

ORACLE

ProductsIndustriesResourcesCustomersPartnersDevelopersCompany


🔍

👤

View Accounts

🗨️

Contact Sales



JDK 20 Release Notes

All JDK Release Notes

Java Development Kit 20 Release Notes

JDK 20 Release Notes

The Java Platform, Standard Edition 20 Development Kit (JDK 20) is a feature release of the Java SE platform. It contains new features and enhancements in many functional areas. The Release Notes below describe the important changes, enhancements, removed APIs and features, deprecated APIs and features, and other information about JDK 20 and Java SE 20. Links to other sources of information about JDK 20 are also provided below.

Note: The Release Notes files are located only on our website.

- [JDK 20 ReadMe](#)
- [JDK 20 Guides and Reference Documentation](#)
- [JDK 20 Specifications](#)
- [Oracle JDK 20 Certified System Configurations](#)
- [JDK 20 Supported Locales](#)
- [Submitting a Bug Report and Available Support Options](#)
- [Copyright and License Terms for Documentation](#)

The following sections are included in these Release Notes:

Jump to category:

▼

Java™ SE Development Kit 20

These notes describe important changes, enhancements, removed APIs and features, deprecated APIs and features, and other information about JDK 20 and Java SE 20. In some cases, the descriptions provide links to additional detailed information about an issue or a change. This page does not duplicate the descriptions provided by the [Java SE 20 \(JSR 395\) Platform Specification](#), which provides informative background for all specification changes and might also include the identification of removed or deprecated APIs and features not described here. The Java SE 20 (JSR 395) specification provides links to:

- **Annex 1:** The complete [Java SE 20 API Specification](#).
- **Annex 2:** An [annotated API specification](#) showing the exact differences between Java SE 17 and Java SE 20. Informative background for these changes may be found in the list of approved Change Specification Requests for this release.
- **Annex 3:** Java SE 20 Editions of [The Java Language Specification](#) and [The Java Virtual Machine Specification](#). The Java SE 20 Editions contain all corrections and clarifications made since the Java SE 17 Editions, as well as additions for new features.

You should be aware of the content in the Java SE 20 (JSR 395) specification as well as the items described in this page.

The descriptions on this Release Notes page also identify potential compatibility issues that you might encounter when migrating to JDK 20. The [Kinds of Compatibility](#) page on the OpenJDK wiki identifies the following three types of potential compatibility issues for Java programs that might be used in these release notes:

- **Source:** Source compatibility preserves the ability to compile existing source code without error.
- **Binary:** Binary compatibility is defined in The Java Language Specification as preserving the ability to link existing class files without error.
- **Behavioral:** Behavioral compatibility includes the semantics of the code that is executed at runtime.

See [CSRs Approved for JDK 20](#) for the list of CSRs closed in JDK 20 and the [Compatibility & Specification Review \(CSR\)](#) page on the OpenJDK wiki for general information about compatibility.

The full version string for this release is build 20+36 (where "+" means "build"). The version number is 20.

IANA Data 2022g

JDK 20 contains IANA time zone data version 2022g. For more information, refer to [Timezone Data Versions in Java Runtimes](#).

[TOP](#)

Major New Functionality

1. Language Feature Previews

[JEP 432](#) Record Patterns (Second Preview)

Enhance the Java programming language with *record patterns* to deconstruct record values. Record patterns and type patterns can be nested to enable a powerful, declarative, and composable form of data navigation and processing. This is a [preview language feature](#).

[JEP 433](#) Pattern Matching for switch (Fourth Preview)

Enhance the Java programming language with pattern matching for `switch` expressions and statements. Extending pattern matching to `switch` allows an expression to be tested against a number of patterns, each with a specific action, so that complex data-oriented queries can be expressed concisely and safely. This is a [preview language feature](#).

2. Libraries Preview

[JEP 434](#) Foreign Function & Memory API (Second Preview)

Introduce an API by which Java programs can interoperate with code and data outside of the Java runtime. By efficiently invoking foreign functions (i.e., code outside the JVM), and by safely accessing foreign memory (i.e., memory not managed by the JVM), the API enables Java programs to call native libraries and process native data without the brittleness and danger of JNI. This is a [preview API](#).

[JEP 438](#) Vector API (Fifth Incubator)

Introduce an API to express vector computations that reliably compile at runtime to optimal vector instructions on supported CPU architectures, thus achieving performance superior to equivalent scalar computations.

3. Concurrency Model Preview and Incubators

[JEP 429](#) Scoped Values (Incubator)

Introduce *scoped values*, which enable the sharing of immutable data within and across threads. They are preferred to thread-local variables, especially when using large numbers of virtual threads. This is an [incubating API](#).

JEP 436 Virtual Threads (Second Preview)

Introduce *virtual threads* to the Java Platform. Virtual threads are lightweight threads that dramatically reduce the effort of writing, maintaining, and observing high-throughput concurrent applications. This is a [preview API](#).

JEP 437 Structured Concurrency (Second Incubator)

Simplify multithreaded programming by introducing an API for *structured concurrency*. Structured concurrency treats multiple tasks running in different threads as a single unit of work, thereby streamlining error handling and cancellation, improving reliability, and enhancing observability. This is an [incubating API](#).

