commands that you can send to a running JVM.

The commands enable you to monitor the efficiency and performance of your Java application. They also help you to obtain information about the performance statistics, Java Flight Recorder (JFR), memory usage, garbage collection, thread stacking, and JVM runtime of the target Java application.

You can find the diagnostic commands in the **Operations** list and configure their parameters to the right. To run it, select a command from the **Operations** list, enter its argument value or values, and click the **Execute** button. The result displays in an output panel.

To see the detailed description of a command, select the command in the **Operations** list and click the **Help** button.

The diagnostic commands are classified according to the performance overhead they cause when executed:

- Standard Commands: These commands are denoted by an icon with the letter **i** in a circle. They provide general diagnostic information. For example, `JFR.dump`.
- Advance Commands: These commands are denoted by an icon with an exclamation mark (!) in a triangle. They require more resources, can impact JVM performance, but provide more information. For example, `JFR.start`.
- Internal Commands: These commands are denoted by an icon with an exclamation mark (!) in an upside-down triangle. These commands provide very detailed diagnostic information, but greatly impact the JVM performance. Their impact depends on the size and contents of the Java heap. For example, `GC.heap_dump`.

When you execute a diagnostic command with medium, high, or unknown impact, a warning message is displayed. You can control this setting through **Preferences**. Deselect the corresponding check boxes if you do not want warnings displayed for diagnostic commands with a specific impact.

7
Flight Recorder

Flight Recorder (JFR) is a profiling and event collection framework built into the JDK.

Flight Recorder allows Java administrators and developers to gather detailed low-level information about how a JVM and Java applications are behaving. You can use JMC, with a plug-in, to visualize the data collected by JFR. Flight Recorder and JMC together create a complete toolchain to continuously collect low-level and detailed runtime information enabling after-the-fact incident analysis.

The advantages of using JFR are:

- It records data about JVM events. You can record events at a particular instance of time.
- Recording events with JFR enables you to preserve the execution states to analyze issues. You can access the data anytime to better understand problems and resolve them.
- JFR can record a large amount of data on production systems while keeping the overhead of the recording process low.
- It is most suited for recording latencies. It records situations where the application is not executing as expected and provide details on the bottlenecks.
- It provides insight into how programs interact with execution environment as a whole, ranging from hardware, operating systems, JVM, JDK, and the Java application environment.

Flight recordings can be started when the application is started or while the application is running. The data is recorded as time-stamped data points called *events*. Events are categorized as follows:

- Duration events: occurs at a particular duration with specific start time and stop time.
- Instant events: occurs instantly and gets logged immediately, for example, a thread gets blocked.
- Sample events: occurs at regular intervals to check the overall health of the system, for example, printing heap diagnostics every minute.
- Custom events: user defined events created using JMC or APIs.

In addition, there are predefined events that are enabled in a *recording template*. Some templates only save very basic events and have virtually no impact on performance. Other templates may come with slight performance overhead and may also trigger garbage collections to gather additional data. The following templates are provided with Flight Recorder in the `<JMC_ROOT>/lib/jfr` directory:

- `default.jfc`: Collects a predefined set of data with low overhead.
- `profile.jfc`: Provides more data than the `default.jfc` template, but with more overhead and impact on performance.

Flight Recorder produces following types of recordings:

- Time fixed recordings: A time fixed recording is also known as a profiling recording that runs for a set amount of time, and then stops. Usually, a time fixed recording has more

events enabled and may have a slightly bigger performance effect. Events that are turned on can be modified according to your requirements. Time fixed recordings will be automatically dumped and opened.

Typical use cases for a time fixed recording are as follows:

- Profile which methods are run the most and where most objects are created.
- Look for classes that use more and more heap, which indicates a memory leak.
- Look for bottlenecks due to synchronization and many more such use cases.

- Continuous recordings: A continuous recording is a recording that is always on and saves, for example, the last six hours of data. During this recording, JFR collects events and writes data to the global buffer. When the global buffer fills up, the oldest data is discarded. The data currently in the buffer is written to the specified file whenever you request a dump, or if the dump is triggered by a rule.

  A continuous recording with the default template has low overhead and gathers a lot of useful data. However, this template doesn't gather heap statistics or allocation profiling.

Start a Flight Recording

Follow these steps to start a flight recording using JMC.

1. Find your JVM in the **JVM Browser**.
2. Right-click the JVM and select **Start Flight Recording...**

   The **Start Flight Recording** window opens.
3. Click **Browse** to find a suitable location and file name to save the recording.
4. Select either **Time fixed recording** (profiling recording), or **Continuous recording**. For continuous recordings, you can specify the maximum size or maximum age of events you want to save.
5. Select the flight recording template in the **Event settings** drop-down list. Templates define the events that you want to record. To create your own templates, click **Template Manager**. However, for most use cases, select either the Continuous template (for very low overhead recordings) or the Profiling template (for more data and slightly more overhead).
6. Click **Finish** to start the recording or click **Next** to modify the event options defined in the selected template.
7. Modify the event options for the flight recording. The default settings provide a good balance between data and performance. You can change these settings based on your requirement.

