JDK for Solaris Developer's Guide

Part No: 817–7970–10
January, 2005

ORACLE

110418@25097

Contents

Preface

This manual is an introduction to and overview of the new features and enhancements in the Java 2 Platform Standard Edition 5, for the Solaris Operating System.

Who Should Use This Book

This document is intended for application developers who use the Java 2 Platform Standard Edition 5, on the Solaris Operating System. The Java software is optimized to deliver superior performance to server-side and client-side Java technology applications in an enterprise environment.

This document is a subset of the J2SE 5 documentation available at http://java.sun.com/j2se/1.5.0/docs/index.html. Upon final release of this product, consider that online documentation to be the definitive description of the Java 2 Platform Standard Edition 5 product.

How This Book Is Organized

lists the features and enhancements of the product.

discusses compatibility issues.

Related Documentation

The following documents also contain information related to this release:

- *J2SE 5 Release Notes* located online at http://java.sun.com/j2se/1.5.0/relnotes.html.
- *J2SE 5 Documentation* located online at http://java.sun.com/j2se/1.5.0/docs/index.html.
- *Java 2 Platform, Standard Edition, v5 API Specification* located online at http://java.sun.com/j2se/1.5.0/docs/api/index.html.

Accessing Sun Documentation Online

The docs.sun.com Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Typographic Conventions

The following table describes the typographic changes used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type **`rm`** *filename*.
AaBbCc123	Book titles, new words, or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide*. These are called *class* options. You must be *root* to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

New Features and Enhancements

Version 1.5.0 of the Java Platform Standard Edition 5 is a major feature release. The features listed below are introduced in 1.5.0 since the previous major release (1.4.0).

For highlights of the new features, also see J2SE 1.5 in a Nutshell (at http://java.sun.com/developer/technicalArticles/releases/j2se15/). For issues, see the JDK 5.0 release notes (at http://java.sun.com/j2se/1.5.0/relnotes.html).

-
-
  -
  -
  -
  -
  -
  -
  -
-
  -
  -
  -
  -
  -
  -
-
  -
  -
  -
  -
  -

Performance Enhancements

For a synopsis of performance enhancements, see Performance Enhancements at http://java.sun.com/j2se/1.5.0/docs/guide/performance/speed.html.

Java Language Features

For more information see New Language Features at http://java.sun.com/j2se/1.5.0/docs/guide/language/index.html.

Generics

This long-awaited enhancement to the type system allows a type or method to operate on objects of various types while providing compile-time type safety. It adds compile-time type safety to the Collections Framework and eliminates the drudgery of casting. Refer to JSR 14 and to the generics documentation at http://java.sun.com/j2se/1.5.0/docs/guide/language/generics.html.

Enhanced for Loop

This new language construct eliminates the drudgery and error-proneness of iterators and index variables when iterating over collections and arrays. Refer to JSR 201 and to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/language/foreach.html.

Autoboxing/Unboxing

This facility eliminates the drudgery of manual conversion between primitive types (such as int) and wrapper types (such as Integer). Refer to JSR 201 and to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/language/autoboxing.html.

Typesafe Enums

This flexible object-oriented enumerated type facility allows you to create enumerated types with arbitrary methods and fields. It provides all the benefits of the Typesafe Enum pattern (*Effective Java*, Item 21) without the verbosity and the error-proneness. Refer to JSR 201 and to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/language/enums.html.

Varargs

This facility eliminates the need for manually boxing up argument lists into an array when invoking methods that accept variable-length argument lists. Refer to JSR 201 and to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/language/varargs.html.

Static Import

This facility lets you avoid qualifying static members with class names without the shortcomings of the *Constant Interface antipattern*. Refer to JSR 201 and to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/language/static-import.html.