[TOP](#)

New Features

This section describes some of the enhancements in Java SE 20 and JDK 20. In some cases, the descriptions provide links to additional detailed information about an issue or a change. The APIs described here are provided with the Oracle JDK. It includes a complete implementation of the Java SE 20 Platform and additional Java APIs to support developing, debugging, and monitoring Java applications. Another source of information about important enhancements and new features in Java SE 20 and JDK 20 is the [Java SE 20 \(JSR 395\)](#) Platform Specification, which documents the changes to the specification made between Java SE 17 and Java SE 20. This document includes descriptions of those new features and enhancements that are also changes to the specification. The descriptions also identify potential compatibility issues that you might encounter when migrating to JDK 20.

core-libs/java.lang

➔ **Support Unicode 15.0 (JDK-8284842)**

This release upgrades the Unicode version to 15.0, which includes updated versions of the Unicode Character Database, Unicode Standard Annexes #9, #15, and #29: The `java.lang.Character` class supports Unicode Character Database, which adds 4,489 characters, for a total of 149,186 characters. These additions include 2 new scripts, for a total of 161 scripts, as well as 20 new emoji characters, and 4,193 CJK (Chinese, Japanese, and Korean) ideographs. The `java.text.Bidi` and `java.text.Normalizer` classes support Unicode Standard Annexes, #9 and #15, respectively. The `java.util.regex` package supports Extended Grapheme Clusters based on the Unicode Standard Annex #29. For more detail about Unicode 15.0, refer to the [Unicode Consortium’s release note](#).

hotspot/gc

➔ **Add GarbageCollectorMXBean for Remark and Cleanup Pause Time in G1 (JDK-8297247)**

A new `GarbageCollectorMXBean` named "G1 Concurrent GC" has been added to the G1 garbage collector.

This `GarbageCollectorMXBean` reports the occurrence and durations of the Remark and Cleanup garbage collection pauses.

Similar to the "CGC" field from `jstat -gcutil`, a complete concurrent mark cycle will increase the bean's collection counter by 2, one for the Remark and one for the Cleanup pauses. These pauses now also update the "G1 Old Gen" `MemoryManagerMXBean` memory pool.

security-libs/java.security

➔ **New JFR Event: `jdk.InitialSecurityProperty` (JDK-8292177)**

A new Java Flight Recorder (JFR) event has been added to record details of initial security properties when loaded via the `java.security.Security` class.

The new event name is `jdk.InitialSecurityProperty` and contains the following fields:

Field name	Field Description
<code>key</code>	Security Property Key

Field name	Field Description
value	Corresponding Security Property Value

This new JFR event is enabled by default. The `java.security.debug=properties` system property will also now print initial security properties to the standard error stream. With this new event and the already available 'jdk.SecurityPropertyModification' event (when enabled since it is not enabled by default), a JFR recording can now monitor the initial settings of all security properties and any subsequent changes.

security-lib/java.security

➔ **New JFR Event: jdk.SecurityProviderService** ([JDK-8254711](#))

A new Java Flight Recorder (JFR) event has been added to record details of `java.security.Provider.getService(String type, String algorithm)` calls.

The new event name is `jdk.SecurityProviderService` and contains the following fields:

Field name	Field Description
type	Type of Service
algorithm	Algorithm Name
provider	Security Provider

This event is disabled by default and can be enabled via the JFR configuration files or via standard JFR options.

security-lib/javax.crypto

➔ **Provide Poly1305 Intrinsic on x86_64 platforms with AVX512 instructions** ([JDK-8288047](#))

This feature delivers optimized intrinsics using AVX512 instructions on x86_64 platforms for the Poly1305 Message Authentication Code algorithm of the SunJCE provider. This optimization is enabled by default on supporting x86_64 platforms, but may be disabled by providing the `-XX:+UnlockDiagnosticVMOptions -XX:-UsePoly1305Intrinsics` command-line options.

security-lib/javax.crypto

➔ **Provide ChaCha20 Intrinsics on x86_64 and aarch64 Platforms** ([JDK-8247645](#))

This feature delivers optimized intrinsic implementations for the ChaCha20 cipher supplied by the SunJCE provider. These optimized routines are designed for x86_64 chipsets that support the AVX, AVX2 and/or AVX512 instruction sets, and aarch64 chips supporting the Advanced SIMD instruction set. These intrinsics are enabled by default on supporting platforms, but may be disabled by providing the `-XX:-UseChaCha20Intrinsics` command-line option to Java. Flags that control intrinsics require the option `-XX:+UnlockDiagnosticVMOptions`.

tools/javac

➔ **Javac Warns about Type Casts in Compound Assignments with Possible Lossy Conversions** ([JDK-8244681](#))

New lint option `lossy-conversions` has been added to `javac` to warn about type casts in compound assignments with possible lossy conversions. If the type of the right-hand operand of a compound assignment is not assignment compatible with the type of the variable, a cast is implied and possible lossy conversion may occur.

The new warnings can be suppressed using `@SuppressWarnings("lossy-conversions")`.

security-lib/javax.net.ssl

➔ **(D)TLS Key Exchange Named Groups** ([JDK-8281236](#))

New Java SE APIs, `javax.net.ssl.SSLParameters.getNamedGroups()` and `javax.net.ssl.SSLParameters.setNamedGroups()`, have been added to allow applications to customize the named groups of key exchange algorithms used in individual TLS or DTLS connections.

The underlying provider may define the default named groups for each TLS or DTLS connection. Applications may also use the existing `jdk.tls.namedGroups` system property to customize the provider-specific default named groups. If not `null`, the named groups passed to the `setNamedGroups()` method will override the default named groups for the specified TLS or DTLS connections.

