







**Note**

Before using this information and the product it supports, read the information in "Notices" on page 75.

**Copyright information**

This edition of the user guide applies to the IBM 31-bit SDK for z/OS, Java Technology Edition, Version 6, product 5655-I98, and to the IBM 64-bit SDK for z/OS, Java Technology Edition, Version 6, product 5655-I99, and to all subsequent releases, modifications, and Service Refreshes, until otherwise indicated in new editions.

© Copyright Sun Microsystems, Inc. 1997, 2007, 901 San Antonio Rd., Palo Alto, CA 94303 USA. All rights reserved.

© **Copyright International Business Machines Corporation 2003, 2008.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Preface</b> . . . . .	<b>v</b>	Determining whether your application is running on a 31-bit or 64-bit JVM . . . . .	32
<b>Chapter 1. Overview</b> . . . . .	<b>1</b>	How the JVM processes signals. . . . .	32
Version compatibility . . . . .	1	Signals used by the JVM . . . . .	33
Migrating from other IBM JVMs. . . . .	1	Linking a native code driver to the signal-chaining library. . . . .	34
Supported hardware. . . . .	2	Writing JNI applications . . . . .	35
The z/OS batch toolkit . . . . .	2	Native formatting of Java types long, double, float. . . . .	36
<b>Chapter 2. Contents of the SDK and Runtime Environment.</b> . . . . .	<b>3</b>	Support for thread-level recovery of blocked connectors. . . . .	36
Contents of the Runtime Environment. . . . .	3	CORBA support. . . . .	37
Contents of the SDK. . . . .	4	System properties for tracing the ORB . . . . .	38
<b>Chapter 3. Installing and configuring the SDK</b> . . . . .	<b>7</b>	System properties for tuning the ORB . . . . .	39
Working with BPXPRM settings . . . . .	7	Java security permissions for the ORB . . . . .	39
Setting the region size . . . . .	8	ORB implementation classes. . . . .	40
Setting MEMLIMIT . . . . .	8	RMI over IIOP . . . . .	40
Setting LE run-time options . . . . .	8	Implementing the Connection Handler Pool for RMI . . . . .	40
Setting LE 31-bit run-time options . . . . .	9	Enhanced BigDecimal . . . . .	41
Setting LE 64-bit run-time options . . . . .	10	Working in a multiple network stack environment . . . . .	41
Marking failures. . . . .	10	Using IBMJCECCA . . . . .	41
Setting the path . . . . .	10	Support for XToolkit . . . . .	43
Setting the class path . . . . .	11	<b>Chapter 6. Applet Viewer</b> . . . . .	<b>45</b>
<b>Chapter 4. Running Java applications</b> . . . . .	<b>13</b>	Distributing Java applications . . . . .	45
The java and javaw commands . . . . .	13	<b>Chapter 7. Class data sharing between JVMs</b> . . . . .	<b>47</b>
Obtaining version information . . . . .	13	Overview of class data sharing . . . . .	47
Specifying Java options and system properties. . . . .	14	Class data sharing command-line options . . . . .	48
Standard options . . . . .	14	Creating, populating, monitoring, and deleting a cache . . . . .	52
Globalization of the java command . . . . .	16	Performance and memory consumption . . . . .	52
The Just-In-Time (JIT) compiler. . . . .	16	Considerations and limitations of using class data sharing. . . . .	53
Disabling the JIT . . . . .	16	Cache size limits. . . . .	53
Enabling the JIT . . . . .	17	Required APAR for Shared Classes . . . . .	53
Determining whether the JIT is enabled . . . . .	17	Working with BPXPRMxx settings. . . . .	53
Specifying garbage collection policy . . . . .	17	Runtime bytecode modification. . . . .	54
Garbage collection options . . . . .	18	Operating system limitations . . . . .	54
More effective heap usage using compressed references . . . . .	18	Using SharedClassPermission . . . . .	55
Pause time . . . . .	19	Adapting custom classloaders to share classes. . . . .	55
Pause time reduction . . . . .	19	<b>Chapter 8. Service and support for independent software vendors</b> . . . . .	<b>57</b>
Environments with very full heaps . . . . .	20	<b>Chapter 9. Accessibility</b> . . . . .	<b>59</b>
Euro symbol support . . . . .	20	Keyboard traversal of JComboBox components in Swing . . . . .	59
Using Indian and Thai input methods . . . . .	20	<b>Chapter 10. Any comments on this user guide?</b> . . . . .	<b>61</b>
<b>Chapter 5. Developing Java applications</b> . . . . .	<b>21</b>	<b>Appendix A. Nonstandard options</b> . . . . .	<b>63</b>
Using XML . . . . .	21		
Migrating to the XL-TXE-J . . . . .	23		
XML reference information . . . . .	25		
Debugging Java applications. . . . .	30		
Java Debugger (JDB) . . . . .	30		
Selective debugging . . . . .	31		

**Appendix B. Known limitations . . . . 71**  
**Notices . . . . . 75**

Trademarks . . . . . 76

---

## Preface

This user guide provides general information about the IBM® 64-bit SDK for z/OS®, Java™ Technology Edition, Version 6 and specific information about any differences in the IBM implementation compared with the Sun implementation.

Read this user guide in conjunction with the more extensive documentation on the Sun Web site: <http://java.sun.com>.

The Diagnostics Guide provides more detailed information about the IBM Virtual Machine for Java.

This user guide is part of a release and is applicable only to that particular release. Make sure that you have the user guide appropriate to the release you are using.

The terms "Runtime Environment" and "Java Virtual Machine" are used interchangeably throughout this user guide.

| Technical changes made for this version of the user guide, other than minor or  
| obvious ones, are indicated by blue chevrons when viewing in an Information  
| Center, in red with vertical bars to the left of the changes when viewing in HTML  
| or in a color-printed copy, or by vertical bars to the left of the changes when  
| viewing as a PDF.

The Program Code is not designed or intended for use in real-time applications such as (but not limited to) the online control of aircraft, air traffic, aircraft navigation, or aircraft communications; or in the design, construction, operation, or maintenance of any nuclear facility.



---

## Chapter 1. Overview

The IBM SDK is a development environment for writing and running applets and applications that conform to the Java 6 Core Application Program Interface (API).

---

### Version compatibility

In general, any application that ran with a previous version of the SDK should run correctly with the IBM 64-bit SDK for z/OS, v6. Classes compiled with this release are not guaranteed to work on previous releases.

For information about compatibility issues between releases, see the Sun web site at:

<http://java.sun.com/javase/6/webnotes/compatibility.html>

<http://java.sun.com/j2se/5.0/compatibility.html>

<http://java.sun.com/j2se/1.4/compatibility.html>

<http://java.sun.com/j2se/1.3/compatibility.html>

If you are using the SDK as part of another product (for example, IBM WebSphere® Application Server), and you upgrade from a previous level of the SDK, perhaps v5.0, serialized classes might not be compatible. However, classes are compatible between service refreshes.

---

### Migrating from other IBM JVMs

From Version 5.0, the IBM Runtime Environment for z/OS contains new versions of the IBM Virtual Machine for Java and the Just-In-Time (JIT) compiler.

If you are migrating from an older IBM Runtime Environment, note that:

- The JVM shared library `libjvm.so` is now stored in `jre/lib/<arch>/j9vm` and `jre/lib/<arch>/classic`.
- From Version 5.0 onwards, the JVM Monitoring Interface (JVMMI) is no longer available. You must rewrite JVMMI applications to use the JVM Tool Interface (JVMTI) instead. The JVMTI is not functionally the equivalent of JVMMI. For information about JVMTI, see <http://java.sun.com/javase/6/docs/technotes/guides/jvmti/> and the Diagnostics Guide.
- From Version 5.0 onwards, the implementation of JNI conforms to the JNI specification, but differs from the Version 1.4.2 implementation. It returns copies of objects rather than pinning the objects. This difference can expose errors in JNI application code. For information about debugging JNI code, see `-Xcheck:jni` in Appendix A, "Nonstandard options," on page 63.
- From Version 5.0 onwards, the format and content of garbage collector verbose logs obtained using `-verbose:gc` have changed. The data is now formatted as XML. The data content reflects the changes to the implementation of garbage collection in the JVM, and most of the statistics that are output have changed. You must change any programs that process the verbose GC output so that they will work with the new format and data. See the Diagnostics Guide for an example of the new verbose GC data.

- SDK 1.4 versions of the IBM JRE included JVM specific classes in a file called core.jar. From Version 5.0 onwards, these are included in a file called vm.jar.
- From Version 6, JVM classes are held in multiple JAR files in the jre/lib directory. This replaces the single rt.jar and core.jar from earlier releases.
- For additional industry compatibility information, see Sun's Java 6 Compatibility Documentation: <http://java.sun.com/javase/6/webnotes/compatibility.html>
- For additional deprecated API information, see Sun's Java 6 Deprecated API List: <http://java.sun.com/javase/6/docs/api/deprecated-list.html>
- All z/OS Java SDK program products can be installed and executed on the same z/OS system. They are independent program products and can coexist in any combination.
- The serial reusability feature of the IBM SDK for z/OS, version 1.4.2 (31-bit) and earlier, invoked using **-Xresettable**, is not supported. If you specify **-Xresettable** the JVM will issue an error message and will not start. The **-Xinitacsh** and **-Xinitth** options, which allowed heap sizes to be specified for the resettable JVM, are ignored. You can share data between JVMs in an address space (the old **-Xjvmset** and **-Xscmax** options) using Chapter 7, "Class data sharing between JVMs," on page 47, a new facility for Version 5.0. If you specify **-Xjvmset** or **-Xscmax** the JVM will issue an error message and will not start.
- The system property **os.arch** for IBM SDK for z/OS, version 1.4.2 (31-bit) versions and earlier had a value of **390**. From Java 5.0 onwards, the value of **os.arch** is **s390**.
- Tracing class dependencies, invoked using **-verbose:Xclassdep**, is not supported. If you specify **-verbose:Xclassdep**, the JVM will issue an error message and will not start.

---

## Supported hardware

The z/OS 31-bit and 64-bit SDKs run on System z9<sup>®</sup> and zSeries<sup>®</sup> hardware.

The SDKs run on the following servers or equivalents:

- z9-109
- z990
- z900
- z890
- z800

---

## The z/OS batch toolkit

From Version 5, Service Refresh 3 onwards, the z/OS products have been enhanced with the JZOS batch toolkit. This toolkit addresses many of the functional and environmental shortcomings in the previous Java batch capabilities on z/OS. The enhancements include a native launcher for running Java applications directly as batch jobs or started tasks and a set of Java methods that make access to traditional z/OS data and key system services directly available from Java applications. Additional system services include console communication, multiline WTO (write to operator), and return code passing capability. For more details, see <http://www.ibm.com/servers/eserver/zseries/software/java/jzos/overview.html> and <http://www.ibm.com/servers/eserver/zseries/software/java/>.

---

## Chapter 2. Contents of the SDK and Runtime Environment

The SDK contains several development tools and a Java Runtime Environment (JRE). This section describes the contents of the SDK tools and the Runtime Environment.

Applications written entirely in Java must have **no** dependencies on the IBM SDK's directory structure (or files in those directories). Any dependency on the SDK's directory structure (or the files in those directories) might result in application portability problems.

The user guides, Javadoc, and the accompanying copyright files, javadoc, and demo directory are the only documentation included in this SDK for z/OS. You can view Sun's software documentation by visiting the Sun Web site, or you can download Sun's software documentation package from the Sun Web site: <http://java.sun.com>. Additional z/OS related information is available on the z/OS Java web site at <http://www.ibm.com/servers/eserver/zseries/software/java/>.

---

### Contents of the Runtime Environment

A list of classes, tools, and other files that you can use with the standard Runtime Environment.

- **Core Classes** - These are the compiled class files for the platform and must remain zipped for the compiler and interpreter to access them. Do not modify these classes; instead, create subclasses and override where you need to.
- **Trusted root certificates from certificate signing authorities** - These certificates are used to validate the identity of signed material. The IBM Runtime Environment for Java contains an expired GTE CyberTrust Certificate for compatibility reasons. This certificate will be removed for Version 7 of the SDK. See <http://www.ibm.com/support/docview.wss?uid=swg21225464> for more information.
- **JRE tools** - The following tools are part of the Runtime Environment and are in the `/usr/lpp/java/J6.0[_64]/jre/bin` directory unless otherwise specified.

#### **ikeyman (iKeyman GUI utility)**

Allows you to manage keys, certificates, and certificate requests. For more information see the accompanying Security Guide and [http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/security/50/GSK7c\\_SSL\\_IKM\\_Guide.pdf](http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/security/50/GSK7c_SSL_IKM_Guide.pdf). The SDK also provides a command-line version of this utility.

#### **java (Java Interpreter)**

Runs Java classes. The Java Interpreter runs programs that are written in the Java programming language.

#### **javaw (Java Interpreter)**

Runs Java classes in the same way as the **java** command does, but does not use a console window.

#### **jextract (Dump extractor)**

Converts a system-produced dump into a common format that can be used by `jdumpview`. For more information, see `jdumpview`.

**keytool (Key and Certificate Management Tool)**

Manages a keystore (database) of private keys and their associated X.509 certificate chains that authenticate the corresponding public keys.

**kinit**

Obtains and caches Kerberos ticket-granting tickets.

**klist**

Displays entries in the local credentials cache and key table.

**ktab**

Manages the principal names and service keys stored in a local key table.

**pack200**

Transforms a JAR file into a compressed pack200 file using the Java gzip compressor.

**policytool (Policy File Creation and Management Tool)**

Creates and modifies the external policy configuration files that define your installation's Java security policy.

**rmid (RMI activation system daemon)**

Starts the activation system daemon so that objects can be registered and activated in a Java virtual machine (JVM).

**rmiregistry (Java remote object registry)**

Creates and starts a remote object registry on the specified port of the current host.

**rnameserv (Common Object Request Broker Architecture (CORBA) transient naming service)**

Starts the CORBA transient naming service.

**unpack200**

Transforms a packed file produced by pack200 into a JAR file.

---

## Contents of the SDK

A list of tools and reference information that is included with the standard SDK.

**The following tools are part of the SDK and are located in the `/usr/lpp/java/J6.0[_64]/bin` directory:**

**appletviewer (Java Applet Viewer)**

Tests and runs applets outside a Web browser.

**apt (Annotation Processing Tool)**

Finds and executes annotation processors based on the annotations present in the set of specified source files being examined.

**extcheck (Extcheck utility)**

Detects version conflicts between a target jar file and currently-installed extension jar files.

**hwkeytool**

Manages a keystore of private keys and their associated X.509 certificate chains authenticating the corresponding public keys.

**idlj (IDL to Java Compiler)**

Generates Java bindings from a given IDL file.

**ikeycmd (iKeyman command-line utility)**

Allows you to manage keys, certificates, and certificate requests from the

command line. For more information see the accompanying *Security Guide* and <http://www.ibm.com/developerworks/java/jdk/security/>.

**jar (Java Archive Tool)**

Combines multiple files into a single Java Archive (JAR) file.

**jarsigner (JAR Signing and Verification Tool)**

Generates signatures for JAR files and verifies the signatures of signed JAR files.

**java (Java Interpreter)**

Runs Java classes. The Java Interpreter runs programs that are written in the Java programming language.

**java-rmi.cgi (HTTP-to-CGI request forward tool)**

Accepts RMI-over-HTTP requests and forwards them to an RMI server listening on any port.

**javac (Java Compiler)**

Compiles programs that are written in the Java programming language into bytecodes (compiled Java code).

**javadoc (Java Documentation Generator)**

Generates HTML pages of API documentation from Java source files.

**javah (C Header and Stub File Generator)**

Enables you to associate native methods with code written in the Java programming language.

**javap (Class File Disassembler)**

Disassembles compiled files and can print a representation of the bytecodes.

**javaw (Java Interpreter)**

Runs Java classes in the same way as the **java** command does, but does not use a console window.

**jconsole (JConsole Monitoring and Management Tool)**

Monitors local and remote JVMs using a GUI. JMX-compliant.

**jdumpview (Cross-platform dump formatter)**

Analyzes dumps. For more information, see the Diagnostics Guide.

**keytool (Key and Certificate Management Tool)**

Manages a keystore (database) of private keys and their associated X.509 certificate chains that authenticate the corresponding public keys.

**native2ascii (Native-To-ASCII Converter)**

Converts a native encoding file to an ASCII file that contains characters encoded in either Latin-1 or Unicode, or both.

**policytool (Policy File Creation and Management Tool)**

Creates and modifies the external policy configuration files that define your installation's Java security policy.

**rmic (Java Remote Method Invocation (RMI) Stub Converter)**

Generates stubs, skeletons, and ties for remote objects. Includes RMI over Internet Inter-ORB Protocol (RMI-IIOP) support.

**rmid (RMI activation system daemon)**

Starts the activation system daemon so that objects can be registered and activated in a Java virtual machine (JVM).

**rmiregistry (Java remote object registry)**

Creates and starts a remote object registry on the specified port of the current host.

**schemagen**

Creates a schema file for each namespace referenced in your Java classes.

**serialver (Serial Version Command)**

Returns the serialVersionUID for one or more classes in a format that is suitable for copying into an evolving class.

**tnameserv (Common Object Request Broker Architecture (CORBA) transient naming service)**

Starts the CORBA transient naming service.

