vms Software

OpenJDK 8 for VSI OpenVMS I64

Release Notes
09-Jun-2020

Introduction

Thank you for your interest in this port of OpenJDK 8 for VSI OpenVMS I64. The current release of OpenJDK for VSI OpenVMS is based on the OpenJDK 8u222 distribution.

OpenJDK (https://openjdk.java.net/) is a free and open source implementation of the Java Platform, Standard Edition (Java SE). OpenJDK is licensed under the GNU General Public License (GNU GPL) Version 2 with a linking exception such that components linked to the Java Class library are not subject to the terms of the GPL license. OpenJDK is the official reference implementation of Java SE since Version 7.

This release notes document contains installation instructions, details of any new features, known issues, and other information specific to this release of the software. This kit can be used to develop and run Java-based programs on VSI OpenVMS I64 Version 8.4-2L1 with latest updates (support for earlier versions of VSI OpenVMS I64 will be provided in a future release).

Please ensure that you understand the copyright and license information before using this release (this information can be found in the top level directory of your OpenJDK installation).

Note that OpenJDK 8 for VSI OpenVMS I64 is not an update for the Java™ Development Kit (JDK) 8 for the OpenVMS Integrity servers Operating System; it is a separate product.

Fixed problems

This release of OpenJDK for VSI OpenVMS is based on the 8u222 release. The OpenJDK release notes document http://mail.openjdk.java.net/pipermail/jdk8u-dev/2019-July/009840.html provides details of problems fixed in this release.

In addition to the fixed problems and updates included in the OpenJDK 8u222 release, a number of general issues encountered with previous versions of Java for OpenVMS have been resolved for this OpenJDK release, including the following:

- Upon certain exceptions (error conditions) the OpenVMS debugger could start unexpectedly. This problem has been fixed.
- Java crash dumps could sometimes be incomplete due to a second crash dump being initiated while the first crash dump was still in progress. This problem has been resolved.
- The code for creating child processes with inherited I/O has been significantly improved and is less error-prone. Issues associated with losing sub-process mailboxes have also been fixed.

- Enhancements to native code generation have eliminated the occurrence of a number of spurious errors and exceptions.
- Not all file versions were processed by `IfExists()` (`java.nio.file.Files` package) when the logical name `JAVA$DELETE_ALL_VERSIONS` was defined. This problem is resolved.
- The `sun.nio.fs.UnixPath.toRealPath()` method did not correctly resolve paths such as "`path_to_dir/.`" and "`path_to_dir/subdir/..`". This problem has been fixed.
- In some cases the `sun.nio.fs.UnixPath.toRealPath()` method incorrectly resolved links. This problem has been fixed.
- The following C RTL features are enabled via `LIB$INITIALIZE`. Note that the logical names to enable some of these features are still included in `java$setup.com` in case users rely on these definitions for other purposes.
  - `DECC$FILE_PERMISSION_UNIX`
  - `DECC$FILENAME_UNIX_NO_VERSION`
  - `DECC$FILE_SHARING`
  - `DECC$FD_LOCKING`
  - `DECC$EFS_CASE_PRESERVE`
  - `DECC$EFS_CHARSET`
  - `DECC$ARGV_PARSE_STYLE`
  - `DECC$READDIR_DROPDOTNOTYPE`
- Atomic operations on improperly aligned values could result in `%SYSTEM-F-ROPRAND` errors. This problem has been resolved.
- The following JVM options have been changed:
  - The `UseCompressedOops` option is not supported in OpenJDK for VSI OpenVMS and is always set to false.
  - Only a subset of values can be used with the `TypeProfileLevel` JVM option. Specifically, for `TypeProfileLevel=x00` `x` may only be 0, 1, or 2.
- Incorrect JVM exit status values were being set in some situations. Exit status values are now set in a correct and consistent manner.
- Problems with inconsistent file cache updates and renaming of cached files have been fixed.
- When remote debugging, programs would not start correctly due to listening sockets being closed when they should not have been. This problem has been resolved.
- When using `sun.nio.fs.UnixNativeDispatcher.opendir()` for an existing file (not a directory) an incorrect exception was being thrown. The correct exception is now reported.