Note that a provider may not have been updated to support the new APIs and in that case may ignore the named groups that are set. The JDK SunJSSE provider supports this method. It is recommended that third party providers add support for these methods when they add support for JDK 19 or later releases.

tools

➔ **New 'jmod --compress' Command Line Option** ([JDK-8293499](#))

A new `--compress` command line option has been added to the `jmod` tool to specify the compression level while creating the JMOD archive. The accepted values are `zip-[0-9]`, where `zip-0` provides no compression, and `zip-9` provides the best compression. Default is `zip-6`.

security-lib/javax.net.ssl

➔ **DTLS Resumption Uses HelloVerifyRequest Messages** ([JDK-8287411](#))

With this fix, the SunJSSE DTLS implementation will exchange cookies for all handshakes, both new and resumed, by default unless the System property `jdk.tls.enabledTlsResumeCookie` is `false`. The property only affects the cookie exchange for resumption.

core-lib/java.io

➔ **Print Warning to Standard Error If Bad java.io.tmpdir Setting Is Detected** ([JDK-8290313](#))

A new warning is now printed to the standard error stream at startup if a custom `java.io.tmpdir` system property is defined but the directory doesn't exist. The warning printed is:

"WARNING: java.io.tmpdir directory does not exist"

[TOP](#)

Removed Features and Options

This section describes the APIs, features, and options that were removed in Java SE 20 and JDK 20. The APIs described here are those that are provided with the Oracle JDK. It includes a complete implementation of the Java SE 20 Platform and additional Java APIs to support developing, debugging, and monitoring Java applications. Another source of information about important enhancements and new features in Java SE 20 and JDK 20 is the [Java SE 20 \(JSR 395\)](#) Platform Specification, which documents changes to the specification made between Java SE 17 and Java SE 20. This document includes the identification of removed APIs and features not described here. The descriptions below might also identify potential compatibility issues that you could encounter when migrating to JDK 20. See [CSRs Approved for JDK 20](#) for the list of CSRs closed in JDK 20.

core-lib/java.lang

➔ **Thread.suspend/resume Changed to Throw UnsupportedOperationException** ([JDK-8249627](#))

The ability to suspend or resume a thread with the `Thread.suspend()` and `Thread.resume()` methods has been removed in this release. The methods have been changed to throw `UnsupportedOperationException`. These methods were inherently deadlock prone and have been deprecated since JDK 1.2 (1998). The corresponding methods in `ThreadGroup`, to suspend or resume a group of threads, were changed to throw `UnsupportedOperationException` in Java 19.

core-lib/java.lang

➔ **Thread.Stop Changed to Throw UnsupportedOperationException** ([JDK-8289610](#))

The ability to "stop" a thread with the `Thread.stop()` method has been removed in this release. The method has been changed to throw `UnsupportedOperationException`. Stopping a thread by causing it to throw `java.lang.ThreadDeath` was inherently unsafe. The `stop` method has been deprecated since JDK 1.2 (1998). The corresponding method in `ThreadGroup`, to "stop" a group of threads, was changed to throw `UnsupportedOperationException` in Java 19.

As part of this change, `java.lang.ThreadDeath` has been deprecated for removal.

tools/javac

➔ **Remove Support for javac -source/-target/--release 7 (JDK-8173605)**

Consistent with the policy outlined in [JEP 182: Policy for Retiring javac -source and -target Options](#), support for the 7/1.7 argument value for javac's `-source`, `-target`, and `--release` flags has been removed.

hotspot/gc

➔ **Improved Control of G1 Concurrent Refinement Threads (JDK-8137022)**

The control of G1 concurrent refinement threads has been completely replaced. The new controller typically allocates fewer threads. It tends to have fewer spikes in refinement thread activity. It also tends to delay refinement, allowing more filtering by the write barrier when there are multiple writes to the same or nearby locations, improving the efficiency of the barrier.

There are a number of command line options used to provide parameter values for the old controller. These aren't relevant to the new controller, and no longer serve any useful purpose. They have all been made obsolete; specifying any of them on the command line will do nothing but print a warning message about the option being obsolete. These arguments are:

```
-XX:-G1UseAdaptiveConcRefinement
-XX:G1ConcRefinementGreenZone=buffer-count
-XX:G1ConcRefinementYellowZone=buffer-count
-XX:G1ConcRefinementRedZone=buffer-count
-XX:G1ConcRefinementThresholdStep=buffer-count
-XX:G1ConcRefinementServiceIntervalMillis=msec
```

These options will be removed entirely in some future release. Use of any of these options after that time will terminate startup of the virtual machine.

[TOP](#)

Deprecated Features and Options

Additional sources of information about the APIs, features, and options deprecated in Java SE 20 and JDK 20 include:

- The [Deprecated API](#) page identifies all deprecated APIs including those deprecated in Java SE 20.
- The [Java SE 20 \(JSR 395\)](#) specification documents changes to the specification made between Java SE 17 and Java SE 20 that include the identification of deprecated APIs and features not described here.
- [JEP 277: Enhanced Deprecation](#) provides a detailed description of the deprecation policy. You should be aware of the updated policy described in this document.

You should be aware of the contents in those documents as well as the items described in this release notes page.

The descriptions of deprecated APIs might include references to the deprecation warnings of `forRemoval=true` and `forRemoval=false`. The `forRemoval=true` text indicates that a deprecated API might be removed from the next major release. The `forRemoval=false` text indicates that a deprecated API is not expected to be removed from the next major release but might be removed in some later release.