**wsgen**

Generates JAX-WS portable artifacts used in JAX-WS web services.

**wsimport**

Generates JAX-WS portable artifacts from a WSDL file.

**xjc**

Compiles XML Schema files.

**Include Files**

C headers for JNI programs.

**Demos**

The demo directory contains a number of subdirectories containing sample source code, demos, applications, and applets that you can use. From Version 6, the RMI-IIOP demo is not included with the SDK.

**copyright**

The copyright notice for the SDK for z/OS software.

---

## Chapter 3. Installing and configuring the SDK

See the z/OS Web site for instructions about ordering, downloading, installing, and verifying the SDK.

<http://www.ibm.com/servers/eserver/zseries/software/java/>

---

### Working with BPXPRM settings

Some of the parameters in PARMLIB member **BPXPRMxx** might affect successful Java operation by imposing limits on resources that are required.

The parameters described here do not cover those required for Class data sharing. See “Considerations and limitations of using class data sharing” on page 53 for the parameters required for Class data sharing.

Enter the z/OS operator command `D OMVS,0` to display the current **BPXPRMxx** settings. Enter the command `D OMVS,L` to show the highwater usage for some of the limits. If you configure the **BPXPRMxx** **LIMMSG** parameter to activate the support, **BPXIxxxI** messages are displayed when the usage approaches and reaches the limits. You can use the `SETOMVS` command to change the settings without requiring an IPL.

Other products might impose their own requirements, but for Java the important parameters and their suggested minimum values are:

*Table 1. BPXPRM settings*

Parameter	Value
MAXPROCSYS	900
MAXPROCUSER	512
MAXUIDS	500
MAXTHREADS	10 000
MAXTHREADTASKS	5 000
MAXASSIZE	2 147 483 647
MAXCPUIME	2 147 483 647
MAXMMAPAREA	40 960
IPCSEMNIDS	500
IPCSEMNSEMS	1 000
SHRLIBRGNSIZE	67 108 864
SHRLIBMAXPAGES	4 096

The lower of **MAXTHREADS** and **MAXTHREADTASKS** limits the number of threads that can be created by a Java process.

**MAXMMAPAREA** limits the number of 4K pages that are available for memory-mapped jar files through the environment variable **JAVA\_MMAP\_MAXSIZE**.

**SHRLIBRGNSIZE** controls how much storage is reserved in each address space for mapping shared DLLs that have the +l extended attribute set. If this storage space is exceeded, DLLs are loaded into the address space instead of using a single copy of USS storage that is shared between the address spaces. Some of the Java SDK DLLs have the +l extended attribute set. The z/OS command D OMVS,L shows the **SHRLIBRGNSIZE** size and peak usage. If this size is set to a much higher value than is needed, Java might have problems acquiring native (z/OS 31-bit) storage, which can cause a z/OS abend, such as 878-10, or a Java OutOfMemoryError.

**SHRLIBMAXPAGES** is only available in z/OS 1.7 and earlier releases. This parameter is similar to **SHRLIBRGNSIZE** except that it is a number of 4K pages and only applies to DLLs that have the .so suffix, but without the +l extended attribute. This feature requires Extended System Queue Area (ESQA), therefore you should use it carefully.

For further information about the use of these parameters, refer to the z/OS MVS™ *Initialization and Tuning Reference (SA22-7592)* at <http://publibz.boulder.ibm.com/epubs/pdf/iea2e280.pdf> and the z/OS *Unix System Services Planning Guide (GA22-7800)* at <http://publibz.boulder.ibm.com/epubs/pdf/bpxzb280.pdf>.

---

## Setting the region size

Java requires a suitable z/OS region size to operate successfully. It is suggested that you do not restrict the region size, but allow Java to use what is necessary. Restricting the region size might cause failures with storage-related error messages or abends such as 878-10.

The region size might be affected by the following factors:

- **JCL REGION** parameter
- **BPXPRMxx MAXASSIZE** parameter
- RACF OMVS segment **ASSIZEMAX** parameter
- IEFUSI

You might want to exclude OMVS from using the IEFUSI exit by setting **SUBSYS(OMVS,NOEXITS)** in PARMLIB member SMFPRMxx.

For further information refer to the documentation of the host product under which Java runs.

---

## Setting MEMLIMIT

z/OS uses region sizes to determine the amount of storage available to running programs. For the 64-bit product, set the **MEMLIMIT** parameter to include at least 1024 MB plus the largest expected JVM heap size value **-Xmx**.

See *Limiting Storage use above the bar in z/Architecture* for information about setting the **MEMLIMIT** parameter: <http://www.ibm.com/support/techdocs/atmsastr.nsf/WebIndex/FLASH10165>.

---

## Setting LE run-time options

LE run-time options can affect both performance and storage usage. The optimum settings will vary according to the host product and the Java application itself, but it is important to have good general settings.

The LE run-time options are documented in *Language Environment® Programming Reference (SA22-7562)* at <http://publibz.boulder.ibm.com/epubs/pdf/ceea3180.pdf>.

Java and other products that are written in C or C++ might have LE run-time options embedded within the main programs by using `#pragma runopts`. These options are chosen to provide suitable default values that assist the performance in a typical environment. Any run-time overrides that you set might alter these values in a way that degrades the performance of Java or the host product. The host product's documentation might provide details of the product's default settings. Changes to the product's `#pragma runopts` might occur as a result of version or release changes. For details of how LE chooses the order of precedence of its run-time options, refer to the *Language Environment Programming Guide (SA22-7561)* at <http://publibz.boulder.ibm.com/epubs/pdf/ceea2180.pdf>.

Use the LE run-time option `RPTOPTS(ON)` as an override to write the options that are in effect, to `stderr` on termination. Refer to the host product documentation and the *Language Environment Programming Guide (SA22-7561)* at <http://publibz.boulder.ibm.com/epubs/pdf/ceea2180.pdf> for details of how to supply LE run-time overrides. Before creating run-time overrides, run the application without overrides, to determine the existing options based on LE defaults and `#pragma` settings.

To tune the options, use the LE run-time option `RPTSTG(ON)` as an override, but be aware that there is an overhead when you use this option. The output for `RPTSTG(ON)` also goes to `stderr` on termination. The *Language Environment Debugging Guide (GA22-7560)* at <http://publibz.boulder.ibm.com/epubs/pdf/ceea1180.pdf> explains `RPTSTG(ON)` output.

## Setting LE 31-bit run-time options

There are a number of LE 31-bit options that are important for successful Java operation.

The following are the important options:

- **ANYHEAP**
- **HEAP**
- **HEAPPOOLS**
- **STACK**
- **STORAGE**
- **THREADSTACK**

You can change any, or all, of these options, however if you set the wrong values this might affect performance. The following values are a suggested starting point for these options:

`ANYHEAP(2M,1M,ANY,FREE)`

`HEAP(80M,4M,ANY,KEEP)`

`HEAPPOOLS(ON,8,10,32,10,128,10,256,10,1024,10,2048,10,0,10,0, 10,0,10,0,10,0,10,0,10)`

`STACK(64K,16K,ANY,KEEP,128K,128K)`

`STORAGE(NONE,NONE,NONE,0K)`

`THREADSTACK(OFF,64K,16K,ANY,KEEP,128K,128K)`

**ANYHEAP** and **HEAP** initial allocations (parameter 1) might be too large for transaction-based systems such as CICS®. Java applications that use many hundreds of threads might need to adjust the **STACK** initial and increment allocations (parameters 1, 2, 5 and 6) based on the `RPTSTG(ON)` output, which shows the maximum stack sizes that are used by a thread within the application.

HEAPOOLS(ON) should improve performance, but the LE-supplied default settings for the cell size and percentage pairs are not optimized for the best performance or storage usage.

For additional information, including how to set the LE run-time options, refer to:

- the Diagnostics Guide
- the *z/OS Language Environment Programming Reference (SA22-7562)* at <http://publibz.boulder.ibm.com/epubs/pdf/ceea3180.pdf>
- the *z/OS Language Environment Programming Guide (SA22-7561)* at <http://publibz.boulder.ibm.com/epubs/pdf/ceea2180.pdf>
- the host product documentation

## Setting LE 64-bit run-time options

There are 64-bit versions of some of the run-time options.

The following are the 64-bit options:

- **HEAP64**
- **HEAPOOLS64**
- **STACK64**
- **THREADSTACK64**

A suggested start point for HEAP64 as an override is HEAP64(512M,4M,KEEP,16M,4M,KEEP,0K,0K,FREE).

The following are LE defaults, and should be appropriate:

STACK64(1M,1M,128M)

THREADSTACK64(OFF,1M,1M,128M)

HEAPOOLS64(OFF,8,4000,32,2000,128,700,256,350.1024,100,2048,50,3072,50,4096,50,8192,25,16384,10,32768,5,65536,5)

Before you set an override for HEAPPOOLS64, use **RPTOPTS(ON)** or **RPTSTG(ON)** and check the result of #pragma runopts. Check this because the host product might have already set cell sizes and numbers that are known to produce good performance.

Also, these settings are dependant on a suitable **MEMLIMIT** setting. Based on these suggested LE 64-bit run-time options, the JVM requirement is a minimum of 512 MB as set for **HEAP64** (which should include **HEAPOOLS64**), plus an initial value for **STACK64** of 1 MB times the expected maximum number of concurrent threads, plus the largest expected JVM heap **-Xmx** value.

---

## Marking failures

The Java launcher can mark the z/OS Task Control Block (TCB) with an ABEND code when the launcher fails to load the VM or is terminated by an uncaught exception. To turn on TCB marking, set the environment variable **IBM\_JAVA\_ABEND\_ON\_FAILURE=Y**.

By default, the Java launcher will not mark the TCB.

---

## Setting the path

If you alter the **PATH** environment variable, you will override any existing Java launchers in your path.

## About this task

The **PATH** environment variable enables z/OS to find programs and utilities, such as `javac`, `java`, and `javadoc`, from any current directory. To display the current value of your **PATH**, type the following at a command prompt:

```
echo $PATH
```

To add the Java launchers to your path:

1. Edit the shell startup file in your home directory (typically `.bashrc`, depending on your shell) and add the absolute paths to the **PATH** environment variable; for example:

```
export PATH=/usr/lpp/java/J6.0[_64]/bin:/usr/lpp/java/J6.0[_64]/jre/bin:$PATH
```

2. Log on again or run the updated shell script to activate the new **PATH** environment variable.

## Results

After setting the path, you can run a tool by typing its name at a command prompt from any directory. For example, to compile the file `Myfile.Java`, at a command prompt, type:

```
javac Myfile.Java
```

---

## Setting the class path

The class path tells the SDK tools, such as `java`, `javac`, and `javadoc`, where to find the Java class libraries.

### About this task

You need to set the class path explicitly only if:

- You require a different library or class file, such as one that you develop, and it is not in the current directory.
- You change the location of the `bin` and `lib` directories and they no longer have the same parent directory.
- You plan to develop or run applications using different runtime environments on the same system.

To display the current value of your **CLASSPATH** environment variable, type the following command at a shell prompt:

```
echo $CLASSPATH
```

If you develop and run applications that use different runtime environments, including other versions that you have installed separately, you must set the **CLASSPATH** and **PATH** explicitly for each application. If you run multiple applications simultaneously and use different runtime environments, each application must run in its own shell prompt.



---

## Chapter 4. Running Java applications

Java applications can be started using the java launcher or through JNI. Settings are passed to a Java application using command-line arguments, environment variables, and properties files.

---

### The java and javaw commands

An overview of the java and javaw commands.

#### Purpose

The java and javaw tools launch a Java application by starting a Java Runtime Environment and loading a specified class.

The javaw command is identical to java, and is supported on z/OS for compatibility with other platforms.

#### Usage

The JVM searches for the initial class (and other classes that are used) in three sets of locations: the bootstrap class path, the installed extensions, and the user class path. The arguments that you specify after the class name or jar file name are passed to the main function.

The java and javaw commands have the following syntax:

```
java [ options ] <class> [ arguments ... ]
java [ options ] -jar <file.jar> [ arguments ... ]
javaw [ options ] <class> [ arguments ... ]
javaw [ options ] -jar <file.jar> [ arguments ... ]
```

#### Parameters

[options]

Command-line options to be passed to the runtime environment.

<class>

Startup class. The class must contain a main() method.

<file.jar>

Name of the jar file to invoke. It is used only with the **-jar** option. The named jar file must contain class and resource files for the application, with the startup class indicated by the Main-Class manifest header.

[arguments ...]

Command-line arguments to be passed to the main() function of the startup class.

### Obtaining version information

You obtain The IBM build and version number for your Java installation using the **-version** option. You can also obtain version information for all jar files on the class path by using the **-Xjarversion** option.

1. Open a shell prompt.
2. Type the following command:

```
java -version
```

You will see information similar to:

```
java version "1.6.0-internal"  
Java(TM) SE Runtime Environment (build 20070405_01)  
IBM J9 VM (build 2.4, J2RE 1.6.0 IBM J9 2.4 z/OS s390x-64 jvmmz6460-20070326_12091 (JIT enabled)  
J9VM - 20070326_12091_bHdSMr  
JIT - dev_20070326_1800  
GC - 20070319_AA)
```

Exact build dates and versions will change.

## What to do next

You can also list the version information for all available jar files on the class path, the boot class path, and in the extensions directory. Type the following command:

```
java -Xjarversion
```

You will see information similar to:

```
...  
/usr/lpp/java/J6.0[_64]/jre/lib/ext/ibmpkcs11impl.jar VERSION: 1.0 build_20070125  
/usr/lpp/java/J6.0[_64]/jre/lib/ext/dfjview.jar  
/usr/lpp/java/J6.0[_64]/jre/lib/ext/xmlencfw.jar VERSION: 1.00, 20061011 LEVEL: -20061011  
...
```

The information available varies for each jar file and is taken from the **Implementation-Version** and **Build-Level** properties in the manifest of the jar file.

## Specifying Java options and system properties

You can specify Java options and system properties on the command line, by using an options file, or by using an environment variable.

### About this task

These methods of specifying Java options are listed in order of precedence. Rightmost options on the command line have precedence over leftmost options; for example, if you specify:

```
java -Xint -Xjit myClass
```

The **-Xjit** option takes precedence.

1. By specifying the option or property on the command line. For example:  

```
java -Dmysysprop1=tcPIP -Dmysysprop2=wait -Xdisablejavadump MyJavaClass
```
2. By creating a file that contains the options, and specifying it on the command line using **-Xoptionsfile=<file>**.
3. By creating an environment variable called **IBM\_JAVA\_OPTIONS** containing the options. For example:  

```
export IBM_JAVA_OPTIONS="-Dmysysprop1=tcPIP -Dmysysprop2=wait -Xdisablejavadump"
```

## Standard options

The definitions for the standard options.

See Appendix A, "Nonstandard options," on page 63 for information about nonstandard (-X) options.

- agentlib:***<libname>*[=*<options>*]  
Loads a native agent library *<libname>*; for example **-agentlib:hprof**. For more information, specify **-agentlib:jdwp=help** and **-agentlib:hprof=help** on the command line.
- agentpath:***libname*[=*<options>*]  
Loads a native agent library by full path name.
- cp** *<directories and zip or jar files separated by :>*  
Sets the search path for application classes and resources. If **-classpath** and **-cp** are not used and the **CLASSPATH** environment variable is not set, the user class path is, by default, the current directory (.).
- classpath** *<directories and zip or jar files separated by :>*  
Sets the search path for application classes and resources. If **-classpath** and **-cp** are not used and the **CLASSPATH** environment variable is not set, the user class path is, by default, the current directory (.).
- D***<property name>*=*<value>*  
Sets a system property.
- help or -?**  
Prints a usage message.
- javaagent:***<jarpath>*[=*<options>*]  
Load a Java programming language agent. For more information, see the `java.lang.instrument` API documentation.
- jre-restrict-search**  
Include user private JREs in the version search.
- no-jre-restrict-search**  
Exclude user private JREs in the version search.
- showversion**  
Prints product version and continues.
- verbose:***<option>*[,*<option>*...]  
Enables verbose output. Separate multiple options using commas. The available options are:
  - class**  
Writes an entry to stderr for each class that is loaded.
  - gc** Writes verbose garbage collection information to stderr. Use **-Xverbosegclog** to control the output. See the Diagnostics Guide for more information.
  - jni**  
Writes information to stderr describing the JNI services called by the application and JVM.
  - sizes**  
Writes information to stderr describing the active memory usage settings.
  - stack**  
Writes information to stderr describing the Java and C stack usage for each thread.
- version**  
Prints product version.
- version:***<value>*  
Requires the specified version to run, for example "1.5".

-X Prints help on nonstandard options.

## Globalization of the java command

The java and javaw launchers accept arguments and class names containing any character that is in the character set of the current locale. You can also specify any Unicode character in the class name and arguments by using Java escape sequences.

To do this, use the **-Xargencoding** command-line option.

### **-Xargencoding**

Use argument encoding. To specify a Unicode character, use escape sequences in the form `\u####`, where # is a hexadecimal digit (0 to 9, A to F).

### **-Xargencoding:utf8**

Use UTF8 encoding.

### **-Xargencoding:latin**

Use ISO8859\_1 encoding.