Compatibility

OpenJDK 8 for VSI OpenVMS I64 is largely compatible with older Java versions for OpenVMS I64 and most existing Java programs will run without change on the OpenJDK platform. However, if you are

moving from Oracle Java 6 for HPE OpenVMS to OpenJDK 8 for VSI OpenVMS there are some significant differences to be aware of.

The following list identifies various differences between Oracle Java 6 for HPE OpenVMS and OpenJDK 8 for VSI OpenVMS that may impact the operation of some programs.

- Exclusive use of 64-bit pointers

  For Oracle Java 6 for HPE OpenVMS, the HotSpot Java Virtual Machine (JVM) utilized 64-bit pointers to facilitate the use of more than 2GB memory; however other binary components such as the launcher and shareable images called into by Java class libraries used only 32-bit pointers. OpenJDK 8 for VSI OpenVMS uses 64-bit pointers exclusively. As a consequence of this, any C or C++ application code using the Java Native Interface (JNI) will need to be recompiled to use 64-bit pointers (`/POINTER_SIZE=64`). Depending on the nature of the application code, this may necessitate some code changes.

- Symbol vector compatibility

  Symbol vectors in sharable images shipped with OpenJDK 8 for VSI OpenVMS will not necessarily match those of the equivalent images provided by Oracle Java 6 for HPE OpenVMS. Any C or C++ application code using the Java Native Interface (JNI) that links with these shareable images will need to be relinked.

- Removal of logical name `JAVA$ENABLE_ENVIRONMENT_EXPANSION`

  Commands to run Java programs can often be very long, and this can cause issues with DCL command line lengths. The logical name `JAVA$ENABLE_ENVIRONMENT_EXPANSION` was used in prior versions of Java for OpenVMS to help get around this issue such that any argument specified on the Java command line beginning with a "`$`" would be assumed to equate to a logical name (without the leading "`$`" character) that could specify a list of values and would be expanded out internally within Java, thereby avoiding issues with command line length. This facility was most commonly used to specify the Java class path (via the `-cp` or `-classpath` command line options), as class paths can often be very long; however the facility was little used for any other purpose.

  In OpenJDK 8 for VSI OpenVMS the Java virtual machine always checks the value supplied with the `-cp` or `-classpath` option to determine whether it equates to a logical name and if so then expansion occurs as before (as if the logical name `JAVA$ENABLE_ENVIRONMENT_EXPANSION` was defined), regardless of whether the argument has a leading "`$`" or not. It should also be noted that OpenJDK for VSI OpenVMS also supports the use of wildcards ("`*`") in class path specifications. This feature can also be used to reduce the length of class path specifications.

- Logical name `JAVA$FILENAME_CONTROLS` defaults to "8"

  The logical name `JAVA$FILENAME_CONTROLS` can be used to control how OpenJDK interprets and maps file names (between UNIX and OpenVMS formats). This logical name now defaults to a value of 8, as this value generally affords greatest flexibility and most predictable results.

  Be sure to define `JAVA$FILENAME_CONTROLS` appropriately for your environment, particularly if an ODS-2 file system is used for `.jar` and/or `.class` files (however the use of ODS-2 file systems is not recommended). See examples in `JAVA$FILENAME_CONTROLS.COM` (found in `SYS$COMMON:[OPENJDK$80.COM]` assuming a default installation) for setting the variable `JAVA$M_MULTI_DOT_KEEP_LAST` to accommodate any particular file name mapping requirements.

- Changes to use of JAVA$FORK_PIPE_STYLE

  In Oracle Java 6 for HPE OpenVMS it was possible to specify values of 0, 1, and 2 for this logical name to control how pipes are established between parent and child processes. The value of 2 would cause sockets to be used instead of OpenVMS mailboxes or standard UNIX-style pipes. If JAVA$FORK_PIPE_STYLE is not defined then a default value of 1 is used (which causes mailboxes to be used for any inter-process communication). This is still the case for OpenJDK on VSI OpenVMS; however the value of 2 is no longer supported, and if a value of 2 or an invalid value is specified, this will not be accepted and the default value of 1 will silently be used.

