IBM WebSphere Everyplace Micro Environment Runtime for Windows Mobile 5.0 CDC 1.1/ Foundation 1.1/Personal Profile 1.1

WebSphere® software

Installation Guide

Version 6.1

Note: Before using this information and the product it supports, read the general information in the Notices section of this guide.

First Edition (March 2006)

This edition applies to version 6.1 of the IBM WebSphere Everyplace Micro Environment for Windows Mobile 5.0 CDC 1.1/Foundation 1.1/ Personal Profile 1.1 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Preface

IBM® WebSphere® Everyplace® Micro Environment (WEME) is a Java™ 2 Micro Edition (J2ME™) certified "Java Powered" runtime environment that provides the foundation for the deployment of applications to a variety of mobile devices. This release optimizes the WEME runtime environment for Windows Mobile™ 5.0 operating systems.

This guide explains how to deploy this WEME release to Windows Mobile 5.0 platforms.

Who should read this guide

This guide is intended for application programmers developing embedded applications for this product.

Readers should be familiar with the following:

- Windows Mobile 5.0: ActiveSync® and File Explorer
- Java related concepts, terminology, and programming fundamentals

What this guide contains

This book contains the following sections:

- **Introducing Websphere Everyplace Micro Environment 6.1** — Introduces WEME and the J9 Virtual Machine (VM). It also describes the release package contents and system requirements necessary for a successful product installation.
- **Deploying J9 to Windows Mobile 5.0** — Describes how to deploy to Windows Mobile 5.0 target systems.
- **Running a demo application** — Describes how to run a demo application.
- **J9 runtime files** — Provides a list of files in the J9 runtime and describes what they are used for.
- **J9 command options** — Provides a list of J9 command line options.

Contacting software support

Before contacting IBM Software Support with a problem, refer to the IBM WEME Software Support site at the following Web site:
http://www.ibm.com/software/wireless/weme/support.html

On this Web site, you can search for technical notes, white papers and other content related to the IBM WEME. For additional help, contact software support by using the methods described in the *IBM Software Support Guide* at the following Web site:
http://techsupport.services.ibm.com/guides/handbook.html

The guide provides the following information:

- Registration and eligibility requirements for receiving support
- Telephone numbers, depending on the country in which you are located

Conventions used in this guide

The following typeface conventions are used in this guide:

Bold	Lowercase commands or mixed case commands that are difficult to distinguish from surrounding text, keywords, parameters, options, names of Java classes, and objects are in bold.
Italic	Variables, titles of publications, and special words or phrases that are emphasized are in italic.
`Monospace`	Code examples, command lines, screen output, file and directory names that are difficult to distinguish from surrounding text, system messages, text that the user must type, and values for arguments or command options are in monospace.

Introducing WebSphere Everyplace Micro Environment 6.1

IBM WebSphere Everyplace Micro Environment (WEME) 6.1 provides the underlying platform for the deployment of e-business applications to small mobile devices.

The J9 VM is the core of WEME. It is the IBM implementation of the Java Virtual Machine (JVM) Specification, Version 1.4. For more on this Java VM Specification, see the following Web site: http://java.sun.com/docs/books/vmspec/

The J9 runtime environment consists of the J9 VM and Java Class Libraries (JCL). It is J2ME compliant and contains Connected Device Configuration (CDC) 1.1, Foundation 1.1 and Personal Profile 1.1 based technologies. The WEME product is supported on a variety of operating systems and hardware architectures. This document covers WEME 6.1 for Windows Mobile 5.0. For information on other platforms contact your IBM Sales Representative.

System requirements

This section lists the minimum product levels you should have installed.

Host system requirements:

- x86-architecture based system running Windows XP service pack 2
- ActiveSync 4.0

Target system requirements:

PDA device running Windows Mobile 5.0

Package contents

This release of WEME is available from the IBM Workplace Client Technology, Micro Edition Web site at: http://www.ibm.com/software/wireless/weme/

This package includes the following files:

Runtime files

- `\bin`
  Includes J9 programs and shared libraries

  Note: These files are listed and described in the **J9 VM and the J9 JCL files** section of this document.

