VSI OpenJDK Version X17.0-13C

Release Notes and Installation Guide

Publication Date: September 2025

Operating System: VSI OpenVMS x86-64 Version 9.2-3

Kit Name: VSI-X86VMS-OPENJDK17-X1700-13C-1.ZIP

VMS Software, Inc. (VSI)
Boston, Massachusetts, USA

VSI OpenJDK Version X17.0-13C Release Notes and Installation Guide

Legal Notice

Table of Contents

1. Introduction

Thank you for your interest in this port of OpenJDK 17 to VSI OpenVMS. The current release of VSI OpenJDK is based on the OpenJDK 17.0.13-ga 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 document contains installation instructions, details of any new features, known issues, and other information specific to this release of the VSI OpenJDK. This kits can be used to develop and run Java-based programs on VSI OpenVMS x86-64 V9.2-3.

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).

2. Fixed Issues and Enhancements

The following known issues were fixed in this release:

- Java Flight Recorder (JFR) support is enabled.
- An issue with inadequate CPU consumption when waiting for user input has been fixed.
- Several important TCP/IP socket bugs have been fixed.
- A issue that was causing a hang when starting a child JVM has been fixed.

3. Compatibility

VSI OpenJDK X17.0-13C is largely compatible with older Java versions for OpenVMS and most existing Java programs will run without change on the OpenJDK platform.

The following list identifies various differences between Oracle Java 6 for HPE OpenVMS and OpenJDK 17 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 17 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 17 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 17 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$17.COM] assuming a default installation) for setting the variable JAVA$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 17 for VSI OpenVMS. The logical name JAVA$DAEMONIZE_MAIN_THREAD therefore serves no purpose and defining it will have no effect on JVM operation.

4. Requirements

VSI OpenJDK X17.0-13C requires the following operating system and layered product software versions:

- VSI OpenVMS x86-64 Version 9.2-3
- 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.

5. Installation

The kits are provided as compressed OpenVMS PCSI kits (VSI-X86VMS-OPENJDK17-X1700-13C-1.ZIP) that can be installed by a suitably privileged user using the following command:

```
$ PRODUCT INSTALL OPENJDK17
```

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

```
Performing product kit validation of signed kits ...
%PCSI-I-VSIVALPASSED, validation of X86$DKA200:[USER]VSI-X86VMS-OPENJDK17-
X1700-13C-1.PCSI$COMPRESSED;1 succeeded

The following product has been selected:
    VSI X86VMS OPENJDK17 X17.0-13C         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 X86VMS OPENJDK17 X17.0-13C: OpenJDK for VSI OpenVMS x86-64

    (c) Copyright 2025 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 X86VMS OPENJDK17 X17.0-13C         DISK$X86SYS:[VMS$COMMON.]

Portion done: 0%...10%...50%...60%...90%...100%

The following product has been installed:
    VSI X86VMS OPENJDK17 X17.0-13C         Layered Product

VSI X86VMS OPENJDK17 X17.0-13C: OpenJDK for VSI OpenVMS x86-64

    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$STARTUP:OPENJDK17$SETUP.COM
$
```

5.1. 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$STARTUP:OPENJDK17$SETUP.COM
$ java -XshowSettings:all
VM settings:
    Max. Heap Size (Estimated): 1.94G
    Using VM: OpenJDK 64-Bit Server VM

Property settings:
    file.encoding = ISO8859-1
    file.separator = /
    java.class.path =
    java.class.version = 61.0
    java.home = /DISK$X86SYS/SYS0/SYSCOMMON/openjdk$17
    java.io.tmpdir = /SYS$SCRATCH
    java.library.path = /usr/lib
    java.runtime.name = OpenJDK Runtime Environment
    java.runtime.version = 17.0.13-ga+11
    java.specification.maintenance.version = 1
    java.specification.name = Java Platform API Specification
    java.specification.vendor = Oracle Corporation
    java.specification.version = 17
```

```
    java.vendor = VMS Software, Inc
    java.vendor.url = http://www.vmssoftware.com
    java.vendor.url.bug = mailto:support@vmssoftware.com
    java.version = 17.0.13-ga
    java.version.date = 2024-10-15
    java.vm.compressedOopsMode = Non-zero based
    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 = 17
    java.vm.vendor = VMS Software, Inc
    java.vm.version = 17.0.13-ga
    jdk.debug = release
    line.separator = \n
    native.encoding = ISO8859-1
    os.arch = x86_64
    os.name = OpenVMS
    os.version = V9.2-3
    path.separator = :
    sun.arch.data.model = 64
    sun.boot.library.path = /DISK$X86SYS/SYS0/SYSCOMMON/openjdk$17/lib
    sun.cpu.endian = little
    sun.io.unicode.encoding = UnicodeLittle
    sun.java.launcher = SUN_STANDARD
    sun.jnu.encoding = ISO8859-1
    sun.management.compiler = HotSpot 64-Bit Tiered Compilers
    sun.stderr.encoding = ISO8859-1
    sun.stdout.encoding = ISO8859-1
    user.dir = /DISK$WORK/User
    user.home = /DISK$WORK/User
    user.language = en
    user.name = USER