- No debug versions of images

  The size of the HotSpot Java Virtual Machine is such that building a debug version is not possible and consequently OpenJDK for VSI OpenVMS does not provide debug versions of executable programs and shareable images.

- Case sensitivity of file names

  OpenJDK for VSI OpenVMS is more sensitive to the case of file names, and in general the names of .java and .class files should match identically the name of the class in question. For example, if you have a Java class named myClass, then the corresponding source file should be named myClass.java. This impacts both the JVM (the java command) and utilities such as the javac compiler. However, when compiling classes it is possible to specify Java source code file name arguments to javac in arbitrary case and the compiler will attempt to determine (and use) the true on-disk filename (which javac will expect to match the public class name).

- Mixed syntax file names

  Oracle Java 6 for HPE OpenVMS allowed mixed-syntax file names (file names containing a combination of UNIX-style and OpenVMS-style syntax). The use of mixed syntax is not supported by OpenJDK for VSI OpenVMS, and in general file names should ideally conform to UNIX-style syntax. For example, the following code will give an exception:

  ```
  File file = new File("[.log]/filetest.log");
  ```

- java.awt.headless system property

  The system property java.awt.headless defaults to "true" for this release of OpenJDK for VSI OpenVMS. For Java applications that use AWT graphical user interface components, it is necessary to explicitly set java.awt.headless to false either via the java command line ("-Djava.awt.headless=false") or programmatically.

  As a specific example, if you use the Archive Backup System (ABS) graphical user interface, the start-up script SYS$COMMON:[MDMS.SYSTEM]MDMS$START_GUI.COM should be modified to include -Djava.awt.headless=false on the Java command line, as follows:

  ```
  $ java "-Xmx64M" "-Djava.awt.headless=false" "absview.ABSView"
  ```

- The CRTL feature DECC$READDIR_DROPDOTNOTYPE is enabled

  This CRTL feature controls how the OpenVMS C RTL treats file names with no extension (no file type). Without this feature enabled, problems can occur when performing operations such as adding a directory containing files with no extension to a jar file such that the files with no extension appear in the jar with a "." appended to the names. This can then cause problems if your Java code specifically tries to access those files in the jar. Appending the "." is the typical C

RTL behaviour when scanning a directory to return a list of file names; this behaviour is overridden by enabling the `DECC$READDIR_DROPDOTNOTYPE` feature.

- Exit status

  Upon normal successful completion, `java`, `javac`, and other executable utilities will consistently exit with a status of "`%X10000001`".

- Location of error logs

  In the event of an unrecoverable error condition, the JVM will attempt to create a log file containing potentially useful information about the crash. Oracle Java 6 for HPE OpenVMS would attempt to create these files in the equivalent of the UNIX/Linux `tmp` directory, which unless otherwise defined, is mapped by the OpenVMS C RTL to `SYS$SCRATCH`. To avoid any ambiguity, this release explicitly uses `SYS$SCRATCH` instead of `tmp`.

- HPE Secure Web Browser compatibility

  OpenJDK for VSI OpenVMS is not compatible with the HPE Secure Web Browser for OpenVMS. A compatible browser plugin may be provided at a later date.

- Not compatible with Availability Manager Analyser

  The Availability Manager Analyser kit includes a compatible JRE (Java Runtime Environment). Availability Manager Analyser will not work correctly with OpenJDK for VSI OpenVMS and the use of the bundled JRE should not be overridden or bypassed in any way. An updated Availability Manager Analyser that can be used with OpenJDK for VSI OpenVMS will be made available in due course.

- `JAVA$DAEMONIZE_MAIN_THREAD` logical name deprecated

  In Oracle Java 6 for HPE OpenVMS this logical name could be used to “daemonize” the main JVM thread, making it less susceptible to various types of interruption (particularly ASTs) that run on the main thread. This is the default for OpenJDK 8 for VSI OpenVMS. The logical name `JAVA$DAEMONIZE_MAIN_THREAD` therefore serves no purpose and defining it will have no effect on JVM operation.