Classes and resources

- `\lib`
  Includes `charconv.zip`

- `\lib\jclFoundation11`
  Includes `classes.zip` and `locale.zip`.

- `\lib\jclFoundation11\ext`
  Includes `j9jce.jar and j9jsse.jar`

- `\lib\jclFoundation11\opt-ext`
  Includes `j9jceprov.jar`

- `\lib\jclFoundation11\ppro11`
  Includes `ppro-ui.jar and ppro-extras.jar`

- `\lib\security`
  Includes `cacerts, java.policy and java.security`

- `\examples`
  Includes `GolfScoreTrackerApp.jar` and `GolfScoreTrackerApp.lnk`. These files are needed to launch the GolfScoreTrackerApp example.

- `\doc`
  Includes this platform installation guide

Java source

- `\lib`
  Includes `charconv-src.zip`

- `\lib\jclFoundation11\source`
  Includes `source.zip` and `local-src.zip`

- `\lib\jclFoundation11\ppro11\source`
  Includes `ppro11-src.zip` and `ppro-ui-src.zip`

- `\licenses`
  Includes licenses, notices and associated files

Deploying J9 to Windows Mobile 5.0

Use the following steps to deploy this WEME Runtime for Windows Mobile 5.0 CDC 1.1/Foundation 1.1/Personal Profile 1.1 on your target device.

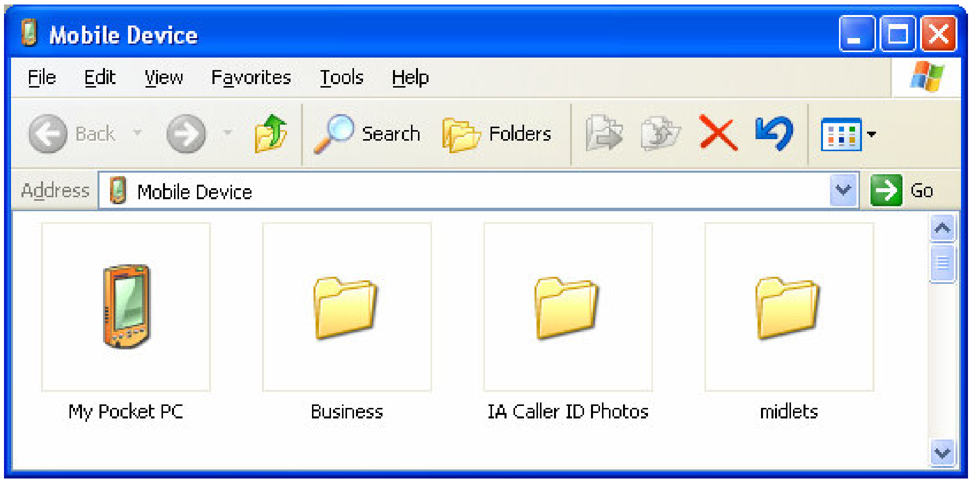
1. Download the product Installer to the Host PC from the following Web site: http://www.ibm.com/software/wireless/weme/.

   Product installer for Windows Mobile 5.0 CDC 1.1/FOUNDATION 1.1/PPRO 1.1: ibm-weme-wm50-arm-ppro11-6.1.0- YYYYMMDD-######-###.exe

   Note: For Linux x86 host platforms, download the corresponding `.bin` file.