The descriptions below also identify potential compatibility issues that you might encounter when migrating to JDK 20. See [CSRs Approved for JDK 20](#) for the list of CSRs closed in JDK 20.

core-libs/java.net

➔ **java.net.URL Constructors Are Deprecated (JDK-8294241)**

The `java.net.URL` constructors are deprecated in this release.

Developers are encouraged to use `java.net.URI` to parse or construct a URL. In cases where an instance of `java.net.URL` is needed to open a connection, `java.net.URI` can be used to construct or parse the URL string, possibly calling `URI::parseServerAuthority()` to validate that the authority component can be parsed as a server-based authority, and then calling `URI::toURL()` to create the URL instance.

A new method, `URL::of(URI, URLStreamHandler)` is provided for the advanced usages where there is a need to construct a URL with a given custom stream handler.

See the [java.net.URL API documentation](#) for more details.

core-svc/javax.management

➔ **Deprecate JMX Management Applets for Removal** ([JDK-8297794](#))

The Java Management Extension (JMX) Management Applet (m-let) feature is deprecated for removal in a future release as it is irrelevant to modern applications - the deprecated public classes in `javax.management.loading` are: `MLet`, `MLetContent`, `PrivateMLet`, `MLetMBean`.

This will have no impact on the JMX agent used for local and remote monitoring, the built-in instrumentation of the Java virtual machine, or tooling that uses JMX.

[TOP](#)

Known Issues

The following notes describe known issues or limitations in this release.

xml/jaxp

➔ **JDK XSLT Transformer Limitations** ([JDK-8290347](#))

Applications using the JDK XSLT transformer to convert stylesheets to Java objects can encounter the following exception:

```
com.sun.org.apache.xalan.internal.xsltc.compiler.util.InternalError: Internal XSLTC error: a
method in the translet exceeds the Java Virtual Machine limitation on the length of a method of
64 kilobytes. This is usually caused by templates in a stylesheet that are very large. Try
restructuring your stylesheet to use smaller templates.
```

Applications will encounter the above exception if the size of the XSL template is too large. It is recommended to split the XSL template into smaller templates. Alternatively, applications can override the JDK XSLT Transformer by providing third-party implementation JAR files in the class path.

hotspot/compiler

➔ **java.lang.Float.floatToFloat16 and java.lang.Float.float16ToFloat May Return Different NaN Results when Optimized by the JIT Compiler** ([JDK-8302976](#))

JDK 20 introduces two new methods which can be used to convert to and from the IEEE 754 binary16 format:

`java.lang.Float.floatToFloat16` and `java.lang.Float.float16ToFloat`.

The new methods may return different NaN results when optimized by the JIT compiler. To disable the JIT compiler optimization of these methods, the following command line options can be used: `-XX:+UnlockDiagnosticVMOptions -XX:DisableIntrinsic=_floatToFloat16,_float16ToFloat`.

install/install

➔ **Installation of Oracle Linux Specific x64 JDK RPMs Pulls in i686 Dependencies** ([JDK-8297475](#) (Not Public))

This issue prevents yum from automatically installing the correct packages required by Oracle Linux specific x86_64 headless and headful JDK packages. Instead of x86_64 packages, it will install i686 packages. To workaround the issue, you may manually install packages with the same names as indicated by yum but with the x86_64 architecture.

After you have the x86_64 headless and/or headful jdk packages installed, you can get the list of required x86_64 packages by running the following script:

```
rpm -qa | grep -E -e '^jdk-.*-headful-.*\.x86_64$' -e '^jdk-.*-headless-.*\.x86_64$' | xargs -r
rpm -q --requires | sort -u | cut -d ' ' -f 1 | grep -v '^rpmlib' | xargs -r rpm -q --
whatprovides | sort -u | grep -e '.i[3456]86$' | xargs -r rpm -q --queryformat '%
{name}.x86_64\n' | xargs -r echo
```

It will output a space-separated list of names of required x86_64 packages to stdout. You can pass this list to a `sudo yum install` command to ensure the installation of the required packages.

[TOP](#)

Other Notes

The following notes describe additional changes and information about this release. In some cases, the following descriptions provide links to additional detailed information about an issue or a change.

core-libs/java.math

➔ **Restore Behavior of `java.math.BigDecimal.movePointLeft()` and `movePointRight()` on a Zero Argument** ([JDK-8289260](#))

When these methods are invoked with a zero argument on a target with a negative *scale*, they return a result which is numerically the same as the target, but with a different *unscaled value* and *scale*.

In earlier releases, they returned a result with the same *unscaled value* and *scale*, which was against the specification.

The behavior is unchanged when the target has a non-negative *scale* or when the argument is not zero.

core-libs/java.net

➔ **HTTP Response Input Streams Will Throw an `IOException` on Interrupt** ([JDK-8294047](#))

HTTP Response Input Streams are `InputStream` instances returned by the `ResponseSubscribers::ofInputStream` method. In this release, the default implementation of their `read` method was changed to throw an `IOException` if the thread performing this operation is interrupted, instead of ignoring the interruption.

If the thread invoking the `read` operation is interrupted while blocking on `read`:

- The request will be cancelled and the `InputStream` will be closed
- The thread interrupt status will be set to `true`
- An `IOException` will be thrown

core-libs/java.net

➔ **URL Constructors Called with Malformed Input May Throw `MalformedURLException` for Cases where It Was Not Thrown Previously** ([JDK-8293590](#))

The parsing of input provided to the `java.net.URL` constructors has changed in this release to be more strict. If the URL constructors are called with malformed input, then `MalformedURLException` may be thrown for cases where it wasn't thrown previously.