Requirements

OpenJDK 8 for VSI OpenVMS I64 requires the operating system and layered product software versions listed below.

- VSI OpenVMS Version 8.4-2L1 with latest updates

  Specifically, the following update kits must be installed (in the order shown) before installing and using OpenJDK 8 for VIS OpenVMS I64:

  – VMS842L1I_UPDATE-V0100
  – VMS842L1I_DPML-V0100
  – VMS82L1_RTL-V0200

  As noted previously, support for earlier versions of VSI OpenVMS I64 will be provided in a future release.

- VSI TCP/IP, HPE TCP/IP Services for OpenVMS, or the Process Software MultiNet TCP/IP stack for network communication

- The software must be installed on an ODS-5-enabled file system (the software cannot be installed on an ODS-2 file system)
- DECWindows Motif V1.5 or higher (note that this is required even if you are not using the Java AWT, as functionality provided by the Motif libraries is used for some non-AWT functions)
- The OpenVMS internationalization data kit (VMSI18N) must be installed in order to use the Java debugger, `jdb`.
- Kernel support for Thread Manager upcalls must be enabled (do not disable Thread Manager upcalls using either the image flags or the `MULTITHREAD` system parameter)

The reader should be familiar with the installation, configuration, and use of open source products in the VSI OpenVMS environment.

Installation

The kit is provided as a compressed OpenVMS PCSI kit (`VSI-I64VMS-OPENJDK80-V0800-222-1.PCSI$COMPRESSED`) that can be installed by a suitably privileged user using the following command:

```
$ PRODUCT INSTALL OPENJDK80
```

The installation will then proceed as follows (output may differ slightly from that shown, depending on various factors):

```
Performing product kit validation of signed kits ...

The following product has been selected:
    VSI I64VMS OPENJDK80 V8.0-222          Layered Product

Do you want to continue? [YES]

Configuration phase starting ...

You will be asked to choose options, if any, for each selected product and for
any products that may be installed to satisfy software dependency requirements.

Configuring VSI I64VMS OPENJDK80 V8.0-222: OpenJDK for VSI OpenVMS Itanium

    © Copyright 2020 VMS Software Inc.

    VMS Software Inc.

* This product does not have any configuration options.

Execution phase starting ...

The following product will be installed to destination:
    VSI I64VMS OPENJDK80 V8.0-222          DISK$I64SYS:[VMS$COMMON.]

Portion done: 0%...10%...50%...80%...90%...100%

The following product has been installed:
    VSI I64VMS OPENJDK80 V8.0-222          Layered Product

VSI I64VMS OPENJDK80 V8.0-222: OpenJDK for VSI OpenVMS Itanium

    Post-installation tasks are required.

    ******************************************************************************
         Note that the VSI OpenVMS internationalization data kit (VMSI18N)
```

```
          must be installed in order to use the Java debugger, jdb;
          however VMSI18N is not required by OpenJDK for any other purpose.
    ******************************************************************************

    To use OpenJDK Java, users must execute the following command:

        $ @SYS$STARTUP:OPENJDK$SETUP.COM
```