For example, to specify a class called HelloWorld using Unicode encoding for both capital letters, use this command:

```
java -Xargencoding '\u0048ello\u0057orld'
```

The java and javaw commands provide translated messages. These messages differ based on the locale in which Java is running. The detailed error descriptions and other debug information that is returned by java is in English.

---

## The Just-In-Time (JIT) compiler

The IBM Just-In-Time (JIT) compiler dynamically generates machine code for frequently used bytecode sequences in Java applications and applets during their execution. The JIT v6 compiler delivers new optimizations as a result of compiler research, improves optimizations implemented in previous versions of the JIT, and provides better hardware exploitation.

Both the IBM SDK and Runtime Environment include the JIT, which is enabled by default in user applications and SDK tools. Normally, you do not invoke the JIT explicitly; the compilation of Java bytecode to machine code occurs transparently. You can disable the JIT to help isolate a problem. If a problem occurs when executing a Java application or an applet, you can disable the JIT to help isolate the problem. Disabling the JIT is a temporary measure only; the JIT is required to optimize performance.

For more information about the JIT, see the Diagnostics Guide.

## Disabling the JIT

The JIT can be disabled in a number of different ways. Both command-line options override the **JAVA\_COMPILER** environment variable.

### About this task

Turning off the JIT is a temporary measure that can help isolate problems when debugging Java applications.

- Set the **JAVA\_COMPILER** environment variable to **NONE** or the empty string before running the java application. Type the following at a shell prompt:

```
export JAVA_COMPILER=NONE
```

- Use the **-D** option on the JVM command line to set the **java.compiler** property to **NONE** or the empty string. Type the following at a shell prompt:

```
java -Djava.compiler=NONE <class>
```

- Use the **-Xint** option on the JVM command line. Type the following at a shell prompt:

```
java -Xint <class>
```

## Enabling the JIT

The JIT is enabled by default. You can explicitly enable the JIT in a number of different ways. Both command-line options override the **JAVA\_COMPILER** environment variable.

- Set the **JAVA\_COMPILER** environment variable to **jitc** before running the Java application. At a shell prompt, enter:

```
export JAVA_COMPILER=jitc
```

If the **JAVA\_COMPILER** environment variable is an empty string, the JIT remains disabled. To disable the environment variable, at the shell prompt, enter:

```
unset JAVA_COMPILER
```

- Use the **-D** option on the JVM command line to set the **java.compiler** property to **jitc**. At a shell prompt, enter:

```
java -Djava.compiler=jitc <class>
```

- Use the **-Xjit** option on the JVM command line. Do **not** specify the **-Xint** option at the same time. At a shell prompt, enter:

```
java -Xjit <class>
```

## Determining whether the JIT is enabled

You can determine the status of the JIT using the **-version** option.

Run the java launcher with the **-version** option. Enter the following at a shell prompt:

```
java -version
```

If the JIT is not in use, a message is displayed that includes the following:

```
(JIT disabled)
```

If the JIT is in use, a message is displayed that includes the following:

```
(JIT enabled)
```

## What to do next

For more information about the JIT, see the Diagnostics Guide.

---

## Specifying garbage collection policy

The Garbage Collector manages the memory used by Java and by applications running within the JVM.

When the Garbage Collector receives a request for storage, unused memory in the heap is set aside in a process called "allocation". The Garbage Collector also checks for areas of memory that are no longer referenced, and releases them for reuse. This is known as "collection".

The collection phase can be triggered by a memory allocation fault, which occurs when no space is left for a storage request, or by an explicit `System.gc()` call.

Garbage collection can significantly affect application performance, so the IBM virtual machine provides various methods of optimizing the way garbage collection is carried out, potentially reducing the effect on your application.

For more detailed information about garbage collection, see the Diagnostics Guide.

## Garbage collection options

The **-Xgcpolicy** options control the behavior of the Garbage Collector. They make trade-offs between throughput of the application and overall system, and the pause times that are caused by garbage collection.

The format of the option and its values is:

**-Xgcpolicy:optthroughput**

(Default and recommended value.) Delivers very high throughput to applications, but at the cost of occasional pauses.

**-Xgcpolicy:optavgpause**

Reduces the time spent in garbage collection pauses and limits the effect of increasing heap size on the length of the garbage collection pause. Use **optavgpause** if your configuration has a very large heap.

**-Xgcpolicy:gencon**

Requests the combined use of concurrent and generational GC to help minimize the time that is spent in any garbage collection pause.

**-Xgcpolicy:subpool**

Uses an improved object allocation algorithm to achieve better performance when allocating objects on the heap. This option might improve performance on large SMP systems.

## More effective heap usage using compressed references

Many Java application workloads depend on the Java heap size. The IBM SDK for Java can use compressed references on 64-bit platforms to decrease the size of Java objects and make more effective use of the available space.

The IBM SDK for Java 64-bit stores object references as 64-bit values. When the same application is run on a 32-bit JVM and a 64-bit JVM, the 64-bit JVM requires a larger heap because the references in the Java objects are larger. The **-Xcompressedrefs** command-line option causes object references to be stored as 32-bit values and maintains the 32-bit object size.

Use **-Xcompressedrefs** in any of these situations:

- When your Java applications does not need more than a 25 GB Java heap.
- When you are using x86-based hardware, which offers a larger set of registers for 64-bit code.
- When your application uses a lot of native memory and needs the JVM to run in a small footprint.

**-Xcompressedrefs** is not recommended for non-x86-based hardware running applications that have enough space in the Java heap provided by the 32-bit JVM.

See the Diagnostics Guide for more detailed information about compressed references.

## Pause time

When an application's attempt to create an object cannot be satisfied immediately from the available space in the heap, the Garbage Collector is responsible for identifying unreferenced objects (garbage), deleting them, and returning the heap to a state in which the immediate and subsequent allocation requests can be satisfied quickly.

Such garbage collection cycles introduce occasional unexpected pauses in the execution of application code. Because applications grow in size and complexity, and heaps become correspondingly larger, this garbage collection pause time tends to grow in size and significance.

The default garbage collection value, **-Xgcpolicy:optthroughput**, delivers very high throughput to applications, but at the cost of these occasional pauses, which can vary from a few milliseconds to many seconds, depending on the size of the heap and the quantity of garbage.

## Pause time reduction

The JVM uses two techniques to reduce pause times: concurrent garbage collection and generational garbage collection.

The **-Xgcpolicy:optavgpause** command-line option requests the use of concurrent garbage collection to reduce significantly the time that is spent in garbage collection pauses. Concurrent GC reduces the pause time by performing some garbage collection activities concurrently with normal program execution to minimize the disruption caused by the collection of the heap. The **-Xgcpolicy:optavgpause** option also limits the effect of increasing the heap size on the length of the garbage collection pause. The **-Xgcpolicy:optavgpause** option is most useful for configurations that have large heaps. With the reduced pause time, you might experience some reduction of throughput to your applications.

During concurrent garbage collection, a significant amount of time is wasted identifying relatively long-lasting objects that cannot then be collected. If garbage collection concentrates on only the objects that are most likely to be recyclable, you can further reduce pause times for some applications. Generational GC reduces pause times by dividing the heap into two generations: the "new" and the "tenure" areas. Objects are placed in one of these areas depending on their age. The new area is the smaller of the two and contains new objects; the tenure is larger and contains older objects. Objects are first allocated to the new area; if they have active references for long enough, they are promoted to the tenure area.

Generational GC depends on most objects not lasting long. Generational GC reduces pause times by concentrating the effort to reclaim storage on the new area because it has the most recyclable space. Rather than occasional but lengthy pause times to collect the entire heap, the new area is collected more frequently and, if the new area is small enough, pause times are comparatively short. However, generational GC has the drawback that, over time, the tenure area might become full. To minimize the pause time when this situation occurs, use a combination of concurrent GC and generational GC. The **-Xgcpolicy:gencon** option requests the combined use of concurrent and generational GC to help minimize the time that is spent in any garbage collection pause.

## Environments with very full heaps

If the Java heap becomes nearly full, and very little garbage can be reclaimed, requests for new objects might not be satisfied quickly because no space is immediately available.

If the heap is operated at near-full capacity, application performance might suffer regardless of which garbage collection options are used; and, if requests for more heap space continue to be made, the application might receive an `OutOfMemoryError`, which results in JVM termination if the exception is not caught and handled. At this point, the JVM produces a Javadump file for use during diagnostics. In these conditions, you are recommended either to increase the heap size by using the `-Xmx` option or to reduce the number of objects in use.

For more information, see the Diagnostics Guide.

---

## Euro symbol support

The IBM SDK and Runtime Environment set the Euro as the default currency for those countries in the European Monetary Union (EMU) for dates on or after 1 January, 2002. From 1 January 2008, Cyprus and Malta also have the Euro as the default currency.

To use the old national currency, specify `-Duser.variant=PREEURO` on the Java command line.

If you are running the UK, Danish, or Swedish locales and want to use the Euro, specify `-Duser.variant=EURO` on the Java command line.

---

## Using Indian and Thai input methods

From Version 6, the Indian and Thai input methods are not available by default. You must manually include the input method jar files in your Java extensions path to use the Indian and Thai input methods.

### About this task

In Version 5.0, the input method jar files were included in the `jre/lib/ext` directory and were automatically loaded by the JVM. In Version 6, the input method jar files are included in the `jre/lib/im` directory and you must manually add them to the Java extensions path to enable Indian and Thai input methods.

- Copy the `indicim.jar` and `thaiim.jar` files from the `jre/lib/im` directory to the `jre/lib/ext` directory.
- Add the `jre/lib/im` directory to the extension directories system property. Use the following command-line option:

```
java -Djava.ext.dirs=/usr/lpp/java/J6.0[_64]/jre/lib/ext:/usr/lpp/java/J6.0[_64]/jre/lib/im <class>
```

### What to do next

If you installed the SDK or Runtime Environment in a different directory, replace `/usr/lpp/java/J6.0[_64]/` with the directory in which you installed the SDK or Runtime Environment.

---

## Chapter 5. Developing Java applications

The SDK for z/OS contains many tools and libraries required for Java software development.

See “Contents of the SDK” on page 4 for details of the tools available.

---

### Using XML

The IBM SDK contains the XML4J and XL XP-J parsers, the XL TXE-J 1.0 XSLT compiler, and the XSLT4J XSLT interpreter. These tools allow you to parse, validate, transform, and serialize XML documents independently from any given XML processing implementation.

Use factory finders to locate implementations of the abstract factory classes, as described in “Selecting an XML processor” on page 22. By using factory finders, you can select a different XML library without changing your Java code.

#### Available XML libraries

The IBM SDK for Java contains the following XML libraries:

##### XML4J 4.5

XML4J is a validating parser providing support for the following standards:

- XML 1.0 (4th edition)
- Namespaces in XML 1.0 (2nd edition)
- XML 1.1 (2nd edition)
- Namespaces in XML 1.1 (2nd edition)
- W3C XML Schema 1.0 (2nd Edition)
- XInclude 1.0 (2nd Edition)
- OASIS XML Catalogs 1.0
- SAX 2.0.2
- DOM Level 3 Core, Load and Save
- DOM Level 2 Core, Events, Traversal and Range
- JAXP 1.4

XML4J 4.5 is based on Apache Xerces-J 2.9.0. See <http://xerces.apache.org/xerces2-j/> for more information.

##### XL XP-J 1.1

XL XP-J 1.1 is a high-performance non-validating parser that provides support for StAX 1.0 (JSR 173). StAX is a bidirectional API for pull-parsing and streaming serialization of XML 1.0 and XML 1.1 documents. See the “XL XP-J reference information” on page 25 section for more details about what is supported by XL XP-J 1.1.

##### XL TXE-J 1.0.1 Beta

For Version 5.0, the IBM SDK for Java included the XSLT4J compiler and interpreter. The XSLT4J interpreter was used by default.

For Version 6, the IBM SDK for Java includes XL TXE-J. XL TXE-J includes the XSLT4J 2.7.8 interpreter and a new XSLT compiler. The new compiler is used by default. The XSLT4J compiler is no longer included with the IBM SDK for Java. See “Migrating to the XL-TXE-J” on page 23 for information about migrating to XL TXE-J.

XL TXE-J provides support for the following standards:

- XSLT 1.0
- XPath 1.0
- JAXP 1.4

## Selecting an XML processor

XML processor selection is performed using service providers. When using a factory finder, Java looks in the following places, in this order, to see which service provider to use:

1. The system property with the same name as the service provider.
2. The service provider specified in a properties file.
  - **For XMLEventFactory, XMLInputFactory, and XMLOutputFactory only.** The value of the service provider in the file `/usr/lpp/java/J6.0[_64]/jre/lib/stax.properties`.
  - **For other factories.** The value of the service provider in the file `/usr/lpp/java/J6.0[_64]/jre/lib/jaxp.properties`.
3. The contents of the `META-INF/services/<service.provider>` file.
4. The default service provider.

The following service providers control the XML processing libraries used by Java:

### **javax.xml.parsers.SAXParserFactory**

Selects the SAX parser. By default, `org.apache.xerces.jaxp.SAXParserFactoryImpl` from the XML4J library is used.

### **javax.xml.parsers.DocumentBuilderFactory**

Selects the document builder. By default, `org.apache.xerces.jaxp.DocumentBuilderFactoryImpl` from the XML4J library is used.

### **javax.xml.datatype.DatatypeFactory**

Selects the datatype factory. By default, `org.apache.xerces.jaxp.datatype.DatatypeFactoryImpl` from the XML4J library is used.

### **javax.xml.stream.XMLEventFactory**

Selects the StAX event factory. By default, `com.ibm.xml.xlsp.api.stax.XMLEventFactoryImpl` from the XL XP-J library is used.

### **javax.xml.stream.XMLInputFactory**

Selects the StAX parser. By default, `com.ibm.xml.xlsp.api.stax.XMLInputFactoryImpl` from the XL XP-J library is used.

### **javax.xml.stream.XMLOutputFactory**

Selects the StAX serializer. By default, `com.ibm.xml.xlsp.api.stax.XMLOutputFactoryImpl` from the XL XP-J library is used.

**javax.xml.transform.TransformerFactory**

Selects the XSLT processor. Possible values are:

**com.ibm.xtq.xslt.jaxp.compiler.TransformerFactoryImpl**

Use the XL TXE-J compiler. This value is the default.

**org.apache.xalan.processor.TransformerFactoryImpl**

Use the XSLT4J interpreter.

**javax.xml.validation.SchemaFactory:http://www.w3.org/2001/XMLSchema**

Selects the schema factory for the W3C XML Schema language. By default, org.apache.xerces.jaxp.validation.XMLSchemaFactory from the XML4J library is used.

**javax.xml.xpath.XPathFactory**

Selects the XPath processor. By default, org.apache.xpath.jaxp.XPathFactoryImpl from the XSLT4J library is used.

## Migrating to the XL-TXE-J

The XL TXE-J compiler has replaced the XSLT4J interpreter as the default XSLT processor. Follow these steps to prepare your application for the new library.

### About this task

The XL TXE-J compiler is faster than the XSLT4J interpreter when you are applying the same transformation more than once. If you perform each individual transformation only once, the XL TXE-J compiler is slower than the XSLT4J interpreter because of the compilation and optimization overhead.

To continue using the XSLT4J interpreter as your XSLT processor, set the **javax.xml.transform.TransformerFactory** service provider to org.apache.xalan.processor.TransformerFactoryImpl.

To migrate to the XL-TXE-J compiler, follow the instructions in this task.

1. Use com.ibm.xtq.xslt.jaxp.compiler.TransformerFactoryImpl when setting the **javax.xml.transform.TransformerFactory** service provider.
2. Regenerate class files generated by the XSLT4J compiler. XL TXE-J cannot execute class files generated by the XSLT4J compiler.
3. Some methods generated by the compiler might exceed the JVM method size limit, in which case the compiler attempts to split these methods into smaller methods.
  - If the compiler splits the method successfully, you receive the following warning:  
Some generated functions exceeded the JVM method size limit and were automatically split into smaller functions. You might get better performance by manually splitting very large templates into smaller templates, by using the 'splitlimit' option to the Process or Compile command, or by setting the 'http://www.ibm.com/xmlns/prod/xltxe-j/split-limit' transformer factory attribute. You can use the compiled classes, but you might get better performance by controlling the split limit manually.
  - If the compiler does not split the method successfully, you receive one of the following exceptions:  
com.ibm.xtq.bcel.generic.ClassGenException: Branch target offset too large for short or

bytecode array size > 65535 at offset=#####Try setting the split limit manually, or using a lower split limit.

To set the split limit, use the **-SPLITLIMIT** option when using the Process or Compile commands, or the <http://www.ibm.com/xmlns/prod/xtxe-j/split-limit> transformer factory attribute when using the transformer factory. The split limit can be between 100 and 2000. When setting the split limit manually, use the highest split limit possible for best performance.

4. XL TXE-J might need more memory than the XSLT4J compiler. If you are running out of memory or performance seems slow, increase the size of the heap using the **-Xmx** option.
5. Migrate your application to use the new attribute keys. The old transformer factory attribute keys are deprecated. The old names are accepted with a warning.