2. Use the **Extraction Wizard** to extract the files. Extract the files to the default location indicated.

3. Select the Explore icon in **ActiveSync** to open a **File Explorer** window for your mobile device.

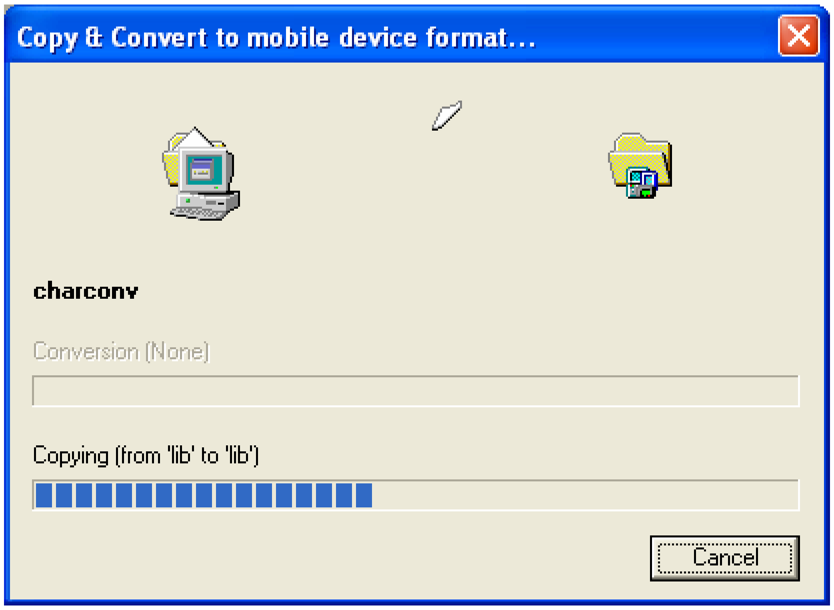


4. Use **File Explorer** to go to the `root` folder on your mobile device and create a new folder named `J9`.

5. Open the `J9` folder and create a `PPRO11` folder.

6. Copy the previously unzipped `bin`, `lib` and `examples` directories to the `PPRO11` folder.

   Note: These directories can be copied to any location you choose on the target file system; however, the `\bin` and `\lib` directories must remain intact. The base J9 executable and its shared objects must remain in the `\bin` directory.



charconv

Conversion (None)

Copying (from 'lib' to 'lib')

Running a demo application

A demo application is included with this product. The LNK and JAR files needed to run this application are located in the following directory:

```
%JAVAHOME%\examples\GolfScoreTrackerApp.lnk
%JAVAHOME%\examples\GolfScoreTrackerApp.jar
```

Note: `%JAVAHOME%` is the location of the J9 VM installation.

You can run this demo from the `\examples` directory on the PDA target file system.

1. In **File Explorer**, go to the `%JAVAHOME%\examples` folder and tap the `GolfScoreTrackerApp.lnk` file. An application window then appears.

J9 VM and J9 JCL files

The J9 VM and the J9 JCL come with a variety of programs and shared libraries. Not all of them are required at runtime in the target. The related files are located in the `%JAVAHOME%\bin` directory. Before loading the runtime files onto the target, you can remove some files from the runtime image to minimize its size. This section organizes these files into three categories:

- Files required on a target
- Optional files on a target
- Development tools

Files required on a target

This section specifies the absolute minimum set of files that must be available on a running target.

`iveppro11.dll`	Natives for the Personal Profile (PPro) 1.1 UI classes.		
`j9vmall23.dll`	All Core DLL files combined into one DLL file for platforms with a limited number of open shared libraries, like the Pocket PC. It includes:		
		`iverel23.dll`	Specifies JXE support, including the relocator and JXE files class loader natives.
		`j9bvc23.dll`	Enables the byte code verifier required when J9 is started with the (default) **–verify** option. Use **–noverify** when removing this file.
		`j9dyn23.dll`	Specifies the dynamic class loader for class files loaded from directories, JAR files or JXE files.
		`j9gc23.dll`	Specifies the Garbage Collector.
		`j9hookable23.dll`	Specifies the link library for hookable components, such as the garbage collector event handling, Just in Time (JIT) compiler, Ahead of Time (AOT) runtime, the debugger, Java Virtual Machine Profiler Interface (JVMPI), MicroAnalyzer, SmartLinker profiler, and verbose output.
		`j9prt23.dll`	Specifies the J9 port library, containing target operating system dependent

	code.
`j9thr23.dll`	Defines support for implementing Java threads in either native OS threads or green threads.
`j9vm23.dll`	Specifies the J9 VM, including implementations of the Java byte codes.
`j9zlib23.dll`	Specifies the zlib data compression library (`(C) 1995-2002 Jean-loup Gailly and Mark Adler)`. This file is required when using compressed JAR files.