   For example:

   - The **Threshold** value is the length of event recording. By default, synchronization events above 10 ms are collected. This means, if a thread waits for a lock for more than 10 ms, an event is saved. You can lower this value to get more detailed data for short contentions.
   - The **Thread Dump** setting gives you an option to perform periodic thread dumps. These are normal textual thread dumps.

8. Click **Finish** to start the recording or click **Next** to modify the event details defined in the selected template.
9. Modify the event details for the selected flight recording template. Event details define whether the event should be included in the recording. For some events, you can also define whether a stack trace should be attached to the event, specify the duration threshold (for duration events) and a request period (for requestable events).
10. Click **Back** if you want to modify any of the settings set in the previous steps or click **Finish** to start the recording.

    The new flight recording appears in the **Progress View**.

> **Note:**
>
> Expand the node in the JVM Browser to view the recordings that are running. Right-click any of the recordings to dump, dump whole, dump last part, edit, stop, or close the recording. Stopping a profiling recording will still produce a recording file and closing a profiling recording will discard the recording.

> **Note:**
>
> You can set up JMC to automatically start a flight recording if a condition is met using the **Triggers** tab in the JMX console. For more information, see Triggers Tab.

Save Current Buffers into a Flight Recording

JFR saves the recorded data to files with the `.jfr` extension. These JFR recordings are binary files for viewing in the JMC. You can manually dump the current contents of the global buffer to a recording file.

1. Right-click a continuous recording in the JVM Browser and then select **Dump**.

   The **Dump Recording** dialog opens.
2. Click the **Browse** button and select the path and file name for the recording.
3. Select to dump the whole recording, only the last part of the recording, or a specified interval of recording.
4. Click **Finish** to create the recording dump file.

Analyze a Flight Recording Using JMC

Once the flight recording file opens in the JMC, you can look at a number of different areas like code, memory, threads, locks and I/O and analyze various aspects of runtime behavior of your application.

The recording file is automatically opened in the JMC when a timed recording finishes or when a dump of a running recording is created. You can also open any recording file by double-clicking it or by opening it through the **File** menu. The flight recording opens in the **Automated Analysis Results** page. This page helps you to diagnose issues quicker. For example, if you're tuning the garbage collection, or tracking down memory allocation issues, then you can use the memory view to get a detailed view on individual garbage collection

events, allocation sites, garbage collection pauses, and so on. You can visualize the latency profile of your application by looking at **I/O** and **Threads** views, and even drill down into a view representing individual events in the recording.

View Automated Analysis Results Page

The Flight Recorder extracts and analyzes the data from the recordings and then displays color-coded report logs on the **Automated Analysis Results** page.

By default, results with yellow and red scores are displayed to draw your attention to potential problems. If you want to view all results in the report, click the **Show OK Results** button (a tick mark) on the top-right side of the page. Similarly, to view the results as a table, click the **Table** button.

The benchmarks are mainly divided into problems related to the following:

- Java Application
- JVM Internals
- Environment

Clicking on a heading in the report, for example, **Java Application**, displays a corresponding page.

> **Note:**
>
> You can select a respective entry in the **Outline** view to navigate between the pages of the automated analysis.

Analyze the Java Application

Java Application dashboard displays the overall health of the Java application.

Concentrate on the parameters having yellow and red scores. The dashboard provides exact references to the problematic situations. Navigate to the specific page to analyze the data and fix the issue.

Threads

The **Threads** page provides a snapshot of all the threads that belong to the Java application. It reveals information about an application's thread activity that can help you diagnose problems and optimize application and JVM performance.

Threads are represented in a table and each row has an associated graph. Graphs can help you to identify the problematic execution patterns. The state of each thread is presented as a **Stack Trace**, which provides contextual information of where you can instantly view the problem area. For example, you can easily locate the occurrence of a deadlock.

Lock Instances

Lock instances provides further details on threads specifying the lock information, that is, if the thread is trying to take a lock or waiting for a notification on a lock. If a thread has taken any lock, the details are shown in the stack trace.

Memory

One way to detect problems with application performance to is to see how it uses memory during runtime.

In the **Memory** page, the graph represents heap memory usage of the Java application. Each cycle consists of a Java heap growth phase that represents the period of heap memory allocations, followed by a short drop that represents garbage collection, and then the cycle starts over. The important inference from the graph is that the memory allocations are short-lived as garbage collector pushes down the heap to the start position at each cycle.