Metadata (Annotations)

This language feature lets you avoid writing boilerplate code under many circumstances by enabling tools to generate it from annotations in the source code. This leads to a *declarative* programming style where the programmer says what should be done and tools emit the code to do it. Also it eliminates the need for maintaining *side files* that must be kept up to date with changes in source files. Instead the information can be maintained in the source file. Refer to JSR 175 and to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html.

Virtual Machine

Class Data Sharing

The class data sharing feature is aimed at reducing application startup time and footprint. The installation process loads a set of classes from the system jar file into a private, internal representation, then dumps that representation to a *shared archive* file. During subsequent JVM invocations, the shared archive is memory-mapped in, saving the cost of loading those classes and allowing much of the JVM's metadata for these classes to be shared among multiple JVM processes. For more information, refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/vm/class-data-sharing.html.

Garbage Collector Ergonomics

The parallel collector has been enhanced to monitor and adapt to the memory needs of the application. You can specify performance goals for applications and the JVM will tune the size of the Java heap to meet those performance goals with the smallest application footprint

consistent with those goals. The goal of this adaptive policy is to eliminate the need to tune command-line options to achieve the best performance. For a synopsis of garbage collection features, refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/vm/gc-ergonomics.html.

Server-Class Machine Detection

At application startup, the launcher can attempt to detect whether the application is running on a *server-class* machine. Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/vm/server-class.html.

Thread Priority Changes

Thread priority mapping has changed somewhat allowing Java threads and native threads that do not have explicitly set priorities to compete on an equal footing. Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/vm/thread-priorities.html.

Fatal Error Handling

The fatal error reporting mechanism has been enhanced to provide improved diagnostic output and reliability.

High-Precision Timing Support

The method `System.nanoTime()` has been added, providing access to a nanosecond-granularity time source for relative time measurements. The actual precision of the time values returned by System.nanoTime() is platform-dependent.

Core Libraries

Lang and Util Packages

For a synopsis of java.lang and java.util enhancements, refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/lang/enhancements.html.

Networking

For a synopsis of added networking features, refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/net/enhancements-1.5.0.html.

Security

This release of J2SE offers significant enhancements for security. It provides better support for security tokens, support for more security standards (SASL, OCSP, TSP), improvements for scalability (SSLEngine) and performance, plus many enhancements in the crypto and Java GSS areas. For details see the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/security/index.html.

Internationalization

Enhancements are as follows:

- Character handling is now based on version 4.0 of the Unicode standard. This affects the `Character` and `String` classes in the `java.lang` package, the collation and bidirectional text analysis functionality in the `java.text` package, character classes in the `java.util.regex` package, and many other parts of the J2SE. As part of this upgrade, support for supplementary characters has been specified by the JSR 204 expert group and implemented throughout the J2SE. See the article Supplementary Characters in the Java Platform, the Java Specification Request 204, and the `Character` class documentation for more information.
- The DecimalFormat class has been enhanced to format and parse `BigDecimal` and `BigInteger` values without loss of precision. Formatting of such values is enhanced automatically; parsing into BigDecimal needs to be enabled using the `setParseBigDecimal` method.
- Vietnamese is now supported in all locale sensitive functionality in the `java.util` and `java.text` packages. See the Supported Locales document for complete information on supported locales and writing systems.

Refer also to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/intl/index.html.

Improved Support for Environment Variables

The `System.getenv(String)` method is no longer deprecated. The new `System.getenv()` method allows access to the process environment as a `Map<String,String>`. Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html#getenv(java.lang.String).

ProcessBuilder

The new `ProcessBuilder` class provides a more convenient way to invoke subprocesses than does Runtime.exec. In particular, ProcessBuilder makes it easy to start a subprocess with a

modified process environment (that is, one based on the parent's process environment, but with a few changes). Refer also to the documentation at http://java.sun.com/j2se/1.5.0/docs/api/java/lang/ProcessBuilder.html.