In previous releases, some of the parsing and validation performed by the JDK built-in `URLStreamHandler` implementations was delayed until `URL::openConnection` or `URLConnection::connect` was called. Some of these parsing and validation actions are now performed early, within URL constructors. An exception caused by a malformed URL that would have been delayed until the connection was opened or connected might now cause a `MalformedURLException` to be thrown at URL construction time.

This change only affects URL instances that delegate to JDK built-in stream handler implementations. Applications relying on custom, third party `URLStreamHandler` implementations should remain unaffected.

A new JDK specific system property `-Djdk.net.url.delayParsing` or `-Djdk.net.url.delayParsing=true` can be specified on the command line to revert to the previous behavior. By default, the property is not set, and the new behavior is in place.

This new property is provided for backward compatibility and may be removed in a future release.

core-libs/java.net

➔ **HttpClient Default Keep Alive Time is 30 Seconds** ([JDK-8297030](#))

In this release, the default idle connection timeout value for the HTTP/1.1 and HTTP/2 connections created by the `java.net.http.HttpClient` has been reduced from 1200 seconds to 30 seconds.

core-libs/java.net

➔ **Idle Connection Timeouts for HTTP/2** ([JDK-8288717](#))

Idle Connection Timeouts for HTTP/2 are added in this release.

The `jdk.httpclient.keepalivetimeout` property can now be used to configure a system-wide value, in seconds, used to close idle connections for both HTTP/1.1 and HTTP/2 when using the `HttpClient`. In addition, developer's can also use the `jdk.httpclient.keepalivetimeout.h2` to specify a timeout value exclusively for use with the HTTP/2 protocol, regardless of whether or not the `jdk.httpclient.keepalivetimeout`` is specified at runtime.

See the [java.net.http](#) module and [Networking Properties](#) in the JDK 20 API documentation for a list of current networking properties.

core-libs/java.nio

➔ **FileChannel Positional Write Is Unspecified in APPEND Mode** ([JDK-6924219](#))

The specification of `java.nio.channels.FileChannel` is updated to clarify that the effect of attempting to write at a specific position using the `FileChannel::write(ByteBuffer, long)` method is system-dependent when the channel is open in append mode. That is, a `java.nio.file.StandardOpenOption.APPEND` is passed to `FileChannel::open` when the channel is opened. In particular, on some operating systems, bytes will be written at the given position, while on other operating systems the given position will be ignored and the bytes will be appended to the file.

core-libs/java.nio

➔ **Do Not Normalize File Paths to Unicode Normalization Format D on macOS** ([JDK-8289689](#))

On macOS, file names are no longer normalized to Apple's variant of Unicode Normalization Format D. File names were normalized on HFS+ prior to macOS 10.13, but on APFS on macOS 10.13 and newer, this normalization is no longer effected. The previous behavior may be enabled by setting the system property `jdk.nio.path.useNormalizationFormD` to "true".

core-libs/java.text

➔ **Grapheme Support in BreakIterator** ([JDK-8291660](#))

Character boundary analysis in `java.text.BreakIterator` now conforms to Extended Grapheme Clusters breaks defined in [Unicode Consortium's Standard Annex #29](#). This change will introduce intentional behavioral changes because the old implementation simply breaks at the code point boundaries for the vast majority of characters. For example, this is a String that contains the US flag and a grapheme for a 4-member-family.

""

This String will be broken into two graphemes with the new implementation:

"", ""

whereas the old implementation simply breaks at the code point boundaries:

"", "", "", "", "", "", "", "", ""

where (zwj) denotes ZERO WIDTH JOINER (U 200D).

core-libs/java.time

➔ **Update Timezone Data to 2022c** ([JDK-8292579](#))

This version includes changes from 2022b that merged multiple regions that have the same timestamp data post-1970 into a single time zone database. All time zone IDs remain the same but the merged time zones will point to a shared zone database.

As a result, pre-1970 data may not be compatible with earlier JDK versions. The affected zones are Antarctica/Vostok, Asia/Brunei, Asia/Kuala_Lumpur, Atlantic/Reykjavik, Europe/Amsterdam, Europe/Copenhagen, Europe/Luxembourg, Europe/Monaco, Europe/Oslo, Europe/Stockholm, Indian/Christmas, Indian/Cocos, Indian/Kerguelen, Indian/Mahe, Indian/Reunion, Pacific/Chuuk, Pacific/Funafuti, Pacific/Majuro, Pacific/Pohnpei, Pacific/Wake, Pacific/Wallis, Arctic/Longyearbyen, Atlantic/Jan_Mayen, Iceland, Pacific/Ponape, Pacific/Truk, and Pacific/Yap.