Table 2. Changes to attribute keys from the XSL4J compiler to the XL TXE-J compiler

XSL4J compiler attribute	XL TXE-J compiler attribute
translet-name	<a href="http://www.ibm.com/xmlns/prod/xtxe-j/translet-name">http://www.ibm.com/xmlns/prod/xtxe-j/translet-name</a>
destination-directory	<a href="http://www.ibm.com/xmlns/prod/xtxe-j/destination-directory">http://www.ibm.com/xmlns/prod/xtxe-j/destination-directory</a>
package-name	<a href="http://www.ibm.com/xmlns/prod/xtxe-j/package-name">http://www.ibm.com/xmlns/prod/xtxe-j/package-name</a>
jar-name	<a href="http://www.ibm.com/xmlns/prod/xtxe-j/jar-name">http://www.ibm.com/xmlns/prod/xtxe-j/jar-name</a>
generate-translet	<a href="http://www.ibm.com/xmlns/prod/xtxe-j/generate-translet">http://www.ibm.com/xmlns/prod/xtxe-j/generate-translet</a>
auto-translet	<a href="http://www.ibm.com/xmlns/prod/xtxe-j/auto-translet">http://www.ibm.com/xmlns/prod/xtxe-j/auto-translet</a>
use-classpath	<a href="http://www.ibm.com/xmlns/prod/xtxe-j/use-classpath">http://www.ibm.com/xmlns/prod/xtxe-j/use-classpath</a>
debug	<a href="http://www.ibm.com/xmlns/prod/xtxe-j/debug">http://www.ibm.com/xmlns/prod/xtxe-j/debug</a>
indent-number	<a href="http://www.ibm.com/xmlns/prod/xtxe-j/indent-number">http://www.ibm.com/xmlns/prod/xtxe-j/indent-number</a>
enable-inlining	<i>Obsolete in new compiler</i>

6. Optional: For best performance, ensure that you are not recompiling XSLT transformations that can be reused. Use one of the following methods to reuse compiled transformations:
  - If your stylesheet does not change at runtime, compile the stylesheet as part of your build process and put the compiled classes on your classpath. Use the `org.apache.xalan.xsltc.cmdline.Compile` command to compile the stylesheet and set the <http://www.ibm.com/xmlns/prod/xtxe-j/use-classpath> transformer factory attribute to `true` to load the classes from the classpath.
  - If your application will use the same stylesheet during multiple runs, set the <http://www.ibm.com/xmlns/prod/xtxe-j/auto-translet> transformer factory attribute to `true` to automatically save the compiled stylesheet to disk for reuse. The compiler will use a compiled stylesheet if it is available, and compile the stylesheet if it is not available or is out-of-date. Use the <http://www.ibm.com/xmlns/prod/xtxe-j/destination-directory> transformer factory attribute to set the directory used to store compiled stylesheets. By default, compiled stylesheets are stored in the same directory as the stylesheet.
  - If your application is a long-running application that reuses the same stylesheet, use the transformer factory to compile the stylesheet and create a Templates object. You can use the Templates object to create Transformer objects without recompiling the stylesheet. The Transformer objects can also be reused but are not thread-safe.

## XML reference information

The XL XP-J and XL TXE-J XML libraries are new for Version 6 of the SDK. This reference information describes the features supported by these libraries.

### XL XP-J reference information

XL XP-J 1.1 is a high-performance non-validating parser that provides support for StAX 1.0 (JSR 173). StAX is a bidirectional API for pull-parsing and streaming serialization of XML 1.0 and XML 1.1 documents.

### Unsupported features

The following optional StAX features are not supported by XL XP-J:

- DTD validation when using an XMLStreamReader or XMLEventReader. The XL XP-J parser is non-validating.
- When using an XMLStreamReader to read from a character stream (java.io.Reader), the Location.getCharaterOffset() method always returns -1. The Location.getCharaterOffset() returns the byte offset of a Location when using an XMLStreamReader to read from a byte stream (java.io.InputStream).

### XMLInputFactory reference

The javax.xml.stream.XMLInputFactory implementation supports the following properties, as described in the XMLInputFactory Javadoc: <http://java.sun.com/javase/6/docs/api/javax/xml/stream/XMLInputFactory.html>.

Table 3.

Property name	Supported?
<code>javax.xml.stream.isValidating</code>	No. The XL XP-J scanner does not support validation.
<code>javax.xml.stream.isNamespaceAware</code>	Yes, supports true and false. For XMLStreamReaders created from DOMSources, namespace processing depends on the methods that were used to create the DOM tree, and this value has no effect.
<code>javax.xml.stream.isCoalescing</code>	Yes
<code>javax.xml.stream.isReplacingEntityReferences</code>	Yes. For XMLStreamReaders created from DOMSources, if entities have already been replaced in the DOM tree, setting this parameter has no effect.
<code>javax.xml.stream.isSupportingExternalEntities</code>	Yes

Table 3. (continued)

Property name	Supported?
<code>javax.xml.stream.supportDTD</code>	<p>True is always supported. Setting the value to false works only if the <code>com.ibm.xml.xpath.support.dtd.compat.mode</code> system property is also set to false.</p> <p>When both properties are set to false, parsers created by the factory throw an <code>XMLStreamException</code> when they encounter an entity reference that requires expansion. This setting is useful for protecting against Denial of Service (DoS) attacks involving entities declared in the DTD.</p> <p>Setting the value to false does not work before Service Refresh 2.</p>
<code>javax.xml.stream.reporter</code>	Yes
<code>javax.xml.stream.resolver</code>	Yes

XL XP-J also supports the optional method `createXMLStreamReader(javax.xml.transform.Source)`, which allows StAX readers to be created from DOM and SAX sources.

XL XP-J also supports the `javax.xml.stream.isSupportingLocationCoordinates` property. If you set this property to true, `XMLStreamReaders` created by the factory return accurate line, column, and character information using `Location` objects. If you set this property to false, line, column, and character information is not available. By default, this property is set to false for performance reasons.

### XMLStreamReader reference

The `javax.xml.stream.XMLStreamReader` implementation supports the following properties, as described in the `XMLStreamReader` Javadoc: <http://java.sun.com/javase/6/docs/api/javax/xml/stream/XMLStreamReader.html>.

Table 4.

Property name	Supported?
<code>javax.xml.stream.entities</code>	Yes
<code>javax.xml.streamnotations</code>	Yes

XL XP-J also supports the `javax.xml.stream.isInterning` property. This property returns a boolean value indicating whether or not XML names and namespace URIs returned by the API calls have been interned by the parser. This property is read-only.

### XMLOutputFactory reference

The `javax.xml.stream.XMLOutputFactory` implementation supports the following properties, as described in the `XMLOutputFactory` Javadoc: <http://java.sun.com/javase/6/docs/api/javax/xml/stream/XMLOutputFactory.html>.

Table 5.

Property name	Supported?
<code>javax.xml.stream.isRepairingNamespaces</code>	Yes

XL XP-J also supports the `javax.xml.stream.XMLOutputFactory.recycleWritersOnEndDocument` property. If you set this property to true, `XMLStreamWriters` created by this factory are recycled when `writeEndDocument()` is called. After recycling, some `XMLStreamWriter` methods, such as `getNamespaceContext()`, must not be called. By default, `XMLStreamWriters` are recycled when `close()` is called. You must call the `XMLStreamWriter.close()` method when you have finished with an `XMLStreamWriter`, even if this property is set to true.

### XMLStreamWriter reference

The `javax.xml.stream.XMLStreamWriter` implementation supports the following properties, as described in the `XMLStreamWriter` Javadoc: <http://java.sun.com/javase/6/docs/api/javax/xml/stream/XMLStreamWriter.html>.

Table 6.

Property name	Supported?
<code>javax.xml.stream.isRepairingNamespaces</code>	Yes

Properties on `XMLStreamWriter` objects are read-only.

XL XP-J also supports the `javax.xml.stream.XMLStreamWriter.isSetPrefixBeforeStartElement` property. This property returns a Boolean indicating whether calls to `setPrefix()` and `setDefaultNamespace()` should occur before calls to `writeStartElement()` or `writeEmptyElement()` to put a namespace prefix in scope for that element. XL XP-J always returns false; calls to `setPrefix()` and `setDefaultNamespace()` should occur after `writeStartElement()` or `writeEmptyElement()`.

### XL TXE-J reference information

XL TXE-J is an XSLT library containing the XSLT4J 2.7.8 interpreter and a XSLT compiler.

### Feature comparison table

Table 7. Comparison of the features in the XSLT4J interpreter, the XSLT4J compiler, and the XL TXE-J compiler.

Feature	XSLT4J interpreter (included)	XSLT4J compiler (not included)	XL TXE-J compiler (included)
<code>http://javax.xml.transform.stream.StreamSource/feature</code>	Yes	Yes	Yes
<code>http://javax.xml.transform.stream.StreamResult/feature</code>	Yes	Yes	Yes
<code>http://javax.xml.transform.dom.DOMSource/feature</code>	Yes	Yes	Yes

Table 7. Comparison of the features in the XSLT4J interpreter, the XSLT4J compiler, and the XL TXE-J compiler. (continued)

Feature	XSLT4J interpreter (included)	XSLT4J compiler (not included)	XL TXE-J compiler (included)
http://javax.xml.transform.dom.DOMResult/feature feature	Yes	Yes	Yes
http://javax.xml.transform.sax.SAXSource/feature feature	Yes	Yes	Yes
http://javax.xml.transform.sax.SAXResult/feature feature	Yes	Yes	Yes
http://javax.xml.transform.stax.StAXSource/feature feature	Yes	No	Yes
http://javax.xml.transform.stax.StAXResult/feature feature	Yes	No	Yes
http://javax.xml.transform.sax.SAXTransformerFactory/feature feature	Yes	Yes	Yes
http://javax.xml.transform.sax.SAXTransformerFactory/feature/xmlfilter feature	Yes	Yes	Yes
http://javax.xml.XMLConstants/feature/secure-processing feature	Yes	Yes	Yes
http://xml.apache.org/xalan/features/incremental attribute	Yes	No	No
http://xml.apache.org/xalan/features/optimize attribute	Yes	No	No
http://xml.apache.org/xalan/properties/source-location attribute	Yes	No	No
translet-name attribute	N/A	Yes	Yes (with new name)
destination-directory attribute	N/A	Yes	Yes (with new name)
package-name attribute	N/A	Yes	Yes (with new name)
jar-name attribute	N/A	Yes	Yes (with new name)
generate-translet attribute	N/A	Yes	Yes (with new name)
auto-translet attribute	N/A	Yes	Yes (with new name)
use-classpath attribute	N/A	Yes	Yes (with new name)
enable-inlining attribute	No	Yes	No (obsolete in TL TXE-J)

Table 7. Comparison of the features in the XSLT4J interpreter, the XSLT4J compiler, and the XL TXE-J compiler. (continued)

Feature	XSLT4J interpreter (included)	XSLT4J compiler (not included)	XL TXE-J compiler (included)
indent-number attribute	No	Yes	Yes (with new name)
debug attribute	No	Yes	Yes (with new name)
Java extensions	Yes	Yes (abbreviated syntax only, xalan:component/xalan:script constructs not supported)	
JavaScript extensions	Yes	No	No
Extension elements	Yes	No	No
EXSLT extension functions	Yes	Yes (excluding dynamic)	Yes (excluding dynamic)
redirect extension	Yes	Yes (excluding redirect:open and redirect:close)	Yes
output extension	No	Yes	Yes
nodeset extension	Yes	Yes	Yes
NodeInfo extension functions	Yes	No	No
SQL library extension	Yes	No	No
pipeDocument extension	Yes	No	No
evaluate extension	Yes	No	No
tokenize extension	Yes	No	No
XML 1.1	Yes	Yes	Yes

## Notes

1. With the Process command, use **-FLAVOR sr2sw** to transform using StAX stream processing, and **-FLAVOR er2ew** for StAX event processing.
2. The new compiler does not look for the `org.apache.xalan.xsltc.dom.XSLTCDTMMManager` service provider. Instead, if `StreamSource` is used, the compiler switches to a high-performance XML parser.
3. Inlining is obsolete in XL TXE-J.
  - The **-XN** option to the Process command is silently ignored.
  - The **-n** option to the Compile command is silently ignored.
  - The **enable-inlining** transformer factory attribute is silently ignored.
4. The `org.apache.xalan.xsltc.trax.SmartTransformerFactoryImpl` class is no longer supported.

## Using an older version of Xerces or Xalan

If you are using an older version of Xerces (before 2.0) or Xalan (before 2.3) in the endorsed override, you might get a `NullPointerException` when you start your application. This exception occurs because these older versions do not handle the `jaxp.properties` file correctly.

## About this task

To avoid this situation, use one of the following workarounds:

- Upgrade to a newer version of the application that implements the latest Java API for XML Programming (JAXP) specification (<https://jaxp.dev.java.net/>).
- Remove the `jaxp.properties` file from `/usr/lpp/java/J6.0[_64]/jre/lib`.
- Uncomment the entries in the `jaxp.properties` file in `/usr/lpp/java/J6.0[_64]/jre/lib`.
- Set the system property for `javax.xml.parsers.SAXParserFactory`, `javax.xml.parsers.DocumentBuilderFactory`, or `javax.xml.transform.TransformerFactory` using the `-D` command-line option.
- Set the system property for `javax.xml.parsers.SAXParserFactory`, `javax.xml.parsers.DocumentBuilderFactory`, or `javax.xml.transform.TransformerFactory` in your application. For an example, see the JAXP 1.4 specification.

- Explicitly set the SAX parser, Document builder, or Transformer factory using the `IBM_JAVA_OPTIONS` environment variable.

```
export IBM_JAVA_OPTIONS=-Djavax.xml.parsers.SAXParserFactory=
org.apache.xerces.jaxp.SAXParserFactoryImpl
```

or

```
export IBM_JAVA_OPTIONS=-Djavax.xml.parsers.DocumentBuilderFactory=
org.apache.xerces.jaxp.DocumentBuilderFactoryImpl
```

or

```
export IBM_JAVA_OPTIONS=-Djavax.xml.transform.TransformerFactory=
org.apache.xmlan.processor.TransformerFactoryImpl
```

---

## Debugging Java applications

To debug Java programs, you can use the Java Debugger (JDB) application or other debuggers that communicate by using the Java Platform Debugger Architecture (JPDA) that is provided by the SDK for z/OS.

More information about problem diagnosis using Java can be found in the Diagnostics Guide.

### Java Debugger (JDB)

The Java Debugger (JDB) is included in the SDK for z/OS. The debugger is invoked by the `jdb` command; it attaches to the JVM using JPDA.

To debug a Java application:

1. Start the JVM with the following options:

```
java -Xdebug -Xrunjdw:transport=dt_socket,server=y,address=<port> <class>
```

The JVM starts up, but suspends execution before it starts the Java application.

2. In a separate session, you can attach the debugger to the JVM:

```
jdb -attach <port>
```

The debugger will attach to the JVM, and you can now issue a range of commands to examine and control the Java application; for example, type `run` to allow the Java application to start.

For more information about JDB options, type:

```
jdb -help
```

For more information about JDB commands:

1. Type `jdb`
2. At the `jdb` prompt, type `help`

You can also use JDB to debug Java applications running on remote machines. JPDA uses a TCP/IP socket to connect to the remote JVM.

1. Start the JVM with the following options:

```
java -Xdebug -Xrunjdpw:transport=dt_socket,server=y,address=<port> <class>
```

The JVM starts up, but suspends execution before it starts the Java application.

2. Attach the debugger to the remote JVM:

```
jdb -attach <host>:<port>
```

The Java Virtual Machine Debugging Interface (JVMDI) is not supported in this release. It has been replaced by the Java Virtual Machine Tool Interface (JVMTI).

For more information about JDB and JPDA and their usage, see these Web sites:

- <http://java.sun.com/products/jpda/>
- <http://java.sun.com/javase/6/docs/technotes/guides/jpda/>
- <http://java.sun.com/javase/6/docs/technotes/guides/jpda/jdb.html>

## Selective debugging

Use the `com.ibm.jvm.Debuggable` annotation to mark classes and methods that should be available for debugging. Use the `-XselectiveDebug` parameter to enable selective debugging at run time. The JVM optimizes methods that do not need debugging to provide better performance in a debugging environment.

### About this task

Selective debugging is useful when Java is being used as a framework for development, for example, as an IDE. The Java code for the IDE is optimized for performance while the user code is debugged.

1. Import the `Debuggable` annotation from the `com.ibm.jvm` package.

```
import com.ibm.jvm.Debuggable;
```

2. Decorate methods using the `Debuggable` annotation.

```
@Debuggable
public int method1() {
    ...
}
```

3. Optional: You can also decorate classes using the `Debuggable` annotation. All methods in the class will remain debuggable.

```
@Debuggable
public class Class1 {
    ...
}
```

4. Enable selective debugging at run time using the `-XselectiveDebug` command-line option.

## Results

Applications will run faster while being debugged because the core Java API and any IDE code can be optimized for performance.

---

## Determining whether your application is running on a 31-bit or 64-bit JVM

Some Java applications must be able to determine whether they are running on a 31-bit JVM or on a 64-bit JVM. For example, if your application has a native code library, the library must be compiled separately in 31- and 64-bit forms for platforms that support both 31- and 64-bit modes of operation. In this case, your application must load the correct library at runtime, because it is not possible to mix 31- and 64-bit code.