`jclfoun11_23.dll`	Natives for the JCL. The JCL natives are provided in one file specific to the JCL (Foundation 1.1) used.

Note: At least one of the following listed launcher programs is required.

`j9.exe`	The J9 VM program.
`j9w.exe`	J9 VM program. Opens J9 VM without a console.

Optional files on a target

This section specifies files that are needed only when you want to enable related J9 options or features.

`j9dbg23.dll`	Defines support for debugging the target. This is required when J9 is started with the **–debug** option. This file requires either `j9dbgi23.dll` or `j9rdbi23.dll`.
`j9fdm23.dll`	Specifies the Freely Distributable LIBM (fdlibm), which is a C math library for machines that support IEEE 754 floating-point arithmetic. It provides the natives required by the `java.lang.StrictMath` class and is only required if this class is used. **Note**: only double precision is supported.
`j9gcchk23.dll`	Concurrent GC Reliability, Availability, Serviceability (RAS) checks. To reproduce GC problems activate with: `-Xrunj9gcchk23[:options]` `To see options run: –Xrunj9gcchk23:help`.
`j9jar2jxe23.dll`	Provides C-calling API for creating JXE files on-the-fly.

`j9jit23.dll`	Just-in-time (JIT) compiler. JIT is enabled if this file is found. JIT is enabled with the **–Xjit** option and disabled with either the **–Xint** or the **–Xnojit** option.
`j9jitd23.dll`	Problem Determination Library for JIT. Used for: regex, limitfile, limit and exclude option processing, JIT log files, option printing, verbose options and internal data structure checking.
`j9jpi23.dll`	Specifies JVMPI, which is required when J9 is started with the **–jvmpi** option.
`j9prf23.dll`	Defines support for micro analysis (profiling) of the target. This is required when J9 is started with the **–analyze** option.
`j9rdbi23.dll`	Specifies access to the remote debug server, which is required when J9 is started with the following options: **–debug**:*options* **–Xrdbginfo**:*host*:*port*
`j9vrb23.dll`	Specifies verbose output, which is required if J9 is started with the **–verbose** option.
`jnichk.dll`	Provides additional checking on JNI functions. Enabled with the following option: `–Xrunjnichk[:verbose,help,profile[=<file>]]` **Note:** This capability is useful when developing, but typically removed at runtime.
`slprof.dll`	Supports SmartLinker profiling of the target. Required by the class: com`.ibm.ive.slprofiler.runtime.TargetProfiling` in `profile.jar`. Requires `j9jpi23.dll`
`java.properties`	Contains all externalized text messages, such as help text and error messages for the default language (English). If this file is missing, numeric error codes are printed.

Development tools

This section describes development tool files that are required only while creating J9 applications on a development machine. These files are not required on the target at runtime.

`j9dbgserv`	Starts the Debug Info Server. On an resource limited target debug information is stored external to the target in the debug info server. The VM communicates with the debug info server, transparent to the tooling. Start the debug info server with: `j9dbgserv –port:<port> –symfiles:<symfile path>.`
`jar2jxe`	Converts JAR files to JXE files.

J9 command options

This section discusses both common and advanced J9 v2.3 command line options.

Note: The Pocket PC platform requires that you use fully qualified path names. Relative path names will not work.

Common options

This section contains common J9 v2.3 command line options.