Select the **Garbage Collection** check box to see the garbage collection pause time in the graph. It indicates that the garbage collector stopped the application during the pause time to do its work. Long pause times lead to poor application performance, which needs to be addressed.

Method Profiling

Method Profiling page enables you to see how often a specific method is run and for how long it takes to run a method. The bottlenecks are determined by identifying the methods that take a lot of time to execute.

As profiling generates a lot of data, it is not turned on by default. Start a new recording and select **Profiling - on server** in the **Event settings** drop-down menu. Do a time fixed recording for a short duration. JFR dumps the recording to the file name specified. Open the **Method Profiling** page in JMC to see the top allocations. Top packages and classes are displayed. Verify the details in the stack trace. Inspect the code to verify if the memory allocation is concentrated on a particular object. JFR points to the particular line number where the problem persists.

JVM Internals

The **JVM Internals** page provides detailed information about the JVM and its behavior.

One of the most important parameters to observe is **Garbage Collections**. Garbage collection is a process of deleting unused objects so that the space can be used for allocation of new objects. The **Garbage Collections** page helps you to better understand the system behavior and garbage collection performance during runtime.

The graphs shows the heap usage as compared to the pause times and how it varies during the specified period. The page also lists all the garbage collection events that occurred during the recording. Observe the longest pause times against the heap. The pause time indicates that garbage collections are taking longer during application processing. It implies that garbage collections are freeing less space on the heap. This situation can lead to memory leaks.

For effective memory management, see the **Compilations** page, which provides details on code compilation along with duration. In large applications, you may have many compiled methods, and memory can be exhausted, resulting in performance issues.

Environment

The **Environment** page provides information about the environment in which the recording was made. It helps to understand the CPU usage, memory, and operating system that is being used.

See the **Processes** page to understand concurrent processes running and the competing CPU usage of these processes. The application performance will be affected if many processes use CPU and other system resources.

Check the **Event Browser** page to see the statistics of all the event types. It helps you to focus on the bottlenecks and take appropriate action to improve application performance.

You can create Custom Pages using the **Event Browser** page. Select the required event type from **Event Type Tree** and click the **Create a new page using the select event type** button in the top right corner of the page. The custom page is listed as a new event page below the event browser page.

8

Accessibility in JDK Mission Control

Oracle is dedicated to providing high quality information technology that is accessible to people with disabilities.

To this end, Oracle has undertaken a substantial project to ensure the accessibility of JMC. Accessibility is a core feature of JMC. Please report any accessibility issues to https://docs.oracle.com/javase/feedback.html or My Oracle Support (for Oracle customers).

Screen Readers

Oracle supports screen readers that translate screen-based information into spoken words to assist vision-impaired users.

For instance, configuration options are currently available for the JAWS screen reader produced by Freedom Scientific, Inc. For information on configuring this product, see Freedom Scientific screen reader website.

> **Note:**
>
> If you are using JAWS, be aware that switching tabs does not work as expected. For a workaround, see Switching Between Tabs.

JDK Mission Control Accessibility Mode

JMC displays performance data dials and graphs that provide a valuable graphical view of the data that can reveal trends and help identify minimum and maximum values for performance metrics.

However, dials and graphs do not convey information in a manner that can be read by a screen reader. JMC provides a way to view dial and graph data in tabular format, which is more accessible for users with visual impairment. To toggle the accessibility mode, click the **Accessibility mode** button in the top right corner of the relevant panel with dials or graphs.

To enable accessibility mode for all panels in JMC, follow these steps:

1. Open the **Window** menu and select **Preferences**.
2. Select **JDK Mission Control**.
3. Under **Accessibility Options**, select **Use accessibility mode**.
4. Click **Apply**.

Show Text Labels on Buttons

By default, JMC displays buttons with graphics but no text labels.

To show the text labels that can be read by screen readers, follow these steps:

1. Open the **Window** menu and select **Preferences**.
2. Select **JDK Mission Control** in the left pane.
3. Under **Accessibility Options**, select **Show text labels on buttons**.
4. Click **Apply**.

Restart any running plug-ins to see the text labels on buttons.

Resize Online Help Text

Vision-impaired users might find it difficult to read the online help documents in the standalone rich client platform (RCP) version of JMC unless the text size is increased. You need an external browser to resize the text.

To change the font size, view the help in an external browser as described in the following steps:

1. Open the **Window** menu and select **Preferences**.
2. Select **Help** in the left pane.
3. Under **Open Modes**, select to open help contents in an external browser.
4. Click **Apply**.

JMC will use the default web browser specified in the operating system, and you cannot specify a different web browser from within JMC. How to specify the default web browser depends on the operating system.

Accessibility Known Issues and Workarounds

This section contains additional instructions for enhancing your experience with JMC's accessibility features.