Locale settings summary:
    Use "-XshowSettings:locale" option for verbose locale settings options
    default locale = English
    default display locale = English
    default format locale = English
    tzdata version = 2024a

Security settings summary:
    See "java -X" for verbose security settings options
    Security provider static configuration: (in order of preference)
        Provider name: SUN
        Provider name: SunRsaSign
        Provider name: SunEC
        Provider name: SunJSSE
        Provider name: SunJCE
        Provider name: SunJGSS
        Provider name: SunSASL
        Provider name: XMLDSig
        Provider name: SunPCSC
        Provider name: JdkLDAP
        Provider name: JdkSASL
        Provider name: SunPKCS11
    Security TLS configuration (SunJSSE provider):
        Enabled Protocols:
            TLSv1.3
            TLSv1.2

Usage: X86$DKA100:[SYS0.SYSCOMMON.openjdk$17.bin]java.exe;1 [options] <mainclass> [args...]
           (to execute a class)
   or  X86$DKA100:[SYS0.SYSCOMMON.openjdk$17.bin]java.exe;1 [options] -jar <jarfile>
 [args...]
           (to execute a jar file)
   or  X86$DKA100:[SYS0.SYSCOMMON.openjdk$17.bin]java.exe;1 [options] -m <module>[/
<mainclass>] [args...]
       X86$DKA100:[SYS0.SYSCOMMON.openjdk$17.bin]java.exe;1 [options] --module <module>[/
<mainclass>] [args...]
```

```
          (to execute the main class in a module)
  or  X86$DKA100:[SYS0.SYSCOMMON.openjdk$17.bin]java.exe;1 [options] <sourcefile> [args]
          (to execute a single source-file program)

Arguments following the main class, source file, -jar <jarfile>,
-m or --module <module>/<mainclass> are passed as the arguments to
main class.

where options include:

   -cp <class search path of directories and zip/jar files>
   -classpath <class search path of directories and zip/jar files>
   --class-path <class search path of directories and zip/jar files>
                A : separated list of directories, JAR archives,
                and ZIP archives to search for class files.
   -p <module path>
   --module-path <module path>...
                A : separated list of directories, each directory
                is a directory of modules.
   --upgrade-module-path <module path>...
                A : separated list of directories, each directory
                is a directory of modules that replace upgradeable
                modules in the runtime image
   --add-modules <module name>[,<module name>...]
                root modules to resolve in addition to the initial module.
                <module name> can also be ALL-DEFAULT, ALL-SYSTEM,
                ALL-MODULE-PATH.
   --enable-native-access <module name>[,<module name>...]
                modules that are permitted to perform restricted native operations.
                <module name> can also be ALL-UNNAMED.
   --list-modules
                list observable modules and exit
   -d <module name>
   --describe-module <module name>
                describe a module and exit
   --dry-run     create VM and load main class but do not execute main method.
                The --dry-run option may be useful for validating the
                command-line options such as the module system configuration.
   --validate-modules
                validate all modules and exit
                The --validate-modules option may be useful for finding
                conflicts and other errors with modules on the module path.
   -D<name>=<value>
                set a system property
   -verbose:[class|module|gc|jni]
                enable verbose output for the given subsystem
   -version      print product version to the error stream and exit
   --version     print product version to the output stream and exit
   -showversion  print product version to the error stream and continue
   --show-version
                print product version to the output stream and continue
   --show-module-resolution
                show module resolution output during startup
   -? -h -help
                print this help message to the error stream
   --help        print this help message to the output stream
   -X            print help on extra options to the error stream
   --help-extra  print help on extra options to the output stream
   -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:jdwp
```

```
                see also -agentlib:jdwp=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
                HiDPI scaled images are automatically supported and used
                if available. The unscaled image filename, e.g. image.ext,
                should always be passed as the argument to the -splash option.
                The most appropriate scaled image provided will be picked up
                automatically.
                See the SplashScreen API documentation for more information
    @argument files
                one or more argument files containing options
    -disable-@files
                prevent further argument file expansion
    --enable-preview
                allow classes to depend on preview features of this release
To specify an argument for a long option, you can use --<name>=<value> or
--<name> <value>.
```

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.

6. 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$17]). 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$17.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$17.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$17.LIB])

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

- C header files (SYS$COMMON:[OPENJDK$17.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$17.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.

7. Known Issues and Limitations

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

- When trying to work with fonts from the FreeType library or pictures in jpg format, errors may occur.
- The exception is sometimes thrown on JVM exit.

  ```
  %NONAME-F-NOMSG, Message number 05F78414
  Improperly handled condition, image exit forced by last chance handler.
  ```

- The redirect stderr to the file does not work yet.
- 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 mixedcase 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 startup 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.