Syntax:

j9 -jcl:ppro11 [*options*…] **–classpath** [*options*] *classname* [*arguments*…]

j9 -jcl:ppro11 [*options*] **–jxe**:*jxe_file* [*arguments*…]

–? or **–help**	Displays help for J9 standard command options.
–classpath *path* **–cp** *path*	Either **–classpath** *path* or **–cp** *path* can be used to set a class path for this invocation of J9. The final value of **–classpath** is determined as follows: • If the **–classpath** option is set, its value is used. • If the **–classpath** option is not set, and the CLASSPATH environment variable is set, its value is used. • If neither of the preceding are set, the current directory `(.)` is used. If the class path includes: • Multiple class path entries, separate them with a semicolon ";" • A JAR, ZIP or JXE file, add the full name of the file to the class path • CLASS files, specify the top-level directory of the CLASS file tree Example: `–classpath \myclasses;\myjars\foo.jar` **CAUTION**: The J9 class libraries and the J9 VM are not compatible with other vendors' class libraries. Because it is possible to have more than one runtime environment installed on your host computer, make sure that you do not mismatch

	these libraries when specifying the class path. In particular, if your CLASSPATH environment variable is set, ensure that other vendor's libraries are not on it. **Note:** java and javax class packages must be on **–Xbootclasspath**, not **–classpath**.
–jxe:*jxe_file*	Reads the specified JXE file, searching for the classes in this file. All classes found in the JXE file are placed at the end of the boot path. For example: `–jxe:hello.jxe` **Note:** When using the **–jxe:** option, do not specify the startup class. Specify the **–jxe:** option as the last option on the command line. **Note**: It is recommended that you use **–classpath** (where applicable) or **–Xbootclasspath** if the JXE file contains boot classes. See the **–Xbootclasspath** option for details.
–D*prop*=*value*	Sets the value of a system property. Example: `–Dmy.property=some.value` Sets the value of `my.property` to `some.value`. If no value is given, **–D***prop* sets the value to `null`. To set values for multiple system properties, repeat the option statement, using a space to separate statements. Example: `j9 –Dprop1=val1 –Dprop2=val2 –Dprop3=val3` **Note:** Spacing is important in this option's syntax. There is no space between the initial **–D**, its property argument, the equal sign, or the *value* argument. Example: `–Dname=John_Smith` **Note:** If *value* contains spaces, enclose the option in double quotes. Example: `"–Dmy.property=value with space"`
–debug:*options*	Enables debug, Java Debug Wire Protocol (JDWP) standard *options*.

–jcl:*config*[*: extralib1 : extralib2*]

When necessary, this command option is used to specify which JCL shared library will provide JNI natives for the class library Java code. For example, if both Foundation 1.1 and PPro 1.1 class libraries are available, to use the PPro 1.1 class libraries you must specify `-jcl:ppro11`.

Note: This option is not required when using the Foundation class library because Foundation 1.1, `-jcl:foun11` is the default.

With J9 2.3 the **–jcl** command line option is extended with the *extralib* parameters. The extralib parameters are used to specify directories off of the *core* library. Anything in the specified directory is added to the bootstrap classpath.

For example: `-jcl:foun11:ppro11`

Loads the JCL Foundation 1.1 library as normal and also includes `jclFoundation11/ppro11/*` on the bootstrap classpath.

A new system property set by the VM, `com.ibm.util.extralibs.properties` contains the value of the extra parameters, in this case: `com.ibm.util.extralibs.properties="ppro11".`

If the **–jcl** option is used without indicating a **–Xbootclasspath:***path*, the value for *path* is assumed to be `$JAVAHOME/lib/jclLibraryName/classes.zip`. However, if the class libraries are stored in a non-default location, you must include the **–Xbootclasspath:***path* to direct the VM to the `classes.zip` file. See the **–Xbootclasspath** option for details.

Note: If the **–Xbootclasspath** and the **`-jcl`** VM options are mismatched, the VM generates an `Incompatible class library` error.

