IBM SDK, Java Technology Edition
Version 6

Windows User Guide

IBM SDK, Java Technology Edition
Version 6

Windows User Guide

Note
Before you use this information and the product it supports, read the information in "Notices" on page 123.

Contents

Preface

This guide provides general information about the IBM® SDK, Java™ Technology Edition, Version 6, for all supported 32-bit Windows architectures. The guide gives specific information about any differences in the IBM implementation compared with the Oracle implementation.

Read this information in conjunction with the documentation on the Oracle Web site: http://www.oracle.com/technetwork/java/index.html.

Late breaking information about this release that is not available in the guide can be found here: http://www.ibm.com/support/docview.wss?uid=swg21587401.

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

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

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

For Service Refresh 9 and earlier
The guide is available in the code package. Technical changes made for a version of the user guide are indicated by blue chevrons.

For Service Refresh 10 and later
The guide is available online. Any modifications made to this user guide to support a later service refresh are indicated by graphic images, for example: 

In this image, SR11 indicates that changes are made for Service Refresh 11. End of changes are marked by 

The guide is also available for download as a PDF. Technical changes made for this version are indicated by vertical bars at the beginning of the line.

To determine the service refresh or fix pack level of an installed version, see "Obtaining version information" on page 13.

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).

The SDK includes the Runtime Environment for Windows, which enables you only to run Java applications. If the SDK was installed, the Runtime Environment is included.

The Runtime Environment contains the Java Virtual Machine and supporting files including class files. The Runtime Environment contains only a subset of the classes that are found in the SDK and allows you to support a Java program at run time but does not provide compilation of Java programs. The Runtime Environment for Windows does not include any of the development tools, for example **appletviewer.exe** or the Java compiler (**javac.exe**), or classes that are only for development systems.

Version compatibility

In general, any applet or application that ran with a previous version of the SDK should run correctly with this release. Classes that are compiled with this release are not guaranteed to work on previous releases.

The 32-bit Windows edition of IBM SDK, Java Technology Edition, Version 6 is built with Microsoft Visual Studio .NET 2003.

For information about compatibility issues between releases, see the Oracle Web site at:

http://www.oracle.com/technetwork/java/javase/compatibility-137541.html

http://www.oracle.com/technetwork/java/javase/compatibility-137462.html

http://www.oracle.com/technetwork/java/javase/compatibility-j2se1-141394.html

http://www.oracle.com/technetwork/java/javase/compatibility-135119.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 earlier IBM SDK or JREs

From Version 5.0, the IBM Runtime Environment for Windows 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 XL TXE-J compiler replaces the XSLT4J interpreter as the default XSLT processor. If you are migrating applications from older versions of Java, see "Migrating to the XL-TXE-J" on page 25.

- The JVM dynamic link library `jvm.dll` is now stored in jre\bin\j9vm and jre\bin\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://docs.oracle.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 "JVM command-line options" on page 83.
- 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 Oracle's Java 6 Compatibility Documentation: http://www.oracle.com/technetwork/java/javase/compatibility-137541.html
- For additional deprecated API information, see Oracle's Java 6 Deprecated API List: http://docs.oracle.com/javase/6/docs/api/deprecated-list.html
- Tracing class dependencies, started using **-verbose:Xclassdep**, is not supported. If you specify **-verbose:Xclassdep**, the JVM will issue an error message and will not start.
- The JVM detects the operating system locale and sets the language preferences accordingly. For example, if the locale is set to **fr_FR**, JVM messages will be printed in French. To avoid seeing JVM messages in the language of the detected locale, remove the file `$SDK/jre/bin/java_xx.properties` where *xx* is the locale, such as **fr**, and the JVM will print messages in English.
- The currency symbol code for Zambia is now corrected to the value "ZMW".

Many new features and capabilities, which might present planning considerations, can be found here: Summary of changes.

Supported environments

This release is supported on certain hardware platforms and operating systems, and is tested on specific virtualization environments.

Hardware platform

The 32-bit release for Windows runs on hardware that supports the Intel 32-bit architecture.

Operating system

The following table shows the latest operating system level tested for each platform architecture. The table indicates whether support for an operating system release was included at the "general availability" (GA) date for the release, or at a later date in a service refresh (SR) or fix pack (FP):

Table 1. Supported Operating Systems

Operating system	**Release supported**
Windows 2000 SP4	GA
Windows XP	GA
Windows Vista	GA
Windows 7	SR6
Windows 8	SR12
Windows 10	SR16 FP7
Windows Server 2003	GA
Windows Server 2003 R2	SR1
Windows Server 2008	SR1
Windows Server 2012	SR12
Windows Server 2012 R2	SR15

Note: Windows 2000 SP4 is supported by Microsoft only under an extended support contract. IPV6 is not supported.

Virtualization software

For information about the virtualization software tested, see "Support for virtualization software" on page 121.

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. Java Native Interface (JNI) applications will have some minor dependencies.

The demo files, and the accompanying license and copyright files are the only documentation included in this SDK for Windows. You can view Oracle's software documentation by visiting the Oracle Web site: http://www.oracle.com/technetwork/java/javase/documentation/index.html.

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 classes are the compiled class files for the platform and must remain compressed 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 might be removed for later versions of the SDK. See "Expired GTE Cybertrust Certificate" on page 118 for more information.
- JRE tools - The following tools are part of the Runtime Environment and are in the `C:\Program Files\IBM\Java60\jre\bin` directory unless otherwise specified.

  `ikeycmd.exe (iKeyman command-line utility)`
  Allows you to manage keys, certificates, and certificate requests from the command line. For more information see the accompanying Security documentation, which includes the *iKeyman User Guide*.

  `ikeyman.exe (iKeyman GUI utility)`
  Allows you to manage keys, certificates, and certificate requests. For more information see the accompanying Security documentation, which includes the *iKeyman User Guide*. There is also a command-line version of this utility.

  `jaaslogon.exe`
  A Windows service that enables JAAS Active Login applications to change their effective user at run time using the JAAS Active Login API.

  `java.exe (Java Interpreter)`
  Runs Java classes. The Java Interpreter runs programs that are written in the Java programming language.

  `javacpl.exe (Java Control Panel)`
  Configures your Runtime Environment.

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

javaws.exe (Java Web Start)
Enables the deployment and automatic maintenance of Java applications. For more information, see "Running Web Start" on page 51.

jextract.exe (Dump extractor)
Converts a system-produced dump into a common format that can be used by **jdmpview**. For more information, see **jdmpview**.

keytool.exe (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.exe
Obtains and caches Kerberos ticket-granting tickets.

klist.exe
Displays entries in the local credentials cache and key table.

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

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

policytool.exe (Policy File Creation and Management Tool)
Creates and modifies the external policy configuration files that define your installation's Java security policy.

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

rmiregistry.exe (Java remote object registry)
Creates and starts a remote object registry on the specified port of the current host.

tnameserv.exe (Common Object Request Broker Architecture (CORBA) transient naming service)
Starts the CORBA transient naming service.

unpack200.exe
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 C:\Program Files\IBM\Java60\bin directory:

appletviewer.exe (Java Applet Viewer)
Tests and runs applets outside a Web browser.

apt.exe (Annotation Processing Tool)
Finds and executes annotation processors based on the annotations present in the set of specified source files being examined.

extcheck.exe (Extcheck utility)
Detects version conflicts between a target jar file and currently-installed extension jar files.

HtmlConverter.exe (Java Plug-in HTML Converter)
Converts an HTML page that contains applets to a format that can use the Java Plug-in.

idlj.exe (IDL to Java Compiler)
Generates Java bindings from a given IDL file.

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

jarsigner.exe (JAR Signing and Verification Tool)
Generates signatures for JAR files and verifies the signatures of signed JAR files.

java.exe (Java Interpreter)
Runs Java classes. The Java Interpreter runs programs that are written in the Java programming language.

java-rmi.exe (HTTP-to-CGI request forward tool)
Accepts RMI-over-HTTP requests and forwards them to an RMI server listening on any port.

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

javadoc.exe (Java Documentation Generator)
Generates HTML pages of API documentation from Java source files.

javah.exe (C Header and Stub File Generator)
Enables you to associate native methods with code written in the Java programming language.

javap.exe (Class File Disassembler)
Disassembles compiled files and can print a representation of the bytecodes.

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

javaws.exe (Java Web Start)
Enables the deployment and automatic maintenance of Java applications. For more information, see "Running Web Start" on page 51.

jconsole.exe (JConsole Monitoring and Management Tool)
Monitors local and remote JVMs using a GUI. JMX-compliant.

jdb.exe (Java Debugger)
Helps debug your Java programs.

jdmpview.exe (Cross-platform dump formatter)
Analyzes dumps. For more information, see "Using system dumps and the dump viewer" in the diagnostic guide.

keytool.exe (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.exe (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.

packager.exe (JavaBean to ActiveX packager)
Packages a JavaBean in a `jar` file for use as an ActiveX control.

policytool.exe (Policy File Creation and Management Tool)
Creates and modifies the external policy configuration files that define your installation's Java security policy.

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

rmiregistry.exe (Java remote object registry)
Creates and starts a remote object registry on the specified port of the current host.

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

serialver.exe (Serial Version Command)
Returns the serialVersionUID for one or more classes in a format that is suitable for copying into an evolving class.

tnameserv.exe (Common Object Request Broker Architecture (CORBA) transient naming service)
Starts the CORBA transient naming service.

wsgen.exe
Generates JAX-WS portable artifacts used in JAX-WS Web services.

wsimport.exe
Generates JAX-WS portable artifacts from a Web Services Description Language (WSDL) file.

xjc.exe
Compiles XML Schema files.

Include Files
C headers for JNI programs.

copyright
The copyright notice for the SDK for Windows software.

License

The License file, `C:\Program Files\IBM\Java60\docs\content\<`*locale*`>\license_<`*locale*`>.txt`, contains the license agreement for the SDK for Windows software (where <*locale*> is the name of your locale, for example en). To view or print the license agreement, open the file in a Web browser.

Chapter 3. Installing and configuring the SDK and Runtime Environment

The SDK is installed as part of an IBM product. Configure the SDK using environment variables, command-line options, and properties files.

Installing and configuring the SDK and Runtime Environment

The IBM SDK, Java Technology Edition, Version 6, for all supported 32-bit Windows architectures is available only as part of an IBM product.

For more information about obtaining, installing and performing initial configuration of the IBM SDK, Java Technology Edition, Version 6, for all supported 32-bit Windows architectures, please contact your IBM support representative.

Enabling the IBM Accessibility Bridge

The IBM Accessibility Bridge is installed but disabled by default. To enable the IBM Accessibility Bridge, uncomment the **assistive_technologies** entry in the `Accessibility.properties` file.

About this task

The `Accessibility.properties` file is in the `jre/lib` directory. Delete the # from the beginning of the following line:

```
#assistive_technologies=JawBridge
```

This Web site tells you more about the Accessibility Utilities:

http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140174.html

Disabling Java Accessibility support

You can disable Java Accessibility support to improve the JVM loading performance of Java applications that do not provide Java assistive technology support, especially over network links. To disable Java Accessibility support, set the **JAVA_ASSISTIVE** environment variable to OFF.

About this task

An assistive technology, such as JawBridge, is not available if this environment variable is set to OFF, even if the technology is enabled in the `Accessibility.properties` file.

Information for European language users

In Windows, a process has two code pages: the ANSI (or Windows) code page and the OEM (or DOS) code page. The **javaw** command always uses the ANSI code page unless the **console.encoding** system property is set.

The command window typically uses the OEM code page. Java console output uses the code page of the command window from which Java is started. However, the **javaw** command always uses the ANSI code page. You specify the code page to use for console output with the **-Dconsole.encoding** option on the **java** or **javaw** launcher. For example, **-Dconsole.encoding=Cp1252** causes all console output to be in the Windows ANSI Latin1 code page (1252).

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 Windows to find programs and utilities, such as **javac**, **java**, and **javadoc** tool, from any current directory. To display the current value of your **PATH**, type the following command at a command prompt:

```
echo
%PATH%
```

To add the Java launchers to your path:

1. If the SDK or Runtime Environment was installed in `C:\Program Files\IBM\Java60\` add the following directories to the **PATH** environment variable:

   `C:\Program Files\IBM\Java60\bin` (SDK only)

   `C:\Program Files\IBM\Java60\jre\bin` (SDK and Runtime Environment)
2. Close and reopen any command prompt windows 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 the **javadoc** tool, where to find the Java class libraries.

About this task

You should 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 command 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 command prompt.

Updating your SDK or JRE for Daylight Saving Time changes

You can apply recent changes to Daylight Saving Time by using the IBM Time Zone Update Utility for Java (JTZU).

About this task

Many countries around the world use a Daylight Saving Time (DST) convention. Typically, clocks move forward by 1 hour during the summer months to create more daylight hours during the afternoon and less during the morning. This practice has many implications, including the need to adjust system clocks in computer systems. Occasionally, countries change their DST start and end dates. These changes can affect the date and time functions in applications because the original start and end dates are programmed into the operating system and in Java software. To avoid this problem, you must update operating systems and Java installations with the new DST information.

The Olson time zone database is an external resource that compiles information about the time zones around the world. This database establishes standard names for time zones, such as "America/New_York", and provides regular updates to time zone information that can be used as reference data. To ensure that IBM developer kits and Runtime Environments contain up to date DST information, IBM incorporates the latest Olson time zone level into every updated release. To find out which Olson time zone level is included for a particular SDK or Runtime level, see https://www.ibm.com/developerworks/java/jdk/dst/olson_table.html.

If a DST change has been introduced since the last IBM update of the SDK or Runtime Environment, you can use JTZU to directly update your Java installation. You can also use this tool to update your installation if you are unable to move straight to the latest SDK or Runtime level. JTZU is available from IBM developerWorks® at the following link: https://www.ibm.com/developerworks/java/jdk/dst/jtzu.html.

Results

After updating your Java installation with any recent DST changes, your application can handle time and date calculations correctly.

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 start a Java application by starting a Java Runtime Environment and loading a specified class.

The **javaw** command is identical to **java**, except that **javaw** has no associated console window. Use **javaw** when you do not want a command prompt window to be displayed. The **javaw** launcher displays a window with error information if it fails.

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 start. 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 by using the **-version** or **-fullversion** options. You can also obtain version information for all `jar` files on the class path by using the **-Xjarversion** option.

Procedure

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

   ```
   java -version
   ```

   You will see information similar to:

```
java version "1.6.0"
Java(TM) SE Runtime Environment (build pwi3260sr10-20111027_02(SR10))
IBM J9 VM (build 2.4, JRE 1.6.0 IBM J9 2.4 Windows 7 x86-32 jvmwi3260sr10-20111026_93491 (JIT enabled,
AOT enabled)
J9VM - 20111026_093491
JIT  - r9_20111021_21134
GC   - 20110519_AA)
JCL  - 20111025_01
```

   The output provides the following information:

   - The first line indicates the Java standard edition class library level.
   - The second line includes information about the build level of the runtime environment. Service refresh (SR), fix pack (FP), and APAR numbers are appended to the build string. In the example, the installed level is service refresh 10.
   - The third line indicates the build level of the IBM J9 virtual machine.
   - Subsequent lines provide detailed information about the levels of IBM components that make up the runtime environment.

   Exact build dates and versions will change.
3. To obtain only the build information for the runtime environment, type the following command:

   ```
   java -fullversion
   ```

   You will see information similar to:

   ```
   java full version "JRE 1.6.0 IBM Windows 32 build pwi3260sr11-20120412_01 (SR11)"
   ```

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:

```
...
C:\Program Files\IBM\Java60\jre\lib\ext\ibmpkcs11impl.jar  VERSION: 1.0 build_20070125
C:\Program Files\IBM\Java60\jre\lib\ext\dtfjview.jar
C:\Program Files\IBM\Java60\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.

| To query the Java version information programmatically, see "Determining which JVM version your application is running on" on page 34.

Specifying Java options and system properties

You can specify Java options and system properties directly on the command line. You can also use an options file or an environment variable.

About this task

The sequence of the Java options on the command line defines which options take precedence during startup. Rightmost options have precedence over leftmost options. In the following example, the **-Xjit** option takes precedence:

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

Use one of more of the options that are shown in the procedure to customize your runtime environment.

Procedure

1. Specify options or system properties on the command line. For example:
   ```
   java -Dmysysprop1=tcpip -Dmysysprop2=wait -Xdisablejavadump MyJavaClass
   ```
2. Create an environment variable that is called **IBM_JAVA_OPTIONS** containing the options. For example:
   ```
   set IBM_JAVA_OPTIONS="-Dmysysprop1=tcpip -Dmysysprop2=wait
   -Xdisablejavadump"
   ```
3. Create a file that contains the options, and specify that file on the command line or in the **IBM_JAVA_OPTIONS** environment variable by using the **-Xoptionsfile** parameter. For more information about constructing this file, see "-Xoptionsfile" on page 89.

Standard options

The definitions for the standard options.

See "JVM command-line options" on page 83 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** (see "Garbage Collector command-line options" on page 101 for more information) to control the output. See Verbose garbage collection logging 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.

Executing a Java file automatically

To set a Java class or jar file to start automatically from the Windows explorer, use the **Tools** > **Folder Options** > **File Type** option of Windows Explorer.

About this task

Alternatively, at a command prompt type:

```
assoc .class=javaclass
ftype javaclass= install_dir\jre\bin\java.exe''%l''%*'
```

Note: The `%l` is the number 1 and not the letter l.

Running Java applications with native assistive technologies

Oracle provides the Java Access Bridge to give native Windows assistive technologies, such as screen readers, access to the Java Accessibility support in a Java application. These native Windows assistive technologies must support calls to the Java Access Bridge.

The Oracle Java Access Bridge includes an installer, which places five files in the correct directories: `access-bridge.jar`, `jaccess.jar`, `accessibility.properties`, `JavaAccessBridge.dll` and `WindowsAccessBridge.dll`. IBM provides a copy of `jaccess.jar` in the appropriate directory for use with JawBridge.

If you have already enabled JawBridge, which allows the Windows 2000 Magnifier to function with Swing applications, and you want to use it at the same time as the Oracle Java Access Bridge, edit the line in the `accessibility.properties` file to read:

```
assistive_technologies=com.sun.java.accessibility.AccessBridge,JawBridge
```

Comment out the line by inserting a leading # to deactivate both bridges.

For information about downloading the Oracle Java Access Bridge, see http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140174.html.

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 compiler delivers new optimizations as a result of compiler research, improves optimizations implemented in previous versions of the JIT, and provides better hardware exploitation.

The JIT is included in both the IBM SDK and Runtime Environment, which is enabled by default in user applications and SDK tools. Typically, you do not start 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 .

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.

Procedure

- Set the **JAVA_COMPILER** environment variable to `NONE` or the empty string before running the **java** application. At the command prompt where the application is run, type:

  ```
  set JAVA_COMPILER=NONE
  ```

  You can also permanently set **JAVA_COMPILER** by using the graphical user interface. Open **Control Panel**, select **System**, and on the **Advanced** tab, select **Environment Variables**.
- Use the **-D** option on the JVM command line to set the **java.compiler** property to `NONE` or the empty string. Type the following command at a command prompt:

  ```
  java -Djava.compiler=NONE <class>
  ```
- Use the **-Xint** option on the JVM command line. Type the following command at a command 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.

Procedure

- Set the **JAVA_COMPILER** environment variable to `jitc` before running the Java application. At the command prompt where the application is run, enter:

  ```
  set JAVA_COMPILER=jitc
  ```

  You can also permanently set **JAVA_COMPILER** by using the graphical user interface. Open **Control Panel**, select **System**, and on the **Advanced** tab, select **Environment Variables**. If the **JAVA_COMPILER** environment variable is an empty string, the JIT remains disabled. To disable the environment variable, at the prompt, enter:

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

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

Determining whether the JIT is enabled

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

Procedure

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

```
java -version
```

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

```
(JIT disabled)
```

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

```
(JIT enabled)
```

What to do next

For more information about the JIT, see The JIT compiler.

Specifying a garbage collection policy

The Garbage Collector manages the memory used by Java and by applications running in 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 remains 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 .

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 is as follows:

```
-Xgcpolicy:<value>
```

The following values are available:

gencon
The generational concurrent (**gencon**) policy uses a concurrent mark phase combined with generational garbage collection to help minimize the time that is spent in any garbage collection pause. This policy is particularly useful for applications with many short-lived objects, such as transactional applications. Pause times can be significantly shorter than with the **optthruput** policy, while still producing good throughput. Heap fragmentation is also reduced.

optavgpause
The "optimize for pause time" (**optavgpause**) policy uses concurrent mark and concurrent sweep phases. Pause times are shorter than with **optthruput**, but

application throughput is reduced because some garbage collection work is taking place while the application is running. Consider using this policy if you have a large heap size (available on 64-bit platforms), because this policy limits the effect of increasing heap size on the length of the garbage collection pause. However, if your application uses many short-lived objects, the **gencon** policy might produce better performance.

optthruput
The "optimize for throughput" (**optthruput**) policy (default) disables the concurrent mark phase. The application stops during global garbage collection, so long pauses can occur. This configuration is typically used for large-heap applications when high application throughput, rather than short garbage collection pauses, is the main performance goal. If your application cannot tolerate long garbage collection pauses, consider using another policy, such as **gencon**.

Increased heap sizes using a split heap

Many Java application workloads depend on the Java heap size. The IBM SDK for Java can use a split heap to work around restrictions in the 32-bit Windows memory space and provide a larger maximum heap size.

By default, the IBM SDK for Java uses a contiguous Java heap to store Java objects. Using the **-Xgc:splitheap** command-line option splits the heap into multiple contiguous memory areas. The use of two areas can increase the maximum allocatable heap size.

The **-Xgc:splitheap** option also forces the use of the “gencon” (generational concurrent) garbage collection policy.

Use **-Xgc:splitheap** for applications that must run on the 32-bit JVM (because of 32-bit JNI libraries, a 32-bit operating system, or 32-bit hardware) but need large Java heaps.

-Xgc:splitheap is not recommended if your application works in the any of the following ways:

- Performs poorly under the gencon garbage collection policy.
- Loads a very large number of classes.
- Uses large amounts of native system memory in JNI libraries; the increased size Java heap might reserve too much of the application's address space.

For more detailed information about a split Java heap, see Split heap.

Pause time

If an object cannot be created from the available space in the heap, the Garbage Collector attempts to tidy the heap. The intention is that subsequent allocation requests can be satisfied quickly.

The Garbage Collector tries to returning the heap to a state in which the immediate and subsequent space requests are successful. The Garbage Collector identifies unreferenced “garbage” objects, and deletes them. This work takes place in a garbage collection cycle. These cycles might introduce occasional, unexpected pauses in the execution of application code. As applications grow in size and complexity, and heaps become correspondingly larger, the garbage collection pause

time tends to grow in size and significance. Pause time can vary from a few milliseconds to many seconds. The actual time depends 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 (GC) 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 GC, 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 diagnostic procedures. 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 .

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.

Procedure

- 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=C:\Program Files\IBM\Java60\jre\lib\ext;
C:\Program Files\IBM\Java60\jre\lib\im <class>
```

What to do next

If the SDK or Runtime Environment was installed in a different directory, replace `C:\Program Files\IBM\Java60\` with the directory in which the SDK or Runtime Environment was installed.

Chapter 5. Developing Java applications

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

See “Contents of the SDK” on page 6 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 24. 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 28 section for more details about what is supported by XL XP-J 1.1.

XL TXE-J 1.0

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 and later, 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 25 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 `C:\Program Files\IBM\Java60\jre\lib\stax.properties`.
   - **For other factories.** The value of the service provider in the file `C:\Program Files\IBM\Java60\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.xlxp.api.stax.XMLEventFactoryImpl from the XL XP-J library is used.

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

`javax.xml.stream.XMLOutputFactory`
Selects the StAX serializer. By default, com.ibm.xml.xlxp.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

From Version 6, the XL TXE-J compiler replaces the XSLT4J interpreter as the default XSLT processor. If you are migrating applications from older versions of Java, 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 compilation and optimization reduce performance.

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.

Procedure

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 decreasing the 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/xltxe-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	http://www.ibm.com/xmlns/prod/xltxe-j/translet-name
destination-directory	http://www.ibm.com/xmlns/prod/xltxe-j/destination-directory
package-name	http://www.ibm.com/xmlns/prod/xltxe-j/package-name
jar-name	http://www.ibm.com/xmlns/prod/xltxe-j/jar-name
generate-translet	http://www.ibm.com/xmlns/prod/xltxe-j/generate-translet
auto-translet	http://www.ibm.com/xmlns/prod/xltxe-j/auto-translet
use-classpath	http://www.ibm.com/xmlns/prod/xltxe-j/use-classpath
debug	http://www.ibm.com/xmlns/prod/xltxe-j/debug
indent-number	http://www.ibm.com/xmlns/prod/xltxe-j/indent-number
enable-inlining	*Obsolete in new compiler*

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 run time, 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/xltxe-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/xltxe-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/xltxe-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.

- If your application uses each stylesheet just once or a very small number of times, or you are unable to make any of the other changes listed in this step, you might want to continue to use the XSLT4J interpreter by setting the javax.xml.transform.TransformerFactory service provider to org.apache.xalan.processor.TransformerFactoryImpl.

Securing Java API for XML processing (JAXP) against malformed input

If your application takes untrusted XML, XSD or XSL files as input, you can enforce specific limits during JAXP processing to protect your application from malformed data. If you specify limits, you must override the default XML parser configuration with a custom configuration.

About this task

To protect your application from malformed data, you can enforce specific limits during JAXP processing. These limits can be set in your `jaxp.properties` file, or by specifying various system properties on the command line. However, for these limits to take effect you must also override the default XML parser configuration with a custom configuration that allows these secure processing limits.

Procedure

1. Select the limits that you want to set for your application.
   - To limit the number of entity expansions in an XML document, see "-Djdk.xml.entityExpansionLimit" on page 77.
   - To limit the maximum size of a general entity, see "-Djdk.xml.maxGeneralEntitySizeLimit" on page 77.
   - To limit the maximum size of a parameter entity, see "-Djdk.xml.maxParameterEntitySizeLimit" on page 79.
   - To limit the length of XML names in XML documents, see "-Djdk.xml.maxXMLNameLimit" on page 79.
   - To limit the total size of all entities that include general and parameter entities, see "-Djdk.xml.totalEntitySizeLimit" on page 81.
   - To define the maximum number of content model nodes that can be created in a grammar, see "-Djdk.xml.maxOccur" on page 78.
   - To control whether external entities are resolved in an XML document, see "-Djdk.xml.resolveExternalEntities" on page 80.
2. To override the default XML parser configuration, set the custom configuration by specifying the following system property on the command line: **`-Dorg.apache.xerces.xni.parser.XMLParserConfiguration=`***config_file*, where *config_file* is **`org.apache.xerces.parsers.SecureProcessingConfiguration`**. For more information about the full override mechanism, see http://xerces.apache.org/xerces2-j/faq-xni.html#faq-2.

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 information: http://docs.oracle.com/javase/6/docs/api/javax/xml/stream/XMLInputFactory.html.

Property name	Supported?
javax.xml.stream.isValidating	No. The XL XP-J scanner does not support validation.
javax.xml.stream.isNamespaceAware	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.
javax.xml.stream.isCoalescing	Yes
javax.xml.stream.isReplacingEntityReferences	Yes. For XMLStreamReaders created from DOMSources, if entities have already been replaced in the DOM tree, setting this parameter has no effect.
javax.xml.stream.isSupportingExternalEntities	Yes
javax.xml.stream.supportDTD	True is always supported. Setting the value to false works only if the **com.ibm.xml.xlxp.support.dtd.compat.mode** system property is also set to false. When both properties are set to false, parsers created by the factory throw an XMLStreamException 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. Setting the value to false does not work before Service Refresh 2.
javax.xml.stream.reporter	Yes
javax.xml.stream.resolver	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://docs.oracle.com/javase/6/docs/api/javax/xml/stream/XMLStreamReader.html.

Property name	Supported?
javax.xml.stream.entities	Yes
javax.xml.stream.notations	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://docs.oracle.com/javase/6/docs/api/javax/xml/stream/XMLOutputFactory.html.

Property name	Supported?
javax.xml.stream.isRepairingNamespaces	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://docs.oracle.com/javase/6/docs/api/javax/xml/stream/XMLStreamWriter.html.

Property name	Supported?
javax.xml.stream.isRepairingNamespaces	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 3. 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)
http://javax.xml.transform.stream.StreamSource/feature feature	Yes	Yes	Yes
http://javax.xml.transform.stream.StreamResult/feature feature	Yes	Yes	Yes
http://javax.xml.transform.dom.DOMSource/feature feature	Yes	Yes	Yes
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)

Table 3. 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)
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)
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)	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.XSLTCDTMManager 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 `C:\Program Files\IBM\Java60\jre\lib`.
- Uncomment the entries in the `jaxp.properties` file in `C:\Program Files\IBM\Java60\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.

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

or

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

or

```
set IBM_JAVA_OPTIONS=-Djavax.xml.transform.TransformerFactory=
  org.apache.xalan.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 the operating system.

More information about problem diagnosis using Java can be found in the Troubleshooting and support.

Java Debugger (JDB)

The Java Debugger (JDB) is included in the SDK. The debugger is started with the **jdb** command; it attaches to the JVM using JPDA.

To debug a Java application:

1. Start the JVM with the following options:
   ```
   java -agentlib:jdwp=transport=dt_shmem,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 workstations. JPDA uses a TCP/IP socket to connect to the remote JVM.

1. Start the JVM with the following options:
   ```
   java -agentlib:jdwp=transport=dt_shmem,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 -connect com.sun.jdi.SocketAttach:hostname=<host>,port=<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://www.oracle.com/technetwork/java/javase/tech/jpda-141715.html
- http://docs.oracle.com/javase/6/docs/technotes/guides/jpda/
- http://docs.oracle.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.

Procedure

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 32-bit or 64-bit JVM

Some Java applications must be able to determine whether they are running on a 32-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 32- and 64-bit forms for platforms that support both 32- and 64-bit modes of operation. In this case, your application must load the correct library at run environmenttime, because it is not possible to mix 32- 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 32-bit mode
- 64 - the JVM is running in 64-bit mode

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

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

Determining which JVM version your application is running on

You can programmatically determine which JVM version your application is running on by querying the **java.runtime.version** system property.

About this task

The system property **java.runtime.version** can be queried at run time to determine the version of the JVM that is running.

Procedure

Use the following call: `System.getProperty("java.runtime.version");`

Results

This call returns a Java String with content similar to:

```
pwi3260sr11-20120412_01 (SR11)
```

The result of this query is similar to the command-line option **java -fullversion**. For more information about finding the Java version using command-line options, see "Obtaining version information" on page 13.

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.

Where a signal is generated externally (for example, when you enter **CTRL-BREAK**), a new thread is created for the signal handler. In this case, the JVM signal handler performs its processing and 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.

For exception and error signals, the JVM either:

- Handles the condition and recovers, or
- Enters a controlled shut down 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 shut down hook
  4. Performs the necessary JVM cleanup

For interrupt signals, the JVM also enters a controlled shut down 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 shut down hooks
3. Calls any application-installed exit hook
4. Performs the necessary JVM cleanup

The shut down is identical to the shut down 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 stop. The only control signal of interest is SIGBREAK, which causes a Javadump to be generated.

Signals used by the JVM

The types of signals are Interrupts, and Controls.

Table 4 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 an unrecoverable 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 shut down.

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

Table 4. Signals used by the JVM

Signal Name	Signal type	Description	Disabled by -Xrs	Disabled by -Xrs:sync
SIGINT (2)	Interrupt	Interactive attention (CTRL-C). JVM exits normally.	Yes	No
SIGTERM (15)	Interrupt	Termination request. JVM will exit normally.	Yes	No
SIGBREAK	Control	A break signal from a terminal. By default, this triggers a Javadump.	Yes	No

Note: A number supplied after the signal name is the standard numeric value for that signal.

The IBM JVM uses structured exception handling and the SetConsoleCtrlHandler() API. These are both disabled with **-Xrs**, however, only structured exception handling is disabled by **-Xrs:sync**. **-Xnosigchain** is ignored on Windows.

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

The signals 2 (SIGINT) and 15 (SIGTERM) on JVM threads causes the JVM to shut down; therefore, an application signal handler should not attempt to recover from this signal 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 jsig.dll before msvcrt.dll. The jsig.dll library ensures that calls to signal() 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 called.

The libjsig.dll 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, *install_dir*.

To use jsig.dll, link it with the application that creates or embeds a JVM.

Writing JNI applications

Valid Java Native Interface (JNI) version numbers that 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 JNI is not supported.

This version number determines only the level of the JNI to use. The actual level of the JVM that is created is specified by the JSE libraries (use the `java -version` command to show the JVM level). The JNI level *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://docs.oracle.com/javase/6/docs/technotes/guides/jni/.

If your application needs two JNI libraries, one built for 32-bit and the other for 64-bit, use the **com.ibm.vm.bitmode** system property to determine if you are running with a 32-bit or 64-bit JVM and choose the appropriate library.

Supported compilers

These compilers have been tested with the IBM SDK.

The Microsoft Visual Studio v8 compiler is supported for Windows 32-bit on Intel.

JNI runtime linking

The Java Native Interface (JNI) enables runtime linking to dynamic native libraries.

For Version 6, runtime linking is supported from service refresh 9. If runtime linking causes a symbol conflict, the application must resolve the conflict by renaming the symbol on the application side, or by turning off runtime linking.

Dynamic linking

When you build a C or C++ program that uses the JNI Invocation API to create a Java virtual machine, and calls Java code, use the linker option `/link /LIBPATH` to do the following task:

- Add *lib_dir* and *lib_dir*\j9vm to the list of directories that are searched for shared objects. These directories contain the Java SDK shared libraries. You also want to link with jvm.dll (by using the **-ljvm** option). Add *lib_dir* and *lib_dir*\j9vm to the list of directories that are searched for shared objects.

On Windows, no special options are required for the command-line compiler cl.exe. Generally, a Microsoft compiler is used, for example VC++. For more information about compiler options, see the documentation for the compiler that is being used. By default, VC++ picks libraries that are present in the environment variable **%LIB%**. The variable must always point at the \lib subdirectory of the VC++ SDK as one of the search paths for linking libraries.

Here is a typical command block to build the invocation API test:

```
cl.exe /I INSTALL_DIR\include
 /FeinvAPITest
 invAPITest.c
 /link /LIBPATH:lib_dir\j9vm
 /LIBPATH:lib_dir
```

When you run a C or C++ program that uses the JNI Invocation API to run Java classes, ensure that the class path is set up correctly to enable the JVM to find your class files. If you modify the Java boot class path, include the SDK files that are necessary to run your applications.

To ensure that a JNI library exports the functions that a Java application must resolve at runtime, you can examine the library by using the dumpbin.exe tool (typically a part of the VC++ SDK installation). For example, a JNI library that is named jnitest.dll, and that contains JNI routines **fooImpl** and **barImpl**, must export the symbols:

```
C:\>dumpbin.exe /EXPORTS jnitest.dll
Dump of file jnitest.dll

File Type: DLL

      Section contains the following exports for JNITEST.dll

00000000 characteristics
5412A472 time date stamp Fri Sep 12 03:44:50 2014
    0.00 version
       1 ordinal base
       5 number of functions
       5 number of names

ordinal hint RVA      name
...
     1   27 0000CE10 Java_mypackage_SampleClass_fooImpl = Java_mypackage_SampleClass_fooImpl
     2   28 000085A0 Java_mypackage_SampleClass_barImpl = Java_mypackage_SampleClass_barImpl
...
```

For more information on dumpbin.exe and its options, see the MSDN documentation.

On Windows, JNI methods are typically stored in dynamic libraries called Dynamic Link Libraries (DLLs). DLLs contain functions, and sometimes data, that can be referenced from outside the library, for example from a dynamic library or an executable program. Native methods are stored in DLLs and are either linked at build time, through the linking process, or at runtime, by dynamically loading the

methods by using the Windows API **LoadLibrary()** and **LoadLibraryEx()** functions. For more information about the LoadLibrary() family of functions, refer to the MSDN documentation.

You can store native methods as follows:

Dynamic link libraries
On Windows, JNI methods are typically stored in dynamic libraries called Dynamic Link Libraries (DLLs). DLLs contain functions and data, which can be referenced from another load module, for example a dynamic library or an executable program. Native methods are stored in DLLs and are either linked at build time, through the linking process, or at runtime, by dynamically loading the methods by using the Windows API **LoadLibrary()** and **LoadLibraryEx()** functions. For more information about the LoadLibrary() family of functions, refer to the MSDN documentation.

Note: To ensure that dynamic linking of native libraries works successfully you can, optionally, implement the lifecycle functions JNI_Onload() and JNI_OnUnload() in the library. If you have implemented JNI_Onload(), the native library must export it otherwise it is not visible to the runtime, and the JVM assumes that the library requires only the JNI version JNI_VERSION_1.1. If JNI_OnUnload() has been implemented, it must also be exported. If JNI_Onload() is implemented and exported, then the latest JNI version is returned; for example, JNI_VERSION_1.8.

Configuring large page memory allocation

You can enable large page support, on systems that support it, by starting Java with the **-Xlp** option.

About this task

Large page usage is primarily intended to provide performance improvements to applications that allocate a great deal of memory and frequently access that memory. The large page performance improvements are a result of the reduced number of misses in the Translation Lookaside Buffer (TLB). The TLB maps a larger virtual storage area range and thus causes this improvement.

For the JVM to use large pages, your system must have an adequate number of contiguous large pages available. If large pages cannot be allocated, even when enough pages are available, possibly the large pages are not contiguous.

To use large pages, the user that runs Java must have the authority to lock pages in memory. To enable this authority, as Administrator go to **Control Panel** > **Administrative Tools** > **Local Security Policy** and then find **Local Policies** > **User Rights Assignment** > **Lock pages in memory**. Alternatively, run **secpol.msc**. Add the user who runs the Java process, and reboot your machine. For more information, see these websites:

- http://msdn.microsoft.com/en-us/library/aa366720(VS.85).aspx
- http://msdn.microsoft.com/en-us/library/aa366568(VS.85).aspx

Large page allocations only succeed if the local administrative policy for the JVM user has the **Lock pages in memory** setting enabled.

On Microsoft Windows Vista and later, and Windows 2008 and later, use of large pages is affected by the User Account Control (UAC) feature. When UAC is

enabled, a regular user (a member of the Users group) can use the **-Xlp** option as normal. However, an administrative user (a member of the Administrators group) must run the application as an administrator to gain the privileges required to lock pages in memory. To run as administrator, right-click the application and click **Run as administrator**. If the user does not have the necessary privileges, the following error message is produced: `System configuration does not support option '-Xlp'`.

For more information about the **-Xlp** options, see "JVM command-line options" on page 83.

To obtain the large page sizes available and the current setting, use the **-verbose:sizes** option. Note the current settings are the requested sizes and not the sizes obtained. For object heap size information, check the **-verbose:gc** output.

CORBA support

The Java Platform, Standard Edition (JSE) supports, at a minimum, the specifications that are defined in the compliance document from Oracle. 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://docs.oracle.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 Oracle Java Core API at this release. Programs that have 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*|*fine*|*finer*|*finest*
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 enable 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 5. 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
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 Oracle 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 start of your `.java` file from: `import com.ibm.math.*;` to `import java.math.*;`.

Support for the Java Attach API

Your application can connect to another "target" virtual machine using the Java Attach API. Your application can then load an agent application into the target virtual machine, for example to perform tasks such as monitoring status. Support for the Java Attach API was added in Java 6 SR 6.

Code for agent applications, such as JMX agents or JVMTI agents, is normally loaded during virtual machine startup by specifying special startup parameters. Requiring startup parameters might not be convenient for using agents on applications that are already running, such as WebSphere Application Servers. You can use the Java Attach API to load an agent at any time, by specifying the process ID of the target virtual machine. The Attach API capability is sometimes called the "late attach" capability.

Support for the Attach API is enabled by default for Java 6 SR 6 and later.

Security considerations

The Java Attach API creates files and directories in a common directory. On Windows, security of the common directory and its subdirectories and files is handled by Windows security mechanisms. This means that only the process owner can connect to their processes.

You must secure access to the Java Attach API capability to ensure that only authorized users or processes can connect to another virtual machine. If you do not intend to use the Java Attach API capability, disable this feature using a Java system property. Set the **com.ibm.tools.attach.enable** system property to the value **no**; for example:

```
-Dcom.ibm.tools.attach.enable=no
```

The Attach API can be enabled by setting the **com.ibm.tools.attach.enable** system property to the value **yes**; for example:

```
-Dcom.ibm.tools.attach.enable=yes
```

Using the Java Attach API

By default, the target virtual machine is identified by its process ID. To use a different target, change the system property **com.ibm.tools.attach.id**; for example:

```
-Dcom.ibm.tools.attach.id=<process_ID>
```

The target process also has a human-readable "display name". By default, the display name is the command line used to start Java. To change the default display name, use the **com.ibm.tools.attach.displayName** system property. The ID and display name cannot be changed after the application has started.

The Attach API creates working files in a common directory, which by default is called `.com_ibm_tools_attach` and is created in the system temporary directory. The system property **java.io.tmpdir** holds the value of the system temporary directory. On Windows systems, the system temporary directory is typically `C:\Documents and Settings\<userid>\Local Settings\Temp`.

You can specify a different common directory from the default, by using the following Java system property:

```
-Dcom.ibm.tools.attach.directory=directory_name
```

This system property causes the specified directory, *directory_name*, to be used as the common directory. If the directory does not already exist, it is created, however the parent directory must already exist. For example, the following system property creates a common directory called `myattachapidir` in the `C:` directory. The `C:` directory must already exist.