Post-installation tasks

Once the installation process has completed, you may wish to verify that the OpenJDK has installed correctly by running the following commands and verifying that the output is similar to that shown below (there may be some differences in the output, depending on operating system version, installation destination, available memory, locale settings, and so on).

```
$ @sys$startup:openjdk$setup.com
$ java -XshowSettings:all
VM settings:
    Max. Heap Size (Estimated): 2.55G
    Ergonomics Machine Class: server
    Using VM: OpenJDK 64-Bit Server VM

Property settings:
    awt.toolkit = sun.awt.X11.XToolkit
    file.encoding = ISO8859-1
    file.encoding.pkg = sun.io
    file.separator = /
    java.awt.graphicsenv = sun.awt.X11GraphicsEnvironment
    java.awt.headless = true
    java.awt.printerjob = sun.print.PSPrinterJob
    java.class.path = .
    java.class.version = 52.0
    java.endorsed.dirs = /disk$i64sys/sys0/syscommon/openjdk$80/jre/lib/endorsed
    java.ext.dirs = /disk$i64sys/sys0/syscommon/openjdk$80/jre/lib/ext
    java.home = /disk$i64sys/sys0/syscommon/openjdk$80/jre
    java.io.tmpdir = /SYS$SCRATCH
    java.library.path = /usr/lib
    java.runtime.name = OpenJDK Runtime Environment
    java.runtime.version = 1.8.0_222-b05
    java.specification.name = Java Platform API Specification
    java.specification.vendor = Oracle Corporation
    java.specification.version = 1.8
    java.vendor = VMS Software, Inc.
    java.vendor.url = http://www.vmssoftware.com
    java.vendor.url.bug = mailto:support@vmssoftware.com
    java.version = 1.8.0_222
    java.vm.info = mixed mode
    java.vm.name = OpenJDK 64-Bit Server VM
    java.vm.specification.name = Java Virtual Machine Specification
    java.vm.specification.vendor = Oracle Corporation
    java.vm.specification.version = 1.8
    java.vm.vendor = VMS Software, Inc
    java.vm.version = 25.222-b05
    line.separator = \n
    os.arch = ia64
    os.name = OpenVMS
    os.version = XE0V-B4N
    path.separator = :
    sun.arch.data.model = 64
    sun.boot.class.path =
/disk$i64sys/sys0/syscommon/openjdk$80/jre/lib/resources.jar
        /disk$i64sys/sys0/syscommon/openjdk$80/jre/lib/rt.jar
        /disk$i64sys/sys0/syscommon/openjdk$80/jre/lib/sunrsasign.jar
        /disk$i64sys/sys0/syscommon/openjdk$80/jre/lib/jsse.jar
        /disk$i64sys/sys0/syscommon/openjdk$80/jre/lib/jce.jar
```

```
        /disk$i64sys/sys0/syscommon/openjdk$80/jre/lib/charsets.jar
        /disk$i64sys/sys0/syscommon/openjdk$80/jre/lib/jfr.jar
        /disk$i64sys/sys0/syscommon/openjdk$80/jre/classes
    sun.boot.library.path =
    sun.cpu.endian = little
    sun.cpu.isalist =
    sun.io.unicode.encoding = UnicodeLittle
    sun.java.launcher = SUN_STANDARD
    sun.jnu.encoding = ISO8859-1
    sun.management.compiler = HotSpot 64-Bit Server Compiler
    sun.os.patch.level = unknown
    user.dir = /disk$i64sys/sys0/syscommon/openjdk$80
    user.home = /sys$sysdevice/biggles
    user.language = en
    user.name = BIGGLES
    user.timezone =

Locale settings:
    default locale = English
    default display locale = English
    default format locale = English
    available locales = , ar, ar_AE, ar_BH, ar_DZ, ar_EG, ar_IQ, ar_JO,
        ar_KW, ar_LB, ar_LY, ar_MA, ar_OM, ar_QA, ar_SA, ar_SD,
        ar_SY, ar_TN, ar_YE, be, be_BY, bg, bg_BG, ca,
        ca_ES, cs, cs_CZ, da, da_DK, de, de_AT, de_CH,
        de_DE, de_GR, de_LU, el, el_CY, el_GR, en, en_AU,
        en_CA, en_GB, en_IE, en_IN, en_MT, en_NZ, en_PH, en_SG,
        en_US, en_ZA, es, es_AR, es_BO, es_CL, es_CO, es_CR,
        es_CU, es_DO, es_EC, es_ES, es_GT, es_HN, es_MX, es_NI,
        es_PA, es_PE, es_PR, es_PY, es_SV, es_US, es_UY, es_VE,
        et, et_EE, fi, fi_FI, fr, fr_BE, fr_CA, fr_CH,
        fr_FR, fr_LU, ga, ga_IE, hi, hi_IN, hr, hr_HR,
        hu, hu_HU, in, in_ID, is, is_IS, it, it_CH,
        it_IT, iw, iw_IL, ja, ja_JP, ja_JP_JP_#u-ca-japanese, ko, ko_KR,
        lt, lt_LT, lv, lv_LV, mk, mk_MK, ms, ms_MY,
        mt, mt_MT, nl, nl_BE, nl_NL, no, no_NO, no_NO_NY,
        pl, pl_PL, pt, pt_BR, pt_PT, ro, ro_RO, ru,
        ru_RU, sk, sk_SK, sl, sl_SI, sq, sq_AL, sr,
        sr_BA, sr_BA_#Latn, sr_CS, sr_ME, sr_ME_#Latn, sr_RS, sr_RS_#Latn,
sr__#Latn,
        sv, sv_SE, th, th_TH, th_TH_TH_#u-nu-thai, tr, tr_TR, uk,
        uk_UA, vi, vi_VN, zh, zh_CN, zh_HK, zh_SG, zh_TW

Usage: java [-options] class [args...]
           (to execute a class)
   or  java [-options] -jar jarfile [args...]
           (to execute a jar file)
where options include:
    -d32          use a 32-bit data model if available
    -d64          use a 64-bit data model if available
    -server       to select the "server" VM
    -client       is a synonym for the "server" VM  [deprecated]
    -hotspot      is a synonym for the "server" VM  [deprecated]
                  The default VM is server,
                  because you are running on a server-class machine.


    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                  A : separated list of directories, JAR archives,
                  and ZIP archives to search for class files.
    -D<name>=<value>
                  set a system property
    -verbose:[class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -version:<value>
                  Warning: this feature is deprecated and will be removed
```

```
                in a future release.
                require the specified version to run
    -showversion  print product version and continue
    -jre-restrict-search | -no-jre-restrict-search
                Warning: this feature is deprecated and will be removed
                in a future release.
                include/exclude user private JREs in the version search
    -? -help      print this help message
    -X            print help on non-standard options
    -ea[:<packagename>...|:<classname>]
    -enableassertions[:<packagename>...|:<classname>]
                enable assertions with specified granularity
    -da[:<packagename>...|:<classname>]
    -disableassertions[:<packagename>...|:<classname>]
                disable assertions with specified granularity
    -esa | -enablesystemassertions
                enable system assertions
    -dsa | -disablesystemassertions
                disable system assertions
    -agentlib:<libname>[=<options>]
                load native agent library <libname>, e.g. -agentlib:hprof
                see also, -agentlib:jdwp=help and -agentlib:hprof=help
    -agentpath:<pathname>[=<options>]
                load native agent library by full pathname
    -javaagent:<jarpath>[=<options>]
                load Java programming language agent, see java.lang.instrument
    -splash:<imagepath>
                show splash screen with specified image
See http://www.oracle.com/technetwork/java/javase/documentation/index.html for more
details.
```