Formatter

An interpreter for printf-style format strings, the `Formatter` class provides support for layout justification and alignment, common formats for numeric, string, and date/time data, and locale-specific output. Common Java types such as `byte`, `java.math.BigDecimal` , and `java.util.Calendar` are supported. Limited formatting customization for arbitrary user types is provided through the `java.util.Formattable` interface.

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/api/java/util/Formatter.html.

Scanner

The `java.util.Scanner` class can be used to convert text into primitives or `Strings`. Since it is based on the `java.util.regex` package, it also offers a way to conduct regular expression based searches on streams, file data, strings, or implementors of the `Readable` interface. Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/api/java/util/Scanner.html.

Reflection

Support for generics, annotations, enums, and convenience methods has been added. Also, `java.lang.Class` has been generified. Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/reflection/enhancements.html.

JavaBeans Component Architecture

A subclass of `PropertyChangeEvent` called `IndexedPropertyChangeEvent` has been added to support bound properties that use an index to identify the part of the bean that changed. Also, methods have been added to the `PropertyChangeSupport` class to support firing indexed property change events. Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/beans/index.html.

Collections Framework

The Collections Framework has been enhanced in the following ways:

- Three new language features are targeted at collections: Generics, Enhanced `for` Loop, and Autoboxing.
- Three new interfaces have been added to the framework (two of which are part of `java.util.concurrent`): `Queue`, `BlockingQueue`, and `ConcurrentMap`.
- Two concrete implementations of `Queue` have been added, as well as one skeletal implementation.
- Five blocking queue implementations have been added, and one `ConcurrentMap` implementation.
- Special-purpose `Map` and `Set` implementations are provided for use with typesafe enums.
- Special-purpose copy-on-write `List` and `Set` implementations have been added.
- Wrapper implementations are provided to add dynamic type-safety for most collection interfaces.
- Several new algorithms are provided for manipulating collections.
- Methods are provided to compute hash codes and string representations for arrays.

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/collections/index.html.

Java API for XML Processing (JAXP)

For details refer to JSR 206 or to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/xml/jaxp/index.html.

Bit Manipulation Operations

The wrapper classes (`Integer`, `Long`, `Short`, `Byte`, and `Char`) now support common bit manipulation operations which include `highestOneBit`, `lowestOneBit`, `numberOfLeadingZeros`, `numberOfTrailingZeros`, `bitCount`, `rotateLeft`, `rotateRight`, `reverse`, `signum`, and `reverseBytes`.

Math

The numerical functionality provided by the libraries has been augmented in several ways:

- The `BigDecimal` class has added support for fixed-precision floating-point computation. Refer to JSR 13.
- The `Math` and `StrictMath` libraries include hyperbolic transcendental functions (sinh, cosh, tanh), cube root, base 10 logarithm, etc.

- Hexadecimal floating-point support - To allow precise and predictable specification of particular floating-point values, hexadecimal notation can be used for floating-point literals and for string to floating-point conversion methods in `Float` and `Double`.

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/math/index.html.

Instrumentation

The new `java.lang.instrument` package provides services that allow Java programming agents to instrument programs running on the Java virtual machine. The intrumentation mechanism is modification of the bytecodes of methods.

Serialization

Support has been added to handle enumerated types which are new in version 1.5.0. The rules for serializing an enum instance differ from those for serializing an ordinary serializable object: the serialized form of an enum instance consists only of its enum constant name, along with information identifying its base enum type. Deserialization behavior differs as well--the class information is used to find the appropriate enum class, and the Enum.valueOf method is called with that class and the received constant name in order to obtain the enum constant to return.

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/serialization/index.html.

Concurrency Utilities

The `java.util.concurrent`, `java.util.concurrent.atomic`, and `java.util.concurrent.locks` packages provide a powerful, extensible framework of high-performance, scalable, thread-safe building blocks for developing concurrent classes and applications, including thread pools, thread-safe collections, semaphores, a task scheduling framework, task synchronization utilities, atomic variables, and locks. The addition of these packages to the core class library frees the programmer from the need to craft these utilities by hand, in much the same manner that the Collections Framework did for data structures. Additionally, these packages provide low-level primitives for advanced concurrent programming which take advantage of concurrency support provided by the processor, enabling programmers to implement high-performance, highly scalable concurrent algorithms in the Java language to a degree not previously possible without resorting to native code.