```
-Dcom.ibm.tools.attach.directory=C:\myattachapidir
```

The common directory must be located on a local drive; specifying a network mounted file system might result in incorrect behavior.

If your Java application ends abnormally, for example, following a crash or a SIGKILL signal, the process subdirectory is not deleted. The Java VM detects and removes obsolete subdirectories where possible. The subdirectory can also be deleted by the owning user ID.

On heavily loaded system, applications might experience timeouts when attempting to connect to target applications. The default timeout is 120 seconds. Use the **`com.ibm.tools.attach.timeout`** system property to specify a different timeout value in milliseconds. For example, to timeout after 60 seconds:

```
-Dcom.ibm.tools.attach.timeout=60000
```

A timeout value of zero indicates an indefinite wait.

For JMX applications, you can disable authentication by editing the `<JAVA_HOME>/jre/lib/management/management.properties` file. Set the following properties to disable authentication in JMX:

```
com.sun.management.jmxremote.authenticate=false
com.sun.management.jmxremote.ssl=false
```

Problems with the Attach API result in one of the following exceptions:

- `com.ibm.tools.attach.AgentLoadException`
- `com.ibm.tools.attach.AgentInitializationException`
- `com.ibm.tools.attach.AgentNotSupportedException`
- `java.io.IOException`

A useful reference for information about the Attach API can be found at http://docs.oracle.com/javase/6/docs/technotes/guides/attach/index.html. The IBM implementation of the Attach API is equivalent to the Oracle Corporation implementation. However, the IBM implementation cannot be used to attach to, or accept attach requests from, non-IBM virtual machines. To use the attach API to attach to target processes from your application, you must add the "tools.jar" library to the application classpath. This library is not required for the target processes to accept attach requests.

Chapter 6. Plug-in, Applet Viewer and Web Start

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.

Using the Java plug-in

The Java plug-in is a web browser plug-in. You use the Java plug-in to run applets in the browser.

Allow enough time for applets to finish loading, otherwise your browser might seem to "stop". For example, if you click **Back** and then click **Forward** while an applet is loading, the HTML pages might be unable to load.

The Java plug-in is documented at: http://docs.oracle.com/javase/6/docs/technotes/guides/jweb/applet/applet_dev_guide.html.

Supported browsers

The Java plug-in supports the following browsers: .

Table 6. Browsers supported by the Java plug-in on Windows

Browser	Supported versions
Internet Explorer	6.0 SP1, 7.0, 8.0, 9.0, 10.0
Firefox	2.0, 3.0, 3.5, 3.6, 4.0, 5.0, 6.0, and later releases

Note: Internet Explorer V10.0 is the default browser on Windows 8 classic edition.

Later minor releases of these browsers are also supported.

Internet Explorer 5.01, the default browser on Windows 2000, is not supported.

Installing the Java plug-in using the Java control panel

You can use the Java control panel to install the Java plug-in for Internet Explorer and Mozilla Firefox.

Before you begin

The IBM SDK for Java must be installed before using this method.

About this task

If the Java plug-in was not installed when the SDK was installed, follow these steps to install the plug-in using the Java control panel:

Procedure

1. Open the Windows Control Panel.
2. Double-click IBM Control Panel for Java.

3. Select the **Advanced** tab.
4. Select **Default Java for Browsers**.
5. To install the Java plug-in for specific browsers, select the corresponding check box. To remove the Java plug-in from specific browsers, clear the corresponding check box. On Windows Vista, Windows Server 2008 R2 and Windows 7, the check box to associate with Internet Explorer is disabled by default. You cannot change this selection. By default, the Next-Generation plug-in is enabled. If you want to switch between a Next-Generation and a First-Generation, or Classic plug-in, follow these steps:
   a. Open the **IBM Control Panel for Java**.
   b. Select **Advanced** > **Java Plug-in** and check or clear the option **Enable the next-generation Java Plug-in**. This action changes the association between old style and Next-Generation plug-ins.

What to do next

Restriction: On Windows Vista with Federal Desktop Core Configuration (FDCC), you must run the Java Control Panel in elevated privilege mode. Browse to the `<SDK>\jre\bin` directory in Explorer, right-click the `javacpl.exe` icon, and select **Run as administrator**.

Secure Static Versioning (SSV) support

Static versioning allows applets to request a specific JVM version to be run under. Secure Static Versioning is used on Internet Explorer because this capability allows applets to use old security vulnerabilities on systems that have been upgraded to a new JVM.

About this task

Secure Static Versioning (SSV) support is not applicable for Next Generation plug-ins.

SSV does not function if third-party browser extensions are disabled in Internet Explorer. To enable third-party browser extensions:

1. Open Internet Explorer.
2. Click **Tools** > **Internet Options**.
3. Click the **Advanced** tab.
4. Select the **Enable third-party browser extensions** check box.

If third-party browser extensions are disabled after SSV has been used, SSV will continue to function.

Common Document Object Model (DOM) support

Because of limitations in particular browsers, you might not be able to implement all the functions of the org.w3c.dom.html package.

One of the following errors is thrown:

- sun.plugin.dom.exception.InvalidStateException
- sun.plugin.dom.exception.NotSupportedException

Using DBCS parameters

The Java plug-in supports double-byte characters (for example, Chinese Traditional BIG-5, Korean, and Japanese) as parameters for the tags <APPLET>, <OBJECT>, and <EMBED>. You must select the correct character encoding for your HTML document so that the Java plug-in can parse the parameter.

About this task

Specify character encoding for your HTML document by using the `<META>` tag in the `<HEAD>` section like this:

```
<meta http-equiv="Content-Type" content="text/html; charset=big5">
```

This example tells the browser to use the Chinese BIG-5 character encoding to parse the HTML file.

Working with applets

With the Applet Viewer, you can run one or more applets that are called by reference in a Web page (HTML file) by using the <APPLET> tag. The Applet Viewer finds the <APPLET> tags in the HTML file and runs the applets, in separate windows, as specified by the tags.

Because the Applet Viewer is for viewing applets, it cannot display a whole Web page that contains many HTML tags. It parses only the <APPLET> tags and no other HTML on the Web page.

Running and debugging applets with the Applet Viewer

Use the following commands to run and debug an applet with the Applet Viewer.

Procedure

- To run an applet with the Applet Viewer, enter the following command: `appletviewer <name>.`

  <*name*> is one of the following options:

  – The file name of an HTML file that calls an applet.

  – The URL of a Web page that calls an applet.

  For example, to start the Applet Viewer on an HTML file that calls an applet, type at a command prompt:

  ```
  appletviewer <demo>\GraphLayout\example1.html
  ```

  Where <*demo*> is replaced by the full path into which you unzipped the demo package.

  To start the Applet Viewer on a Web page, type at a command prompt:

  ```
  appletviewer http://mywebpage.com/demo/applets/MyApplet/example1.html
  ```

  The Applet Viewer does not recognize the **charset** option of the `<META>` tag. If the file that the Applet Viewer loads is not encoded as the system default, an I/O exception might occur. To avoid the exception, use the **-encoding** option when you run **appletviewer**. For example:

  ```
  appletviewer -encoding JISAutoDetect sample.html
  ```

- To debug an applet with the Applet Viewer, use the `debug` parameter with the `appletviewer` command.

  For example:

```
>
cd demo\applets\TicTacToe
..\..\..\bin\appletviewer -debug example1.html
```

You can find documentation about how to debug applets using the Applet Viewer at the Oracle Web site: http://docs.oracle.com/javase/6/docs/technotes/guides/plugin/developer_guide/debugger.html

Unique CLSIDs

A unique set of CLSIDs have been added to the IBM JVM from Version 6.

The new CLSIDs are as follows:

```
1ACECAFE-0016-0000-0000-ABCDEFFEDCBA
1ACECAFE-0016-0000-0000-ABCDEFFEDCBB
1ACECAFE-0016-0000-0000-ABCDEFFEDCBC
```

You can refer to these CLSIDs in the OBJECT Tag for your applets.

In addition, the following existing CLSIDs are also supported for compatibility purposes:

```
CAFEEFAC-0016-0000-0000-ABCDEFFEDCBA
CAFEEFAC-0016-0000-0000-ABCDEFFEDCBB
CAFEEFAC-0016-0000-0000-ABCDEFFEDCBC
```

Using Web Start

Java Web Start is used for Java application deployment.

With Web Start, you can start and manage applications directly from the Web. Applications are cached to minimize installation times. Applications are automatically upgraded when new versions become available.

Web Start supports these command-line arguments documented at http://docs.oracle.com/javase/6/docs/technotes/guides/javaws/developersguide/syntax.html#resources:

- -verbose
- -version
- -showversion
- -help
- -X
- -ea
- -enableassertions
- -da
- -disableassertions
- -esa
- -enablesystemassertions
- -dsa
- -disablesystemassertions
- -Xint
- -Xnoclassgc
- -Xdebug
- -Xfuture

- -Xrs
- -Xms
- -Xmx
- -Xss

Web Start also supports **-Xgcpolicy** to set the garbage collection policy.

From service refresh 10, the Autodownload option in the Java Control Panel is set to **Always** by default. This option enables a user without administration privileges to download the JRE from the location specified in the JNLP file.

For more information about Web Start, see:

- http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136112.html and
- http://docs.oracle.com/javase/6/docs/technotes/guides/javaws/index.html.

For more information about deploying applications, see:

- http://docs.oracle.com/javase/6/docs/technotes/guides/deployment/index.html.

Running Web Start

Web Start can be run from a Web page or the command line. Web Start applications are stored in the Java Application Cache.

About this task

You can start Web Start in a number of different ways.

Procedure

- Select a link on a Web page that refers to a `.jnlp` file. If your browser does not have the correct association to run Web Start applications, select the `C:\Program Files\IBM\Java60\jre\bin\javaws.exe` command from the **Open/Save** window to start the Web Start application.
- At a command prompt, type:

  ```
  javaws <URL>
  ```

  Where <*URL*> is the location of a `.jnlp` file.
- If you have used Java Web Start to open the application in the past, use the Java Application Cache Viewer. At a command prompt, type:

  ```
  C:\Program Files\IBM\Java60\jre\bin\javaws -viewer
  ```

  All Java Web Start applications are stored in the Java Application Cache. An application is downloaded only if the latest version is not in the cache.

WebStart Secure Static Versioning

Static versioning allows Web Start applications to request a specific JVM version on which those applications will run. Because static versioning also allows applications to exploit old security vulnerabilities on systems that have been upgraded to a new JVM, Secure Static Versioning (SSV) is now used by default.

With SSV, the user is warned before running any unsigned Web Start application that requests a specific JVM, if the requested JVM is installed. Signed applications and applications that request the latest version of the JVM run as usual.

You can disable SSV by setting the **deployment.javaws.ssv.enabled** property in the `deployment.properties` file to false.

SSV is not supported for Next-Generation plug-ins. SSV is applicable only for First-generation plug-ins.

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

To run your application, a user needs the Runtime Environment for Windows. The SDK for Windows software contains a Runtime Environment. However, you cannot assume that your users have the SDK for Windows software installed.

Your SDK for Windows software license does **not** allow you to redistribute any of the SDK files with your application. You must ensure that a licensed version of the SDK for Windows is installed on the target workstation.

Chapter 7. Class data sharing between JVMs

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

You can share class data between Java Virtual Machines (JVMs) by storing it in a memory-mapped cache file on disk. Sharing reduces the overall virtual storage 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 running JVM and persists until it is deleted.

A shared cache can contain:

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

The format of classes stored in the shared classes cache is changed in this release at service refresh 13. As a result, there is a different shared cache generation number, which causes the JVM to create a new shared classes cache, rather than re-creating or reusing an existing cache. To save space, all existing shared caches should be removed unless they are in use by an earlier release of IBM SDK, Java Technology Edition, Version 6. For more information about deleting a shared classes cache, see "Class data sharing command-line options" on page 55.

Overview of class data sharing

Class data sharing provides a method of reducing memory footprint and improving JVM start 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 class loaders share classes automatically if they extend the application class loader. Otherwise, they must use the Java Helper API provided with the JVM to access the cache. See "Adapting custom class loaders to share classes" on page 61.

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. The amount of AOT code stored in the cache is determined heuristically. You cannot control which methods get stored in the cache. You can set maximum and minimum limits on the amount of cache space used for AOT code, or you can disable AOT caching completely. See "Class data sharing command-line options" on page 55 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 60 for more information.

Dynamic updating of the cache

The shared class cache persists beyond the lifetime of any JVM. Therefore, 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 independent of the application using it.

Cache security

Access to the shared class cache is limited by operating system permissions and Java security permissions.

Only a class loader that has registered to share class data can update the shared class cache.

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 class loaders, must be granted permission to share classes. Grant permission by adding SharedClassPermission lines to the `java.policy` file. See "Using SharedClassPermission" on page 61 for more information. The RuntimePermission createClassLoader restricts the creation of new class loaders 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 deleted using a suboption to the **-Xshareclasses** command or the cache file is deleted manually.

Cache utilities

All cache utilities are suboptions to the **-Xshareclasses** command. See "Class data sharing command-line options" on page 55 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 technology 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 that 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 60.)

-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 that are 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.

cacheDir=<*directory*>
Sets the directory in which cache data is read and written. By default, <*directory*> is the user's `C:\Documents and Settings\<username>\Local Settings\Application Data\javasharedresources` directory. The user must have sufficient permissions in <*directory*>. The JVM writes persistent cache files directly into the directory specified. Persistent cache files can be safely moved and deleted from the file system. Non-persistent 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 Java virtual machine does not create a new cache with this suboption. Opening a cache read-only prevents the VM from making any updates to the cache. If you specify this suboption, the VM can connect to caches that were created by other users or groups without requiring write access.

By default, this suboption is not specified.

persistent (default)
Uses a persistent cache. The cache is created on disk, which persists beyond operating system restarts. Non-persistent and persistent caches can have the same name.

nonpersistent
Uses a non-persistent cache. The cache is deleted when the operating system shuts down. Non-persistent and persistent caches can have the same name. You must always use the **nonpersistent** suboption when running utilities such as **destroy** on a non-persistent cache.

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 by using the **noaot** suboption.

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

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

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 run time. 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 that is chosen by the user; for example, "myModification1". This option partitions the cache, so that only JVMs that use 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. For more information, see "Runtime bytecode modification" on page 60.

reset
Causes a cache to be destroyed and then re-created 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.

Note: On z/OS, when the **destroyAll** option is invoked from a 31-bit JVM, 64-bit caches are not destroyed. Similarly, when the **destroyAll** option is invoked from a 64-bit JVM, 31-bit caches are not destroyed. The following message is displayed:

```
JVMSHRC735I: Use a nn-bit JVM to perform the requested operation on the
nn-bit shared cache \"cachename\" as the nn-bit JVM
cannot verify that the shared memory was created by the JVM.
```

expire=<*time in minutes*>
Destroys all caches that have been unused for the time that is specified before loading shared classes. This option is not a utility option because it does not cause the JVM to exit. On NTFS file systems, the expire option is accurate to the nearest hour.

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 that is specified by the **name**,

cacheDir, and **nonpersistent** suboptions. The most useful information that is displayed is how full the cache is and how many classes it contains. Stale classes are classes that are updated on the file system and which the cache has therefore marked as "stale". Stale classes are not purged from the cache and can be reused. For more information, see printStats utility.

printAllStats (Utility option)
Displays detailed information for the cache that is 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.

For more information, see printAllStats utility.

mprotect=[all | default | none]
By default, the memory pages that contain the cache are always protected, 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 performance cost. Specifying `all` ensures that all the cache pages are protected, including the header. Specifying `none` disables the page.

Note: Specifying `all` has a negative impact on performance. You should specify `all` only for problem diagnosis, and not for production.

noBootclasspath
Prevents storage of classes that are loaded by the bootstrap class loader in the shared classes cache. Can be used with the SharedClassURLFilter API to control exactly which classes get cached. For more information about shared class filtering, see Using the SharedClassHelper API.

cacheRetransformed
Enables caching of classes that are transformed by 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 Java virtual machine (VM) 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 that are being loaded until the cache becomes full. If two or more VMs are started concurrently, they populate the cache concurrently.

To check that the cache is 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 VM ends or in another command window.

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

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

Caches can be deleted if they contain many stale classes or if the cache is full and you want to create a bigger cache. To delete a cache, run `java -Xshareclasses:[name=<name>],destroy`. If you want to delete a 64-bit non-compressed references cache, run `java -Xshareclasses:[name=<name>],destroy -Xnocompressedrefs`.

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, by using **-Xscmx**. Then, run the application and use the **printStats** option to determine how much class data is 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 VM, it is best to do this analysis after the application ends. However, a full cache does not have a negative affect on the performance or capability of any VMs connected to it. Therefore, you can choose a cache size that is smaller than required.

If a cache becomes full, a message is displayed on the command line of any VMs that are using the **verbose** suboption. All VMs sharing the full cache can then load 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 real storage consumption. It is also useful on systems that frequently start and shut down JVMs, which benefit from the improvement in startup time.

The processor and memory usage required 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 storage 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 the cache you can specify is limited by the amount of available disk space and available virtual address space.

The cache is limited by the following factors:

- Available disk space. A file is created to store the class data in a directory called `javasharedresources`. This directory is created in the user's profile directory, which is typically `C:\Documents and Settings\<username>\Local Settings\Application Data\javasharedresources\`. The shared file is deleted every time Windows is restarted.
- Available virtual address space. Because the virtual address space of a process is shared between the shared classes cache and the Java heap, increasing the maximum size of the Java heap reduces the size of the shared classes cache you can create.

JVMTI RetransformClasses() is unsupported

You cannot run RetransformClasses() on classes loaded from the shared class cache.

The JVM might throw the exception UnmodifiableClassException if you attempt to run RetransformClasses(). It does not work because class file bytes are not available for classes loaded from the shared class cache. If you must use RetransformClasses(), ensure that the classes to be transformed are not loaded from the shared class cache, or disable the shared class cache feature.

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 affect on performance. Using a modification context with a JVMTI agent avoids the need for extra checks and therefore has no affect 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 Dealing with runtime bytecode modification for more detail on this topic.

Operating system limitations

You cannot share classes between 32-bit and 64-bit Java virtual machines (VM). Temporary disk space must be available to hold cache information. The operating system enforces cache permissions.

For operating systems that can run both 32-bit and 64-bit applications, class data sharing is not allowed between 32-bit and 64-bit VMs. The **listAllCaches** suboption lists 32-bit and 64-bit caches, depending on the address mode and compressed references mode of the VM being used.

The shared class cache requires disk space to store identification information about the caches that exist on the system. This information is in the user profile directory. If the identification information directory is deleted, the VM cannot identify the shared classes on the system and must re-create the 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.

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 class loaders to share classes

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

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

The API documentation for this package is available here: API documentation

See Using the Java Helper API 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 usual method of access or on the Web at: http://www.ibm.com/partnerworld/.

If you have purchased a service contract (that is, the IBM 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

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

IBM strives to provide products with usable access for everyone, regardless of age or ability.

For example, you can operate the IBM SDK, Java Technology Edition, Version 6 without a mouse, by using only the keyboard.

Keyboard navigation

This product uses standard Microsoft Windows navigation keys.

For users who require keyboard navigation, a description of useful keystrokes for Swing applications can be found here: Swing Key Bindings.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility.

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.

Web Start accessibility

From Version 5.0, Java Web Start contains several accessibility and usability improvements, including better support for screen readers and improved keyboard navigation.

You can use the command line to start a Java application that is enabled for Web Start. To change preference options, you must edit a configuration file, `Application Data\IBM\Java\Deployment\deployment.properties` in the user's home directory. Take a backup before you edit this file. Not all of the preferences that can be set in the Java Application Cache Viewer are available in the configuration file.

Chapter 10. General note about security

You can obtain JCE unrestricted jurisdiction policy files from the ibm.com® web site. Documentation about the IBM security packages JCE, JCEFIPS, JSSE2, JSSEFIPS, JGSS, JAAS, and hardware cryptography is available in the Security documentation.

Appendix. Appendixes

Reference information.

Command-line options

You can specify the options on the command line while you are starting Java. They override any relevant environment variables. For example, using **-cp <dir1>** with the Java command completely overrides setting the environment variable **CLASSPATH=<dir2>**.

This chapter provides the following information:

Specifying command-line options

Although the command line is the traditional way to specify command-line options, you can also pass options to the Java virtual machine (VM) by using options files and environment variables.

The sequence of the Java options on the command line defines which options take precedence during startup. Rightmost options have precedence over leftmost options. In the following example, the **-Xjit** option takes precedence:

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

Use single or double quotation marks for command-line options only when explicitly directed to do so. Single and double quotation marks have different meanings on different platforms, operating systems, and shells. Do not use **'-X<option>'** or **"-X<option>"**. Instead, you must use **-X<option>**. For example, do not use `'-Xmx500m'` and `"-Xmx500m"`. Write this option as `-Xmx500m`.

At startup, the list of VM arguments is constructed in the following order, with the lowest precedence first:

1. Environment variables that are described in ../com.ibm.java.doc.diagnostics.60/diag/appendixes/env_var/env_jvm.dita are translated into command-line options. For example, the following environment variable adds the parameter **-Xrs** to the list of arguments:

   ```
   set IBM_NOSIGHANDLER=<non_null_string>
   ```

2. The **IBM_JAVA_OPTIONS** environment variable. You can set command-line options using this environment variable. The options that you specify with this environment variable are added to the command line when a JVM starts in that environment.

   The environment variable can contain multiple blank-delimited argument strings, but must not contain comments. For example:

   ```
   set IBM_JAVA_OPTIONS="-Dmysysprop1=tcpip -Dmysysprop2=wait -Xdisablejavadump"
   ```

Note: The environment variable **JAVA_TOOLS_OPTIONS** is equivalent to **IBM_JAVA_OPTIONS** and is available for compatibility with JVMTI.

3. Certain options are created automatically by the JVM. These specify arguments such as search paths and version information.
4. Options that are specified on the command line. For example:

   ```
   java -Dmysysprop1=tcpip -Dmysysprop2=wait -Xdisablejavadump MyJavaClass
   ```

   The Java launcher adds some automatically generated arguments to this list, such as the names of the main class.

You can also use the **-Xoptionsfile** parameter to specify JVM options. This parameter can be used on the command line, or as part of the **IBM_JAVA_OPTIONS** environment variable. The contents of an option file are expanded in place during startup. For more information about the structure and contents of this type of file, see "-Xoptionsfile" on page 89.

To troubleshoot startup problems, you can check which options are used by a JVM. Append the following command-line option, and inspect the Javadump file that is generated:

```
-Xdump:java:events=vmstart
```

Here is an extract from a Javadump file that shows the options that are used:

```
....
    2CIUSERARG              -Xdump:java:file=/home/test_javacore.txt,events=vmstop
    2CIUSERARG              -Dtest.cmdlineOption=1
    2CIUSERARG              -XXallowvmshutdown:true
    2CIUSERARG              -Xoptionsfile=test1.test_options_file
....
```

General command-line options

Use these options to print help on assert-related options, set the search path for application classes and resources, print a usage method, identify memory leaks inside the JVM, print the product version and continue, enable verbose output, and print the product version.

-cp, -classpath <directories and compressed or .jar files separated by : (; on Windows)>
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 classpath is, by default, the current directory (`.`).

-help, -?
Prints a usage message.

-fullversion
Prints the build and version information for the JVM.

-showversion
Prints product version and continues.

-verbose:<*option*>[,<*option*>...]
Enables verbose output. Separate multiple options using commas. These options are available:

class
Writes an entry to `stderr` for each class that is loaded.

dynload
Provides detailed information as each bootstrap class is loaded by the JVM:

- The class name and package

- For class files that were in a .jar file, the name and directory path of the .jar
- Details of the size of the class and the time taken to load the class

The data is written out to `stderr`. An example of the output on a Windows platform follows:

```
<Loaded java/lang/String from C:\sdk\jre\lib\vm.jar>
<Class size 17258; ROM size 21080; debug size 0>
<Read time 27368 usec; Load time 782 usec; Translate time 927 usec>
```

gc Provide verbose garbage collection information.

init
Writes information to `stderr` describing JVM initialization and termination.

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 the full build and version information for the JVM.

System property command-line options

Use the system property command-line options to set up your system.

-D<*name*>=<*value*>
Sets a system property.

-Dcom.ibm.CORBA.CommTrace

This system property turns on wire tracing for the Object Request Broker (ORB), which is also known as *Comm* tracing.

-Dcom.ibm.CORBA.CommTrace=true|false
When you set this option to `true`, every incoming and outgoing GIOP message is sent to the trace log. You can set this property independently from **-Dcom.ibm.CORBA.Debug**. Use this property if you want to look only at the flow of information, and you do not want to debug the internal information. The default value for this property is `false`.

Related reference:

"-Dcom.ibm.CORBA.Debug"
This system property enables debugging for the Object Request Broker (ORB) and includes tracing options that control how much information is recorded.

"-Dcom.ibm.CORBA.Debug.Output" on page 72
This system property redirects Object Request Broker (ORB) trace output to a file, which is known as a trace log.

-Dcom.ibm.CORBA.Debug

This system property enables debugging for the Object Request Broker (ORB) and includes tracing options that control how much information is recorded.

-Dcom.ibm.CORBA.Debug=value
Where `value` is one of the following options:

false No output is produced. This option is the default value.

true Messages and traces for the entire ORB code flow

Note: If you use this property without specifying a value, tracing is enabled.

Related reference:

"-Dcom.ibm.CORBA.Debug.Output"
This system property redirects Object Request Broker (ORB) trace output to a file, which is known as a trace log.

"-Dcom.ibm.CORBA.CommTrace" on page 71
This system property turns on wire tracing for the Object Request Broker (ORB), which is also known as *Comm* tracing.

-Dcom.ibm.CORBA.Debug.Output

This system property redirects Object Request Broker (ORB) trace output to a file, which is known as a trace log.

-Dcom.ibm.CORBA.Debug.Output=*filename*

Where *filename* is the name you want to specify for your trace log. If this property is not specified or the value of *filename* is empty, the file name defaults to the following format:

```
orbtrc.DDMMYYYY.HHmm.SS.txt
```

Where:

- D = day
- M = month
- Y = year
- H = hour (24 hour format)
- M = minutes
- S = seconds

If the application or applet does not have the privilege that it requires to write to a file, the trace entries go to `stderr`.

Related reference:

"-Dcom.ibm.CORBA.Debug" on page 71
This system property enables debugging for the Object Request Broker (ORB) and includes tracing options that control how much information is recorded.

"-Dcom.ibm.CORBA.CommTrace" on page 71
This system property turns on wire tracing for the Object Request Broker (ORB), which is also known as *Comm* tracing.

-Dcom.ibm.dbgmalloc

This option provides memory allocation diagnostic information for class library native code.

-Dcom.ibm.dbgmalloc=true

When an application is started with this option, a javadump records the amount of memory allocated by the class library components. You can use this option together with the **-Xcheck:memory** option to obtain information about class library call sites and their allocation sizes. Enabling this option has an impact on throughput performance. The information does not include allocation information from Abstract Windows Toolkit (AWT), ZLIB data compression library, and libwrapper ASCII to EBCDIC conversion library.

-Dcom.ibm.HTTPSPNEGOCrossRealm

From Java 6 service refresh 10, the HTTP/SPNEGO Cross Realm support can be enabled by setting this property to *true*.

-Dcom.ibm.HTTPSPNEGOCrossRealm=true
Support is turned off by default. When HTTP/SPNEGO Cross Realm support is enabled, delegating credentials is turned off.

-Dcom.ibm.jsse2.renegotiate

If your Java application uses JSSE for secure communication, you can disable TLS renegotiation by installing APAR IZ65239.

-Dcom.ibm.jsse2.renegotiate=[ALL | NONE | ABBREVIATED]

ALL Allow both abbreviated and unabbreviated (full) renegotiation handshakes.

NONE
Allow no renegotiation handshakes. This value is the default setting.

ABBREVIATED
Allow only abbreviated renegotiation handshakes.

-Dcom.ibm.lang.management.verbose

Enables verbose information from java.lang.management operations to be written to the output channel during VM operation.

-Dcom.ibm.lang.management.verbose
There are no options for this system property.

-Dcom.ibm.IgnoreMalformedInput

From Java 6 SR9, any invalid UTF8 or malformed byte sequences are replaced with the standard unicode replacement character `\uFFFD`.

-Dcom.ibm.IgnoreMalformedInput=true
To retain the old behavior, where invalid UTF8 or malformed byte sequences are ignored, set this system property to *true*.

-Dcom.ibm.mappedByteBufferForce

Setting this value to true forces data to be committed to disk during system failure.

-Dcom.ibm.mappedByteBufferForce=[true | false]
During system failure, the MappedByteBuffer.force API does not commit data to disk, which prevents data integrity issues. Setting this value to `true` forces data to be committed to disk during system failure. Because this setting can cause performance degradation, this switch is not enabled by default.

-Dcom.ibm.rational.mvfs.checking

Use this property to improve the performance of Multi Version File System (MVFS) file systems.

-Dcom.ibm.rational.mvfs.checking=[true | false]
The WinNTFilesystem methods getModifiedTime and getBooleanAttributes use the windows methods API_wstati64() and _wfindfirsti64() instead of the defaults. This property is not enabled by default because it can cause performance degradation on local file systems. The property also causes degradation on remote Windows shares where there is no Windows directory cache for the remote file system.

-Dcom.ibm.signalhandling.ignoreLogoff

This property controls the way the JVM handles a CTRL_LOGOFF_EVENT signal when the JVM is running as an interactive Windows service.

-Dcom.ibm.signalhandling.ignoreLogoff=[true|false]

Windows issues a CTRL_LOGOFF_EVENT when a user logs out of an interactive Windows service. By default, the JVM ends when this signal is received. Setting this property to `true` prevents the JVM ending when a CTRL_LOGOFF_EVENT signal is received. The default value for this property is `false`.

-Dcom.ibm.streams.CloseFDWithStream

Determines whether the close() method of a stream object closes a native file descriptor even if the descriptor is still in use by another stream object.

-Dcom.ibm.streams.CloseFDWithStream=[true | false]

Usually, you create a FileInputStream or FileOutputStream instance by passing a String or a File object to the stream constructor method. Each stream then has a separate file descriptor. However, you can also create a stream by using an existing FileDescriptor instance, for example one that you obtain from a RandomAccessFile instance, or another FileInputStream or FileOutputStream instance. Multiple streams can then share the same file descriptor.

If you set this option to `false`, when you use the close() method of the stream, the associated file descriptor is also closed only if it is not in use by any other streams. If you set the option to `true`, the file descriptor is closed regardless of any other streams that might still be using it.

The default setting is `true`.

Note: Before version 6 service refresh 14, the default behavior was to close the file descriptor only when all the streams that were using it were also closed. This system property exists so that you can revert to this previous default behavior if necessary. This system property will be removed in a future release, so you should adjust your applications to use the new default behavior before you upgrade to a later release.

-Dcom.ibm.tools.attach.enable

Enable the Attach API for this application.

-Dcom.ibm.tools.attach.enable=yes

The Attach API allows your application to connect to a virtual machine. Your application can then load an agent application into the virtual machine. The agent can be used to perform tasks such as monitoring the virtual machine status.

-Dcom.ibm.UseCLDR16

This property reverts behavior to an earlier release.

-Dcom.ibm.UseCLDR16

From IBM SDK, Java Technology Edition, Version 6 service refresh 10, changes are made to the locale translation files to make them consistent with Oracle JDK 6. To understand the differences in detail, see http://www.ibm.com/support/docview.wss?uid=swg21568667. Include the **-Dcom.ibm.UseCLDR16** system property on the command-line to revert to the locale translation files used in earlier releases.

-Dcom.ibm.xtq.processor.overrideSecureProcessing

This system property affects the XSLT processing of extension functions or extension elements when Java security is enabled.

Purpose

From IBM SDK, Java Technology Edition, Version 6 service refresh 14, the use of extension functions or extension elements is not allowed when Java security is enabled. This change is introduced to enhance security. The system property can be used to revert to the behavior in earlier releases.

Parameters

com.ibm.xtq.processor.overrideSecureProcessing=*true*
To revert to the behavior in earlier releases of the IBM SDK, set this system property to *true*.

-Dcom.ibm.zipfile.closeinputstreams

The Java.util.zip.ZipFile class allows you to create InputStreams on files held in a compressed archive.

-Dcom.ibm.zipfile.closeinputstreams=true
Under some conditions, using ZipFile.close() to close all InputStreams that have been opened on the compressed archive might result in a 56-byte-per-InputStream native memory leak. Setting the **-Dcom.ibm.zipfile.closeinputstreams=true** forces the JVM to track and close InputStreams without the memory impact caused by retaining native-backed objects. Native-backed objects are objects that are stored in native memory, rather than the Java heap. By default, the value of this system property is not enabled.

-Dfile.encoding

Use this property to define the file encoding that is required.

-Dfile.encoding=*value*
Where *value* defines the file encoding that is required.

By default the IBM GBK converter follows Unicode 3.0 standards. To force the IBM GBK converter to follow Unicode 2.0 standards, use a value of *bestfit936*.

-Dibm.jvm.bootclasspath

The value of this property is used as an additional search path.

-Dibm.jvm.bootclasspath
The value of this property is used as an additional search path, which is inserted between any value that is defined by **-Xbootclasspath/p:** and the `bootclass` path. The `bootclass` path is either the default or the one that you defined by using the **-Xbootclasspath:** option.

-Dibm.stream.nio

This option addresses the ordering of IO and NIO converters.

-Dibm.stream.nio=[true | false]
When this option is set to `true`, the NIO converters are used instead of the IO converters. By default the IO converters are used.

-Djava.compiler

Disables the Java compiler by setting to NONE.

-Djava.compiler=[NONE | j9jit<vm_version>]
Enable JIT compilation by setting to j9jit<vm_version> (Equivalent to **–Xjit**).

-Djava.util.Arrays.useLegacyMergeSort

Changes the implementation of java.util.Collections.sort(list, comparator) in this release.

The Java SE 6 implementation of java.util.Collections.sort(list, comparator) relies on the Comparator function, which implements the conditions greater than, less than, and equal. However, the Java SE 5.0 implementation of java.util.Collections.sort(list, comparator) can accept the Comparator function, which implements only the conditions greater than and less than. From IBM SDK, Java Technology Edition, Version 6 service refresh 16 fix pack 1 onwards, you can switch between the Java SE 5.0 and Java SE 6 implementation.

-Djava.util.Arrays.useLegacyMergeSort=[true | false]
Setting the value to `true` changes the Comparator function to the Java SE 5.0 implementation. The default for this setting is `false`.

-Djavax.xml.namespace.QName.useCompatibleHashCodeAlgorithm

Use this property to turn off an enhanced hashing algorithm for javax.xml.namespace.QName.hashCode().

-Djavax.xml.namespace.QName.useCompatibleHashCodeAlgorithm=1.0
From Java 6 SR11 an enhanced hashing algorithm is used for javax.xml.namespace.QName.hashCode(). This algorithm can change the iteration order of items returned from hash maps. For compatibility, you can restore the earlier hashing algorithm by setting the system property **-Djavax.xml.namespace.QName.useCompatibleHashCodeAlgorithm=1.0**.

-Djdk.map.althashing.threshold

This system property controls the use of an enhanced hashing algorithm for hashed maps.

-Djdk.map.althashing.threshold=*value*
This alternative hashing algorithm is used for string keys when a hashed data structure has a capacity larger than *value*.

A value of *1* ensures that this algorithm is always used, regardless of the hashed map capacity. A value of *-1* prevents the use of this algorithm, which is the default value.

The hashed map structures affected by this threshold are: java.util.HashMap, java.util.Hashtable, java.util.LinkedHashMap, java.util.WeakHashMap, and java.util.concurrent.ConcurrentHashMap.

The capacity of a hashed map is related to the number of entries in the map, multiplied by the load factor. Because the capacity of a hashed map is rounded up to the next power of two, setting the threshold to intermediate values has no affect on behavior. For example, threshold values of 600, 700, and 1000 have the same effect. However, values of 1023 and 1024 cause a difference in behavior. For a more detailed description of the capacity and load factor, see http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html.

When entries are removed from a hashed map the capacity does not shrink. Therefore, if the map ever exceeds the threshold to use alternative hashing for Strings, the map always uses alternative hashing for Strings. This behavior does not change, even if entries are later removed or the map is emptied using clear().

The enhanced hashing algorithm is available from Java 6 SR11

-Djdk.xml.entityExpansionLimit

This option provides limits for Java API for XML processing (JAXP). Use this option to limit the number of entity expansions in an XML document.

-Djdk.xml.entityExpansionLimit=*value*

where *value* is a positive integer. The default value is 64,000.

A value of *0* or a negative number sets no limit.

You can also set this limit by adding the following line to your `jaxp.properties` file:

`jdk.xml.entityExpansionLimit=value`

Related reference:

"-Djdk.xml.maxGeneralEntitySizeLimit"
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the maximum size of a general entity.

"-Djdk.xml.maxOccur" on page 78
This option provides limits for Java API for XML processing (JAXP). This option defines the maximum number of content model nodes that can be created in a grammar.

"-Djdk.xml.maxParameterEntitySizeLimit" on page 79
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the maximum size of a parameter entity.

"-Djdk.xml.maxXMLNameLimit" on page 79
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the length of XML names in XML documents.

"-Djdk.xml.totalEntitySizeLimit" on page 81
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the total size of all entities, including general and parameter entities.

"-Djdk.xml.resolveExternalEntities" on page 80
This option provides limits for Java API for XML processing (JAXP). Use this option to control whether external entities are resolved in an XML document.

-Djdk.xml.maxGeneralEntitySizeLimit

This option provides limits for Java API for XML processing (JAXP). Use this option to limit the maximum size of a general entity.

To protect an application from malformed XML, set this value to the minimum size possible.

-Djdk.xml.maxGeneralEntitySizeLimit=*value*

Where *value* is the maximum size that is allowed for a general entity. The default value is 0.

A value of *0* or a negative number sets no limits.

You can also set this limit by adding the following line to your `jaxp.properties` file:

`jdk.xml.maxGeneralEntitySizeLimit=value`

Related reference:

"-Djdk.xml.entityExpansionLimit"
This option provides limits for Java API for XML processing (JAXP). Use this

option to limit the number of entity expansions in an XML document.

"-Djdk.xml.maxOccur"
This option provides limits for Java API for XML processing (JAXP). This option defines the maximum number of content model nodes that can be created in a grammar.

"-Djdk.xml.maxParameterEntitySizeLimit" on page 79
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the maximum size of a parameter entity.

"-Djdk.xml.maxXMLNameLimit" on page 79
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the length of XML names in XML documents.

"-Djdk.xml.totalEntitySizeLimit" on page 81
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the total size of all entities, including general and parameter entities.

"-Djdk.xml.resolveExternalEntities" on page 80
This option provides limits for Java API for XML processing (JAXP). Use this option to control whether external entities are resolved in an XML document.

-Djdk.xml.maxOccur

This option provides limits for Java API for XML processing (JAXP). This option defines the maximum number of content model nodes that can be created in a grammar.

When building a grammar for a W3C XML schema, use this option to limit the number of content model nodes that can be created when the schema defines attributes that can occur multiple times.

`-Djdk.xml.maxOccur=`*`value`*

Where *value* is a positive integer. The default value is 5,000.

A value of *0* or a negative number sets no limits.

You can also set this limit by adding the following line to your `jaxp.properties` file:

`jdk.xml.maxoccur=`*`value`*

Related reference:

"-Djdk.xml.entityExpansionLimit" on page 77
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the number of entity expansions in an XML document.

"-Djdk.xml.maxGeneralEntitySizeLimit" on page 77
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the maximum size of a general entity.

"-Djdk.xml.maxParameterEntitySizeLimit" on page 79
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the maximum size of a parameter entity.

"-Djdk.xml.maxXMLNameLimit" on page 79
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the length of XML names in XML documents.

"-Djdk.xml.totalEntitySizeLimit" on page 81
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the total size of all entities, including general and parameter entities.

"-Djdk.xml.resolveExternalEntities" on page 80
This option provides limits for Java API for XML processing (JAXP). Use this option to control whether external entities are resolved in an XML document.

-Djdk.xml.maxParameterEntitySizeLimit

This option provides limits for Java API for XML processing (JAXP). Use this option to limit the maximum size of a parameter entity.

To protect an application from malformed XML, set this value to the minimum size possible.

-Djdk.xml.maxParameterEntitySizeLimit=*value*

Where *value* is the maximum size that is allowed for a parameter entity. The default value is 0.

A value of *0* or a negative number sets no limits.

You can also set this limit by adding the following line to your `jaxp.properties` file:

`jdk.xml.maxParameterEntitySizeLimit=value`

Related reference:

"-Djdk.xml.entityExpansionLimit" on page 77
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the number of entity expansions in an XML document.

"-Djdk.xml.maxGeneralEntitySizeLimit" on page 77
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the maximum size of a general entity.

"-Djdk.xml.maxOccur" on page 78
This option provides limits for Java API for XML processing (JAXP). This option defines the maximum number of content model nodes that can be created in a grammar.

"-Djdk.xml.maxXMLNameLimit"
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the length of XML names in XML documents.

"-Djdk.xml.totalEntitySizeLimit" on page 81
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the total size of all entities, including general and parameter entities.

"-Djdk.xml.resolveExternalEntities" on page 80
This option provides limits for Java API for XML processing (JAXP). Use this option to control whether external entities are resolved in an XML document.

-Djdk.xml.maxXMLNameLimit

This option provides limits for Java API for XML processing (JAXP). Use this option to limit the length of XML names in XML documents.

-Djdk.xml.maxXMLNameLimit=*value*

Where *value* is a positive integer.

A value of 0 or a negative number sets no limits. The default value is 0.

You can also set this limit by adding the following line to your `jaxp.properties` file:

`jdk.xml.maxXMLNameLimit=value`

Related reference:

"-Djdk.xml.entityExpansionLimit" on page 77
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the number of entity expansions in an XML document.

"-Djdk.xml.maxGeneralEntitySizeLimit" on page 77
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the maximum size of a general entity.

"-Djdk.xml.maxOccur" on page 78
This option provides limits for Java API for XML processing (JAXP). This option defines the maximum number of content model nodes that can be created in a grammar.

"-Djdk.xml.maxParameterEntitySizeLimit" on page 79
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the maximum size of a parameter entity.

"-Djdk.xml.resolveExternalEntities"
This option provides limits for Java API for XML processing (JAXP). Use this option to control whether external entities are resolved in an XML document.

"-Djdk.xml.totalEntitySizeLimit" on page 81
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the total size of all entities, including general and parameter entities.

-Djdk.xml.resolveExternalEntities

This option provides limits for Java API for XML processing (JAXP). Use this option to control whether external entities are resolved in an XML document.

-Djdk.xml.resolveExternalEntities=*value*

Where *value* is boolean. The default value is *true*.

A value of *false* turns off the resolution of XML external entities.

You can also set this limit by adding the following line to your `jaxp.properties` file:

`jdk.xml.resolveExternalEntities=`*value*

Related reference:

"-Djdk.xml.entityExpansionLimit" on page 77
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the number of entity expansions in an XML document.

"-Djdk.xml.maxGeneralEntitySizeLimit" on page 77
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the maximum size of a general entity.

"-Djdk.xml.maxOccur" on page 78
This option provides limits for Java API for XML processing (JAXP). This option defines the maximum number of content model nodes that can be created in a grammar.

"-Djdk.xml.maxXMLNameLimit" on page 79
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the length of XML names in XML documents.

"-Djdk.xml.totalEntitySizeLimit" on page 81
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the total size of all entities, including general and parameter entities.

"-Djdk.xml.maxParameterEntitySizeLimit" on page 79
This option provides limits for Java API for XML processing (JAXP). Use this

option to limit the maximum size of a parameter entity.

-Djdk.xml.totalEntitySizeLimit

This option provides limits for Java API for XML processing (JAXP). Use this option to limit the total size of all entities, including general and parameter entities.

-Djdk.xml.totalEntitySizeLimit=*value*

Where *value* is the collective size of all entities. The default value is 5x10^7 (50 000 000).

A value of *0* or a negative number sets no limits.

You can also set this limit by adding the following line to your `jaxp.properties` file:

`jdk.xml.totalEntitySizeLimit=value`

Related reference:

"-Djdk.xml.entityExpansionLimit" on page 77
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the number of entity expansions in an XML document.

"-Djdk.xml.maxGeneralEntitySizeLimit" on page 77
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the maximum size of a general entity.

"-Djdk.xml.maxOccur" on page 78
This option provides limits for Java API for XML processing (JAXP). This option defines the maximum number of content model nodes that can be created in a grammar.

"-Djdk.xml.maxParameterEntitySizeLimit" on page 79
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the maximum size of a parameter entity.

"-Djdk.xml.maxXMLNameLimit" on page 79
This option provides limits for Java API for XML processing (JAXP). Use this option to limit the length of XML names in XML documents.

"-Djdk.xml.resolveExternalEntities" on page 80
This option provides limits for Java API for XML processing (JAXP). Use this option to control whether external entities are resolved in an XML document.

-Dsun.awt.keepWorkingSetOnMinimize

The **-Dsun.awt.keepWorkingSetOnMinimize=true** system property stops the JVM trimming an application when it is minimized.

-Dsun.awt.keepWorkingSetOnMinimize=true

When a Java application using the Abstract Windowing Toolkit (AWT) is minimized, the default behavior is to "trim" the "working set". The working set is the application memory stored in RAM. Trimming means that the working set is marked as being available for swapping out if the memory is required by another application. The advantage of trimming is that memory is available for other applications. The disadvantage is that a "trimmed" application might experience a delay as the working set memory is brought back into RAM.

The default behavior is to trim an application when it is minimized.

-Dsun.net.client.defaultConnectTimeout

Specifies the default value for the connect timeout for the protocol handlers used by the java.net.URLConnection class.

-Dsun.net.client.defaultConnectTimeout=<*value in milliseconds*>
The default value set by the protocol handlers is -1, which means that no timeout is set.

When a connection is made by an applet to a server and the server does not respond properly, the applet might seem to hang. The delay might also cause the browser to hang. The apparent hang occurs because there is no network connection timeout. To avoid this problem, the Java Plug-in has added a default value to the network timeout of 2 minutes for all HTTP connections. You can override the default by setting this property.

-Dsun.net.client.defaultReadTimeout

Specifies the default value for the read timeout for the protocol handlers used by the java.net.URLConnection class when reading from an input stream when a connection is established to a resource.

-Dsun.net.client.defaultReadTimeout=<*value in milliseconds*>
The default value set by the protocol handlers is -1, which means that no timeout is set.

-Dsun.nio.MaxDirectMemorySize

Limits the native memory size for nio Direct Byte Buffer objects to the value specified.

-Dsun.nio.MaxDirectMemorySize=<*value*>
Specify <*value*> in bytes.

-Dsun.reflect.inflationThreshold

Controls inflation from the JNI implementation of reflection to the Java implementation of reflection.

When your application uses Java reflection, the JVM has two methods of accessing the information on the class being reflected. It can use a JNI accessor, or a Java bytecode accessor. If your application uses reflection extensively, you might want to force the JVM to use the JNI accessor because the Java bytecode accessor can use a significant amount of native memory.

-Dsun.reflect.inflationThreshold=<*value*>
Where a <*value*> sets the number of times to use the JNI accessor before the JVM changes to use the Java bytecode accessor, a process that is known as *inflation*. A value of 0 causes reflection never to inflate from the JNI accessor to the Java bytecode accessor.

Note: The Oracle implementation of this system property is different. Setting the value to 0 causes reflection to inflate from the JNI implementation of reflection to the Java implementation of reflection after the first usage. If you want to force the use of the Java implementation of reflection, use **-Dsun.reflect.noInflation=true**.

-Dsun.rmi.transport.tcp.connectionPool

Enables thread pooling for the RMI ConnectionHandlers in the TCP transport layer implementation.

-Dsun.rmi.transport.tcp.connectionPool=*val*
val is either *true* or a value that is not null.

-Dswing.useSystemFontSettings

This option addresses compatibility problems for Swing programs.

-Dswing.useSystemFontSettings=[false]
By default, Swing programs running with the Windows Look and Feel render with the system font set by the user instead of a Java-defined font. As a result, fonts differ from the fonts in earlier releases. This option addresses compatibility problems like these for programs that depend on the old behavior. By setting this option, v1.4.1 fonts and those of earlier releases are the same for Swing programs running with the Windows Look and Feel.

JVM command-line options

Use these options to configure your JVM. The options prefixed with **-X** are nonstandard.

Options that relate to the JIT are listed under "JIT and AOT command-line options" on page 98. Options that relate to the Garbage Collector are listed under "Garbage Collector command-line options" on page 101.

-X

Displays help on nonstandard options.

-X Displays help on nonstandard options.

-Xaggressive

Enables performance optimizations.

-Xaggressive
Enables performance optimizations that are expected to be the default in future releases.

-Xargencoding

Include Unicode escape sequences in the argument list.

-Xargencoding
You can use the Unicode escape sequences in the argument list that you pass to this option. To specify a Unicode character, use escape sequences in the form \u####, where # is a hexadecimal digit (0 - 9, A to F).

-Xargencoding:utf8
Use utf8 encoding.

-Xargencoding:latin
Use ISO8859_1 encoding.

To specify a class that is called HelloWorld and use Unicode encoding for both capital letters, specify this command:

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

-Xbootclasspath

Sets the search path for bootstrap classes and resources.

-Xbootclasspath:<*directories and compressed or Java archive files separated by : (; on* Windows*)*>
The default is to search for bootstrap classes and resources in the internal VM directories and .jar files.

-Xbootclasspath/a:

Appends to the end of the search path for bootstrap classes.

-Xbootclasspath/a:<*directories and compressed or Java archive files separated by : (; on* Windows*)*>
Appends the specified directories, compressed files, or .jar files to the end of the bootstrap class path. The default is to search for bootstrap classes and resources in the internal VM directories and `.jar` files.

-Xbootclasspath/p:

Adds a prefix to the search path for bootstrap classes.

-Xbootclasspath/p:<*directories and compressed or Java archive files separated by : (; on* Windows*)*>
Adds a prefix of the specified directories, compressed files, or Java archive files to the front of the bootstrap class path. Do not deploy applications that use the **-Xbootclasspath:** or the **-Xbootclasspath/p:** option to override a class in the standard API. The reason is that such a deployment contravenes the Java 2 Runtime Environment binary code license. The default is to search for bootstrap classes and resources in the internal VM directories and `.jar` files.

-Xcheck

You can use the **-Xcheck** option to run checks during JVM startup, such as memory checks or checks on JNI functions.

-Xcheck:<*option*>
The options available are detailed in separate topics.

-Xcheck:classpath:

Displays a warning message if an error is discovered in the class path.

-Xcheck:classpath
Checks the classpath and reports if an error is discovered; for example, a missing directory or JAR file.

-Xcheck:gc:

Runs additional checks on garbage collection.

-Xcheck:gc[:<*scan options*>][:<*verify options*>][:<*misc options*>]
By default, no checks are made. See the output of **-Xcheck:gc:help** for more information.

-Xcheck:jni:

Runs additional checks for JNI functions.

-Xcheck:jni[**:help**][:<*option*>=<*value*>]
This option is equivalent to **-Xrunjnichk**. By default, no checks are made.

-Xcheck:memory:

Identifies memory leaks inside the JVM.

-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. The available options are as follows:

all
Enables checking of all allocated and freed blocks on every free and

allocate call. This check of the heap is the most thorough. It typically causes the JVM to exit on nearly all memory-related problems soon after they are caused. This option has the greatest affect on performance.

callsite=<*number of allocations*>

Displays callsite information every <*number of allocations*>. De-allocations are not counted. Callsite information is presented in a table with separate information for each callsite. Statistics include:

- The number and size of allocation and free requests since the last report.
- The number of the allocation request responsible for the largest allocation from each site.

Callsites are presented as `sourcefile:linenumber` for C code and assembly function name for assembler code.

Callsites that do not provide callsite information are accumulated into an "unknown" entry.

failat=<*number of allocations*>

Causes memory allocation to fail (return NULL) after <*number of allocations*>. Setting <*number of allocations*> to 13 causes the 14th allocation to return NULL. De-allocations are not counted. Use this option to ensure that JVM code reliably handles allocation failures. This option is useful for checking allocation site behavior rather than setting a specific allocation limit.

ignoreUnknownBlocks

Ignores attempts to free memory that was not allocated using the **-Xcheck:memory** tool. Instead, the **-Xcheck:memory** statistics that are printed out at the end of a run indicates the number of “unknown” blocks that were freed.

mprotect=<*top*|*bottom*>

Locks pages of memory on supported platforms, causing the program to stop if padding before or after the allocated block is accessed for reads or writes. An extra page is locked on each side of the block returned to the user.

If you do not request an exact multiple of one page of memory, a region on one side of your memory is not locked. The **top** and **bottom** options control which side of the memory area is locked. **top** aligns your memory blocks to the top of the page (lower address), so buffer underruns result in an application failure. **bottom** aligns your memory blocks to the bottom of the page (higher address) so buffer overruns result in an application failure.

Standard padding scans detect buffer underruns when using **top** and buffer overruns when using **bottom**.

nofree

Keeps a list of blocks that are already used instead of freeing memory. This list, and the list of currently allocated blocks, is checked for memory corruption on every allocation and deallocation. Use this option to detect a dangling pointer (a pointer that is "dereferenced" after its target memory is freed). This option cannot be reliably used with long-running applications (such as WebSphere Application Server), because “freed” memory is never reused or released by the JVM.

noscan

Checks for blocks that are not freed. This option has little effect on

performance, but memory corruption is not detected. This option is compatible only with **subAllocator**, **callsite**, and **callsitesmall**.

quick
Enables block padding only and is used to detect basic heap corruption. Every allocated block is padded with sentinel bytes, which are verified on every allocate and free. Block padding is faster than the default of checking every block, but is not as effective.

skipto=<*number of allocations*>
Causes the program to check only on allocations that occur after <*number of allocations*>. De-allocations are not counted. Use this option to speed up JVM startup when early allocations are not causing the memory problem. The JVM performs approximately 250+ allocations during startup.

subAllocator[=<*size in MB*>]
Allocates a dedicated and contiguous region of memory for all JVM allocations. This option helps to determine if user JNI code or the JVM is responsible for memory corruption. Corruption in the JVM **subAllocator** heap suggests that the JVM is causing the problem; corruption in the user-allocated memory suggests that user code is corrupting memory. Typically, user and JVM allocated memory are interleaved.

zero
Newly allocated blocks are set to 0 instead of being filled with the `0xE7E7xxxxxxxxE7E7` pattern. Setting these blocks to 0 helps you to determine whether a callsite is expecting zeroed memory, in which case the allocation request is followed by `memset(pointer, 0, size)`.

Note: The **-Xcheck:memory** option cannot be used in the **-Xoptionsfile**.

-Xclassgc

Enables dynamic unloading of classes by the JVM. Garbage collection of class objects occurs only on class loader changes.

-Xclassgc
Dynamic unloading is the default behavior. To disable dynamic class unloading, use the **-Xnoclassgc** option.

-Xcompressedrefs

Enables the use of compressed references.

-Xcompressedrefs
(64-bit only) To disable compressed references, use the **-Xnocompressedreferences** option. For more information, see Compressed references.

Compressed references are disabled by default.

You cannot include this option in an options file. You must specify this option on the command line, or by using the **IBM_JAVA_OPTIONS** environment variable.

-Xdbg

Loads debugging libraries to support the remote debugging of applications.

-Xdbg:<*options*>
This option is deprecated in the IBM SDK, Java Technology Edition, Version 6. By default, the debugging libraries are not loaded, and the VM instance is not enabled for debug.

The preferred method to enable the debugger is **-agentlib:jdwp=**<*options*>. For more information about using the Java debugger, see the IBM SDK, Java Technology Edition, Version 6 user guide.

-Xdiagnosticscollector

Enables the Diagnostics Collector.

-Xdiagnosticscollector[:settings=<filename>]
See The Diagnostics Collector for more information. The settings option allows you to specify a different Diagnostics Collector settings file to use instead of the default `dc.properties` file in the JRE.

-Xdisablejavadump

Turns off Javadump generation on errors and signals.

-Xdisablejavadump
By default, Javadump generation is enabled.

-Xdump

Use the **-Xdump** option to add and remove dump agents for various JVM events, update default dump settings (such as the dump name), and limit the number of dumps that are produced.

-Xdump
See Using dump agents for more information.

-Xenableexplicitgc

This options tells the VM to trigger a garbage collection when a call is made to System.gc().

-Xenableexplicitgc
Signals to the VM that calls to System.gc() trigger a garbage collection. This option is enabled by default.

-Xfastresolve

Tune performance by improving the resolution time for classes.

-Xfastresolve<*n*>
This option is used to tune performance by improving the resolution time for classes when the field count exceeds the threshold specified by <*n*>. If profiling tools show significant costs in field resolution, change the threshold until the costs are reduced. If you enable this option, additional memory is used when the threshold is exceeded.

-Xfuture

Turns on strict class-file format checks.

-Xfuture
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.

-Xiss

Sets the initial stack size for Java threads.

-Xiss<*size*>
By default, the stack size is set to 2 KB. Use the **-verbose:sizes** option to output the value that the VM is using.

-Xjarversion

Produces output information about the version of each .jar file.

-Xjarversion

Produces output information about the version of each .jar file in the class path, the boot class path, and the extensions directory. Version information is taken from the Implementation-Version and Build-Level properties in the manifest of the .jar file.

Note: The **-Xjarversion** option cannot be used in the **-Xoptionsfile**.

-Xjni

Sets JNI options.

-Xjni:<*suboptions*>

You can use the following suboption 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.

-Xlinenumbers

See also **-Xnolinenumbers**. By default, line numbers are on.

-Xlog

Enables message logging.

-Xlog

To prevent message logging, use the **-Xlog:none** option. By default, logging is enabled. This option is available from Java 6 SR5. See JVM Messages.

-Xlp

Requests the JVM to allocate the Java object heap and JIT code cache memory with large pages.

-Xlp[<*size*>]

Windows: Requests the JVM to allocate the Java object heap with large pages. This command is available only on Windows Server 2003 and later, and Windows Vista, and later releases.

For more information, see "Configuring large page memory allocation" on page 39.

All platforms: To obtain the large page sizes available and the current setting, use the **-verbose:sizes** option. Note the current settings are the requested sizes and not the sizes obtained. For object heap size information, check the **-verbose:gc** output.

The JVM ends if there are insufficient operating system resources to satisfy the request. However, an error message is not issued. This limitation and a workaround for verifying the page size that is used can be found in Known limitations.

-Xmso

Sets the initial stack size for operating system threads.

-Xmso<*size*>

The default value can be determined by running the command:

```
java -verbose:sizes
```

The maximum value for the stack size varies according to platform and specific machine configuration. If you exceed the maximum value, a `java/lang/StackOverflowError` message is reported.

-Xnoagent

Disables support for the old JDB debugger.

-Xnoagent
Disables support for the old JDB debugger.

-Xnoclassgc

Disables class garbage collection.

-Xnoclassgc
This option switches off garbage collection of storage associated with Java technology classes that are no longer being used by the JVM. The default behavior is as defined by **-Xclassgc**. Enabling this option is not recommended except under the direction of the IBM support team. The reason is the option can cause unlimited native memory growth, leading to out-of-memory errors.

-Xnocompressedrefs

Disables the use of compressed references.

-Xnocompressedrefs
(64-bit only)

This option disables the use of compressed references.

You cannot include this option in an options file. You must specify this option on the command line, or by using the **IBM_JAVA_OPTIONS** environment variable.

To enable compressed references, use the **-Xcompressedreferences** option. For more information, see Compressed references.

-Xnolinenumbers

Disables the line numbers for debugging.

-Xnolinenumbers
See also **-Xlinenumbers**. By default, line number are on.

-Xnosigcatch

Disables JVM signal handling code.

-Xnosigcatch
See also **-Xsigcatch**. By default, signal handling is enabled.

-Xnosigchain

Disables signal handler chaining.

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

-Xoptionsfile

Specifies a file that contains VM options and definitions.

-Xoptionsfile=<*file*>
where <*file*> contains options that are processed as if they had been entered directly as command-line options. By default, a user option file is not used.

Here is an example of an options file:

```
#My options file
-X<option1>
-X<option2>=\
<value1>,\
<value2>
-D<sysprop1>=<value1>
```

The options file does not support these options:

- **-assert**
- **-fullversion**
- **-help**
- **-showversion**
- **-version**
- **-Xcompressedrefs**
- **-Xcheck:memory**
- **-Xjarversion**
- **-Xoptionsfile**

Although you cannot use **-Xoptionsfile** recursively within an options file, you can use **-Xoptionsfile** multiple times on the same command line to load more than one options files.

Some options use quoted strings as parameters. Do not split quoted strings over multiple lines using the forward slash line continuation character (\). The Yen symbol (¥) is not supported as a line continuation character. For example, the following example is not valid in an options file:

```
-Xevents=vmstop,exec="cmd /c \
echo %pid has finished."
```

The following example is valid in an options file:

```
-Xevents=vmstop, \
exec="cmd /c echo %pid has finished."
```

Related information:

"Specifying command-line options" on page 69
Although the command line is the traditional way to specify command-line options, you can also pass options to the Java virtual machine (VM) by using options files and environment variables.

TITLE, GPINFO, and ENVINFO sections

-Xoss

Sets the maximum Java stack size for any thread.

-Xoss<*size*>
Recognized but deprecated. Use **-Xss** and **-Xmso** instead. The maximum value for the stack size varies according to platform and specific machine configuration. If you exceed the maximum value, a `java/lang/OutOfMemoryError` message is reported.

-Xrdbginfo

Loads the remote debug information server with the specified host and port.

-Xrdbginfo:<*host*>:<*port*>
By default, the remote debug information server is disabled.

-Xrs

Disables signal handling in the JVM.

-Xrs

Setting **-Xrs** prevents the Java run time environment from handling any internally or externally generated signals such as SIGSEGV and SIGABRT. Any signals that are raised are handled by the default operating system handlers. Disabling signal handling in the JVM reduces performance by approximately 2-4%, depending on the application.

-Xrs:sync

On Windows systems, hardware exceptions are not handled by the JVM when this option is specified. However, the Windows CTRL_BREAK_EVENT signal, triggered by the Ctrl-Break key combination, is still handled by the JVM. As with **-Xrs**, the use of **-Xrs:sync** reduces performance by approximately 2-4%, depending on the application.

Note: Setting this option prevents dumps being generated by the JVM for signals such as SIGSEGV and SIGABRT, because the JVM is no longer intercepting these signals.

-Xrun

This option loads helper libraries, but has been superseded by the **-agentlib** option.

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

This option has been superseded; use the **-agentlib** option instead. For more information about **-agentlib**, see Using the JVMTI.

-Xrun 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.

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

Loads debugging libraries to support the remote debugging of applications. This option is the same as **-Xdbg**.

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

Deprecated. Use **-Xcheck:jni** instead.

-Xscmx

Specifies cache size.

-Xscmx<*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. Minimum cache size is 4 KB. Maximum cache size is platform-dependent. The size of cache that you can specify is limited by the amount of physical memory and paging space available to the system. The virtual address space of a process is shared between the shared classes cache and the Java heap. Increasing the maximum size of the Java heap reduces the size of the shared classes cache that you can create.

-XselectiveDebug

Enables selective debugging.

-XselectiveDebug

Use the com.ibm.jvm.Debuggable annotation to mark classes and methods that must be available for debugging. The JVM optimizes methods that do not need

debugging to provide better performance in a debugging environment. See the *User Guide* for your platform for more information.

-Xshareclasses

Enables class sharing. This option can take a number of suboptions, some of which are cache utilities.

-Xshareclasses:<*suboptions*>

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.

Note: 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 that are 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:

cacheDir=<*directory*>
Sets the directory in which cache data is read and written. By default, <*directory*> is the user's `C:\Documents and Settings\<username>\Local Settings\Application Data\javasharedresources` directory on Windows or `/tmp/javasharedresources` on Linux, AIX®, z/OS®, and IBM i. You must have sufficient permissions in <*directory*>. The JVM writes persistent cache files directly into the directory specified. Persistent cache files can be safely moved and deleted from the file system. Nonpersistent 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**.

cacheRetransformed
Enables caching of classes that are transformed by using the JVMTI RetransformClasses function. See JVMTI redefinition and retransformation of classes for more information.

destroy (Utility option)
Destroys a cache that is 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*> (Utility option)
Destroys all caches that are unused for the time that is specified before loading shared classes. This option is not a utility option because it does not cause the JVM to exit. On NTFS file systems, the `expire` option is accurate to the nearest hour.

groupAccess
Sets operating system permissions on a new cache to allow group access to the cache. Group access can be set only when permitted by the operating system **umask** setting. The default is user access only.

help
Lists all the command-line options.

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.

mprotect=[default | all | none]
Where:

- `default`: By default, the memory pages that contain the cache are always protected, 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 performance cost.
- `all`: This option ensures that all the cache pages are protected, including the header.
- `none`: Specifying this option disables the page protection.

Note: Specifying `all` has a negative impact on performance. You should specify `all` only for problem diagnosis and not for production.

modified=<*modified context*>
Used when a JVMTI agent is installed that might modify bytecode at run time. 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 an application with a modification context and then run it again with a different modification context, all classes are stored twice in the cache. See Dealing with runtime bytecode modification for more information.

name=<*name*>
Connects to a cache of a given name, creating the cache if it does not exist. This option is 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.

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

noBootclasspath
Disables the storage of classes loaded by the bootstrap class loader in the shared classes cache. Often used with the SharedClassURLFilter API to control exactly which classes are cached. See Using the SharedClassHelper API for more information about shared class filtering.

none
Added to the end of a command line, disables class data sharing. This suboption overrides class sharing arguments found earlier on the command line.

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.

nonpersistent
Uses a nonpersistent cache. The cache is lost when the operating system shuts down. Nonpersistent and persistent caches can have the same name. You must always use the **nonpersistent** suboption when running utilities such as **destroy** on a nonpersistent cache.

persistent (default for Windows and Linux platforms)
Uses a persistent cache. The cache is created on disk, which persists beyond operating system restarts. Nonpersistent and persistent caches can have the same name.

printAllStats (Utility option)
Displays detailed information about the contents of the cache that is specified in the **name=**<*name*> suboption. If the name is not specified, statistics are displayed about the default cache. Every class is listed in chronological order with a reference to the location from which it was loaded. See printAllStats utility for more information.

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

readonly
Opens an existing cache with read-only permissions. The Java virtual machine does not create a new cache with this suboption. Opening a cache read-only prevents the VM from making any updates to the cache. If you specify this suboption, the VM can connect to caches that were created by other users or groups without requiring write access.

By default, this suboption is not specified.

reset
Causes a cache to be destroyed and then re-created when the JVM starts up. This option can be added to the end of a command line as **-Xshareclasses:reset**.

safemode
Forces the JVM to load all classes from disk and apply the modifications to those classes (if applicable). For more information, see Using the `safemode` option.

This suboption is deprecated in IBM SDK, Java Technology Edition, Version 6.

silent
Disables all shared class messages, including error messages. Unrecoverable error messages, which prevent the JVM from initializing, are displayed.

verbose
Gives detailed output on the cache I/O activity, listing information about classes that are stored and found. Each class loader is given a unique ID

(the bootstrap loader is always 0) and the output shows the class loader hierarchy at work, where class loaders must ask their parents for a class before they can load it themselves. It is typical to see many failed requests; this behavior is expected for the class loader hierarchy. The standard option **-verbose:class** also enables class sharing verbose output if class sharing is enabled.

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 that is generated at all for a small application. You can disable AOT caching using the **noaot** suboption. See the IBM JVM Messages Guide for a list of the messages produced.

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

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

-Xsigcatch

Enables VM signal handling code.

-Xsigcatch
See also **-Xnosigcatch**. By default, signal handling is enabled.

-Xsigchain

Enables signal handler chaining.

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

-Xss

Sets the maximum stack size for Java threads.

-Xss<*size*>
The default is 256 KB for 32-bit JVMs and 512 KB for 64-bit JVMs. The maximum value varies according to platform and specific machine configuration. If you exceed the maximum value, a `java/lang/OutOfMemoryError` message is reported.

-Xssi

Sets the stack size increment for Java threads.

-Xssi<*size*>
When the stack for a Java thread becomes full it is increased in size by this value until the maximum size (**-Xss**) is reached. The default is 16 KB.

-Xthr

-Xthr:<*suboptions*>

-Xthr:minimizeUserCPU
Minimizes user-mode CPU usage in thread synchronization where possible. The reduction in CPU usage might be a trade-off in exchange for decreased performance.

-XtlhPrefetch (64-bit)

Speculatively prefetches bytes in the thread local heap (TLH) ahead of the current allocation pointer during object allocation.

-XtlhPrefetch
This option helps reduce the performance cost of subsequent allocations.

-Xtrace

Trace options.

-Xtrace[**:help**] | [**:**<*option*>**=**<*value*>**, ...**]
See Controlling the trace for more information.

-Xtune:virtualized

Optimizes JVM function for virtualized environments, such as a cloud.

-Xtune:virtualized
Optimizes JVM function for virtualized environments, such as a cloud.

-Xverify

Use this option to enable or disable the verifier.

-Xverify[:<*option*>**]**
With no parameters, enables the verifier, which is the default. Therefore, if used on its own with no parameters, for example, **-Xverify**, this option does nothing. Optional parameters are as follows:

- **all** - enable maximum verification
- **none** - disable the verifier
- **remote** - enables strict class-loading checks on remotely loaded classes

The verifier is on by default and must be enabled for all production servers. Running with the verifier off is not a supported configuration. If you encounter problems and the verifier was turned off using **-Xverify:none**, remove this option and try to reproduce the problem.

-Xzero

Enables reduction of the memory footprint of the Java runtime environment when concurrently running multiple Java invocations.

-Xzero[:<*option*>**]**
-Xzero might not be appropriate for all types of applications because it changes the implementation of `java.util.ZipFile`, which might cause extra memory usage. **-Xzero** includes the optional parameters:

- **j9zip** - enables the j9zip sub option
- **noj9zip** - disables the j9zip sub option
- **sharezip** - enables the sharezip sub option
- **nosharezip** - disables the sharezip sub option
- **none** - disables all sub options
- **describe** - prints the sub options in effect

Because future versions might include more default options, **-Xzero** options are used to specify the sub options that you want to disable. By default, **-Xzero** enables **j9zip** and **sharezip**. A combination of **j9zip** and **sharezip** enables all .jar files to have shared caches:

- **j9zip** - uses a new `java.util.ZipFile` implementation. This suboption is not a requirement for **sharezip**; however, if **j9zip** is not enabled, only the bootstrap .jar files have shared caches.
- **sharezip** - puts the j9zip cache into shared memory. The j9zip cache is a map of zip entry names to file positions used to quickly find entries in the .zip file. You must enable **-Xshareclasses** to avoid a warning message. When using the **sharezip** suboption, note that this suboption allows every opened .zip file and .jar file to store the j9zip cache in shared memory, so you might fill the shared memory when opening multiple new .zip files and .jar files. The affected API is `java.util.zip.ZipFile` (superclass of `java.util.jar.JarFile`). The .zip and .jar files do not have to be on a class path.

The system property com.ibm.zero.version is defined, and has a current value of 2. Although **-Xzero** is accepted on all platforms, support for the sub options varies by platform:

- **-Xzero** with all other sub options are available only on Windows x86-32 and Linux x86-32 platforms.

.

JVM -XX command-line options

JVM command-line options that are specified with -XX are not recommended for casual use.

These options are subject to change without notice.

-XXallowvmshutdown

This option is provided as a workaround for customer applications that cannot shut down cleanly, as described in APAR IZ59734.

-XXallowvmshutdown:[false|true]

Customers who need this workaround should use **-XXallowvmshutdown:false**. The default option is **-XXallowvmshutdown:true**for Java 6 SR5 onwards.

-XX:codecachetotal

Use this option to set the maximum size limit for the JIT code cache.

-XX:codecachetotal=<*size*>

This option is an alias for the "-Xcodecachetotal" on page 99 option.

-XX:MaxDirectMemorySize

This option sets a limit on the amount of memory that can be reserved for all Direct Byte Buffers.

-XX:MaxDirectMemorySize=<*size*>

Where <*size*> is the limit on memory that can be reserved for all Direct Byte Buffers. If a value is set for this option, the sum of all Direct Byte Buffer sizes cannot exceed the limit. After the limit is reached, a new Direct Byte Buffer can be allocated only when enough old buffers are freed to provide enough space to allocate the new buffer.

By default, the JVM does not set a limit on how much memory is reserved for Direct Byte Buffers. A soft limit of 64 MB is set, which the JVM automatically expands in 32 MB chunks, as required.

-XX:-StackTraceInThrowable

This option removes stack traces from exceptions.

-XX:-StackTraceInThrowable

By default, stack traces are available in exceptions. Including a stack trace in exceptions requires walking the stack and that can affect performance. Removing stack traces from exceptions can improve performance but can also make problems harder to debug.

When this option is enabled, Throwable.getStackTrace() returns an empty array and the stack trace is displayed when an uncaught exception occurs. Thread.getStackTrace() and Thread.getAllStackTraces() are not affected by this option.

-XX:[+|-]UseCompressedOops (64-bit only)

This option enables or disables compressed references in 64-bit JVMs, and is provided to help when porting applications from the Oracle JVM to the IBM JVM. This option might not be supported in subsequent releases.

-XX:[+|-]UseCompressedOops

The **-XX:+UseCompressedOops** option enables compressed references in 64-bit JVMs. The **-XX:+UseCompressedOops** option is similar to specifying **-Xcompressedrefs**, which is detailed in the topic "JVM command-line options" on page 83.

The **-XX:-UseCompressedOops** option prevents the use of compressed references in 64-bit JVMs.

JIT and AOT command-line options

Use these JIT and AOT compiler command-line options to control code compilation.

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 more information about JIT and AOT, see JIT and AOT problem determination.

-Xaot

Use this option to control the behavior of the AOT compiler.

-Xaot[:<*parameter*>=<*value*>, ...]

With no parameters, enables the AOT compiler. The AOT compiler is enabled by default but is not active unless shared classes are enabled. Using this option on its own has no effect. The following parameters are useful:

count=<*n*>

Where <*n*> is the number of times a method is called before it is compiled or loaded from an existing shared class cache. For example, setting `count=0` forces the AOT compiler to compile everything on first execution.

exclude={<*method*>}

Where <*method*> is the Java method you want to exclude when AOT code is compiled or loaded from the shared classes cache. You can use this option if the method causes the program to fail.

limitFile=(<*filename*>,<*m*>,<*n*>)
Compile or load only the methods listed on lines <*m*> to <*n*> in the specified limit file. Methods not listed in the limit file and methods listed on lines outside the range are not compiled or loaded.

loadExclude=<*methods*>
Do not load methods beginning with <*methods*>.

loadLimit=<*methods*>
Load methods beginning with <*methods*> only.

loadLimitFile=(<*filename*>,<*m*>,<*n*>)
Load only the methods listed on lines <*m*> to <*n*> in the specified limit file. Methods not listed in the limit file and methods listed on lines outside the range are not loaded.

verbose
Reports information about the AOT and JIT compiler configuration and method compilation.

-Xcodecache

This option is used to tune performance.

-Xcodecache<*size*>
This option sets the size of each block of memory that is allocated to store the native code of compiled Java methods. By default, this size is selected internally according to the processor architecture and the capability of your system. The maximum value a user can specify is 32 MB. If you set a value larger than 32 MB, the JIT ignores the input and sets the value to 32 MB.

Note: The JIT compiler might allocate more than one code cache for an application. Use the **-Xcodecachetotal** option to set the maximum amount of memory that is used by all code caches.

-Xcodecachetotal

Use this option to set the maximum size limit for the JIT code cache.

-Xcodecachetotal<*size*>
See "JIT and AOT command-line options" on page 98 for more information about the <*size*> parameter.

By default, the total size of the JIT code cache is determined by your operating system, architecture, and the version of the IBM SDK that you are using. Long-running, complex, server-type applications can fill the JIT code cache, which can cause performance problems because not all of the important methods can be JIT-compiled. Use the **-Xcodecachetotal** option to increase the maximum code cache size beyond the default setting, to a setting that suits your application.

The value that you specify is rounded up to a multiple of the code cache block size, as specified by the "-Xcodecache" option. If you specify a value for the **-Xcodecachetotal** optoin that is smaller than the default setting, that value is ignored.

The maximum size limits, for both the JIT code and data caches, that are in use by the JVM are shown in Javadump output. Look for lines that begin with `1STSEGLIMIT`. Use this information together with verbose JIT tracing to determine suitable values for this option on your system. For example Javadump output, see Storage Management (MEMINFO).

Related reference:

"-Xjit"

Use the JIT compiler command line option to produce verbose JIT trace output.

Related information:

Using Javadump

-Xint

This option makes the JVM use the Interpreter only, disabling the Just-In-Time (JIT) and Ahead-Of-Time (AOT) compilers.

-Xint
By default, the JIT compiler is enabled. By default, the AOT compiler is enabled, but is not used by the JVM unless shared classes are also enabled.

-Xjit

Use this option to control the behavior of the JIT compiler.

-Xjit[:<*option*>=<*value*>, ...]
The JIT compiler is enabled by default. Therefore, specifying **-Xjit** with no options, has no effect. These options can be used to modify behavior:

count=<*n*>
Where <*n*> is the number of times a method is called before it is compiled. For example, setting `count=0` forces the JIT compiler to compile everything on first execution.

exclude={<*method*>}
Excludes the specified method from compilation.

limitFile=(<*filename*>, <*m*>, <*n*>)
Compile only the methods that are listed on lines <*m*> to <*n*> in the specified limit file. Methods that are not listed in the limit file and methods that are listed on lines outside the range are not compiled.

optlevel=[noOpt | cold | warm | hot | veryHot | scorching]
Forces the JIT compiler to compile all methods at a specific optimization level. Specifying **optlevel** might have an unexpected effect on performance, including reduced overall performance.

verbose[={compileStart|compileEnd}]
Reports information about the JIT and AOT compiler configuration and method compilation.

The **={compileStart|compileEnd}** option reports when the JIT starts to compile a method, and when it ends.

vlog=<*filename*>
Sends verbose output to a file. If you do not specify this parameter, the output is sent to the standard error output stream (STDERR).

Related information:

Diagnosing a JIT or AOT problem

-Xnoaot

This option turns off the AOT compiler and disables the use of AOT-compiled code.

-Xnoaot
By default, the AOT compiler is enabled but is active only when shared classes are also enabled. Using this option does not affect the JIT compiler.

-Xnojit

This option turns off the JIT compiler.

-Xnojit
By default, the JIT compiler is enabled. This option does not affect the AOT compiler.

-Xquickstart

This option causes the JIT compiler to run with a subset of optimizations.

-Xquickstart
The effect is faster compilation times that improve startup time, but longer running applications might run slower. When the AOT compiler is active (both shared classes and AOT compilation enabled), **-Xquickstart** causes all methods to be AOT compiled. The AOT compilation improves the startup time of subsequent runs, but might reduce performance for longer running applications. **-Xquickstart** can degrade performance if it is used with long-running applications that contain hot methods. The implementation of **-Xquickstart** is subject to change in future releases. By default, **-Xquickstart** is disabled..

-XsamplingExpirationTime

Use this option to disable JIT sampling after a specified amount of time.

-XsamplingExpirationTime<*time*>
Disables the JIT sampling thread after <*time*> seconds. When the JIT sampling thread is disabled, no processor cycles are used by an idle JVM.

-Xscmaxaot

When you create a shared classes cache, you can use this option to apply a maximum number of bytes in the class cache that can be used for AOT data.

-Xscmaxaot<*size*>
This option is useful if you want a certain amount of cache space guaranteed for non-AOT data. If this option is not specified, by default the maximum limit for AOT data is the amount of free space in the cache. The value of this option must not be smaller than the value of **-Xscminaot** and must not be larger than the value of **-Xscmx**.

-Xscminaot

When you create a shared classes cache, you can use this option to apply a minimum number of bytes in the class cache to reserve for AOT data.

-Xscminaot<*size*>
If this option is not specified, no space is reserved for AOT data. However, AOT data is still 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** equals the value of **-Xscmx**, no class data or AOT data can be stored.

Garbage Collector command-line options

Use these Garbage Collector command-line options to control garbage collection.

You might need to read Memory management to understand some of the references that are given here.

The **-verbose:gc** option detailed in Verbose garbage collection logging is the main diagnostic aid that is available for runtime analysis of the Garbage Collector. However, additional command-line options are available that affect the behavior of the Garbage Collector and might aid diagnostic data collection.

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.

-Xalwaysclassgc

Always perform dynamic class unloading checks during global collection.

-Xalwaysclassgc
The default behavior is as defined by **-Xclassgc**.

-Xclassgc

Enables dynamic unloading of classes by the JVM. Garbage collection of class objects occurs only on class loader changes.

-Xclassgc
Dynamic unloading is the default behavior. To disable dynamic class unloading, use the **-Xnoclassgc** option.

-Xcompactexplicitgc

Enables full compaction each time System.gc() is called.

-Xcompactexplicitgc
Enables full compaction each time System.gc() is called.

-Xcompactgc

Compacts on all garbage collections (system and global).

-Xcompactgc
The default (no compaction option specified) makes the GC compact based on a series of triggers that attempt to compact only when it is beneficial to the future performance of the JVM.

-Xconcurrentbackground

Specifies the number of low-priority background threads attached to assist the mutator threads in concurrent mark.

-Xconcurrentbackground<*number*>
The default is 0 for Linux on z Systems™ and 1 on all other platforms.

-Xconcurrentlevel

Specifies the allocation "tax" rate.

-Xconcurrentlevel<*number*>
This option indicates the ratio between the amount of heap allocated and the amount of heap marked. The default is 8.

-Xconcurrentslack

Attempts to keep the specified amount of the heap space free in concurrent collectors by starting the concurrent operations earlier.

-Xconcurrentslack<*size*>
Using this option can sometimes alleviate pause time problems in concurrent

collectors at the cost of longer concurrent cycles, affecting total throughput. The default value is 0, which is optimal for most applications.

-Xconmeter

This option determines the usage of which area, LOA (Large Object Area) or SOA (Small Object Area), is metered and hence which allocations are taxed during concurrent mark.

-Xconmeter:<*soa* | *loa* | *dynamic*>

Using **-Xconmeter:soa** (the default) applies the allocation tax to allocations from the small object area (SOA). Using **-Xconmeter:loa** applies the allocation tax to allocations from the large object area (LOA). If **-Xconmeter:dynamic** is specified, the collector dynamically determines which area to meter based on which area is exhausted first, whether it is the SOA or the LOA.

-Xdisableexcessivegc

Disables the throwing of an OutOfMemory exception if excessive time is spent in the GC.

-Xdisableexcessivegc

Disables the throwing of an OutOfMemory exception if excessive time is spent in the GC.

-Xdisableexplicitgc

Disables System.gc() calls.

-Xdisableexplicitgc

Many applications still make an excessive number of explicit calls to System.gc() to request garbage collection. In many cases, these calls degrade performance through premature garbage collection and compactions. However, you cannot always remove the calls from the application.

The **-Xdisableexplicitgc** parameter allows the JVM to ignore these garbage collection suggestions. Typically, system administrators use this parameter in applications that show some benefit from its use.

By default, calls to System.gc() trigger a garbage collection.

-Xdisablestringconstantgc

Prevents strings in the string intern table from being collected.

-Xdisablestringconstantgc

Prevents strings in the string intern table from being collected.

-Xenableexcessivegc

If excessive time is spent in the GC, the option returns null for an allocate request and thus causes an OutOfMemory exception to be thrown.

-Xenableexcessivegc

The OutOfMemory exception is thrown only when the heap has been fully expanded and the time spent is making up at least 95%. This behavior is the default.

You can control the percentage that triggers an excessive GC event with the **-Xgc:excessiveGCratio** option. For more information, see "-Xgc" on page 104.

-Xenablestringconstantgc

Enables strings from the string intern table to be collected.

-Xenablestringconstantgc
This option is on by default.

-Xgc
Options that change the behavior of the Garbage Collector (GC). These options are deprecated.

-Xgc:<excessiveGCratio | verbose | compact | nocompact | scvNoAdaptiveTenure | scvTenureAge>

excessiveGCratio=*value*
Where *value* is a percentage. The default value is 95. This option can be used only when **-Xenableexcessivegc** is set. For more information, see "-Xenableexcessivegc" on page 103.

scvNoAdaptiveTenure
This option turns off the adaptive tenure age in the generational concurrent GC policy. The initial age that is set is maintained throughout the run time of the Java virtual machine. See **scvTenureAge**.

scvTenureAge=<*n***>**
This option sets the initial scavenger tenure age in the generational concurrent GC policy. The range is 1 - 14 and the default value is 10. For more information, see Tenure age.

Options **verbose**, **compact**, and **nocompact** are deprecated.

-Xgc:splitheap
Allocates the new and old areas of the generational Java heap in separate areas of memory.

Using a split heap forces the Garbage Collector to use the **gencon** policy and disables resizing of the new and old memory areas. See for more information.See Split heap for more information. By default, the Java heap is allocated in a single contiguous area of memory.

-Xgcpolicy
Controls the behavior of the Garbage Collector.

-Xgcpolicy:< gencon | optavgpause | optthruput | subpool (AIX, Linux and IBM i on IBM POWER® architecture, Linux and z/OS on zSeries) >

gencon
The generational concurrent (**gencon**) policy uses a concurrent mark phase combined with generational garbage collection to help minimize the time that is spent in any garbage collection pause. This policy is particularly useful for applications with many short-lived objects, such as transactional applications. Pause times can be significantly shorter than with the **optthruput** policy, while still producing good throughput. Heap fragmentation is also reduced.

optavgpause
The "optimize for pause time" (**optavgpause**) policy uses concurrent mark and concurrent sweep phases. Pause times are shorter than with **optthruput**, but application throughput is reduced because some garbage collection work is taking place while the application is running. Consider using this policy if you have a large heap size (available on 64-bit platforms), because this policy limits the effect of increasing heap size on the length of the garbage collection pause. However, if your application uses many short-lived objects, the **gencon** policy might produce better performance.

optthruput
The "optimize for throughput" (**optthruput**) policy (default) disables the concurrent mark phase. The application stops during global garbage collection, so long pauses can occur. This configuration is typically used for large-heap applications when high application throughput, rather than short garbage collection pauses, is the main performance goal. If your application cannot tolerate long garbage collection pauses, consider using another policy, such as **gencon**.

-Xgcthreads

Sets the number of threads that the Garbage Collector uses for parallel operations.

-Xgcthreads<*number*>
The total number of GC threads is composed of one application thread with the remainder being dedicated GC threads. By default, the number is set to *n-1*, where *n* is the number of reported CPUs. Where SMT or hyperthreading is in place, the number of reported CPUs is larger than the number of physical CPUs. Likewise, where virtualization is in place, the number of reported CPUs is the number of virtual CPUs assigned to the operating system. To set it to a different number, for example 4, use **-Xgcthreads4**. The minimum valid value is 1, which disables parallel operations, at the cost of performance. No advantage is gained if you increase the number of threads to more than the default setting.

On systems running multiple JVMs or in LPAR environments where multiple JVMs can share the same physical CPUs, you might want to restrict the number of GC threads used by each JVM. The restriction helps prevent the total number of parallel operation GC threads for all JVMs exceeding the number of physical CPUs present, when multiple JVMs perform garbage collection at the same time.

-Xgcworkpackets

Specifies the total number of work packets available in the global collector.

-Xgcworkpackets<*number*>
If you do not specify a value, the collector allocates a number of packets based on the maximum heap size.

-Xloa

This option enables the large object area (LOA).

-Xloa
By default, allocations are made in the small object area (SOA). If there is no room in the SOA, and an object is larger than 64KB, the object is allocated in the LOA.

By default, the LOA is enabled for all GC policies except for subpool, where the LOA is not available.

-Xloainitial

Specifies the initial percentage (between 0 and 0.95) of the current tenure space allocated to the large object area (LOA).

-Xloainitial<*percentage*>
The default value is 0.05, which is 5%.

-Xloamaximum

Specifies the maximum percentage (between 0 and 0.95) of the current tenure space allocated to the large object area (LOA).

-Xloamaximum<*percentage*>
The default value is 0.5, which is 50%.

-Xloaminimum

Specifies the minimum percentage (between 0 and 0.95) of the current tenure space allocated to the large object area (LOA).

-Xloaminimum<*percentage*>
The LOA does not shrink to less than this value. The default value is 0, which is 0%.

-Xmaxe

Sets the maximum amount by which the garbage collector expands the heap.

-Xmaxe<*size*>
Typically, the garbage collector expands the heap when the amount of free space falls to less than 30% (or by 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

Specifies the maximum percentage of heap that must be free after a garbage collection.

-Xmaxf<*percentage*>
If the free space exceeds this amount, the JVM tries to shrink the heap. The default value is 0.6 (60%).

-Xmaxt

Specifies the maximum percentage of time to be spent in Garbage Collection.

-Xmaxt<*percentage*>
If the percentage of time exceeds this value, the JVM tries to expand the heap. The default value is 13%.

-Xmca

Sets the expansion step for the memory allocated to store the RAM portion of loaded classes.

-Xmca<*size*>
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 determine the value that the VM is using. If the expansion step size you choose is too large, `OutOfMemoryError` is reported. The exact value of a "too large" expansion step size varies according to the platform and the specific machine configuration.

-Xmcrs

Sets an initial size for an area in memory that is reserved for compressed references within the lowest 4 GB memory area.

Native memory OutOfMemoryError exceptions might occur when using compressed references if the lowest 4 GB of address space becomes full, particularly when loading classes, starting threads, or using monitors. This option secures space for any native classes, monitors, and threads that are used by compressed references.

-Xmcrs<*mem_size*>
Where <*mem_size*> is the initial size. You can use the **-verbose:sizes** option to find out the value that is being used by the VM. If you are not using compressed references and this option is set, the option is ignored and the output of **-verbose:sizes** shows `-Xmcrs0`.

The following option sets an initial size of 200 MB for the memory area:
`-Xmcrs200M`

-Xmco
Sets the expansion step for the memory allocated to store the ROM portion of loaded classes.

-Xmco<*size*>
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 determine the value that the VM is using. If the expansion step size you choose is too large, `OutOfMemoryError` is reported. The exact value of a "too large" expansion step size varies according to the platform and the specific machine configuration.

-Xmine
Sets the minimum amount by which the Garbage Collector expands the heap.

-Xmine<*size*>
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
Specifies the minimum percentage of heap to remain free after a garbage collection.

-Xminf<*percentage*>
If the free space falls to less than this amount, the JVM attempts to expand the heap. The default value is 30%.

-Xmint
Specifies the minimum percentage of time to spend in Garbage Collection.

-Xmint<*percentage*>
If the percentage of time drops to less than this value, the JVM tries to shrink the heap. The default value is 5%.

-Xmn
Sets the initial and maximum size of the new area to the specified value when using **-Xgcpolicy:gencon**.

-Xmn<*size*>
Equivalent to setting both **-Xmns** and **-Xmnx**. If you set either **-Xmns** or **-Xmnx**, you cannot set **-Xmn**. If you try to set **-Xmn** with either **-Xmns** or **-Xmnx**, the VM does not start, returning an error. By default, **-Xmn** is not set. If the scavenger is disabled, this option is ignored.

-Xmns
Sets the initial size of the new area to the specified value when using **-Xgcpolicy:gencon**.

-Xmns<*size*>
By default, this option is set to 25% of the value of the **-Xms** option. This option returns an error if you try to use it with **-Xmn**. You can use the **-verbose:sizes** option to find out the values that the VM is currently using. If the scavenger is disabled, this option is ignored.

-Xmnx

Sets the maximum size of the new area to the specified value when using **-Xgcpolicy:gencon**.

-Xmnx<*size*>
By default, this option is set to 25% of the value of the **-Xmx** option. This option returns an error if you try to use it with **-Xmn**. You can use the **-verbose:sizes** option to find out the values that the VM is currently using. If the scavenger is disabled, this option is ignored.

-Xmo

Sets the initial and maximum size of the old (tenured) heap to the specified value when using **-Xgcpolicy:gencon**.

-Xmo<*size*>
Equivalent to setting both **-Xmos** and **-Xmox**. If you set either **-Xmos** or **-Xmox**, you cannot set **-Xmo**. If you try to set **-Xmo** with either **-Xmos** or **-Xmox**, the VM does not start, returning an error. By default, **-Xmo** is not set.

-Xmoi

Sets the amount the Java heap is incremented when using **-Xgcpolicy:gencon**.

-Xmoi<*size*>
If set to zero, no expansion is allowed. By default, the increment size is calculated on the expansion size, set by **-Xmine** and **-Xminf**.

-Xmos

Sets the initial size of the old (tenure) heap to the specified value when using **-Xgcpolicy:gencon**.

-Xmos<*size*>
By default, this option is set to 75% of the value of the **-Xms** option. This option returns an error if you try to use it with **-Xmo**. You can use the **-verbose:sizes** option to find out the values that the VM is currently using.

-Xmox

Sets the maximum size of the old (tenure) heap to the specified value when using **-Xgcpolicy:gencon**.

-Xmox<*size*>
By default, this option is set to the same value as the **-Xmx** option. This option returns an error if you try to use it with **-Xmo**. You can use the **-verbose:sizes** option to find out the values that the VM is currently using.

-Xmr

Sets the size of the Garbage Collection "remembered set".

-Xmr<*size*>
The Garbage Collection "remembered set" is a list of objects in the old (tenured) heap that have references to objects in the new area. By default, this option is set to 16 K.

-Xmrx

Sets the remembered maximum size setting.

-Xmrx<*size*>
Sets the remembered maximum size setting.

-Xms

Sets the initial Java heap size.

-Xms*size*
size can be specified in megabytes (m) or gigabytes (g). For example: **-Xms2g** sets an initial Java heap size of 2GB. The minimum size is 1 MB.

You can also use the **-Xmo** option.

If the scavenger is enabled, **-Xms** >= **-Xmn** + **-Xmo**.

If the scavenger is disabled, **-Xms** >= **-Xmo**.

Note: The **-Xmo** option is not supported by the balanced garbage collection policy.

-Xmx

Sets the maximum memory size for the application (**-Xmx** >= **-Xms**).

-Xmx*size*
size can be specified in megabytes (m) or gigabytes (g). For example: **-Xmx2g** sets a maximum heap size of 2GB.

For information about default values, see "Default settings for the JVM" on page 112.

If you are allocating the Java heap with large pages, read the information provided for the "-Xlp" on page 88 option.

Examples of the use of **-Xms** and **-Xmx**:

-Xms2m -Xmx64m
Heap starts at 2 MB and grows to a maximum of 64 MB.

-Xms100m -Xmx100m
Heap starts at 100 MB and never grows.

-Xms20m -Xmx1024m
Heap starts at 20 MB and grows to a maximum of 1 GB.

-Xms50m
Heap starts at 50 MB and grows to the default maximum.

-Xmx256m
Heap starts at default initial value and grows to a maximum of 256 MB.

If you exceed the limit set by the **-Xmx** option, the JVM generates an `OutofMemoryError`.

-Xnoclassgc

Disables class garbage collection.

-Xnoclassgc
This option switches off garbage collection of storage associated with Java technology classes that are no longer being used by the JVM. The default behavior is as defined by **-Xclassgc**. Enabling this option is not recommended except under the direction of the IBM support team. The reason is the option can cause unlimited native memory growth, leading to out-of-memory errors.

-Xnocompactexplicitgc

Disables compaction on System.gc() calls.

-Xnocompactexplicitgc
Compaction takes place on global garbage collections if you specify **-Xcompactgc** or if compaction triggers are met. By default, compaction is enabled on calls to System.gc().

-Xnocompactgc

Disables compaction on all garbage collections (system or global).

-Xnocompactgc
By default, compaction is enabled.

-Xnoloa

Prevents allocation of a large object area; all objects are allocated in the SOA.

-Xnoloa
See also **-Xloa**.

-Xnopartialcompactgc

Disables incremental compaction.

-Xnopartialcompactgc
See also **-Xpartialcompactgc**.

-Xpartialcompactgc

Enables incremental compaction.

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

-Xsoftmx

This option sets a "soft" maximum limit for the initial size of the Java heap.

-Xsoftmx<*size*>(AIX only)
Use the **-Xmx** option to set a "hard" limit for the maximum size of the heap. By default, **-Xsoftmx** is set to the same value as **-Xmx**. The value of **-Xms** must be less than, or equal to, the value of **-Xsoftmx**. See the introduction to this topic for more information about specifying <*size*> parameters.

You can set this option on the command line, then modify it at run time by using the MemoryMXBean.setMaxHeapSize() method in the com.ibm.lang.management API. By using this API, Java applications can dynamically monitor and adjust the heap size as required. This function can be useful in virtualized or cloud environments, for example, where the available memory might change dynamically to meet business needs. When you use the API, you must specify the value in bytes, such as `2147483648` instead of `2g`.

For example, you might set the initial heap size to 1 GB and the maximum heap size to 8 GB. You might set a smaller value, such as 2 GB, for **-Xsoftmx**, to limit the heap size that is used initially:

```
–Xms1g –Xsoftmx2g –Xmx8g
```

You can then use the com.ibm.lang.management API from within a Java application to increase the **-Xsoftmx** value during run time, as load increases. This change allows the application to use more memory than you specified initially.

-Xsoftrefthreshold

Sets the value used by the garbage collector to determine the number of garbage collections after which a soft reference is cleared if its referent has not been marked.

-Xsoftrefthreshold<*number*>

The default is 32, meaning that the soft reference is cleared after 32 * (percentage of free heap space) garbage collection cycles where its referent was not marked. For example, if **-Xsoftrefthreshold** is set to 32, and the heap is 50% free, soft references are cleared after 16 garbage collection cycles.

-Xtgc

Provides garbage collection tracing options.

-Xtgc:<*arguments*>

<*arguments*> is a comma-separated list containing one or more of the following arguments:

backtrace

Before a garbage collection, a single line is printed containing the name of the master thread for garbage collection, as well as the value of the osThread slot in the J9VMThread structure.

compaction

Prints extra information showing the relative time spent by threads in the “move” and “fixup” phases of compaction

concurrent

Prints extra information showing the activity of the concurrent mark background thread

dump

Prints a line of output for every free chunk of memory in the system, including "dark matter" (free chunks that are not on the free list for some reason, typically because they are too small). Each line contains the base address and the size in bytes of the chunk. If the chunk is followed in the heap by an object, the size and class name of the object is also printed. This argument has a similar effect to the **terse** argument.

freeList

Before a garbage collection, prints information about the free list and allocation statistics since the last garbage collection. Prints the number of items on the free list, including "deferred" entries (with the scavenger, the unused space is a deferred free list entry). For TLH and non-TLH allocations, prints the total number of allocations, the average allocation size, and the total number of bytes discarded during allocation. For non-TLH allocations, also included is the average number of entries that were searched before a sufficiently large entry was found.

parallel

Produces statistics on the activity of the parallel threads during the mark and sweep phases of a global garbage collection.

references

Prints extra information every time that a reference object is enqueued for finalization, showing the reference type, reference address, and referent address.

scavenger

Prints extra information after each scavenger collection. A histogram is produced showing the number of instances of each class, and their relative

ages, present in the survivor space. The information is obtained by performing a linear walk-through of the space.

terse
Dumps the contents of the entire heap before and after a garbage collection. For each object or free chunk in the heap, a line of trace output is produced. Each line contains the base address, "a" if it is an allocated object, and "f" if it is a free chunk, the size of the chunk in bytes, and, if it is an object, its class name.

-Xverbosegclog

Causes **-verbose:gc** output to be written to a specified file.

-Xverbosegclog[:<*file*>[,<X>,<Y>]]
If the file cannot be found, **-verbose:gc** tries to create the file, and then continues as normal if it is successful. If it cannot create the file (for example, if an invalid filename is passed into the command), it redirects the output to stderr.

If you specify <*X*> and <*Y*> the **-verbose:gc** output is redirected to X files, each containing Y GC cycles.

The dump agent tokens can be used in the filename. See Dump agent tokens for more information. If you do not specify <*file*>, `verbosegc.%Y%m%d.%H%M%S.%pid.txt` is used.

By default, no verbose GC logging occurs.

Default settings for the JVM

This appendix shows the default settings that the JVM uses. These settings affect how the JVM operates if you do not apply any changes to its environment. The tables show the JVM operation and the default setting.

These tables are a quick reference to the state of the JVM when it is first installed. The last column shows how the default setting can be changed:

c The setting is controlled by a command-line parameter only.

e The setting is controlled by an environment variable only.

ec The setting is controlled by a command-line parameter or an environment variable. The command-line parameter always takes precedence.

JVM setting	Default	Setting affected by
Javadump	Enabled	ec
Heapdump	Disabled	ec
System dump	Enabled	ec
Snap traces	Enabled	ec
Verbose output	Disabled	c
Boot classpath search	Disabled	c
JNI checks	Disabled	c
Remote debugging	Disabled	c
Strict conformance checks	Disabled	c
Quickstart	Disabled	c

JVM setting	Default	Setting affected by
Remote debug info server	Disabled	c
Reduced signaling	Disabled	c
Signal handler chaining	Enabled	c
Classpath	Not set	ec
Class data sharing	Disabled	c
Accessibility support	Enabled	e
JIT compiler	Enabled	ec
AOT compiler (AOT is not used by the JVM unless shared classes are also enabled)	Enabled	c
JIT debug options	Disabled	c
Java2D max size of fonts with algorithmic bold	14 point	e
Java2D use rendered bitmaps in scalable fonts	Enabled	e
Java2D freetype font rasterizing	Enabled	e
Java2D use AWT fonts	Disabled	e

JVM setting	AIX	IBM i	Linux	Windows	z/OS	Setting affected by
Default locale	None	None	None	N/A	None	e
Time to wait before starting plug-in	N/A	N/A	Zero	N/A	N/A	e
Temporary directory	`/tmp`	`/tmp`	`/tmp`	`c:\temp`	`/tmp`	e
Plug-in redirection	None	None	None	N/A	None	e
IM switching	Disabled	Disabled	Disabled	N/A	Disabled	e
IM modifiers	Disabled	Disabled	Disabled	N/A	Disabled	e
Thread model	N/A	N/A	N/A	N/A	Native	e
Initial stack size for Java Threads 32-bit. Use: **-Xiss<size>**	2 KB	2 KB	2 KB	2 KB	2 KB	c
Maximum stack size for Java Threads 32-bit. Use: **-Xss<size>**	256 KB	256 KB	256 KB	256 KB	256 KB	c
Stack size for OS Threads 32-bit. Use **-Xmso<size>**	256 KB	256 KB	256 KB	32 KB	256 KB	c
Initial stack size for Java Threads 64-bit. Use: **-Xiss<size>**	2 KB	N/A	2 KB	2 KB	2 KB	c
Maximum stack size for Java Threads 64-bit. Use: **-Xss<size>**	512 KB	N/A	512 KB	512 KB	512 KB	c
Stack size for OS Threads 64-bit. Use **-Xmso<size>**	256 KB	N/A	256 KB	256 KB	256 KB	c
Initial heap size. Use **-Xms<size>**	4 MB	4 MB	4 MB	4 MB	4 MB	c

JVM setting	AIX	IBM i	Linux	Windows	z/OS	Setting affected by
Maximum Java heap size. Use **-Xmx<size>**	Half the available memory with a minimum of 16 MB and a maximum of 512 MB	2 GB	Half the available memory with a minimum of 16 MB and a maximum of 512 MB	Half the real memory with a minimum of 16 MB and a maximum of 2 GB	Half the available memory with a minimum of 16 MB and a maximum of 512 MB	c

"Available memory" is defined as being the smallest of two values:

- The real or "physical" memory.
- The **RLIMIT_AS** value.

Known issues and limitations

Known issues or limitations that you might encounter in specific system environments, or configurations.

Font problems in supported locales

The release supports the following locales:

- Bengali (bn_IN)
- Malayalam (ml_IN)
- Oriya (or_IN)

However the fonts from these locales might not work on AWT components.

Use of sockets with IPv6

The release supports IPv6. However, because the current IPv6 support in Windows is not dual-stack, the release emulates dual-stack behavior on an IPv6 enabled system. Your Java technology application might use up to twice as many sockets because of the nature of the emulation. To disable this emulation, disable IPv6 support in the release by setting the system property **java.net.preferIPv4Stack** to `true`.

JConsole monitoring tool Local tab

In the IBM 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 start the application that you want to monitor, set these command-line options:

-Dcom.sun.management.jmxremote.port=<*value*>
Specifies the port the management agent listens on.

-Dcom.sun.management.jmxremote.authenticate=false
Disables authentication unless you have created a user name 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 in 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.

Web Start and Java technology 1.3 applications

The release version of Web Start does not support launching Java technology 1.3 applications.

Input Method Editor (IME)

When working with an Input Method Editor (IME), complete the character composition and select the candidate before using the workspace for any other operation.

When a user types text in an AWT TextArea while using an Input Method Editor (IME), and then resizes the application window *before* committing the text, the text is committed automatically.

Slow DSA key pair generation

Creating DSA key pairs of unusual lengths can take a significant amount of time on slow machines. Do not interpret the delay as a stop or endless loop, because the process finishes if sufficient time is allowed. The DSA key generation algorithm has been optimized to generate standard key lengths (for instance, 512, 1024) more quickly than others.

Personal firewalls

Personal firewalls can cause problems for the Windows NIO code, causing particular operations to fail. For example, the method call Selector.open() can throw a "java.io.IOException: Unable to establish loopback connection" with a cause of "java.net.ConnectException: Connection refused: connect". The exception is caused by the operating system connecting on a port that is being blocked by the firewall. The JVM tries the connect operation again, asking the operating system to select a different port number. If it still cannot connect after several attempts, a ConnectException is thrown.

If you see this exception, you can set the system property **java.nio.debug=pipe** to see which port numbers are being blocked.

File handle exhaustion

On Windows 2000 and XP, the default value of the number of files that you can have simultaneously opened is too low and causes problems to applications that are I/O intensive. To fix this limitation, edit the file *<windows>*\system32\CONFIG.NT and set the following values:

```
files=200
buffers=60
```

where <*windows*> is the directory where Windows is installed.

DirectDraw and mouse pointer problems

On Windows 2000, with a 32-bit color depth, the DirectDraw mechanism of the JVM does not repaint the region under the mouse pointer. The effect is that gray or black squares are displayed on menus after the mouse has been there. The workaround is either to switch off direct draw (**-Dsun.java2d.noddraw**), or to change your screen color resolution to some other value, such as 256 color.

NIO connection problems

The NIO SocketChannel finishConnect() method can return `true` (the channel is connected) or `false` (the connection process is not yet complete), or can throw an exception. On Windows 2000, when using non-blocking connections, `false` might be returned even after a previous select() call has implied that a channel is ready for processing.

The methods `setReadOnly()` and `setWritable(false)` do not work on Windows directories

From Version 6 Service Refresh 10, if you use these methods on a directory on the Windows operating system, they return the value `false.`

Note: In the same situation in earlier releases, these methods set the DOS read-only attribute to prevent the directory from being deleted. However, this behaviour does not make the directory read-only, therefore the only changes in behavior are that the methods now return the value `false`, and the read-only attribute is not set.

Stack range of the main thread

You cannot alter the stack range of the main thread (also known as the primordial thread) at run time. The main thread has a fixed size of 256 KB, determined as the optimum value for performance reasons. You can use the **-Xss** option to modify the stack range only on threads other than the main one. Do not use the main thread for heavily recursive calculations because the main thread is more prone to stack overflow than other threads.

DBCS characters

If you are typing DBCS characters in a JTextArea, JTextField, or JFileChooser, switching from some Chinese IMEs (in particular, Chinese Internal Code and Zhengma) to Intelligent ABC IME might cause a core dump to be produced.

Czech language installation

For Czech users, note that the language selection panel of the installation program offers one translated entry in an installation that is otherwise not translated. This limitation is caused by the installation program. The string is picked up from the operating system based on the code page. Because Polish (for which the installation is translated) and Czech both have code page 1250, the installation program attempts to retrieve a language list from the system for both languages, resulting in this string in the language list.

Traditional Chinese and the `more` command

If you use Traditional Chinese, do not pipe the output from your application directly into the **more** command. Instead, direct the output to a temporary file and view the file separately.

Accent corruption for Catalan users

For Catalan users of Windows 2000, use the Lucida Console font to avoid corruption of accented capital letters.

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.

Unicode Shift_JIS code page alias

Japanese users only:

The Unicode code page alias "\u30b7\u30d5\u30c8\u7b26\u53f7\u5316\u8868\u73fe" for Shift_JIS has been removed. If you use this code page in your applications, replace it with Shift_JIS.

`-Xshareclasses:<options>`

Shared classes cache and control files are not compatible between Version 6 SR 4 and previous releases.

Java Kernel installation

The kernel aims to reduce the startup time imposed by an application when it finds that the installed release needs an update. When this situation occurs, the kernel automatically downloads only the components that are needed directly from the Oracle Web site. Automated download is currently not possible with the IBM implementation of the Oracle update.

Java Deployment Toolkit

The toolkit implements the JavaScript **DeployJava.js**, which can be used to automatically generate any HTML needed to deploy applets and Java Web Start applications. However, the automatic generation is not possible with this release, because the process involves downloading and running the specific release from a public site, using public functions.

Next-Generation Java Plug-In Technology

There are some known limitations relating to Next-Generation Java Plug-In Technology:

1. When using the Next-Generation Plug-In, the Console option previously available in the Microsoft Internet Explorer tools menu is not available.
2. "Secure Static Versioning (SSV) support" on page 48 is not provided for Next-Generation plug-ins.

Expired GTE Cybertrust Certificate

The release contains an expired GTE CyberTrust Certificate in the `CACERTS` file for compatibility reasons. The `CACERTS` file is provided as a default truststore. Some common public certificates are provided as a convenience.

If no applications require the certificate, you can leave it in the `CACERTS` file. Alternatively, the certificate can be deleted. If applications do require the certificate, modify them to use the newer GTE CyberTrust Global root certificate that expires in 2018.

This certificate might be removed for later versions of the release.

Java Communications API (JavaComm) parallel port detection

The JavaComm API gives applications a platform-independent way of performing serial and parallel port communications for technologies such as voice mail, fax, and smart cards. On Microsoft Windows 7 and Microsoft Windows Server 2008 R2, the parallel port is not detected by the operating system. This means that the JavaComm API for parallel ports cannot be used on these operating systems. However, the serial port can be used for communication.

Release installation on Windows 7 on Intel 32-bit architecture

This limitation applies only to Version 6 SR6.

The first installation and uninstallation of the release is successful. However, repeat installation and uninstallation cycles might generate one or more of the following error messages during installation or uninstallation:

```
"Error 1316. A network error
                  occurred while reading the file. F:\Users\ADMINI~1
                  \AppData\Local\Temp\
                  {DEF2FE6F-D233-45AD-94F4-3D050F1685D3}\IBM 32-bit SDK for
                  Java v6 .msi"
msci missing
msi file missing
```

This problem applies to the Development Kit only.

Using Web Start to launch a JNLP application

When using Web Start to launch a Java Network Launching Protocol (JNLP) application that requires an older version of Java technology, you might see an error containing the following message:

```
java.lang.NoClassDefFoundError: com/sun/deploy/util/BlackList
```

This results from a check controlled by the deployment configuration property **`deployment.security.blacklist.check`** The property is enabled using the `Enable blacklist revocation check` option in the Java Control Panel.

To work around the problem:

1. Launch the Java Control Panel.
2. Select **Advanced tab** > **Security**.
3. Clear the `Enable blacklist revocation check` option.

Using -Xshareclasses:destroy during JVM startup

When running the command **java -Xshareclasses:destroy** on a shared cache that is being used by a second JVM during startup, you might have the following issues:

- The second JVM fails.
- The shared cache is deleted.

Windows 7 requires approval to run ssvagent.exe when Next-Generation Java Plug-In is disabled

If Next-Generation Java Plug-in is disabled on Windows 7, Windows Vista or Windows Server 2008, a UAC (User Account Control) dialog requesting user consent to run `ssvagent.exe` might be seen when Internet Explorer is launched. This behavior is enforced by the operating system when the user is not logged on as Administrator, or does not run Internet Explorer with Administrator privileges. You can give consent to run this executable file.

Problems accessing archive files created using java.util.Zip*

The `java.util.Zip*` files can create archive files that are larger than 4 GB. However, some third-party compression tools have file size limitations, and cannot access files larger than 4 GB.

Chinese characters stored as ? in an Oracle database

When you configure an Oracle database to use the ZHS16GBK character set, some Chinese characters or symbols that are encoded with the GBK character set are incorrectly stored as a question mark (?). This problem is caused by an incompatibility of the GBK undefined code range Unicode mapping between the Oracle ZHS16GBK character set and the IBM GBK converter. To fix this problem, use a new code page, MS936A, by including the following system property when you start the JVM:

```
-Dfile.encoding=MS936A
```

For IBM WebSphere Application Server users, this problem might occur when web applications that use JDBC configure Oracle as the WebSphere Application Server data source. To fix this problem, use a new code page, MS936A, as follows:

1. Use the following system property when you start the JVM:

   ```
   -Dfile.encoding=MS936A
   ```

2. Add the following lines to the *WAS_HOME*`/properties/converter.properties` file, where *WAS_HOME* is your WebSphere Application Server installation directory.

   ```
   GBK=MS936A
   GB2312=MS936A
   ```

Issues with the XL TXE-J XSLT compiler

A low split limit might cause compilation errors.

Avoid calling Java technology extension functions that have side effects because the order of execution is not guaranteed.

Versions of Ant before 1.7.0 do not work with the XL TXE-J compiler. Instead, use the XSLT4J interpreter by running the release with the following system property:

- `-Djavax.xml.transform.TransformerFactory=org.apache.xalan.processor.TransformerFactoryImpl.`

Large page request fails

There is no error message issued when the JVM is unable to honor the **-Xlp** request.

There are a number of reasons why the JVM cannot honor a large page request. For example, there might be insufficient large pages available on the system at the time of the request. To check whether the **-Xlp** request was honored, you can review the output from **-verbose:gc**. Look for the attributes `requestedPageSize` and `pageSize` in the **-verbose:gc** log file. The attribute `requestedPageSize` contains the value specified by **-Xlp**. The attribute `pageSize` is the actual page size used by the JVM.

Unexpected `CertificateException`

Version 6 Service Refresh 13 fix pack 1 and later releases contain a security enhancement to correctly validate certificates on jar files of applications. After upgrading, a `CertificateException` occurs for any applications in one of the following scenarios:

- The application jar is not properly signed.
- The application jar has incorrect certificates.
- A certificate in the certificate chain is revoked.

To avoid these exceptions, make sure that your application jars are signed with valid certificates before you upgrade from an earlier release. This issue relates to APAR IV38456.

Unexpected application errors with RMI

If your application uses RMI and you experience unexpected errors after updating to Version 6 Service Refresh 13 fix pack 2, or later releases, the problem might be associated with a change to the default value of the RMI property java.rmi.server.useCodebaseOnly. For more information, see http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/enhancements-7.html.

Startup issues on applications that contain spaces in the program name

After installing Version 6 Service Refresh 13 fix pack 2 or later releases, applications that contain spaces in the program name, or that use quotation marks incorrectly, might fail to start. This issue might be caused by improvements to the way Runtime.exec decodes command strings. For more information, including guidance on resolving problems, see http://www.oracle.com/technetwork/java/javase/7u21-relnotes-1932873.html#jaruntime.

Unexpected XSLT error on extension elements or extension functions when security is enabled

From Version 6 Service Refresh 14, any attempt to use extension elements or extension functions when security is enabled, results in a `javax.xml.transform.TransformerException` error during XSLT processing. This change in behavior is introduced to enhance security.

The following XSLT message is generated when extension functions are used: `Use of the extension function '<method name>' is not allowed when security is enabled. To override this, set the com.ibm.xtq.processor.overrideSecureProcessing property to true. This override only affects XSLT processing.`

The following XSLT message is generated when extension elements are used: `Use of the extension element '<element name>' is not allowed when security is enabled. To override this, set the com.ibm.xtq.processor.overrideSecureProcessing property to true. This override only affects XSLT processing.`

To allow extensions when security is enabled, set the **com.ibm.xtq.processor.overrideSecureProcessing** system property to *true*. For more information about this system property, see "-Dcom.ibm.xtq.processor.overrideSecureProcessing" on page 75.

Incorrect value for Windows 8.1 and Windows 10 in the `java -version` output

The executable files in this release do not contain the manifest information that is required to properly display the Windows version information in the output from the **java -version** command. Windows 8.1 and Windows 10 are incorrectly reported as Windows 8.

Support for virtualization software

This release is tested with a number of virtualized server products.

This release has been tested with the following virtualization software:

Table 7. Virtualization software tested

Vendor	Architecture	Server virtualization	Version
IBM	z Systems	PR/SM™	z13, z10™, z11, z196, zEC12
IBM	z Systems	z/VM®	6.1, 6.2
IBM	z Systems	KVM for IBM z Systems	1.1.0
IBM	POWER	PowerVM® Hypervisor	Power® 6, Power 7, Power 8
VMware	x86-64	VMware ESX and ESXi Server	4.1, 5.0
Red Hat	x86-64	Red Hat Enterprise Virtualization (RHEV)	2.1, 3.0
SUSE	x86-64	SUSE KVM	SLES 11
Microsoft	x86-64	Hyper-V	Server 2012
Docker, Inc	x86-64	Docker	V1.6 or later (see note)

Note: IBM supports all versions of the SDK that run in Docker containers, provided that the Docker images are based on supported operating systems. To find out which operating systems are supported for the IBM SDK, see

http://www.ibm.com/support/knowledgecenter/SSYKE2_6.0.0/com.ibm.java.doc.user.lnx.60/user/supported_env.html .

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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 described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

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

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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 jurisdictions 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 publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide 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:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

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.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

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.

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year).
Portions of this code are derived from IBM Corp. Sample Programs.
© Copyright IBM Corp. _enter the year or years_.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Intel, Intel logo, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see: (i) IBM’s Privacy Policy at http://www.ibm.com/privacy ; (ii) IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details (in particular, the section entitled “Cookies, Web Beacons and Other Technologies”); and (iii) the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Printed in USA