Assuming that the installation was successful and OpenJDK is functioning as expected, you can now use the OpenJDK to compile and run your Java-based applications.

Contents of the kit

This section provides a general summary of the files and directories that are created by the installation process. For simplicity, it is assumed that OpenJDK was installed using the default location (namely `SYS$COMMON:[OPENJDK$80]`). If you installed the kit in that alternate location, substitute that location for the default while reading the examples in this document.

- Development tools (`SYS$COMMON:[OPENJDK$80.BIN]`)

  This area contains programs that will help you develop, execute, debug, and document programs written in the Java programming language.

- Runtime environment (JRE) (`SYS$COMMON:[OPENJDK$80.JRE]`)

  An implementation of the Runtime Environment (JRE). The runtime environment includes a virtual machine for Java, class libraries, and other files that support the execution of programs written in the Java programming language.

- Additional libraries (`SYS$COMMON:[OPENJDK$80.LIB]`)

  Additional class libraries and support files required by the development tools.

- C header files (`SYS$COMMON:[OPENJDK$80.INCLUDE]`)

Header files that support native-code programming using the Java Native Interface (JNI) and the JVM Tools Interface.

- JNI example code (`SYS$COMMON:[OPENJDK$80.examples.jni]`)

  Simple example code that illustrates using the JNI to call C code from Java and to call Java (invoke a JVM instance) from C.