For more details, refer to the announcement of [2022b](#)

core-libs/java.util.collections

➔ **IdentityHashMap's Remove and Replace Methods Use Object Identity** ([JDK-8178355](#))

The `remove(key, value)` and `replace(key, oldValue, newValue)` implementations of `IdentityHashMap` have been corrected. In previous releases, the value arguments were compared with values in the map using `equals`. However, `IdentityHashMap` specifies that all such comparisons should be made using object identity (`==`). These methods' implementations now conform to the specification.

core-libs/java.util.i18n

➔ **Support for CLDR Version 42** ([JDK-8284840](#))

Locale data based on Unicode Consortium's CLDR has been upgraded to version 42. For the detailed locale data changes, please refer to the [Unicode Consortium's CLDR release notes](#). Some of the notable changes in the upstream that may affect formatting are:

- [NBSP/NNBSP prefixed to AM/PM in time format, instead of a normal space](#)
- [" at " is no longer used for standard date/time format](#)
- [Fix first day of week info for China \(CN\)](#)
- [Japanese: Support numbers up to 9999京](#)

core-libs/javax.naming

➔ **Update Default Value and Extend the Scope of `com.sun.jndi ldap.object.trustSerialData` System Property** ([JDK-8290367](#))

In this release, the JDK implementation of the LDAP provider no longer supports deserialization of Java objects by default:

- The default value of the `com.sun.jndi.ldap.object.trustSerialData` system property has been updated to `false`.
- The scope of the `com.sun.jndi.ldap.object.trustSerialData` system property has been extended to cover the reconstruction of RMI remote objects from the `javaRemoteLocation` LDAP attribute.

The transparent deserialization of Java objects from an LDAP context will now require an explicit opt-in. Applications that rely on reconstruction of Java objects or RMI stubs from the LDAP attributes would need to set the `com.sun.jndi.ldap.object.trustSerialData` system property to `true`.

core-libs/javax.naming

➔ **Introduce LDAP and RMI Protocol Specific Object Factory Filters to JNDI Implementation** ([JDK-8290368](#))

In this release, new system and security properties are introduced to allow more granular control over the set of JNDI object factories allowed to reconstruct Java objects from JNDI/LDAP and JNDI/RMI contexts:

- The new `jdk.jndi.ldap.object.factoriesFilter` property specifies which object factory classes are allowed to instantiate Java objects from object references returned by JNDI/LDAP contexts. Its default value only allows object factories defined in the `java.naming` module.
- The new `jdk.jndi.rmi.object.factoriesFilter` property specifies which object factory classes are allowed to instantiate Java objects from object references returned by JNDI/RMI contexts. Its default value only allows object factories defined in the `jdk.rmi` module.

These new factory filter properties complement the `jdk.jndi.object.factoriesFilter` global factories filter property by determining if a specific object factory is permitted to instantiate objects for the LDAP or RMI protocols used in JNDI.

An application depending on custom object factories to recreate Java objects from JNDI/LDAP or JNDI/RMI contexts will need to supply a security or system property with an updated value to allow such third-party object factories to reconstruct LDAP or RMI objects. If usage of a factory is denied, the lookup operation may result in a plain instance of `javax.naming.Reference` instance returned, which may lead to a `ClassCastException` being thrown in the application.

For more information, see the [java.naming](#) and [jdk.naming.rmi](#) module-info documentation.

core-svc/debugger

➔ **`com.sun.jdi.ObjectReference::setValue` Specification Should Prohibit Any Final Field Modification** ([JDK-8280798](#))

The specification of the Java Debug Interface (JDI) method `ObjectReference.setValue` has changed in this release to require the given field be non-final. The method was previously specified to require static fields be non-final but was silent on final instance fields. The JDK’s implementation of JDI has never allowed final instance fields to be changed with this method so this change has no impact on debuggers or tools using the JDK’s JDI implementation. Maintainers of JDI implementations should take note of this change so they can align their implementation with the updated specification.

core-svc/javax.management

➔ **JMX Connections Use an `ObjectInputFilter` by Default** ([JDK-8283093](#))

The default JMX agent now sets an `ObjectInputFilter` on the RMI connection to restrict the types that the server will deserialize. This should not affect normal usage of the MBeans in the JDK. Applications which register their own MBeans in the Platform MBeanServer may need to extend the filter to support any additional types that their MBeans accept as parameters. The default filter already covers any type that OpenMBeans and MXBeans might use.

The filter pattern is set in `JDK/conf/management/management.properties` using the property `com.sun.management.jmxremote.serial.filter.pattern`. If there are additional Java types that need to be passed, the default can be overridden by running with `-Dcom.sun.management.jmxremote.serial.filter.pattern=.`

Serialization Filtering and the filter pattern format are described in detail in the Core Libraries guide.

hotspot/gc

➔ **G1: Disable Preventive GCs by Default** ([JDK-8293861](#))

In JDK 17, G1 added "preventive" garbage collections (GCs). These are speculative garbage collections, with the goal of avoiding costly evacuation failures due to allocation bursts when the heap is almost full.

However, these speculative collections have the consequence of additional garbage collection work, as object aging is based on number of GCs with additional GCs causing premature promotion into the old generation, which leads to more data in the old generation, and more garbage collection work to remove these objects. This has been compounded by the current prediction to trigger preventive garbage collections being very conservative; which means these garbage collections are often triggered unnecessarily.

In the majority of cases this feature is a net loss, and as evacuation failures are now handled more quickly, there is no longer any reason for this feature and it has been disabled by default, it may be re-enabled by `-XX:+UnlockDiagnosticVMOptions -XX:+G1UsePreventiveGC`.

hotspot/jvmti

➔ **appendToClassPathForInstrumentation Must Be Used in a Thread-Safe Manner** ([JDK-8296472](#))

When running an application with a Java agent (e.g. `-javaagent:myagent.jar`) and a custom system class loader (e.g. `-Djava.system.class.loader=MyClassLoader`), and the Java agent invokes the `Instrumentation.appendToSystemClassLoaderSearch` API to append to the class loader search, then the custom system class loader `appendToClassPathForInstrumentation` method will be invoked to add the JAR file to the custom system class loader's search path.