Navigating in a Tree Table with Only One Row

When navigating in a tree table component containing only one row, press the Space key or the Shift key and use the Up and Down keys to get to the row.

Switching Between Tabs

When reaching a tab component, JAWS erroneously tells the user “to switch pages, press Ctrl+Tab.” The correct way to switch between tabs is to use the Left or Right keys.

Reading Table Data with a Screen Reader

To read table data more efficiently with screen reading software, copy and paste the table data into a text editor and read it from there. To copy the data, select the rows that you are interested in, right-click those rows and select **Copy**. Then paste the data into a text editor.

9
Troubleshooting Tricks

This topic lists some known issues and ways to troubleshoot them.

JVM

- JDK Mission Control fails to find any local JVMs:
  To resolve the issue, consider the following for Windows platform:
  - Ensure that you start the JMC client from `<JMC_ROOT>\bin` directory.
  - Set the JDK 8 (and later) system path (`<jdk_installation_path>\bin`) in the PATH environment variable.
  - Ensure that you have access permission to the PATH, while opening `jmc.exe`. If you don't have access, **Run as Administrator**.
  - If you are running JMC from Eclipse, then ensure that Eclipse is running on a JDK (not JRE).
  - Ensure that there is a directory named `hsperfdata_username` in the system's `tmp` directory that it is writable by you running JMC and that the file system supports access control lists (ACLs).

  > **Note:**
  >
  > For information about Linux and macOS configurations, see JDK Mission Control 8 Installation Instructions.

- Unable to connect to JVM:
  Verify the following:
  - Are you using the correct protocol?

    Ensure that the version of the JVM that you want to monitor and the JVM running the JMC application are the same.

    The format of the service URL is:

    ```
    service:jmx:rmi:///jndi/rmi://<hostname>/jmxrmi
    ```
  - Are the correct ports opened?

    Running JMX over RMI requires two ports and that one of the ports will not be known beforehand.
  - Is the communication blocked by a firewall?

    For more information, see JDK Mission Control Communication.
- When attempting to connect to a JVM, stack trace appears indicating that the JVM attempts to communicate with a strange IP or host name:
  Sometimes RMI can have a problem determining which address to use. This can happen due to any of the following scenarios:
  - There are access restrictions in the Security manager.

  - The machine is multihomed and RMI is selecting the wrong interface.
  - There is a misconfigured `hosts` file or a number of different network related configuration problems.

    If none of the above scenarios work, you can try setting the `java.rmi.server.hostname` system property. Note that this can affect applications running on the JVM.

Starting JMC

- Class not found exception when you start JMC:
  To resolve this issue, ensure that you start JMC from the following locations:
  - Windows: `<JMC_ROOT>\bin`
  - Linux: `<JMC_ROOT>/bin`
  - macOS: `<JMC_ROOT>/JDK Mission Control.app`
- If JMC eclipse plugin is started with JDK 16 or higher, then it may fail to detect locally running JVMs:
  To resolve this issue, add `sun.jvmstat.monitor` from `jdk.internal.jvmstat` to `All module` in the `eclipse.ini` file. You can also launch eclipse with the following command:

```
.\eclipse.exe -vmargs --add-exports=jdk.internal.jvmstat/
sun.jvmstat.monitor=ALL-UNNAMED"
--add-exports=jdk.internal.jvmstat/sun.jvmstat.monitor=ALL-UNNAMED
```

Best Practices

Here are some tips or best practices you can use to avoid issues:

- After setting the preference options like Keystore, Email and so on, restart JMC for the configurations to take effect.
- If you want to clear the locally persisted configurations, data, and so on, delete `<user-home>/.jmc` file and restart JMC. The log file is also available in the same location.

A

Security Compliance for JDK Mission Control

Follow these security recommendations to improve the processes of administering and managing JDK Mission Control.

- Use the latest security baseline JDK version for starting JMC.
- Enable Java Discovery Protocol (JDP) only in a secure environment: JDP is a protocol that enables technologies, in particular, JDK Mission Control and JDK Flight Recorder, to discover manageable JVMs across the same network subnet. It enables the JVM browser to list all JVM instances on the network.
  You can enable the preferences for JDP protocol using the **auto-discover** option in JMC. This option is disabled by default. To enable the option, go to **Windows**, **Preferences**, **JDK Mission Control**, **JVM Browser**, and then **JDP**. Click **auto-discover**.
- Use SSL for JMXRMI connections: While connecting to JMX remotely, ensure that you connect to JVMs having **SSL** and **Authentication** options enabled. This will ensure server side and client side security.
  Provide the Key store and Trust store credentials. To set the values, go to **Windows**, **Preferences**, **JDK Mission Control**, and then **JMXRMI**.