### About this task

The system property **com.ibm.vm.bitmode** allows applications to determine the mode in which your JVM is running. It returns the following values:

- 32 - the JVM is running in 31-bit mode
- 64 - the JVM is running in 64-bit mode

You can inspect the **com.ibm.vm.bitmode** property from within your application code using the call:

```
System.getProperty("com.ibm.vm.bitmode");
```

---

## How the JVM processes signals

When a signal is raised that is of interest to the JVM, a signal handler is called. This signal handler determines whether it has been called for a Java or non-Java thread.

If the signal is for a Java thread, the JVM takes control of the signal handling. If an application handler for this signal is installed and you did not specify the **-Xnosigchain** command-line option, the application handler for this signal is called after the JVM has finished processing.

If the signal is for a non-Java thread, and the application that installed the JVM had previously installed its own handler for the signal, control is given to that handler. Otherwise, if the signal is requested by the JVM or Java application, the signal is ignored or the default action is taken.

For exception and error signals, the JVM either:

- Handles the condition and recovers, or
- Enters a controlled shutdown sequence where it:
  1. Produces dumps, to describe the JVM state at the point of failure
  2. Calls your application's signal handler for that signal
  3. Calls any application-installed unexpected-shutdown hook
  4. Performs the necessary JVM cleanup

For information about writing a launcher that specifies the above hooks, see: <http://www.ibm.com/developerworks/java/library/i-signalhandling/>. This item was written for Java V1.3.1, but still applies to later versions.

For interrupt signals, the JVM also enters a controlled shutdown sequence, but this time it is treated as a normal termination that:

1. Calls your application's signal handler for that signal
2. Calls all application shutdown hooks
3. Calls any application-installed exit hook
4. Performs the necessary JVM cleanup

The shutdown is identical to the shutdown initiated by a call to the Java method `System.exit()`.

Other signals that are used by the JVM are for internal control purposes and do not cause it to terminate. The only control signal of interest is SIGQUIT, which causes a Javdump to be generated.

## Signals used by the JVM

The types of signals are Exceptions, Errors, Interrupts, and Controls.

Table 8 below shows the signals that are used by the JVM. The signals are grouped in the table by type or use, as follows:

### Exceptions

The operating system synchronously raises an appropriate exception signal whenever a fatal condition occurs.

**Errors** The JVM raises a SIGABRT if it detects a condition from which it cannot recover.

### Interrupts

Interrupt signals are raised asynchronously, from outside a JVM process, to request shutdown.

### Controls

Other signals that are used by the JVM for control purposes.

*Table 8. Signals used by the JVM*

Signal Name	Signal type	Description	Disabled by -Xrs
SIGBUS (7)	Exception	Incorrect access to memory (data misalignment)	Yes
SIGSEGV (11)	Exception	Incorrect access to memory (write to inaccessible memory)	Yes
SIGILL (4)	Exception	Illegal instruction (attempt to invoke an unknown machine instruction)	Yes
SIGFPE (8)	Exception	Floating point exception (divide by zero)	Yes
SIGABRT (6)	Error	Abnormal termination. The JVM raises this signal whenever it detects a JVM fault.	Yes
SIGINT (2)	Interrupt	Interactive attention (CTRL-C). JVM exits normally.	Yes

Table 8. Signals used by the JVM (continued)

Signal Name	Signal type	Description	Disabled by -Xrs
SIGTERM (15)	Interrupt	Termination request. JVM will exit normally.	Yes
SIGHUP (1)	Interrupt	Hang up. JVM exits normally.	Yes
SIGQUIT (3)	Control	By default, this triggers a Javdump.	Yes
SIGRECONFIG (58)	Control	Reserved to detect any change in the number of CPUs, processing capacity, or physical memory.	Yes
SIGTRAP (5)	Control	Used by the JIT.	Yes
SIGCHLD (17)	Control	Used by the SDK for internal control.	No
SIGUSR1	Control	Used by the SDK.	No

Use the **-Xrs** (reduce signal usage) option to prevent the JVM from handling most signals. For more information, see Sun's Java application launcher page.

Signals 1 (SIGHUP), 2 (SIGINT), 4 (SIGILL), 7 (SIGBUS), 8 (SIGFPE), 11 (SIGSEGV), and 15 (SIGTERM) on JVM threads cause the JVM to shut down; therefore, an application signal handler should not attempt to recover from these unless it no longer requires the JVM.

## Linking a native code driver to the signal-chaining library

The Runtime Environment contains signal-chaining. Signal-chaining enables the JVM to interoperate more efficiently with native code that installs its own signal handlers.

### About this task

Signal-chaining enables an application to link and load the shared library `libjsig.so` before the system libraries. The `libjsig.so` library ensures that calls such as `signal()`, `sigset()`, and `sigaction()` are intercepted so that their handlers do not replace the JVM's signal handlers. Instead, these calls save the new signal handlers, or "chain" them behind the handlers that are installed by the JVM. Later, when any of these signals are raised and found not to be targeted at the JVM, the preinstalled handlers are invoked.

If you install signal handlers that use `sigaction()`, some **sa\_flags** are not observed when the JVM uses the signal. These are:

- `SA_NOCLDSTOP` - This is always unset.
- `SA_NOCLDWAIT` - This is always unset.
- `SA_RESTART` - This is always set.

The `libjsig.so` library also hides JVM signal handlers from the application. Therefore, calls such as `signal()`, `sigset()`, and `sigaction()` that are made after the JVM has started no longer return a reference to the JVM's signal handler, but instead return any handler that was installed before JVM startup.

The environment variable `JAVA_HOME` should be set to the location of the SDK, for example, `/usr/lpp/java/J6.0[_64]/`.

To use `libjsig.a`:

- Link it with the application that creates or embeds a JVM:

```
cc_r -q64 <other compile/link parameter> -L/usr/lpp/java/J6.0[_64]/jre/bin -ljsig  
-L/usr/lpp/java/J6.0[_64]/jre/bin/j9vm -ljvm java_application.c
```

**Note:** Use `xlc_r` or `xlc_r` in place of `cc_r` if that is how you normally invoke the compiler or linker.

---

## Writing JNI applications

Valid JNI version numbers that native programs can specify on the `JNI_CreateJavaVM()` API call are: `JNI_VERSION_1_2(0x00010002)` and `JNI_VERSION_1_4(0x00010004)`.

**Restriction:** Version 1.1 of the Java Native Interface (JNI) is not supported.

This version number determines only the level of the JNI native interface to use. The actual level of the JVM that is created is specified by the JSE libraries (that is, v6). The JNI interface API *does not* affect the language specification that is implemented by the JVM, the class library APIs, or any other area of JVM behavior. For more information, see <http://java.sun.com/javase/6/docs/technotes/guides/jni/>.

If your application needs two JNI libraries, one built for 31- and the other for 64-bit, use the `com.ibm.vm.bitmode` system property to determine if you are running with a 31- or 64-bit JVM and choose the appropriate library.

For more information about writing 64-bit applications, see the IBM Redpaper *z/OS 64-bit C/C++ and Java Programming Environment* at <http://www.redbooks.ibm.com/abstracts/redp9110.html>.

### ASCII and EBCDIC issues

On z/OS, the Java Virtual Machine is essentially an EBCDIC application. Enhanced ASCII methods are C or C++ code that has been compiled with ASCII compiler options. If you create JNI routines as enhanced ASCII C or C++ methods you will be operating in a bimodal environment; your application will be crossing over between ASCII and EBCDIC environments.

The inherent problem with bimodal programs is that, in the z/OS runtime, threads are designated as either EBCDIC or enhanced ASCII and are not intended to be switched between these modes in normal use. Enhanced ASCII is not designed to handle bimodal issues. You might get unexpected results or experience failures when the active mode does not match that of the compiled code. There are z/OS runtime calls that allow applications to switch the active mode between EBCDIC and enhanced ASCII (the `__ae_thread_swapmode()` and `__ae_thread_setmode()` functions are documented in Language Environment Vendor Interfaces, see the *SA22-7568-06 Red Book*: <http://publibz.boulder.ibm.com/epubs/pdf/ceev1160.pdf>). However, even if an application is carefully coded to switch modes correctly, other bimodal issues might exist.

## Native formatting of Java types long, double, float

The latest C/C++ compilers and runtimes can convert jlong, jdouble, and jfloat data types to strings by using printf()-type functions.

Previous versions of the SDK for z/OS 31-bit had a set of native conversion functions and macros for formatting large Java data types. These functions and macros were:

### ll2str() function

Converts a jlong to an ASCII string representation of the 64-bit value.

### flt2dbl() function

Converts a jfloat to a jdouble.

### dbl2nat() macro

Converts a jdouble to an ESA/390 native double.

### dbl\_sqrt() macro

Calculates the square root of a jdouble and returns it as a jdouble.

### dbl2str() function

Converts a jdouble to an ASCII string representation.

### flt2str() function

Converts a jfloat to an ASCII string representation.

These functions and macros are no longer supported by Version 6 of the SDK for z/OS. To provide a migration path, the functions have been moved to the demos area of the SDK and the appropriate demo code for these functions has been updated to reflect the changes.

The functions ll2str(), dbl2str(), and flt2str() are provided in the following object files:

- /usr/lpp/java/J6.0[\_64]/demo/jni/JNINativeTypes/c/convert.o (For 31-bit)
- /usr/lpp/java/J6.0[\_64]/demo/jni/JNINativeTypes/c/convert64.o (For 64-bit)

The function flt2dbl() and the macros dbl2nat() and dbl\_sqrt() are not defined. However, the following macros give their definitions:

```
#include <math.h>
#define flt2dbl(f) ((double)f)
#define dbl2nat(a) ((a))
#define dbl_sqrt(a) (sqrt(a))
```

---

## Support for thread-level recovery of blocked connectors

Four new IBM-specific SDK classes have been added to the com.ibm.jvm package to support the thread-level recovery of Blocked connectors. The new classes are packaged in core.jar.

These classes allow you to unblock threads that have become blocked on networking or synchronization calls. If an application does not use these classes, it must end the whole process, rather than interrupting an individual blocked thread.

The classes are:

### **public interface InterruptibleContext**

Defines two methods, isBlocked() and unblock(). The other three classes implement InterruptibleContext.

**public class InterruptibleLockContext**

A utility class for interrupting synchronization calls.

**public class InterruptibleIOContext**

A utility class for interrupting network calls.

**public class InterruptibleThread**

A utility class that extends `java.lang.Thread`, to allow wrapping of interruptible methods. It uses instances of `InterruptibleLockContext` and `InterruptibleIOContext` to perform the required `isBlocked()` and `unlock()` methods depending on whether a synchronization or networking operation is blocking the thread.

Both `InterruptibleLockContext` and `InterruptibleIOContext` work by referencing the current thread. Therefore if you do not use `InterruptibleThread`, you must provide your own class that extends `java.lang.Thread`, to use these new classes.

The Javadoc for these classes is provided with the SDK in the `docs/content/apidoc` directory.

---

## CORBA support

The Java Platform, Standard Edition (JSE) supports, at a minimum, the specifications that are defined in the compliance document from Sun. In some cases, the IBM JSE ORB supports more recent versions of the specifications.

The minimum specifications supported are defined in the Official Specifications for CORBA support in Java SE 6: <http://java.sun.com/javase/6/docs/api/org/omg/CORBA/doc-files/compliance.html>.

### Support for GIOP 1.2

This SDK supports all versions of GIOP, as defined by chapters 13 and 15 of the CORBA 2.3.1 specification, OMG document *formal/99-10-07*.

<http://www.omg.org/cgi-bin/doc?formal/99-10-07>

Bidirectional GIOP is not supported.

### Support for Portable Interceptors

This SDK supports Portable Interceptors, as defined by the OMG in the document *ptc/01-03-04*, which you can obtain from:

<http://www.omg.org/cgi-bin/doc?ptc/01-03-04>

Portable Interceptors are hooks into the ORB that ORB services can use to intercept the normal flow of execution of the ORB.

### Support for Interoperable Naming Service

This SDK supports the Interoperable Naming Service, as defined by the OMG in the document *ptc/00-08-07*, which you can obtain from:

<http://www.omg.org/cgi-bin/doc?ptc/00-08-07>

The default port that is used by the Transient Name Server (the `tnameserv` command), when no **ORBInitialPort** parameter is given, has changed from *900* to *2809*, which is the port number that is registered with the IANA (Internet Assigned Number Authority) for a CORBA Naming Service. Programs that depend on this default might have to be updated to work with this version.

The initial context that is returned from the Transient Name Server is now an `org.omg.CosNaming.NamingContextExt`. Existing programs that narrow the reference to a context `org.omg.CosNaming.NamingContext` still work, and do not need to be recompiled.

The ORB supports the **-ORBInitRef** and **-ORBDefaultInitRef** parameters that are defined by the Interoperable Naming Service specification, and the `ORB::string_to_object` operation now supports the ObjectURL string formats (`corbaloc:` and `corbaname:`) that are defined by the Interoperable Naming Service specification.

The OMG specifies a method `ORB::register_initial_reference` to register a service with the Interoperable Naming Service. However, this method is not available in the Sun Java Core API at Version 6. Programs that need to register a service in the current version must invoke this method on the IBM internal ORB implementation class. For example, to register a service "MyService":

```
((com.ibm.CORBA.iiop.ORB)orb).register_initial_reference("MyService",
serviceRef);
```

Where `orb` is an instance of `org.omg.CORBA.ORB`, which is returned from `ORB.init()`, and `serviceRef` is a CORBA Object, which is connected to the ORB. This mechanism is an interim one, and is not compatible with future versions or portable to non-IBM ORBs.

## System properties for tracing the ORB

A runtime debug feature provides improved serviceability. You might find it useful for problem diagnosis or it might be requested by IBM service personnel.

### Tracing Properties

**com.ibm.CORBA.Debug=true**

Turns on ORB tracing.

**com.ibm.CORBA.CommTrace=true**

Adds GIOP messages (sent and received) to the trace.

**com.ibm.CORBA.Debug.Output=<file>**

Specify the trace output file. By default, this is of the form `orbtrc.DDMMYYYY.HHmm.SS.txt`.

### Example of ORB tracing

For example, to trace events and formatted GIOP messages from the command line, type:

```
java -Dcom.ibm.CORBA.Debug=true
      -Dcom.ibm.CORBA.CommTrace=true <myapp>
```

### Limitations

Do not turn on tracing for normal operation, because it might cause performance degradation. Even if you have switched off tracing, FFDC (First Failure Data

Capture) is still working, so serious errors are reported. If a debug output file is generated, examine it to check on the problem. For example, the server might have stopped without performing an `ORB.shutdown()`.

The content and format of the trace output might vary from version to version.

## System properties for tuning the ORB

The ORB can be tuned to work well with your specific network. The properties required to tune the ORB are described here.

**com.ibm.CORBA.FragmentSize**=<size in bytes>

Used to control GIOP 1.2 fragmentation. The default size is 1024 bytes.

To disable fragmentation, set the fragment size to 0 bytes:

```
java -Dcom.ibm.CORBA.FragmentSize=0 <myapp>
```

**com.ibm.CORBA.RequestTimeout**=<time in seconds>

Sets the maximum time to wait for a CORBA Request. By default the ORB waits indefinitely. Do not set the timeout too low to avoid connections ending unnecessarily.

**com.ibm.CORBA.LocateRequestTimeout**=<time in seconds>

Set the maximum time to wait for a CORBA LocateRequest. By default the ORB waits indefinitely.

**com.ibm.CORBA.ListenerPort**=<port number>

Set the port for the ORB to read incoming requests on. If this property is set, the ORB starts listening as soon as it is initialized. Otherwise, it starts listening only when required.

## Java security permissions for the ORB

When running with a Java SecurityManager, invocation of some methods in the CORBA API classes might cause permission checks to be made, which might result in a SecurityException. If your program uses any of these methods, ensure that it is granted the necessary permissions.

Table 9. Methods affected when running with Java SecurityManager

Class/Interface	Method	Required permission
org.omg.CORBA.ORB	init	java.net.SocketPermission resolve
org.omg.CORBA.ORB	connect	java.net.SocketPermission listen
org.omg.CORBA.ORB	resolve_initial_references	java.net.SocketPermission connect
org.omg.CORBA. portable.ObjectImpl	_is_a	java.net.SocketPermission connect
org.omg.CORBA. portable.ObjectImpl	_non_existent	java.net.SocketPermission connect
org.omg.CORBA. portable.ObjectImpl	OutputStream _request (String, boolean)	java.net.SocketPermission connect
org.omg.CORBA. portable.ObjectImpl	_get_interface_def	java.net.SocketPermission connect
org.omg.CORBA. Request	invoke	java.net.SocketPermission connect

Table 9. Methods affected when running with Java SecurityManager (continued)

Class/Interface	Method	Required permission
org.omg.CORBA. Request	send_deferred	java.net.SocketPermission connect
org.omg.CORBA. Request	send_oneway	java.net.SocketPermission connect
javax.rmi. PortableRemoteObject	narrow	java.net.SocketPermission connect

## ORB implementation classes

A list of the ORB implementation classes.

The ORB implementation classes in this release are:

- org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
- org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton
- javax.rmi.CORBA.UtilClass=com.ibm.CORBA.iiop.UtilDelegateImpl
- javax.rmi.CORBA.StubClass=com.ibm.rmi.javax.rmi.CORBA.StubDelegateImpl
- javax.rmi.CORBA.PortableRemoteObjectClass=com.ibm.rmi.javax.rmi.PortableRemoteObject

These are the default values, and you are advised not to set these properties or refer to the implementation classes directly. For portability, make references only to the CORBA API classes, and not to the implementation. These values might be changed in future releases.