The JVM no longer synchronizes on the custom class loader while calling `appendToClassPathForInstrumentation`. The `appendToClassPathForInstrumentation` method in the custom class loader must add to the class search path in a thread-safe manner.

hotspot/jvmti

➔ **GetLocalXXX/SetLocalXXX Specification Should Require Suspending Target Thread** ([JDK-8288387](#))

The JVM TI specification of the `GetLocalXXX/SetLocalXXX` functions has been changed to require the target thread to be either suspended or the current thread. The error code `JVMTI_ERROR_THREAD_NOT_SUSPENDED` will be returned if this requirement is not satisfied. The JVM TI agents which use the `GetLocalXXX/SetLocalXXX` API's need to be updated to suspend the target thread if it is not the current thread. The full list of impacted JVM TI functions is:

- `GetLocalObject`, `GetLocalInt`, `GetLocalLong`, `GetLocalFloat`, `GetLocalDouble`, `GetLocalInstance`,
- `SetLocalObject`, `SetLocalInt`, `SetLocalLong`, `SetLocalFloat`, `SetLocalDouble`

hotspot/runtime

➔ **Deprecate and Disable Legacy Parallel Class Loading Workaround for Non-Parallel-Capable Class Loaders** ([JDK-8295673](#))

Some user-defined, older class loaders would workaround a deadlock issue by releasing the class loader lock during the loading process. To prevent these loaders from encountering a “`java.lang.LinkageError: attempted duplicate class definition`” while loading the same class by parallel threads, the HotSpot Virtual Machine introduced a workaround in JDK 6 that serialized the load attempts, causing the subsequent attempts to wait for the first to complete.

The need for class loaders to work this way was removed in JDK 7 when parallel-capable class loaders were introduced, but the workaround remained in the VM. That workaround is finally being removed and as a first step has been deprecated and disabled by default. If you start seeing “`java.lang.LinkageError: attempted duplicate class definition`”, then you may have an affected legacy class loader. The flag `-XX:+EnableWaitForParallelLoad` can be used to temporarily restore the old behavior in this release of the JDK, but the legacy class loader will need to be updated for future releases.

See the CSR request ([JDK-8295848](#)) for more background and details.

security-libs/java.security

➔ **Added Constructors (String, Throwable) and (Throwable) to InvalidParameterException** ([JDK-8296226](#))

Constructors `InvalidParameterException(String, Throwable)` and `InvalidParameterException(Throwable)` have been added to the `java.security.InvalidParameterException` class to support easy construction of `InvalidParameterException` objects with a cause.

security-libs/java.security

➔ **Throw Error If Default java.security File Fails to Load** ([JDK-8155246](#))

A behavioral change has been made in the case where the default `conf/security/java.security` security configuration file fails to load. In such a scenario, the JDK will now throw an `InternalError`.

Such a scenario should never occur. The default security file should always be present. Prior to this change, a static and outdated security configuration was loaded.

security-libs/javax.net.ssl

➔ **Disabled TLS_ECDH_* Cipher Suites** (JDK-8279164)

The TLS_ECDH_* cipher suites have been disabled by default, by adding "ECDH" to the `jdk.tls.disabledAlgorithms` security property in the `java.security` configuration file. The TLS_ECDH_* cipher suites do not preserve forward-secrecy and are rarely used in practice. Note that some TLS_ECDH_* cipher suites were already disabled because they use algorithms that are disabled, such as 3DES and RC4. This action disables the rest. Any attempts to use cipher suites starting with "TLS_ECDH_" will fail with an `SSLHandshakeException`. Users can, at their own risk, re-enable these cipher suites by removing "ECDH" from the `jdk.tls.disabledAlgorithms` security property.

security-libs/javax.net.ssl

➔ **Disabled DTLS 1.0** (JDK-8256660)

DTLS 1.0 has been disabled by default, by adding "DTLSv1.0" to the `jdk.tls.disabledAlgorithms` security property in the `java.security` configuration file. DTLS 1.0 has weakened over time and lacks support for stronger cipher suites. Any attempts to use DTLSv1.0 will fail with an `SSLHandshakeException`. Users can, at their own risk, re-enable the version by removing "DTLSv1.0" from the `jdk.tls.disabledAlgorithms` security property.

security-libs/javax.security

➔ **Remove Thread Text from Subject.current** (JDK-8297276)

The specification of `Subject.current` has been changed in this release to drop the expectation that the `Subject` is inherited when creating a thread. At this time, the `Subject` is stored in the `AccessControlContext` and is inherited when creating platform threads. Virtual threads do not capture the caller context at thread creation time and the `AccessControlContext` is not inherited. Inheritance will be re-examined in a future release in advance of removing support for the `SecurityManager` and the inherited `AccessControlContext`.

security-libs/javax.security

➔ **New Implementation Note for LoginModule on Removing Null from a Principals or Credentials Set** (JDK-8282730)

The `Set` implementation that holds principals and credentials in a JAAS `Subject` prohibits null elements and any attempt to add, query, or remove a null element will result in a `NullPointerException`. This is especially important when trying to remove principals or credentials from the subject at the logout phase but they are null because of a previous failed login. Various JDK `LoginModule` implementations have been fixed to avoid the exception. An Implementation Note has also been added to the `logout()` method of the `LoginModule` interface. Developers should verify, and if necessary update, any custom `LoginModule` implementations to be compliant with this implementation advice.

tools/javac

➔ **An Exhaustive Switch over an Enum Class Should Throw MatchException Rather Than IncompatibleClassChangeError If No Switch Label Applies at Runtime** (JDK-8297118)

In this release, when preview features are enabled with `--enable-preview`, a `switch` expression over an enum type will throw a `MatchException` rather than an `IncompatibleClassChangeError` should the selector expression yield an unexpected enum constant value. This can only happen if the enum class has been changed by adding a new enum constant *after* compilation of the `switch`.

