OpenJDK 17

Release notes for OpenJDK 17.0.1

Last Updated: 2021-11-24

Legal Notice

Abstract

This document provides an overview of new features in OpenJDK 17, and a list of potential known issues and possible workarounds.

Table of Contents

PREFACE

OpenJDK (Open Java Development Kit) is a free and open source implementation of the Java Platform, Standard Edition (Java SE).

Packages for the Red Hat build of OpenJDK are made available on Red Hat Enterprise Linux and Microsoft Windows and shipped as a JDK and JRE in the Red Hat Ecosystem Catalog.

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. SUPPORT POLICY FOR OPENJDK

Red Hat supports select major versions of OpenJDK in its products. For consistency, these versions remain similar to Oracle JDK versions that are designated as long-term support (LTS).

Red Hat supports a major version of OpenJDK for a minimum of six years from the time Red Hat first introduces OpenJDK.

OpenJDK 17 is supported on Microsoft Windows and Red Hat Enterprise Linux until November 2027.

NOTE

RHEL 6 has reached the end of life in November 2020. OpenJDK 17 is not supported on RHEL 6.

Additional resources

For more information, see the OpenJDK Life Cycle and Support Policy .

CHAPTER 2. DIFFERENCES FROM UPSTREAM OPENJDK 17

OpenJDK in Red Hat Enterprise Linux contains a number of structural changes from the upstream distribution of OpenJDK. The Windows version of OpenJDK attempts to follow Red Hat Enterprise Linux updates as closely as possible.

The following list details the most notable Red Hat OpenJDK 17 changes:

- FIPS support. Red Hat OpenJDK 17 automatically detects whether the RHEL system is in FIPS mode and automatically configures OpenJDK 17 to operate in that mode. This change does not apply to OpenJDK builds for Microsoft Windows.
- Cryptographic policy support. Red Hat OpenJDK 17 obtains the list of enabled cryptographic algorithms and key size constraints, which are used by for the TLS, a certificate path validation, and signed JARs, from the Red Hat Enterprise Linux system configuration. You can set different security profiles to balance safety and compatibility. This change does not apply to OpenJDK builds for Microsoft Windows.
- Red Hat Enterprise Linux dynamically links against native libraries such as **zlib** for archive format support and **libjpeg-turbo**, **libpng**, and **giflib** for image support. RHEL also dynamically links against **Harfzbuzz** and **Freetype** for font rendering and management.
- The **src.zip** file includes the source for all of the JAR libraries shipped with OpenJDK.
- Red Hat Enterprise Linux uses system-wide timezone data files as a source for timezone information.
- Red Hat Enterprise Linux uses system-wide CA certificates.
- Microsoft Windows includes the latest available timezone data from Red Hat Enterprise Linux.
- Microsoft Windows uses the latest available CA certificate from Red Hat Enterprise Linux.

Additional resources

- For more information about detecting if a system is in FIPS mode, see the Improve system FIPS detection example on the Red Hat RHEL Planning Jira web page.
- For more information about cryptographic policies, see Using system-wide cryptographic policies in the Red Hat Enterprise Linux *Security hardening* guide.

CHAPTER 3. DEPRECATED AND UNSUPPORTED OPENJDK CAPABILITIES

The OpenJDK upstream project removed support for some technologies due to low community interest and better alternative solutions.

Ensure that you review the following deprecated and unsupported features before you install OpenJDK 17.

Applet API

OpenJDK 17 has deprecated the Applet API, because common web browsers do not support the Java plug-in for the Applet API. OpenJDK 17 also deprecates classes and API elements related to the Applet API.

For more information about the deprecated Applet API, see JEP 398: Deprecate the Applet API for Removal.

Concurrent mark-and-sweep (CMS) garbage collector (GC)

OpenJDK 17 has removed support for the CMS GC. CMS was deprecated from a previous release of OpenJDK.

For more information about removal of the CMS GC, see JEP 363: Remove the Concurrent Mark Sweep (CMS) Garbage Collector.

Deprecate and disable biased locking

OpenJDK 17 deprecates biased locking and all of its command-line options.

For more information about deprecated biased locking feature, see JEP 374: Deprecate and Disable Biased Locking.

Graal JIT compiler and Java AOT compiler

OpenJDK 17 removed the Graal just-in-time (JIT) compiler, so the following modules were removed from OpenJDK 17:

- **jaotc**
- **jdk.aot**
- **jdk.internal.vm.compiler**
- **jdk.internal.vm.compiler.management**

You can continue to use the GrallVM for ahead-of-time (AOT) compilation tasks, and Mandrel for integrating GraalVM, OpenJDK and RHEL capabilities.

Additionally, OpenJDK 17 removes the Java ahead-of-time (AOT) compiler, also known as the **jaotc** tool.