Known issues and changes

This section provides descriptions of the known issues and limitations that exist in OpenJDK 8u222 for VSI OpenVMS. These issues include the following:

- Use of the `JAVA$READDIR_CASE_DISABLE` logical name:

  Java program performance may be improved by defining the `JAVA$READDIR_CASE_DISABLE` logical name. This logical name allows the user to turn off the case-sensitive filename extraction feature, if it is not needed. In such cases, for ODS-2 filename formats the Java language compiler (`javac`) fails with the "cannot find symbol" error when referencing Java programs with mixed-case class names.

- To set the receive or send buffer size using the `socket.setReceiveBufferSize(int)` or `socket.setSendBufferSize(int)` methods, processes must have one (or more) of `SYSPRV`, `BYPASS`, or `OPER` privileges. This restriction is imposed by TCP/IP services.

  Without one of these process privileges, these Java methods behave as follows:

  - If the receive or send buffer size requested is greater than the default receive or send buffer size set on the system, the methods will fail.
  - If the receive or send buffer size requested is less than or equal to the default receive or send buffer size set on the system, the system returns the default receive or send buffer size.

  Alternatively, you can modify the default buffer size value in the system.

- If the process does not have either of the `SYSPRV`, `BYPASS`, or `OPER` OpenVMS process privileges, invocation of the `DatagramSocket setBroadcast(boolean)` method fails.

- The OpenJDK debugger (`jdb`) fails with "`UTF ERROR`" at start-up if the VMSI18N kit for VSI OpenVMS is not installed.

  The `jdb` utility uses the C RTL `iconv` family of functions to perform UTF-8 character conversions; however the database files required by the RTL for these conversions are not installed by default on all VSI OpenVMS operating system versions that support OpenJDK. To overcome this issue, you must ensure that the VMSI18N kit is installed on your system (note that VMSI18N is installed by default for OpenVMS 8.4-2 and higher).

- OpenJDK will not operate properly after the DCL command `set process/case=sensitive` is executed.

- OpenJDK will not operate correctly if either of the logical names `DECC$FILENAME_UNIX_ONLY` or `DECC$DISABLE_TO_VMS_LOGNAME_TRANSLATION` are defined. Running Java programs with these logical names defined is not supported. Other `DECC$*` logical names (or combinations of such logical names) may also result in incorrect operation of the Java virtual machine.

- Upon encountering a fatal error, the JVM may try to create a log file containing potentially useful information regarding the crash. Unless specified otherwise (using the `-XX:ErrorFile` command line option) such log files will be created in the directory pointed to by the logical name `SYS$SCRATCH` (which is generally your login directory). However, it should be noted that the JVM will report that the file has been created in `/tmp` (the standard scratch area on UNIX and Linux systems). If `tmp` is not defined as a logical name, the OpenVMS C RTL will map `/tmp` to your `SYS$SCRATCH` directory. If `tmp` is defined, the log file may be found in the corresponding directory (assuming the directory exists). For example, the following definition would cause log files to be created in `SYS$SYSDEVICE:[LOGS]` (assuming the user has write permission for this directory):

  ```
  $ define tmp SYS$SYSDEVICE:[LOGS]
  ```

- Splash screens may only work with small image files. For larger image files, the image may be only partially displayed.

- This release of OpenJDK for VSI OpenVMS provides an option that can be used to limit the maximum length of XML names in XML documents processed by the Java API for XML processing (JAXP).

  The maximum length can be changed by using the `-Djdk.xml.maxXMLNameLimit=value` option, where `value` is a positive integer. A value of 0 or a negative number sets no limits (0 is the default). It is also possible to set this limit by adding the following line to your `jaxp.properties` file:

  ```
  jdk.xml.maxXMLNameLimit=value
  ```

- Defining the logical name `JAVA$FILE_OPEN_MODE` to "3" can cause problems with some Java applications and should not be used. Note that this logical name is deprecated and may be removed in future releases.

- The logical name `JAVA$XCOMP_SAFE_MODE` has been added

  In rare situations Java programs run with the `-Xcomp` option can crash with an `ACCVIO` error caused by a race condition between threads. The logical name `JAVA$XCOMP_SAFE_MODE` can be defined (to anything) to prevent this race condition from occurring, at the expense of a small performance penalty.