---

## RMI over IIOP

Java Remote Method Invocation (RMI) provides a simple mechanism for distributed Java programming. RMI over IIOP (RMI-IIOP) uses the Common Object Request Broker Architecture (CORBA) standard Internet Inter-ORB Protocol (IIOP) to extend the base Java RMI to perform communication. This allows direct interaction with any other CORBA Object Request Brokers (ORBs), whether they were implemented in Java or another programming language.

The following documentation is available:

- The RMI-IIOP Programmer's Guide is an introduction to writing RMI-IIOP programs.
- The *Java Language to IDL Mapping* document is a detailed technical specification of RMI-IIOP: <http://www.omg.org/cgi-bin/doc?ptc/00-01-06.pdf>.

---

## Implementing the Connection Handler Pool for RMI

Thread pooling for RMI Connection Handlers is not enabled by default.

### About this task

To enable the connection pooling implemented at the RMI TCPTransport level, set the option

```
-Dsun.rmi.transport.tcp.connectionPool=true
```

This version of the Runtime Environment does not have a setting that you can use to limit the number of threads in the connection pool.

---

## Enhanced BigDecimal

From Java 5.0, the IBM BigDecimal class has been adopted by Sun as `java.math.BigDecimal`. The `com.ibm.math.BigDecimal` class is reserved for possible future use by IBM and is currently deprecated. Migrate existing Java code to use `java.math.BigDecimal`.

The new `java.math.BigDecimal` uses the same methods as both the previous `java.math.BigDecimal` and `com.ibm.math.BigDecimal`. Existing code using `java.math.BigDecimal` continues to work correctly. The two classes do not serialize.

To migrate existing Java code to use the `java.math.BigDecimal` class, change the import statement at the top of your `.java` file from: `import com.ibm.math.*;` to `import java.math.*;`

---

## Working in a multiple network stack environment

In a multiple network stack environment (CINET), when one of the stacks fails, no notification or Java exception occurs for a Java program that is listening on an `INADDR_ANY` socket. Also, when new stacks become available, the Java application does not become aware of them until it rebinds the `INADDR` socket.

To avoid this situation, when a TCP/IP stack comes online:

- If the `ibm.socketserver.recover` property is set to false (which is the default), an exception (`NetworkRecycledException`) is thrown to the application to allow it either to fail or to attempt to rebind.
- If the `ibm.socketserver.recover` property is set to true, Java attempts to redrive the socket connection on the new stack if listening on all addresses (`addrs`). If the socket bind cannot be replayed at that time, an exception (`NetworkRecycledException`) is thrown to the application to allow it either to fail or to attempt to rebind.

Both `ServerSocket.accept()` and `ServerSocketChannel.accept()` can throw `NetworkRecycledException`.

While a socket is listening for new connections, it maintains a queue of incoming connections. When `NetworkRecycledException` is thrown and the system attempts to rebind the socket, the connection queue is reset and connection requests in this queue are dropped.

---

## Using IBMJCECCA

IBMJCECCA uses ICSF services during processing. You need to have the correct CSFSERV access to use the ICSF services and IBMJCECCA.

Table 10. CSFSERV access permissions required to use ICSF services

ICSF APIs used by IBMJCECCA	CSF access required
CSNBSYE and CSNESYE (64-bit) Symmetric key encipher	CSFENC Encipher callable service CSFCVE Cryptographic variable encipher callable service
CSNBSYD and CSNESYD (64-bit) Symmetric key decipher	CSFDEC Decipher callable service

Table 10. CSFSERV access permissions required to use ICSF services (continued)

ICSF APIs used by IBMJCECCA	CSF access required
CSNBOWH and CSNEOWH (64-bit) One-way hash generate	<b>CSFOWH</b> One-way hash generate callable service
CSNBRNG and CSNERNG (64-bit) Random number generate	<b>CSFRNG</b> Random number generate callable service
CSNDKRC and CSNFKRC (64-bit) PKDS record create	<b>CSFPKRC</b> PKDS record create callable service <b>CSFKRC</b> Key record create callable service
CSNDKRD and CSNFKRD (64-bit) PKDS record delete	<b>CSFPKRD</b> PKDS record delete callable service <b>CSFKRD</b> Key record delete callable service
CSNDRKD and CSNFRKD (64-bit) Retained key delete	<b>CSFRKD</b> Retained key delete callable service
CSNDPKG and CSNFPKG (64-bit) PKA key generate	<b>CSFPKG</b> PKA key generate callable service
CSNDDSG and CSNFDSG (64-bit) Digital signature generate	<b>CSFDSG</b> Digital signature generate service
CSNDDSV and CSNFDSV (64-bit) Digital signature verify	<b>CSFDSV</b> Digital signature verify callable service
CSNDPKB and CSNFPKB (64-bit) PKA key token build	<b>CSFPKG</b> PKA key generate callable service <b>CSFPKTC</b> PKA key token change callable service
CSNDRKL and CSNFRKL (64-bit) Retained key list	<b>CSFRKL</b> Retained key list callable service
CSNDPKX and CSNFPKX (64-bit) PKA public key extract	<b>CSFPKX</b> PKA Public Key Extract callable service
CSNBENC and CSNEENC (64-bit) Encipher	<b>CSFENC</b> Encipher callable service
CSNBDEC and CSNEDEC (64-bit) Decipher	<b>CSFDEC</b> Decipher callable service
CSNDPKE and CSNFPKE (64-bit) PKA encrypt	<b>CSFPKE</b> PKA encrypt callable service
CSNDPKD and CSNFPKD (64-bit) PKA decrypt	<b>CSFPKD</b> PKA decrypt callable service
CSNDPKI and CSNFPKI (64-bit) PKA key import	<b>CSFPKI</b> PKA key import callable service
CSNBCKM and CSNECKM (64-bit) Multiple clear key import	<b>CSFCKM</b> Multiple clear key import callable service
CSNBKGN and CSNEKGN (64-bit) Key generate	<b>CSFKGN</b> Key generate callable service
CSNDSYI Symmetric key import	<b>CSFSYI</b> Symmetric key import callable service
CSNDSYX Symmetric key export	<b>CSFSYX</b> Symmetric key export callable service

---

## Support for XToolkit

The IBM 64-bit SDK for z/OS, v6 includes XToolkit by default. You need XToolkit when using the Eclipse's SWT\_AWT bridge to build an application that uses both SWT and Swing.

**Restriction:** Motif is no longer supported and will be removed in a later release.

Related links:

- An example, Integrating Swing into Eclipse RCPs: <http://eclipsezone.com/eclipse/forums/t45697.html>
- Reference Information in the Eclipse information center: [http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/api/org/eclipse/swt/awt/SWT\\_AWT.html](http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/api/org/eclipse/swt/awt/SWT_AWT.html)
- Set up information on the Sun Web site: <http://java.sun.com/javase/6/docs/technotes/guides/awt/1.5/xawt.html>



---

## Chapter 6. Applet Viewer

The Java plug-in is used to run Java applications in the browser. The appletviewer is used to test applications designed to be run in a browser. Java Web Start is used to deploy desktop Java applications over a network, and provides a mechanism for keeping them up-to-date.

---

### Distributing Java applications

Java applications typically consist of class, resource, and data files.

When you distribute a Java application, your software package probably consists of the following parts:

- Your own class, resource, and data files
- An installation procedure or program

Your SDK for z/OS software license does **not** allow you to redistribute any of the SDK's files with your application. You must ensure that a licensed version of the SDK for z/OS is installed on the target machine.

When distributing your application for use on a z/OS platform, make the z/OS SDK a prerequisite, because z/OS does not have a separate JRE.



---

## Chapter 7. Class data sharing between JVMs

Class data sharing allows multiple JVMs to share a single space in memory.

The Java Virtual Machine (JVM) allows you to share class data between JVMs by storing it in a cache in shared memory. Sharing reduces the overall virtual memory consumption when more than one JVM shares a cache. Sharing also reduces the startup time for a JVM after the cache has been created. The shared class cache is independent of any active JVM and persists until it is destroyed or the system is IPL'd.

A shared cache can contain:

- Bootstrap classes
- Application classes
- Metadata that describes the classes
- Ahead-of-time (AOT) compiled code

---

### Overview of class data sharing

Class data sharing provides a transparent method of reducing memory footprint and improving JVM start-up time. Java 6 provides new and improved features in cache management, isolation, and performance.

#### Enabling class data sharing

Enable class data sharing by using the `-Xshareclasses` option when starting a JVM. The JVM connects to an existing cache or creates a new cache if one does not exist.

All bootstrap and application classes loaded by the JVM are shared by default. Custom classloaders share classes automatically if they extend the application classloader; otherwise, they must use the Java Helper API provided with the JVM to access the cache. See “Adapting custom classloaders to share classes” on page 55.

The JVM can also store ahead-of-time (AOT) compiled code in the cache for certain methods to improve the startup time of subsequent JVMs. The AOT compiled code is not shared between JVMs, but is cached to reduce compilation time when the JVM starts up. The amount of AOT code stored in the cache is determined heuristically. You cannot control which methods get stored in the cache, but you can set upper and lower limits on the amount of cache space used for AOT code, or you can choose to disable AOT caching completely. See “Class data sharing command-line options” on page 48 for more information.

#### Cache access

A JVM can access a cache with either read-write or read-only access. Any JVM connected to a cache with read-write access can update the cache. Any number of JVMs can concurrently read from the cache, even while another JVM is writing to it.

You must take care if runtime bytecode modification is being used. See “Runtime bytecode modification” on page 54 for more information.

## Dynamic updating of the cache

Because the shared class cache persists beyond the lifetime of any JVM, the cache is updated dynamically to reflect any modifications that might have been made to JARs or classes on the file system. The dynamic updating makes the cache transparent to the application using it.

## Cache security

Access to the shared class cache is limited by operating system permissions and Java security permissions. The shared class cache is created with user access by default unless the **groupAccess** command-line suboption is used. Only a classloader that has registered to share class data can update the shared class cache.

*(31-bit only)* The cache memory is protected against accidental or deliberate corruption using memory page protection. This protection is not an absolute guarantee against corruption because the JVM must unprotect pages to write to them. The only way to guarantee that a cache cannot be modified is to open it read-only.

If a Java SecurityManager is installed, classloaders, excluding the default bootstrap, application, and extension classloaders, must be granted permission to share classes by adding SharedClassPermission lines to the java.policy file. See “Using SharedClassPermission” on page 55. The RuntimePermission createClassLoader restricts the creation of new classloaders and therefore also restricts access to the cache.

## Cache lifespan

Multiple caches can exist on a system and you specify them by name as a suboption to the **-Xshareclasses** command. A JVM can connect to only one cache at any one time.

You can override the default cache size on startup using **-Xscmx<n><size>**. This size is then fixed for the lifetime of the cache. Caches exist until they are explicitly destroyed using a suboption to the **-Xshareclasses** command or until the system is IPL'd.

## Cache utilities

All cache utilities are suboptions to the **-Xshareclasses** command. See “Class data sharing command-line options” or use **-Xshareclasses:help** to see a list of available suboptions.

---

## Class data sharing command-line options

Class data sharing and the cache management utilities are controlled using command-line options to the Java launcher.

For options that take a *<size>* parameter, suffix the number with “k” or “K” to indicate kilobytes, “m” or “M” to indicate megabytes, or “g” or “G” to indicate gigabytes.

**-Xscmaxaot<size>**

Sets the maximum number of bytes in the cache that can be used for AOT

data. Use this option to ensure a certain amount of cache space is available for non-AOT data. By default, the maximum limit for AOT data is the amount of free space in the cache. The value of this option should not be smaller than the value of **-Xscminaot** and must not be larger than the value of **-Xscmx**.

**-Xscminaot**<size>

Sets the minimum number of bytes in the cache to reserve for AOT data. By default, no space is reserved for AOT data, although AOT data is written to the cache until the cache is full or the **-Xscmaxaot** limit is reached. The value of this option must not exceed the value of **-Xscmx** or **-Xscmaxaot**. The value of **-Xscminaot** must always be considerably less than the total cache size because AOT data can be created only for cached classes. If the value of **-Xscminaot** is equal to the value of **-Xscmx**, no class data or AOT data is stored because AOT data must be associated with a class in the cache.

**-Xscmx**<size>

Specifies cache size. This option applies only if a cache is being created and no cache of the same name exists. The default cache size is platform-dependent. You can find out the size value being used by adding **-verbose:sizes** as a command-line argument. The minimum cache size is 4 KB. The maximum cache size is also platform-dependent. (See “Cache size limits” on page 53.)

**-Xshareclasses**:<suboption>[,<suboption>...]

Enables class data sharing. Can take a number of suboptions, some of which are cache utilities. Cache utilities perform the required operation on the specified cache, without starting the VM. You can combine multiple suboptions, separated by commas, but the cache utilities are mutually exclusive. When running cache utilities, the message `Could not create the Java virtual machine` is expected. Cache utilities do not create the virtual machine.

Some cache utilities can work with caches from previous Java versions or caches created by JVMs with different bit-widths. These caches are referred to as “incompatible” caches.

You can use the following suboptions with the **-Xshareclasses** option:

**help**

Lists all the command-line suboptions.

**name**=<name>

Connects to a cache of a given name, creating the cache if it does not already exist. Also used to indicate the cache that is to be modified by cache utilities; for example, **destroy**. Use the **listAllCaches** utility to show which named caches are currently available. If you do not specify a name, the default name “sharedcc\_%u” is used. %u in the cache name inserts the current user name. You can specify “%g” in the cache name to insert the current group name.

**cacheDir**=<directory>

Sets the directory in which cache data is read and written. By default, <directory> is `/tmp/javasharedresources`. The user must have sufficient permissions in <directory>. Caches are stored in shared memory and have control files that describe the location of the memory. Control files are stored in a `javasharedresources` subdirectory of the **cacheDir** specified. Do not move or delete control files in this directory. The **listAllCaches** utility, the **destroyAll** utility, and the **expire** suboption work only in the scope of a given **cacheDir**.

**readonly**

Opens an existing cache with read-only permissions. The JVM does not create a new cache with this suboption. Opening a cache read-only prevents the JVM from making any updates to the cache. It also allows the JVM to connect to caches created by other users or groups without requiring write access. By default, this suboption is not specified.

**groupAccess**

Sets operating system permissions on a new cache to allow group access to the cache. The default is user access only.

**verbose**

Enables verbose output, which provides overall status on the shared class cache and more detailed error messages.

**verboseAOT**

Enables verbose output when compiled AOT code is being found or stored in the cache. AOT code is generated heuristically. You might not see any AOT code generated at all for a small application. You can disable AOT caching using the **noaot** suboption.

**verboseIO**

Gives detailed output on the cache I/O activity, listing information on classes being stored and found. Each classloader is given a unique ID (the bootstrap loader is always 0) and the output shows the classloader hierarchy at work, where classloaders must ask their parents for a class before they can load it themselves. It is normal to see many failed requests; this behavior is expected for the classloader hierarchy.

**verboseHelper**

Enables verbose output for the Java Helper API. This output shows you how the Helper API is used by your ClassLoader.

**silent**

Turns off all shared classes messages, including error messages. Unrecoverable error messages, which prevent the JVM from initializing, are displayed.

**nonfatal**

Allows the JVM to start even if class data sharing fails. Normal behavior for the JVM is to refuse to start if class data sharing fails. If you select **nonfatal** and the shared classes cache fails to initialize, the JVM attempts to connect to the cache in read-only mode. If this attempt fails, the JVM starts without class data sharing.

**none**

Can be added to the end of a command line to disable class data sharing. This suboption overrides class sharing arguments found earlier on the command line.

**modified=<modified context>**

Used when a JVMTI agent is installed that might modify bytecode at runtime. If you do not specify this suboption and a bytecode modification agent is installed, classes are safely shared with an extra performance cost. The *<modified context>* is a descriptor chosen by the user; for example, "myModification1". This option partitions the cache, so that only JVMs using context myModification1 can share the same classes. For instance, if you run HelloWorld with a modification context and then run it again with a different modification context, all classes are stored twice in the cache. See "Runtime bytecode modification" on page 54 for more information.

**reset**

Causes a cache to be destroyed and then recreated when the JVM starts up. Can be added to the end of a command line as `-Xshareclasses:reset`.

**destroy (Utility option)**

Destroys a cache specified by the **name**, **cacheDir**, and **nonpersistent** suboptions. A cache can be destroyed only if all JVMs using it have shut down, and the user has sufficient permissions.

**destroyAll (Utility option)**

Tries to destroy all caches available using the specified **cacheDir** and **nonpersistent** suboptions. A cache can be destroyed only if all JVMs using it have shut down, and the user has sufficient permissions.

**expire=<time in minutes>**

Destroys all caches that have been unused for the time specified before loading shared classes. This option is not a utility option because it does not cause the JVM to exit.

**listAllCaches (Utility option)**

Lists all the compatible and incompatible caches that exist in the specified cache directory. If you do not specify **cacheDir**, the default directory is used. Summary information, such as Java version and current usage is displayed for each cache.

