OpenJDK 8

Release notes for OpenJDK 8.0.292

Last Updated: 2021-05-21

Legal Notice

Abstract

This document provides an overview of new features in OpenJDK 8, as well as 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). The Red Hat build of OpenJDK is available in two versions, OpenJDK 8u and OpenJDK 11u.

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.

CHAPTER 1. SUPPORT POLICY

Red Hat will support select major versions of OpenJDK in its products. For consistency, these versions will be the same ones that Oracle designates 'LTS' for the Oracle JDK.

A major version of OpenJDK will be supported for a minimum of six years from the time it is first introduced.

OpenJDK 8 is supported on Microsoft Windows and Red Hat Enterprise Linux until May 2026.

NOTE

RHEL 6 has reached the end of life in November 2020. Due to this, OpenJDK is not supporting RHEL 6 as a supporting configuration.

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

CHAPTER 2. DIFFERENCES FROM UPSTREAM OPENJDK 8

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

The most notable changes are the following:

- On Red Hat Enterprise Linux, we dynamically link against native libraries such as **zlib** for archive format support and **libjpeg-turbo**, **libpng**, and **giflib** for image support. Likewise, we dynamically link against **Harfzbuzz** and **Freetype** for font rendering and management.
  On Microsoft Windows, these libraries are built from the sources of the corresponding Red Hat Enterprise Linux RPMs.

- On Red Hat Enterprise Linux, system-wide timezone data files are used as a source for timezone information.
  On Microsoft Windows, the latest available timezone data from Red Hat Enterprise Linux is included.

- On Red Hat Enterprise Linux, system-wide CA certificates are used.
  On Microsoft Windows, the latest available CA certificate from Red Hat Enterprise Linux is used.

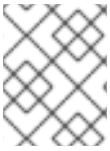
- The Windows distribution includes the DejaVu set of TrueType Fonts imported from Red Hat Enterprise Linux.

- The **src.zip** file includes the source for all of the JAR libraries shipped with OpenJDK.

CHAPTER 3. OPENJDK FEATURES

3.1. NEW FEATURES AND ENHANCEMENTS

This section describes the new features introduced in this release. It also contains information about changes in the existing features.



NOTE

For all the other changes and security fixes, see https://mail.openjdk.java.net/pipermail/jdk8u-dev/2021-April/013680.html

3.1.1. Added two HARICA root CA certificates

The following two root certificates are added to the cacerts truststore:

- Alias Name: haricarootca2015
  Distinguished Name: CN=Hellenic Academic and Research Institutions RootCA 2015, O=Hellenic Academic and Research Institutions Cert. Authority, L=Athens, C=GR

- Alias Name: haricaeccrootca2015
  Distinguished Name: CN=Hellenic Academic and Research Institutions ECC RootCA 2015, O=Hellenic Academic and Research Institutions Cert. Authority, L=Athens, C=GR

For more information, see JDK-8260597.

3.1.2. Weak-named curves in TLS, CertPath, and signed JAR are disabled by default

Weak-named curves are disabled by default by adding them to the following **disabledAlgorithms** security properties:

- **jdk.tls.disabledAlgorithms**
- **jdk.certpath.disabledAlgorithms**
- **jdk.jar.disabledAlgorithms**

Red Hat has always disabled many of the curves provided by upstream, so the only addition in this release is **secp256k1**.

The following curves remain enabled:

- secp256r1
- secp384r1
- secp521r1
- X25519
- X448

When large numbers of weak-named curves need to be disabled, adding individual named curves to each **disabledAlgorithms** property would be overwhelming. To relieve this, a new security property, **jdk.disabled.namedCurves**, is implemented that can list the named curves common to all of the

disabledAlgorithms properties. To use the new property in the **disabledAlgorithms** properties, precede the full property name with the keyword **include**. Users can still add individual named curves to **disabledAlgorithms** properties separate from this new property. No other properties can be included in the **disabledAlgorithms** properties.

To restore the named curves, remove the **include jdk.disabled.namedCurves** either from specific or from all **disabledAlgorithms** security properties. To restore one or more curves, remove the specific named curve(s) from the **jdk.disabled.namedCurves** property.

For more information, see JDK-8236730.

3.1.3. Updated tools warn if weak algorithms are used

The **keytool** and **jarsigner** tools are updated to warn users when weak cryptographic algorithms are used in keys, certificates, and signed JARs before they are disabled. The weak algorithms are set in the **jdk.security.legacyAlgorithms** security property in the **java.security** configuration file. In this release, the tools issue warnings for the SHA-1 hash algorithm and 1024-bit RSA/DSA keys.

For more information, see JDK-8244286.

3.1.4. Disabled TLS 1.0 and 1.1 versions

TLS 1.0 and 1.1 versions of the TLS protocol that are no longer considered secure and are superseded by more secure and modern TLS 1.2 and 1.3 versions.

TLS 1.0 and 1.1 versions are now disabled by default. If you encounter issues, you can re-enable the versions (at your own risk) by removing **TLSv1** or **TLSv1.1** from the **jdk.tls.disabledAlgorithms** security property in the **java.security** configuration file.

For more information, see JDK-8256490.

3.1.5. Added new system properties to configure the TLS signature schemes

Two new system properties are added to customize the TLS signature schemes in JDK. The **jdk.tls.client.SignatureSchemes** is added for the TLS client side and the **jdk.tls.server.SignatureSchemes** is added for the server side.

Each system property contains a comma-separated list of supported signature scheme names specifying the signature schemes that can be used for the TLS connections.

The names are described in the **Signature Schemes** section of the **Java Security Standard Algorithm Names Specification**.

For more information, see JDK-8242147.

3.1.6. Several incorporation steps are silently failing when an error should be reported

Reporting previously silent errors found during incorporation **JLS 8 §18.3** was supposed to be a clean-up with performance only implications. But consider the following test case:

```
import java.util.Arrays;
import java.util.List;
```

```
class Klass {
  public static <A> List<List<A>> foo(List<? extends A>... lists) {
    return foo(Arrays.asList(lists));
    }

  public static <B> List<List<B>> foo(List<? extends List<? extends B>> lists) {
    return null;
  }
}
```

This code was not accepted before the patch, but after this patch the compiler is accepting it. Accepting this code is the right behavior as not reporting incorporation errors was a bug in the compiler. While determining the applicability of **method: <B> List<List<B>> foo(List<? extends List<? extends B>> lists)** for which we have the constraints **b <: Object t <: List<? extends B> t<:Object List<? extends A> <: t** first, inference variable b is selected for **instantiation: b = CAP1 of ? extends A** so this implies that **t <: List<? extends CAP1 of ? extends A> t<: Object List<? extends A> <: t**

Now all the bounds are checked for consistency. While checking if **List<? extends A>** is a subtype of **List<? extends CAP1 of ? extends A>** a bound error is reported. Before the compiler was just swallowing it. As now the error is reported while inference variable **b** is being instantiated, the bound set is rolled back to it's initial state, **b** is instantiated to Object, and with this instantiation the constraint set is solvable, the method is applicable, it's the only applicable one and the code is accepted as correct. The compiler behavior in this case is defined at **JLS 8 §18.4**

This fix has source compatibility impact, right now code that wasn't being accepted is now being accepted by the **javac** compiler. Currently there are no reports of any other kind of incompatibility.

For more information, see JDK-8177368.

CHAPTER 4. ADVISORIES RELATED TO THIS RELEASE

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

- RHSA-2021:1298
- RHSA-2021:1299
- RHSA-2021:1301
- RHSA-2021:1444
- RHSA-2021:1445