You can continue to use the experimental Java-level JVM compiler interface (JVMCI) for JIT compilation tasks related to externally-built versions of the compiler.

For more information about removal of the AOT and JIT Compilers, see JEP 410: Remove the Experimental AOT and JIT Compiler.

PACK200 tools and APIs

OpenJDK 17 removed support for the PACK200 tools and the PACK200 APIs, which were both deprecated from a previous OpenJDK release.

The removed PACK200 tools are as follows:

- **pack200**
- **unpack200**

The removed PACK200 APIs are as follows:

- **java.util.jar.Pack200**
- **java.util.jar.Pack200.Packer**
- **java.util.jar.Pack200.Unpacker**

These tools and APIs were removed because of improved technological download speeds and reduced browser support for Pack200 plug-ins.

For OpenJDK compatible applications, you can shrink application JARs by using either the **jlink** tool or the **jpackage** tool.

For more information about removal of PACK200 tools and APIs, see JEP 367: Remove the Pack200 Tools and API.

For more information about the **jlink** tool, see the OpenJDK 11 Using **jlink** to customize Java runtime environment guide.

Parallel Scavenge and Serial Old GC algorithms

OpenJDK 17 has deprecated the combination of the Parallel Scavenge and Serial Old garbage collector (GC) algorithms.

For more information about this GC algorithm combination, see JEP 366: Deprecate the ParallelScavenge + SerialOld GC Combination.

Nashorn JavaScript engine

OpenJDK 17 has removed support for the Nashorn JavaScript engine, its APIs, and the **jjs** tool.

For more information about removal of the Nashorn JavaScript engine, see JEP 372: Remove the Nashorn JavaScript Engine.

Remote Method Invocation (RMI)

OpenJDK 17 has removed the (RMI) activation mechanism and its **java.rmi.activation** API package. You can continue to use any other RMI features with OpenJDK 17.

For more information about removal of the RMI activation mechanism, see JEP 407: Remove RMI Activation.

Security Manager

OpenJDK 17 has deprecated the Security Manager feature, which includes its classes and its methods, due to low community interest.

For more information about deprecating security manager, see JEP 411: Deprecate the Security Manager for Removal.

Solaris and SPARC Ports

OpenJDK 17 has removed the source code and build support for the Solaris/SPARC, Solaris/x64, and Linux/SPARC ports.

For more information about removal of Solaris and SPARC Ports, see JEP 381: Remove the Solaris and SPARC Ports.

Value-based class warnings

OpenJDK 17 designates the primitive wrapper classes as value-based and deprecates their constructors. This creates new deprecation warnings that prompt you about improper attempts to synchronize on value-based classes in the Java Platform.

For more information about value-based class warnings, see JEP 390: Warnings for Value-Based Classes.

CHAPTER 4. NEW FEATURES FOR OPENJDK

OpenJDK 17 includes new features that enhance the use of your Java applications.

OpenJDK 17 includes the following new features:

Context-specific deserialization filters

OpenJDK 17 provides the capability for Java programs to configure context-specific and dynamically-selected deserialization filters by using a JVM-wide filter factory. A Java program invokes this filter factory to select a filter for each deserialization operation.

For more information about context-specific deserialization filters, see JEP 415: Context-Specific Deserialization Filters.

Edwards-Curve Digital Signature Algorithm (EdDSA)

OpenJDK 17 supports using an EdDSA algorithm to implement cryptographic signatures.

For more information about EdDSA, see JEP 339: Edwards-Curve Digital Signature Algorithm (EdDSA) .

File mapping modes

OpenJDK 17 includes JDK-specific file mapping modes to the **FileChannel** API, so that you can create a **MappedByteBuffer** instance that maps to non-volatile memory (NVM).

For more information about the JDK-specific file mapping modes, see JEP 352: Non-Volatile Mapped Byte Buffers.

Foreign Linker API (Incubator feature)

OpenJDK 17 introduces the Foreign Linker API. This API provides Java programs with the following capabilities:

- Statically-typed Java access to native code.
- Simplified binding to a native library.

For more information about the Foreign Linker API , see JEP 389: Foreign Linker API .

Foreign-Memory Access API (Incubator feature)

OpenJDK 17 introduces the Foreign-Memory Access API that Java programs can safely and efficiently use to access foreign memory that exists outside of the Java heap.

For more information about the Foreign-Memory Access API, see JEP 393: Foreign-Memory Access API.

Foreign Function and Memory API (Incubator feature)

OpenJDK 17 introduces the Foreign Function and Memory API. Java programs can use this API to interact with code and data outside of the Java runtime.

By efficiently invoking foreign functions, such as code outside the JVM, and by safely accessing foreign memory, such as memory not managed by the JVM, the API provides the following capabilities:

- Enable Java programs to call native libraries.
- Process native data without experiencing any common Java Native Interface (JNI) issues.