–verbose[:**class, gc, stack, sizes**]

Enables verbose output. Parameters are as follows:

- **class** displays each fully-qualified class name as it is loaded (that is, enable verbose class loading). This is the default value.
- **gc** displays garbage collection information.
- **stack** displays stack information.
- **sizes** displays default VM sizes.

–verify	Enables class file (byte code) verification. **Note:** The **–verify** option is true by default. To disable byte code verification, specify **–noverify**.
–version	Each VM build is identified by a version string of the form: `YYYYMMDD_#####_flags` Example: `20051103_03795_lEdFGq` The first 8 digits indicate the date the VM was built on. The next 5 digits indicate the build ID. The flags indicate the configuration: 1st letter: l: little endian b: big endian L: 64-bit little endian B: 64-bit big endian 2nd letter: E: emulated FPU H: hardware FPU 3rd letter: s: static linkage d: dynamic linkage 4th letter: C: CLDC F: Foundation S: J2SE 5th letter: M: Desktop GC m: Tiny GC G: Embedded 6th letter: i: no JIT a: AOT only r: large JIT q: small JIT V: Micro JIT R: large JIT + MicroJIT Q: small JIT + MicroJIT A: AOT + MicroJIT
–X	Prints help for non-standard (advanced) options.

Advanced options

The following advanced J9 v2.3 command line options are non-standard and subject to change without notice.

–Xbootclasspath:*path*	Sets the bootstrap class path to *path*. For example: `-jcl:ppro11 –Xbootclasspath:%JAVAHOME%\lib\ jclFoundation11\classes.zip` **Note:** When using this command line option, the **–jcl:LibraryName** option must be used, as shown in the above example, to indicate which class library natives the application should use.
–Xbootclasspath/p:*path*	Prepends the classes in *path* to the bootstrap class path. This option is useful for applying temporary fixes and/or adding to the bootstrap class path.
–Xbootclasspath/a:*path*	Appends the classes in *path* to the bootstrap class path. This option is useful for applying temporary fixes to application classes and/or adding to the bootstrap classpath.
–Xdbg:*options*	Enables standard Java Debug Wire Protocol (JDWP) debug options.
–Xdbginfo:*symbol_file_path*	Enables the debug info server.
–Xrdbginfo:*host*:*port*	Enables the remote debug info server.
–Xrunjdwp:*options*	Enables standard JDWP debug options. **Note:** Starts a JDWP server.
–Xfuture	Turns on strict class-file format checks. These checks enforce closer conformance to the class-file format specification.
–Xss*x*	Sets the maximum Java thread stack size to *x.*
–Xmso*x*	Sets the operating system thread stack size to *x.*
–Xint	Run interpreted only.

–Xjit[:**count**=*x*, **code**=*x*]

With no parameters, **–Xjit** enables the JIT. Useful parameters are:

count=*x*, where *x* is the upper limit of the number of times a method is invoked before it is compiled. **Example:** `–Xjit:count=0`, forces the JIT to compile everything on first execution.

code=*x*, where *x* sets the size of the JIT code cache, in kilobytes. **Example:** `–Xjit:code=1024`, sets the size of the JIT code cache to 1MB. The code cache will grow dynamically if required.

–Xoptionsfile=*filename* **StartupClass [arguments..]**

J9 VM Version 2.3 on all platforms supports an option file for the purpose of reducing the length of the command line.

Example: `%JAVAHOME%\bin\j9 –Xoptionsfile= vm.options com.ibm.myapps.MyApp1`

- An option file is a text file with one option per line.
- Lines starting with `#` are ignored and can be used for comments.
- The `\` character can be used as a continuation so that a single option can span multiple lines.
- The following command line options must be converted into their J9 internal form when used in the options file:

Command line option	**J9 internal form**
–analyze	–Xanalyze:NULL
–analyze:	–Xanalyze:
–classpath *path*	–Djava.class.path=*path*
–dbginfo:	–Xdbginfo:
–debug:	–Xdbg:
–jcl:*config*	–Xjcl:jcl*config*_23
–noverify	–Xverify:none
–rdbginfo:	–Xrdbginfo:
–verify	–Xverify
–verify:	–Xverify:
–Xrunjdwp:	–Xdbg:

- The following options must be entered on the command line (they are ignored when listed in the options file):

–jar
–jxe
–jxe:
–jxespace:
–Xoptionsfile=
Note: Embedded options files are not supported.