**printStats (Utility option)**

Displays summary information for the cache specified by the **name**, **cacheDir**, and **nonpersistent** suboptions. The most useful information displayed is how full the cache is and how many classes it contains. Stale classes are classes that have been updated on the file system and which the cache has therefore marked "stale". Stale classes are not purged from the cache and can be reused. See the Diagnostics Guide for more information.

**printAllStats (Utility option)**

Displays detailed information for the cache specified by the **name**, **cacheDir**, and **nonpersistent** suboptions. Every class is listed in chronological order, with a reference to the location from which it was loaded. AOT code for class methods is also listed.

See the Diagnostics Guide for more information.

**(31-bit only) mprotect=[ all | default | none ]**

By default, the memory pages containing the cache are protected at all times, unless a specific page is being updated. This protection helps prevent accidental or deliberate corruption to the cache. The cache header is not protected by default because this protection has a small performance cost. Specifying `all` ensures that all the cache pages are protected, including the header. Specifying `none` disables the page protection.

**noBootclasspath**

Prevents storage of classes loaded by the bootstrap classloader in the shared classes cache. Can be used with the `SharedClassLoaderFilter` API to control exactly which classes get cached. See the Diagnostics Guide for more information about shared class filtering.

**cacheRetransformed**

Enables caching of classes that have been transformed using the `JVMTI RetransformClasses` function.

### **noaot**

Disables caching of AOT code. AOT code already in the shared data cache can be loaded.

---

## **Creating, populating, monitoring, and deleting a cache**

An overview of the lifecycle of a shared class data cache including examples of the cache management utilities.

To enable class data sharing, add **-Xshareclasses[:name=<name>]** to your application command line.

The JVM either connects to an existing cache of the given name or creates a new cache of that name. If a new cache is created, it is populated with all bootstrap and application classes being loaded until the cache becomes full. If two or more JVMs are started concurrently, they populate the cache concurrently.

To check that the cache has been created, run `java -Xshareclasses:listAllCaches`. To see how many classes and how much class data is being shared, run `java -Xshareclasses:[name=<name>],printStats`. You can run these utilities after the application JVM has terminated or in another command window.

For more feedback on cache usage while the JVM is running, use the **verbose** suboption. For example, `java -Xshareclasses:[name=<name>],verbose`.

To see classes being loaded from the cache or stored in the cache, add `-Xshareclasses:[name=<name>],verboseIO` to your application command line.

To delete the cache, run `java -Xshareclasses:[name=<name>],destroy`. You usually delete caches only if they contain many stale classes or if the cache is full and you want to create a bigger cache.

You should tune the cache size for your specific application, because the default is unlikely to be the optimum size. To determine the optimum cache size, specify a large cache, using **-Xscmx**, run the application, and then use **printStats** to determine how much class data has been stored. Add a small amount to the value shown in **printStats** for contingency. Because classes can be loaded at any time during the lifetime of the JVM, it is best to do this analysis after the application has terminated. However, a full cache does not have a negative impact on the performance or capability of any JVMs connected to it, so it is acceptable to decide on a cache size that is smaller than required.

If a cache becomes full, a message is displayed on the command line of any JVMs using the **verbose** suboption. All JVMs sharing the full cache then loads any further classes into their own process memory. Classes in a full cache can still be shared, but a full cache is read-only and cannot be updated with new classes.

---

## **Performance and memory consumption**

Class data sharing is particularly useful on systems that use more than one JVM running similar code; the system benefits from reduced virtual memory consumption. It is also useful on systems that frequently start up and shut down JVMs, which benefit from the improvement in startup time.

The overhead to create and populate a new cache is minimal. The JVM startup cost in time for a single JVM is typically between 0 and 5% slower compared with a

system not using class data sharing, depending on how many classes are loaded. JVM startup time improvement with a populated cache is typically between 10% and 40% faster compared with a system not using class data sharing, depending on the operating system and the number of classes loaded. Multiple JVMs running concurrently show greater overall startup time benefits.

Duplicate classes are consolidated in the shared class cache. For example, class A loaded from myClasses.jar and class A loaded from myOtherClasses.jar (with identical content) is stored only once in the cache. The **printAllStats** utility shows multiple entries for duplicated classes, with each entry pointing to the same class.

When you run your application with class data sharing, you can use the operating system tools to see the reduction in virtual memory consumption.

---

## Considerations and limitations of using class data sharing

Consider these factors when deploying class data sharing in a product and using class data sharing in a development environment.

### Cache size limits

The maximum theoretical cache size is 2 GB. The size of cache you can specify is limited by the amount of physical memory and swap space available to the system.

The cache for sharing classes is allocated using the System V IPC Shared memory mechanism.

Because the virtual address space of a process is shared between the shared classes cache and the Java heap, if you increase the maximum size of the Java heap you might reduce the size of the shared classes cache you can create.

### Required APAR for Shared Classes

You must apply z/OS APAR OA11519, available for z/OS R1.6 and onwards, to any z/OS system where shared classes are used. This APAR ensures that multiple shmat requests for the same shared segment will map to the same virtual address across multiple JVMs.

Without this APAR, there is a problem with using shared memory when multiple JVMs reside in a single address space. Each shmat call consumes a separate virtual address range. This is not acceptable because shared classes will run out of shared memory pages prematurely.

### Working with BPXPRMxx settings

Some of the **BPXPRMxx** parmlib settings affect shared classes performance. Using the wrong settings can stop shared classes from working. These settings might also have performance implications.

For further information about performance implications and use of these parameters, refer to the *z/OS MVS Initialization and Tuning Reference (SA22-7592)* at <http://publibz.boulder.ibm.com/epubs/pdf/iea2e280.pdf> and the *z/OS Unix System Services Planning Guide (GA22-7800)* at <http://publibz.boulder.ibm.com/epubs/pdf/bpxzb280.pdf>. The most significant **BPXPRMxx** parameters that affect the operation of shared classes are:

- **MAXSHAREPAGES**, **IPCSHMSPAGES**, **IPCSHMMPAGES**, and **IPCSHMMSSEGS**. These settings affect the amount of shared memory pages available to the JVM. The JVM uses these memory pages for the shared classes cache. If you request large cache sizes, you might have to increase the amount of shared memory pages available.

The shared page size for a z/OS Unix System Service is fixed at 4 KB for 31-bit and 1 MB for 64-bit. Shared classes try to create a 16 MB cache by default on both 31- and 64-bit platforms. Therefore set **IPCSHMMPAGES** greater than 4096 on a 31-bit system.

If you set a cache size using **-Xscmx**, the VM will round up the value to the nearest megabyte. You must take this into account when setting **IPCSHMMPAGES** on your system.

- **IPCSEMNIDS**, and **IPCSEMNSEMS**. These settings affect the amount of SystemV IPC semaphore available to Unix processes. IBM shared classes use System V IPC semaphores to communicate between the JVMs.

## Runtime bytecode modification

Any JVM using a JVM Tool Interface (JVMTI) agent that can modify bytecode data must use the **modified=<modified\_context>** suboption if it wants to share the modified classes with another JVM.

The modified context is a user-specified descriptor that describes the type of modification being performed. The modified context partitions the cache so that all JVMs running under the same context share a partition.

This partitioning allows JVMs that are not using modified bytecode to safely share a cache with those that are using modified bytecode. All JVMs using a given modified context must modify bytecode in a predictable, repeatable manner for each class, so that the modified classes stored in the cache have the expected modifications when they are loaded by another JVM. Any modification must be predictable because classes loaded from the shared class cache cannot be modified again by the agent.

If a JVMTI agent is used without a modification context, classes are still safely shared by the JVM, but with a small impact on performance. Using a modification context with a JVMTI agent avoids the need for extra checks and therefore has no impact on performance. A custom `ClassLoader` that extends `java.net.URLClassLoader` and modifies bytecode at load time without using JVMTI automatically stores that modified bytecode in the cache, but the cache does not treat the bytecode as modified. Any other VM sharing that cache loads the modified classes. You can use the **modified=<modification\_context>** suboption in the same way as with JVMTI agents to partition modified bytecode in the cache. If a custom `ClassLoader` needs to make unpredictable load-time modifications to classes, that `ClassLoader` must not attempt to use class data sharing.

See the Diagnostics Guide for more detail on this topic.

## Operating system limitations

Temporary disk space must be available to hold cache information. The operating system enforces cache permissions.

The shared class cache requires disk space to store identification information about the caches that exist on the system. This information is stored in `/tmp/javasharedresources`. If the identification information directory is deleted, the

JVM cannot identify the shared classes on the system and must re-create the cache. Use the `ipcs` command to view the memory segments used by a JVM or application.

Users running a JVM must be in the same group to use a shared class cache. The operating system enforces the permissions for accessing a shared class cache. If you do not specify a cache name, the user name is appended to the default name so that multiple users on the same system create their own caches by default.

## Using SharedClassPermission

If a SecurityManager is being used with class data sharing and the running application uses its own class loaders, you must grant these class loaders shared class permissions before they can share classes.

You add shared class permissions to the `java.policy` file using the `ClassLoader` class name (wildcards are permitted) and either `"read"`, `"write"`, or `"read,write"` to determine the access granted. For example:

```
permission com.ibm.oti.shared.SharedClassPermission
    "com.abc.customclassloaders.*", "read,write";
```

If a `ClassLoader` does not have the correct permissions, it is prevented from sharing classes. You cannot change the permissions of the default bootstrap, application, or extension class loaders.

---

## Adapting custom classloaders to share classes

Any classloader that extends `java.net.URLClassLoader` can share classes without modification. You must adopt classloaders that do not extend `java.net.URLClassLoader` to share class data.

You must grant all custom classloaders shared class permissions if a SecurityManager is being used; see "Using SharedClassPermission." IBM provides several Java interfaces for various types of custom classloaders, which allow the classloaders to find and store classes in the shared class cache. These classes are in the `com.ibm.oti.shared` package.

The Javadoc for this package is provided with the SDK in the `docs/content/apidoc` directory.

See the Diagnostics Guide for more information about how to use these interfaces.



---

## Chapter 8. Service and support for independent software vendors

Contact points for service:

If you are entitled to services for the Program code pursuant to the IBM Solutions Developer Program, contact the IBM Solutions Developer Program through your normal method of access or on the Web at: <http://www.ibm.com/partnerworld/>.

If you have purchased a service contract (that is, IBM's Personal Systems Support Line or equivalent service by country), the terms and conditions of that service contract determine what services, if any, you are entitled to receive with respect to the Program.



---

## Chapter 9. Accessibility

The user guides that are supplied with this SDK and the Runtime Environment have been tested using screen readers.

To change the font sizes in the user guides, use the function that is supplied with your browser, typically found under the **View** menu option.

For users who require keyboard navigation, a description of useful keystrokes for Swing applications is in *Swing Key Bindings* at <http://www.ibm.com/developerworks/java/jdk/additional/>.

---

### Keyboard traversal of JComboBox components in Swing

If you traverse the drop-down list of a JComboBox component with the cursor keys, the button or editable field of the JComboBox does not change value until an item is selected. This is the correct behavior for this release and improves accessibility and usability by ensuring that the keyboard traversal behavior is consistent with mouse traversal behavior.



---

## Chapter 10. Any comments on this user guide?

If you have any comments about this user guide, contact us through one of the following channels. Note that these channels are not set up to answer technical queries, but are for comments about the documentation only.

Send your comments:

- By e-mail to [idrcf@hursley.ibm.com](mailto:idrcf@hursley.ibm.com).
- By fax:
  - From the UK: 01962 842327
  - From elsewhere: +44 1962 842327
- By mail to:

IBM United Kingdom Ltd  
User Technologies,  
Mail Point 095  
Hursley Park  
Winchester  
Hampshire  
SO21 2JN  
United Kingdom

**The fine print.** By choosing to send a message to IBM, you acknowledge that all information contained in your message, including feedback data, such as questions, comments, suggestions, or the like, shall be deemed to be non-confidential and IBM shall have no obligation of any kind with respect to such information and shall be free to reproduce, use, disclose, and distribute the information to others without limitation. Further, IBM shall be free to use any ideas, concepts, know-how or techniques contained in such information for any purpose whatsoever, including, but not limited to, developing, manufacturing and marketing products incorporating such information.



---

## Appendix A. Nonstandard options

The **-X** options listed below are nonstandard and subject to change without notice.

For options that take a *<size>* parameter, suffix the number with "k" or "K" to indicate kilobytes, "m" or "M" to indicate megabytes, or "g" or "G" to indicate gigabytes.

For options that take a *<percentage>* parameter, use a number from 0 to 1. For example, 50% is 0.5.

### **-Xargencoding**

Allows you to put Unicode escape sequences in the argument list. This option is set to off by default.

### **-Xbootclasspath:<directories and zip or jar files separated by :>**

Sets the search path for bootstrap classes and resources. The default is to search for bootstrap classes and resources within the internal VM directories and .jar files.

### **-Xbootclasspath/a:<directories and zip or jar files separated by :>**

Appends the specified directories, zip, or jar files to the end of bootstrap class path. The default is to search for bootstrap classes and resources within the internal VM directories and .jar files.

### **-Xbootclasspath/p:<directories and zip or jar files separated by :>**

Prepends the specified directories, zip, or jar files to the front of the bootstrap class path. Do not deploy applications that use the **-Xbootclasspath:** or **-Xbootclasspath/p:** option to override a class in the standard API, because such a deployment would contravene the Java Runtime Environment binary code license. The default is to search for bootstrap classes and resources within the internal VM directories and .jar files.

### **-Xcheck:classpath**

Displays a warning message if an error is discovered in the class path, for example, a missing directory or JAR file.

### **-Xcheck:gc[:<scan options>][:<verify options>][:<misc options>]**

Performs additional checks on garbage collection. By default, no checking is performed. See the output of **-Xcheck:gc:help** for more information.

### **-Xcheck:jni**

Performs additional checks for JNI functions. By default, no checking is performed.

### **-Xcheck:memory[:<option>]**

Identifies memory leaks inside the JVM using strict checks that cause the JVM to exit on failure. If no option is specified, **all** is used by default. See the output of **-Xcheck:memory:help** or the Diagnostics Guide for more information.

### **-Xcheck:nabounds**

Performs additional checks for JNI functions. By default, no checking is performed.

### **-Xclassgc**

Enables collection of class objects at every garbage collection. See also **-Xnoclassgc**. By default, this option is enabled.

- Xcodecache<size>**  
Sets the unit size of which memory blocks are allocated to store native code of compiled Java methods. An appropriate size can be chosen for the application being run. By default, this is selected internally according to the CPU architecture and the capability of your system.
- Xcompactexplicitgc**  
Performs a compaction for every call to `System.gc()`. See also **-Xnocompactexplicitgc**. By default, compaction occurs only when triggered internally.
- Xcompactgc**  
Performs a compaction for every garbage collection. See also **-Xnocompactgc**. By default, compaction occurs only when triggered internally.
- Xcompressedrefs**  
(64-bit only) Uses 32-bit values for references. See “More effective heap usage using compressed references” on page 18 for more information. By default, references are 64-bit.
- Xconcurrentbackground<number>**  
Specifies the number of low priority background threads attached to assist the mutator threads in concurrent mark. The default is 1.
- Xconcurrentlevel<number>**  
Specifies the allocation “tax” rate. It indicates the ratio between the amount of heap allocated and the amount of heap marked. The default is 8.
- Xconmeter:<soa | loa | dynamic>**  
Determines which area’s usage, LOA (Large Object Area) or SOA (Small Object Area), is metered and hence which allocations are taxed during concurrent mark. The allocation tax is applied to the selected area. If **-Xconmeter:dynamic** is specified, the collector dynamically determines the area to meter based on which area is exhausted first. By default, the option is set to **-Xconmeter:soa**.
- Xdbg:<options>**  
Loads debugging libraries to support the remote debugging of applications. See “Debugging Java applications” on page 30 for more information. Specifying **-Xrunjdw** provides the same support.
- Xdebug**  
Starts the JVM with the debugger enabled. By default, the debugger is disabled.
- Xdisableexcessivegc**  
Disables the throwing of an `OutOfMemoryError` if excessive time is spent in the GC. By default, this option is off.
- Xdisableexplicitgc**  
Signals to the VM that calls to `System.gc()` should have no effect. By default, calls to `System.gc()` trigger a garbage collection.
- Xdisablestringconstantgc**  
Prevents strings in the string intern table from being collected. By default, this option is disabled.
- Xdisablejavadump**  
Turns off Javacore generation on errors and signals. By default, Javacore generation is enabled.
- Xenableexcessivegc**  
If excessive time is spent in the GC, this option returns `NULL` for an allocate

request and thus causes an `OutOfMemoryError` to be thrown. This action occurs only when the heap has been fully expanded and GC is taking 95% of the available time. This behavior is the default.

**-Xenableexplicitgc**

Signals to the VM that calls to `System.gc()` should trigger a garbage collection. This is the default.

**-Xenablestringconstantgc**

Enables strings from the string intern table to be collected. By default, this option is enabled.

**-Xfuture**

Turns on strict class-file format checks. Use this flag when you are developing new code because stricter checks will become the default in future releases. By default, strict format checks are disabled.

**-Xgcpolicy:***<optthruput | optavgpause | gencon | subpool>*

Controls the behavior of the Garbage Collector. See “Garbage collection options” on page 18 for more information.