This change is required to unify the treatment of erroneous exhaustive switches introduced by enhancing `switch` with pattern labels.

tools/javadoc(tool)

➔ **Improved Preview API Page** (JDK-8287597)

The Preview API page in the documentation generated by JavaDoc now provides detailed information about the JEPs the preview features belong to.

tools/javadoc(tool)

➔ **Auto-Generated IDs in JavaDoc Headings** (JDK-8289332)

JavaDoc now generates `id` attributes for all HTML headings in documentation comments that may be used as link anchors.

tools/javadoc(tool)

➔ **Generalize 'see' and 'link' Tags for User-Defined Anchors** (JDK-8200337)

The `{@link}`, `{@linkplain}`, and `@see` tags have been enhanced to allow linking to arbitrary anchors in the JavaDoc-generated documentation for an element. To distinguish these references from member references, a double hash mark (``##``) is used to separate the element name from the URI fragment.

hotspot/runtime

➔ **The JNI Specification Omits an Update to the JNI Version** ([JDK-8290482](#))

This bug updated the JNI Specification in relation to the `DestroyJavaVM` function. As part of that work, the JNI Specification version number was incremented. That change to the JNI Specification version should also have been reflected in the `GetVersion` function but was not. The `GetVersion` function for JDK 20 and later has been updated to read `JNI_VERSION_20`. This new version has also been documented in `jni.h` as:

```
#define JNI_VERSION_20 0x00140000
```

core-libs/java.util.jar

➔ **Weaken the InflaterInputStream Specification in Order to Allow Faster Zip Implementations** ([JDK-8282648](#))

The specification of the `read` methods defined by `java.util.zip.InflaterInputStream`, `ZipInputStream`, and `GZIPInputStream` have changed to allow these methods deviate from `InputStream.read` for the case that a “short read” occurs.

A “short read” is the case where the user provides a buffer with space for `M` bytes but only `N` bytes (where $0 < N < M$) are read. The long standing specification for `InputStream.read` is that `N` bytes are stored in the buffer provided by the user and elements at offset `off+N` to `off+M-1` are not changed.

The deviation allows the `read` method to use the elements at `off+N` to `off+M-1` as temporary storage. Code using these APIs can no longer depend on these elements being unaffected when reading uncompressed data into a byte array.

core-libs/java.net

➔ **New System Property to Limit the Number of Open Connections to com.sun.net.httpserver.HttpServer** (JDK-8286918 (not public))

A new system property named `jdk.httpserver.maxConnections` has been introduced to allow users to configure the `com.sun.net.httpserver.HttpServer` to limit the maximum number of open connections to the server at any given time. This system property takes an integer value and can be configured to be a positive integer. If the property is absent, set to 0, or a negative value, the server will not limit the number of open connections. By default, this system property is not set.

[TOP](#)

Differences Between Oracle JDK and OpenJDK

Although we have stated the goal to have Oracle JDK and OpenJDK binaries be as close to each other as possible, there remain several differences between the two options.

The current differences are:

- Oracle JDK offers "installers" (`msi`, `rpm`, `deb`, etc.) which not only place the JDK binaries in your system but also contain update rules and in some cases handle some common configurations like set common environmental variables (such as, `JAVA_HOME` in Windows) and establish file associations (such as, use `java` to launch `.jar` files). OpenJDK is offered only as compressed archive (`tar.gz` or `.zip`).
- Usage Logging is only available in Oracle JDK.
- Oracle JDK requires that third-party cryptographic providers be signed with a Java Cryptography Extension (JCE) Code Signing Certificate. OpenJDK continues allowing the use of unsigned third-party crypto providers.
- The output of `java -version` is different. Oracle JDK returns `java` and includes the Oracle-specific identifier. OpenJDK returns `OpenJDK` and does not include the Oracle-specific identifier.
- Oracle JDK 17 and later are released under the [Oracle No-Fee Terms and Conditions License](#). OpenJDK is released under [GPLv2wCP](#). License files included with each will therefore be different.
- Oracle JDK distributes FreeType under the FreeType license and OpenJDK does so under GPLv2. The contents of `\legal\java.desktop\freetype.md` is therefore different.
- Oracle JDK has Java cup and steam icons and OpenJDK has Duke icons.
- Oracle JDK source code includes "ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms." Source code distributed with OpenJDK refers to the GPL license terms instead.

Resources for

Careers

Developers

Investors

Partners

Researchers

Students and Educators

Why Oracle

Analyst Reports

Best cloud-based ERP

Cloud Economics

Corporate Responsibility

Diversity and Inclusion

Security Practices

Learn

What is cloud computing?

What is CRM?

What is Docker?

What is Kubernetes?

What is Python?

What is SaaS?

News and Events

News

Oracle CloudWorld

Oracle CloudWorld Tour

Oracle Health Conference

DevLive Level Up

Search all events

Contact Us

US Sales: +1.800.633.0738

How can we help?

Subscribe to emails

Integrity Helpline


© 2023 Oracle


Privacy / Do Not Sell My Info


Cookie Preferences


Ad Choices


Careers









 Country/Region