For more information about the Foreign Function and Memory API, see JEP 412: Foreign Function & Memory API.

Hidden classes

OpenJDK 17 supports hidden classes. Bytecode in other classes cannot use such hidden classes.

A framework can generate these classes at run time and use them indirectly through a process known as *reflection*. A hidden class can be defined as a member of an access control nest.

For more information about the hidden classes, see JEP 371: Hidden Classes.

HotSpot class-metadata memory

OpenJDK 17 promptly returns unused HotSpot class-metadata memory, such as metaspace, to your operating system. This reduces high off-heap memory usage and simplifies the amount of metaspace code.

For more information about HotSpot class-metadata memory, see JEP 387: Elastic Metaspace.

Java package

OpenJDK 17 includes the new package **java.lang.invoke.constant**. This package includes an API that you can use to model nominal descriptions for class files and run time artifacts, such as constants that are loadable from the constant pool.

For more information about the **java.lang.invoke.constant** package, see JEP 334: JVM Constants API.

JDK Flight Recorder (JFR) data

OpenJDK 17 provides JFR data in a data stream format that you can use to continuously monitor your application's performance.

For more information about JFR data streams, see JEP 349: JFR Event Streaming.

jpackage tool

OpenJDK 17 includes the **jpackage** tool that you can use to package self-contained Java applications.

For more information about the **jpackage** tool, see JEP 392: Packaging Tool.

Pattern matching for the instanceof operator

The **instanceof** operator supports pattern matching. Pattern matching supports common logic in a program. The **instanceof** operator can use pattern matching to conditionally extract components from objects in a concise and safe manner.

For more information about pattern matching, see JEP 394: Pattern Matching for **instanceof**.

Pseudo-random number generators

OpenJDK 17 includes additional interface types and implementations for pseudo-random number generators (PRNGs). Java programs can use any of these interface types to reduce duplicate code in existing PRNGs or reuse PRNG algorithms in other applications. These interface types are listed as follows:

- **SplittableRandomGenerator**
- **JumpableRandomGenerator**
- **LeapableRandomGenerator**
- **ArbitrarilyJumpableRandomGenerator**

For more information about new interface types, see JEP 356: Enhanced Pseudo-Random Number Generators.

Record classes
You can use record classes to enhance your Java code. Record classes provide a compact syntax for declaring classes, because record classes act as transparent classes for immutable data.

For more information about record classes, see JEP 395: Records.

Sealed classes
You can use sealed classes and their interfaces to enhance your Java code. Sealed classes and their interfaces restrict which other classes or interfaces can extend or implement them.

For more information about sealed classes, see JEP 409: Sealed Classes.

Text blocks
OpenJDK 17 includes text blocks. A text block is a multi-line string literal that provides the following functionality:

- Formats a string automatically and predictably.
- Provides a user with the option to format a string.
- Reduces the need for escape sequencing in Java code.

For more information about text blocks see JEP 378: Text Blocks.

Vector API (Incubator feature)
OpenJDK 17 provides an initial iteration of an incubator module, **jdk.incubator.vector**, to express vector computations that reliably compile at run time.

Vector computations provide optimal vector hardware instructions on supported CPU architectures, so that these architectures perform better when compared to architectures with scalar computations.

For more information about vector APIs, see JEP 414: Vector API.

Unix domain socket channels
OpenJDK 17 adds support for the Unix-domain, **AF_UNIX**, socket to the socket channel and server-socket channel APIs in the **java.nio.channels** package. This extends the inherited channel mechanism to support Unix-domain socket channels and server socket channels.

For more information about UNIX domain socket channels see JEP 380: Unix-Domain Socket Channels.

Z Garbage Collector (ZGC)
OpenJDK 17 includes the ZGC as a product feature that you can use as a low-latency collector. ZGC in OpenJDK 17 moves thread-stack processing from safepoints to a concurrent phase, which is similar to that of the Shenandoah GC.

Additionally, OpenJDK 17 provides the following enhancements to the ZGC as Technology Preview features:

- Port the ZGC to the Windows operating system. See JEP 365: ZGC on Windows.
- Return unused committed memory automatically from heap memory to an idle operating system. See JEP 351: ZGC: Uncommit Unused Memory.

IMPORTANT

Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

For more information about the ZGC, see JEP 377: ZGC: A Scalable Low-Latency Garbage Collector .

For more information about concurrent thread-stack processing, see JEP 376: ZGC: Concurrent Thread-Stack Processing.

CHAPTER 5. OPENJDK ENHANCEMENTS

OpenJDK 17 provides enhancements to features originally created in previous releases of OpenJDK.

Dynamic Class Data Sharing (CDS) archives

OpenJDK 17 extends application class-data sharing that enables dynamic archiving of classes at the end of Java program execution. The archived classes now include all loaded application classes and library classes. These classes were not available in the default base-layer CDS archive.