**-Xgcthreads***<number of threads>*

Sets the number of helper threads that are used for parallel operations during garbage collection. By default, the number of threads is one less than the number of physical CPUs present, with a minimum of 1. The maximum value is the number of physical CPUs present.

**-Xgcworkpackets***<number>*

Specifies the total number of work packets available in the global collector. If not specified, the collector allocates a number of packets based on the maximum heap size.

**-Xifa:***[on | off | force]*

z/OS R6 can run Java applications on a new type of special-purpose assist processor called the *eServer™ zSeries Application Assist Processor (zAAP)*. The zSeries Application Assist Processor is also known as an IFA (Integrated Facility for Applications).

The **-Xifa** option enables Java applications to run on IFAs if these are available. Only Java code and system native methods can be on IFA processors.

If not specified, this option is set to on.

**-Xint**

Makes the JVM use only the Interpreter, disabling the Just-In-Time (JIT) compiler. By default, the JIT compiler is enabled.

**-Xiss***<size>*

Sets the initial Java thread stack size. 2 KB by default. Use the **-verbose:sizes** option to output the value that the VM is using.

**-Xjarversion**

See “Obtaining version information” on page 13.

**-Xjit:***<suboption>,<suboption>...*

Enables the JIT. For details of the sub-options, see the Diagnostics Guide. See also **-Xnojit**. By default, the JIT is enabled.

**-Xjni:***<suboptions>*

Sets JNI options. You can use the following suboptions with the **-Xjni** option:

**-Xjni:arrayCacheMax=***[<size in bytes> | unlimited]*

Sets the maximum size of the array cache. The default size is 8096 bytes.

**-Xlinenumbers**

Displays line numbers in stack traces, for debugging. See also **-Xnolinenumbers**. By default, line numbers are on.

**-Xloa**

Allocates a large object area (LOA). Objects will be allocated in this LOA rather than the SOA. By default, the LOA is enabled for all GC policies except for subpool, where the LOA is not available. See also **-Xnoloa**.

**-Xloainitial**<percentage>

<percentage> is between 0 and 0.95, which specifies the initial percentage of the current tenure space allocated to the large object area (LOA). The default is 0.05 or 5%.

**-Xloamaximum**<percentage>

<percentage> is between 0 and 0.95, which specifies the maximum percentage of the current tenure space allocated to the large object area (LOA). The default is 0.5 or 50%.

**-Xlp**

Requests the JVM to allocate the Java heap with large pages. If large pages are not available, the JVM will not start, displaying the error message GC: system configuration does not support option --> '-Xlp'. Large page support requires z/OS V1.9 or later with APAR OA25485 and a System z10 processor or later. z/OS needs to be configured for large pages by the system programmer. Users who require large pages must be authorized to the IARRSM.LRGPAGES resource in the RACF (or an equivalent security product) FACILITY class with read authority.

**-Xmaxe**<size>

Sets the maximum amount by which the garbage collector expands the heap. Typically, the garbage collector expands the heap when the amount of free space falls below 30% (or the amount specified using **-Xminf**), by the amount required to restore the free space to 30%. The **-Xmaxe** option limits the expansion to the specified value; for example **-Xmaxe10M** limits the expansion to 10 MB. By default, there is no maximum expansion size.

**-Xmaxf**<percentage>

Specifies the maximum percentage of heap that must be free after a garbage collection. If the free space exceeds this amount, the JVM attempts to shrink the heap. The default value is 0.6 (60%).

**-Xmca**<size>

Sets the expansion step for the memory allocated to store the RAM portion of loaded classes. Each time more memory is required to store classes in RAM, the allocated memory is increased by this amount. By default, the expansion step is 32 KB. Use the **-verbose:sizes** option to output the value that the VM is using.

**-Xmco**<size>

Sets the expansion step for the memory allocated to store the ROM portion of loaded classes. Each time more memory is required to store classes in ROM, the allocated memory is increased by this amount. By default, the expansion step is 128 KB. Use the **-verbose:sizes** option to output the value that the VM is using.

**-Xmine**<size>

Sets the minimum amount by which the Garbage Collector expands the heap. Typically, the garbage collector expands the heap by the amount required to restore the free space to 30% (or the amount specified using **-Xminf**). The

**-Xmine** option sets the expansion to be at least the specified value; for example, **-Xmine50M** sets the expansion size to a minimum of 50 MB. By default, the minimum expansion size is 1 MB.

**-Xminf**<percentage>

Specifies the minimum percentage of heap that should be free after a garbage collection. If the free space falls below this amount, the JVM attempts to expand the heap. By default, the minimum value is 0.3 (30%).

**-Xmn**<size>

Sets the initial and maximum size of the new area to the specified value when using **-Xgcpolicy:gencon**. Setting **-Xmn** is equivalent to setting **-Xmns** and **-Xmnx**. If you set either **-Xmns** or **-Xmnx**, you cannot set **-Xmn**. If you attempt to set **-Xmn** with either **-Xmns** or **-Xmnx**, the VM will not start, returning an error. By default, **-Xmn** is selected internally according to your system's capability. Use the **-verbose:sizes** option to output the value that the VM is using.

**-Xmns**<size>

Sets the initial size of the new area to the specified value when using **-Xgcpolicy:gencon**. By default, this option is selected internally according to your system's capability. This option will return an error if you try to use it with **-Xmn**. Use the **-verbose:sizes** option to output the value that the VM is using.

**-Xmnx**<size>

Sets the maximum size of the new area to the specified value when using **-Xgcpolicy:gencon**. By default, this option is selected internally according to your system's capability. This option will return an error if you try to use it with **-Xmn**. Use the **-verbose:sizes** option to output the value that the VM is using.

**-Xmo**<size>

Sets the initial and maximum size of the old (tenured) heap to the specified value when using **-Xgcpolicy:gencon**. Equivalent to setting both **-Xmos** and **-Xmox**. If you set either **-Xmos** or **-Xmox**, you cannot set **-Xmo**. If you attempt to set **-Xmo** with either **-Xmos** or **-Xmox**, the VM will not start, returning an error. By default, **-Xmo** is selected internally according to your system's capability. Use the **-verbose:sizes** option to output the value that the VM is using.

**-Xmoi**<size>

Sets the amount the Java heap is incremented when using **-Xgcpolicy:gencon**. If set to zero, no expansion is allowed. By default, the increment size is calculated on the expansion size, **-Xmine** and **-Xminf**.

**-Xmos**<size>

Sets the initial size of the old (tenure) heap to the specified value when using **-Xgcpolicy:gencon**. By default, this option is selected internally according to your system's capability. This option will return an error if you try to use it with **-Xmo**. Use the **-verbose:sizes** option to output the value that the VM is using.

**-Xmox**<size>

Sets the maximum size of the old (tenure) heap to the specified value when using **-Xgcpolicy:gencon**. By default, this option is selected internally according to your system's capability. This option will return an error if you try to use it with **-Xmo**. Use the **-verbose:sizes** option to output the value that the VM is using.

- Xmr<size>**  
Sets the size of the Garbage Collection "remembered set" when using **-Xgcpolicy:gencon**. This is a list of objects in the old (tenured) heap that have references to objects in the new area heap. By default, this option is set to 16 kilobytes. Use the **-verbose:sizes** option to output the value that the VM is using.
- Xmrx<size>**  
Sets the remembered maximum size setting.
- Xms<size>**  
Sets the initial Java heap size. You can also use **-Xmo**. The default is set internally according to your system's capability. Use the **-verbose:sizes** option to output the value that the VM is using.
- Xmso<size>**  
Sets the C stack size for forked Java threads. By default, this option is set to 32 KB on 32-bit platforms and 256 KB on 64-bit platforms. Use the **-verbose:sizes** option to output the value that the VM is using.
- Xmx<size>**  
Sets maximum Java heap size. By default, this option is set internally according to your system's capability. Use the **-verbose:sizes** option to output the value that the VM is using.
- Xnocsclassgc**  
Disables class garbage collection. This option switches off garbage collection of storage associated with Java classes that are no longer being used by the JVM. See also **-Xclassgc**. By default, class garbage collection is performed.
- Xnocompactexplicitgc**  
Disables compaction on a call to `System.gc()`. See also **-Xcompactexplicitgc**. By default, compaction is enabled on calls to `System.gc()`.
- Xnocompactgc**  
Disables compaction for the Garbage Collector. See also **-Xcompactgc**. By default, compaction is enabled.
- Xnojit**  
Disables the JIT compiler. See also **-Xjit**. By default, the JIT compiler is enabled.
- Xnolinenumbers**  
Disables the line numbers for debugging. See also **-Xlinenumbers**. By default, line number are on.
- Xnoloa**  
Prevents allocation of a large object area (LOA). All objects will be allocated in the SOA. By default, the LOA is enabled for all GC policies except for subpool, where the LOA is not available. See also **-Xloa**.
- Xnopartialcompactgc**  
Disables incremental compaction. See also **-Xpartialcompactgc**.
- Xnosigcatch**  
Disables JVM signal handling code. See also **-Xsigcatch**. By default, signal handling is enabled.
- Xnosigchain**  
Disables signal handler chaining. See also **-Xsigchain**. By default, the signal handler chaining is enabled.

**-Xoptionsfile=<file>**

Specifies a file that contains JVM options and defines. By default, no option file is used.

The options file does not support the following options:

- **-version**
- **-fullversion**
- **-Xjarversion**
- **-memorycheck**
- **-assert**
- **-help**

**-Xoss<size>**

Sets the Java stack size and C stack size for any thread. This option is provided for compatibility and is equivalent to setting both **-Xss** and **-Xmso** to the specified value.

**-Xpartialcompactgc**

Enables partial compaction. By default, this option is not set, so all compactions are full. See also **-Xnopartialcompactgc**.

**-Xquickstart**

Improves startup time by delaying JIT compilation and optimizations. By default, quickstart is disabled and there is no delay in JIT compilation.

**-Xrdbginfo:<host>:<port>**

Loads and passes options to the remote debug information server. By default, the remote debug information server is disabled.

**-Xrs**

Reduces the use of operating system signals. By default, the VM makes full use of operating system signals, see “Signals used by the JVM” on page 33.

**-Xrun<library name>[:<options>]**

Loads helper libraries. To load multiple libraries, specify it more than once on the command line. Examples of these libraries are:

**-Xrunhprof[:help] | [:<option>=<value>, ...]**

Performs heap, CPU, or monitor profiling. For more information, see the Diagnostics Guide.

**-Xrunjdp[:help] | [:<option>=<value>, ...]**

Loads debugging libraries to support the remote debugging of applications. See **-Xdbg** for more information.

**-Xrunjchk[:help] | [:<option>=<value>, ...]**

Deprecated, use **-Xcheck:jni**.

**-Xscmx<size>**

For details of **-Xscmx**, see “Class data sharing command-line options” on page 48.

**-XselectiveDebug**

Enables selective debugging. Use the `com.ibm.jvm.Debuggable` annotation to mark classes and methods that should be available for debugging. The JVM optimizes methods that do not need debugging to provide better performance in a debugging environment. See “Selective debugging” on page 31 for more information.

**-Xshareclasses:***<options>*

For details of the **-Xshareclasses** options, see “Class data sharing command-line options” on page 48.

**-Xsigcatch**

Enables VM signal handling code. See also **-Xnosigcatch**. By default, signal handling is enabled.

**-Xsigchain**

Enables signal handler chaining. See also **-Xnosigchain**. By default, signal handler chaining is enabled.

**-Xsoftrefthreshold***<number>*

Sets the hint used by the GC to determine the number of GCs after which a soft reference will be cleared if its referent has not been marked. The default is 32, meaning that the soft reference will be cleared after 32\*(percent of free heap space) GC cycles where its referent was not marked.

**-Xss***<size>*

Sets the maximum Java stack size for any thread. By default, this option is set to 256 KB. Use the **-verbose:sizes** option to output the value that the VM is using.

**-Xverbosegclog:***<path to file>*[*X,Y*]

Causes verbose garbage collection (GC) output to be written to the specified file. If the file exists, it is overwritten. Otherwise, if an existing file cannot be opened or a new file cannot be created, the output is redirected to stderr. If you specify the arguments *X* and *Y* (both are integers) the verbose GC output is redirected to *X* number of files, each containing *Y* number of gc cycles worth of verbose GC output. These files have the form *filename1*, *filename2*, and so on. By default, no verbose GC logging occurs.

See the Diagnostics Guide for more information about verbose GC output.

**-Xverify**

Enables strict class checking for every class that is loaded. By default, strict class checking is disabled.

**-Xverify:none**

Disables strict class checking. By default, strict class checking is disabled.

---

## Appendix B. Known limitations

Known limitations on the SDK and Runtime Environment for z/OS.

If you find a problem, see the “Hints and Tips” pages, at <http://www.ibm.com/servers/eserver/zseries/software/java/javafaq.html>.

If you find a problem that you have been unable to solve after looking through the “Hints and Tips” pages, see <http://www.ibm.com/servers/eserver/zseries/software/java/services.html> for advice and information on how to raise problems.

You can find more help with problem diagnosis in the *Diagnostics Guide* at <http://www.ibm.com/developerworks/java/jdk/diagnosis/60.html>.

### Limitation on classpath length

If there are more than 2031 characters in your classpath the shell truncates your classpath to 2031 characters. If you need a classpath longer than 2031 characters use the extension classloader option to reference directories containing your .jar files. For example:

```
-Djava.ext.dirs=<directory>
```

Where <directory> is the directory containing your .jar files.

### JConsole monitoring tool Local tab

In IBM’s JConsole tool, the **Local** tab, which allows you to connect to other Virtual Machines on the same system, is not available. Also, the corresponding command-line **pid** option is not supported. Instead, use the **Remote** tab in JConsole to connect to the Virtual Machine that you want to monitor. Alternatively, use the **connection** command-line option, specifying a host of localhost and a port number. When you launch the application that you want to monitor, set these command-line options:

```
-Dcom.sun.management.jmxremote.port=<value>
```

Specifies the port the management agent should listen on.

```
-Dcom.sun.management.jmxremote.authenticate=false
```

Disables authentication unless you have created a username file.

```
-Dcom.sun.management.jmxremote.ssl=false
```

Disables SSL encryption.

### Incorrect stack traces when loading new classes after an Exception is caught

If new classes are loaded after an Exception has been caught, the stack trace contained within the Exception might become incorrect. The stack trace becomes incorrect if classes in the stack trace are unloaded, and new classes are loaded into their memory segments.

## ThreadMXBean Thread User CPU Time limitation

There is no way to distinguish between user mode CPU time and system mode CPU time on this platform. `ThreadMXBean.getThreadUserTime()`, `ThreadMXBean.getThreadCpuTime()`, `ThreadMXBean.getCurrentThreadUserTime()`, and `ThreadMXBean.getCurrentThreadCpuTime()` all return the total CPU time for the required thread.

## NullPointerException with the GTK Look and Feel

### DBCS environments only

If your application fails with a `NullPointerException` using the GTK Look and Feel, unset the `GNOME_DESKTOP_SESSION_ID` environment variable.

## ASCII to EBCDIC

Because z/OS uses the EBCDIC character encoding instead of the more common ASCII encoding, sometimes there are portability problems with Java code written on z/OS. Within the scope of the JVM, all character and string data is stored and manipulated in Unicode, and I/O data outside of the virtual machine (disk, network, and so on) is converted to the native platform encoding. However, Java applications that implicitly assume ASCII in specific situations might require some alterations to run as expected under z/OS. For example, a platform-neutral application might have hard coded dependencies, such as literals in ASCII.

The Java language contains the abstractions necessary to handle the switch between character encodings. The various `Reader` and `Writer` classes in the `java.io` package provide alternate constructors with a specified codepage. This mechanism is used for internationalization support, and it can also be used to force ASCII (or other) I/O where required. Not all I/O needs to be overridden; for example, character output to the display should remain in the native encoding.

In addition to the `Reader` and `Writer` classes, there are a few specific situations that might require additional care. For example, the `String` class has an overloaded `getBytes()` method that takes an encoding as an additional parameter. This is useful for direct string manipulation when you are implementing custom data streams or network protocols directly in Java.

In general, straightforward workarounds are available for character encoding problems. Some encoding problems are not visible to the application because they are handled within programs running on z/OS. An example of this is Java Database Connectivity (JDBC).

## IPv6 multicast support

z/OS V1R6 currently does not support IPv4-mapped Multicast addresses. If you are using an IPv4 Multicast address, you cannot join a Multicast group unless you disable IPv6 support by setting the `java.net.preferIPv4Stack` property to true.

Use the following to set this property on the command line:

```
java -Djava.net.preferIPv4Stack=true <classname>
```

## Unicode Shift\_JIS codepage alias

### Japanese users only

The unicode codepage alias “\u30b7\u30d5\u30c8\u7b26\u53f7\u5316\u8868\u73fe” for Shift\_JIS has been removed. If you use this codepage in your applications, replace it with Shift\_JIS.



---

## Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area.

Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive, Armonk  
NY 10504-1758 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

JIMMAIL@uk.ibm.com

[Hursley Java Technology Center (JTC) contact]

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

---

## Trademarks

IBM, the IBM logo, and `ibm.com` are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (<sup>®</sup> or <sup>™</sup>), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at <http://www.ibm.com/legal/copytrade.shtml>.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.







Printed in USA