- Undocumented options are ignored when listed in the options file. To be used, they must be added to the command line.

- All other options, such as **–D**, **–Xint**, **–Xmx** etc. are the same on the command line and in the options file.

Environment variables are not supported in the options file. For example the following works on the command line but not in the options file : –`Dmy.property=some.value`

- Options listed in the option file override options on the command line regardless of their position.

Example: `vm.options` file:

```
-Xint
-Xanalyze:st=true,ia=192.168.1.100,ms=100000
-Djava.security.manager
-Djava.security.policy=my.policy
#-Djava.class.path=my.jar
```

–Xrun*dll*[:*options*]	Loads helper libraries, such as those used with JVMPI.

J9 V2.3 GC command line options

The Garbage Collector for IBM WEME Runtime for Windows Mobile 5.0 CDC 1.1/Foundation 1.1/PPro 1.1 provides global GC with compaction. The following related command line options are provided:

Option parameter key

x	integer value in bytes, or append with ‘k’ or ‘M’ for large values
percentage	integer value in the range of 0--100 (inclusive)
age	integer value in the range of 1--14 (inclusive)
time	integer value (in milliseconds)

–Xmx*x*	Sets memory object heap memory size to *x*. **Xmx** >= **NewSpace** size plus **OldSpace** size Scavenger enabled: minimum size 1536 bytes on 32-bit architectures, 6072 bytes on 64-bit architectures Scavenger disabled: minimum size 512 bytes on 32-bit architectures, 2048 bytes on 64-bit architectures
–Xms*x*	Sets the initial memory size to *x.* scavenger enabled: minimum size 4 kilobytes on 32-bit architectures, 8 kilobytes on 64-bit architectures **Xms** >= **Xmn** + **Xmo** scavenger disabled: minimum size 4 kilobytes on 32-bit architectures, 8 kilobytes on 64-bit architectures **Xms** >= **Xmos**
–Xmos*x*	Sets the initial **OldSpace** size to *x.* **Note:** Minimum size 512 bytes on 32-bit architectures, 2048 bytes on 64-bit architectures
–Xmox*x*	Sets the maximum **OldSpace** size to *x.*
–Xmo*x*	Sets the initial and maximum **OldSpace** size to *x.* **Note:** Attempts to set **–Xmo** and **–Xmos**, or **–Xmo** and **–Xmox** are rejected

–Xmca*x*	Sets the RAM class segment increment to *x.* RAM class segments contain the portion of the Java classes that needs to be modified at runtime, such as the pointers to the class loader, super classes, implemented interfaces, statics, first instance and so forth. There is at least one RAM class segment per Class Loader. If more space for the RAM classes is needed, the J9 VM allocates a new segment in the same size.
–Xmcox	Sets the ROM class segment increment to *x*. The code of Java classes loaded from class files is stored in special ROM classes segment types called Dynamically Loaded Classes, whose size is determined by this parameter. There is at least one ROM class segment per class loader. If more space for the dynamically loaded classes is needed, the J9 VM allocates a new segment in the size determined by the class loader. Note: This option does not apply to classes loaded from a JXE file. The ″rom.classes″ entry in the JXE file is mapped directly to a ROM class segment.
–Xmoi*x*	Sets the **OldSpace** increment to *x.* This value is used to expand the **OldSpace.** A value of *0* means no expansion is allowed. If **–Xmoi** is not specified, there are no restrictions on the expansion size of **OldSpace**.

Notices

Trademarks

Notices for Excluded Components

These notices are provided in the `notices.txt` file, located in the `\licenses` directory.