For more information about Dynamic CDS archives, see JEP 350: Dynamic CDS Archives .

Floating-point operations

OpenJDK 17 sets floating-point operations to be consistently strict with the **strictfp** semantic, which is a change from the traditional default setting of **strictfp** semantics and the subtly different default floating-point semantics.

The **strictfp** semantic restores the original floating-point semantics to the language and virtual machine (VM) that match the semantics introduced before strict and default floating-point modes in Java SE 1.2.

For more information about floating-point operations, see JEP 306: Restore Always-Strict Floating-Point Semantics.

G1 garbage collector (GC)

OpenJDK 17 enhances the G1 GC with the following capabilities:

- Abort mixed collections if these collections exceed the set paused time.
- Implement Non-Uniform Memory Access (NUMA) aware memory allocation.
- Return unused committed memory automatically from heap memory to an idle operating system.

For more information about aborting mixed collections, see JEP 344: Abortable Mixed Collections for G1.

For more information about NUMA-aware memory allocation for the G1 GC, JEP 345: NUMA-Aware Memory Allocation for G1.

For more information about returning unused committed memory from the G1 GC, see JEP 346: Promptly Return Unused Committed Memory from G1.

Legacy DatagramSocket API

OpenJDK 17 replaces the underlying implementations of the **java.net.DatagramSocket** and **java.net.MulticastSocket** APIs with a new implementation. The new implementation includes the following enhancements:

- Better compatibility with virtual threads.
- Easier maintenance and debug offerings.
- Simpler implementation.

For more information about the Legacy DatagramSocket API, see JEP 373: Reimplement the Legacy DatagramSocket API.

For more information about virtual threads, see Loom - Fibers, Continuations and Tail-Calls for the JVM.

Legacy Socket API

OpenJDK 17 replaces the implementation used by the **java.net.Socket** and **java.net.ServerSocket** APIs with a new implementation. The new implementation includes the following enhancements:

- Better compatibility with user-mode threads, such as fibers.
- Easier maintenance and debug offerings.
- Simpler implementation.

For more information about the reintroduction of the Legacy Socket API, see JEP 353: Reimplement the Legacy Socket API.

For more information about fibers, see Loom - Fibers, Continuations and Tail-Calls for the JVM .

Helpful NullPointerExceptions

OpenJDK 17 improves the use of **NullPointerException** exceptions that were generated by the Java Virtual Machine (JVM) by precisely describing where the **null** value occurred when the exception was raised.

For more information about improvements to **NullPointerException** exceptions, see JEP 358: Helpful NullPointerExceptions.

Shenandoah garbage collector

OpenJDK 17 improves the Shenandoah garbage collector (GC). Shenandoah can now reduce GC pause times by running concurrently with Java threads.

With OpenJDK 17, the Shenandoah GC can achieve sub-millisecond pause times. Pause times with Shenandoah are independent of the Java heap size.

For more information about the Shenandoah GC, see JEP 379: Shenandoah: A Low-Pause-Time Garbage Collector.

Strongly encapsulate JDK internals

OpenJDK 17 provides a stronger encapsulation mechanism for internal elements of JDK by default, except critical internal APIs, such as **sun.misc.Unsafe**.

In previous releases of OpenJDK, you could reduce the level of encapsulation for all internal JDK elements, such as critical and non-critical elements. You cannot reduce this level of encapsulation for internal elements in OpenJDK 17.

For more information about strongly encapsulating internal elements of JDK, see JEP 403: Strongly Encapsulate JDK Internals.

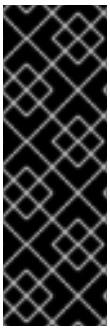
Switch expressions

OpenJDK 17 extends the capability of the **switch** statement, so you can use it either in statement form or in expression form. This capability simplifies coding and provides some pattern matching capabilities with the **switch** expression.

Both forms support **traditional** and **simplified** scoping and control flow behavior. Additionally, both forms can use either the traditional **case… :** labels or the new **case…** → labels, with new statements providing a value from a **switch** expression.

Additionally, OpenJDK 17 supports extending pattern matching to **switch**, so that you can test an

expression against a number of patterns, each with a specific action, so that you can perform complex data-oriented queries concisely and safely. For OpenJDK 17, this feature is available as Technology Preview only.



IMPORTANT

For OpenJDK 17, extending pattern matching to **switch** is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

For more information about **switch** expression capabilities, see JEP 361: Switch Expressions .

For more information about Java-related technology preview features, see JEP 12: Preview Features .

CHAPTER 6. ADVISORIES RELATED TO THIS RELEASE

The following advisories have been issued to bugfixes and CVE fixes included in this release:

- RHSA-2021:4135-01
- RHEA-2021:4136-04
- RHSA-2021:4532-01
- RHSA-2021:4531-01

Revised on 2021-11-24 12:28:15 UTC