Refer to JSR 166 and to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/concurrency/index.html.

Threads

The `java.lang.Thread` class has the following enhancements:

- Thread priority handling has changed; see the above link for details.
- `Thread.State` enum class and the new `getState()` API are provided for querying the execution state of a thread.
- The new thread dump API - the `getStackTrace` and `getAllStackTraces` methods in the `Thread` class - provides a programmatic way to obtain the stack trace of a thread or all threads.
- The `uncaughtExceptionHandler` mechanism, previously available only through the ThreadGroup class, is now available directly through the Thread class.
- A new form of the `sleep()` method is provided which allows for sleep times smaller than one millisecond.

Monitoring and Management

This release of J2SE offers significant enhancements for monitoring and management for the Java platform.

- Monitoring and management API for the Java virtual machine The new `java.lang.management` package provides the interface for monitoring and managing the Java virtual machine.
- Monitoring and management API for the logging facility The new `java.util.logging.LoggingMXBean` interface is the management interface for the logging facility.
- JMX instrumentation of the Java virtual machine The Java virtual machine (JVM) has built-in instrumentation that enables you to monitor and manage it using JMX. You can easily start a JMX agent for monitoring and managing remote or local Java VMs instrumentation or of any application with JMX instrumentation. See Monitoring and Management Using JMX at http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html for details.
- The SNMP agent publishes the standard MIB for the Java virtual machine instrumentation as defined by JSR 163. For more information, see SNMP Monitoring and Management.
- Java Management Extensions JMX API version 1.2 and the RMI connector of the JMX Remote API version 1.0 are included in J2SE 5 release. The JMX API allows you to instrument libraries and applications for monitoring and management. The RMI connector allows this instrumentation to be remotely accessible. For more details, see the JMX documentation at http://java.sun.com/j2se/1.5.0/docs/guide/management/index.html.

Integration Libraries

Remote Method Invocation (RMI)

RMI has been enhanced in the following areas:

- Dynamic Generation of Stub Classes - This release adds support for the dynamic generation of stub classes at runtime, obviating the need to use the Java Remote Method Invocation (Java RMI) stub compiler, rmic, to pregenerate stub classes for remote objects. Note that rmic must still be used to pregenerate stub classes for remote objects that need to support clients running on earlier versions.
- Standard SSL/TLS Socket Factory Classes - This release adds standard Java RMI socket factory classes, `javax.rmi.ssl.SslRMIClientSocketFactory` and `javax.rmi.ssl.SslRMIServer SocketFactory`, which communicate over the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocols using the Java Secure Socket Extension (JSSE).
- Launching rmid or a Java RMI Server from inetd/xinetd - A new feature, provided by the `System.inheritedChannel` method, allows an application to obtain a channel (`java.nio.channels.SocketChannel` or `java.nio.channels.ServerSocketChannel`, for example) inherited from the process that launched the virtual machine (VM). Such an inherited channel can be used to either service a single incoming connection (as with `SocketChannel`) or accept multiple incoming connections (as with `ServerSocketChannel`). Therefore, Java networking applications launched by inetd (Solaris(tm) Operating System) or xinetd (Linux) can now obtain the `SocketChannel` or `ServerSocketChannel` inherited from inetd/xinetd.

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/rmi/index.html.

Java Database Connectivity (JDBC)

`RowSet` interface, part of the `javax.sql` package, introduced in J2SE version 1.4, provides a lightweight means of passing data between components.

At this release, as an aid to developers, the `RowSet` interface has been implemented (as JSR 114) in five of the more common ways a `RowSet` object can be used. These implementations provide a standard that developers are free to use as is or to extend. Following are the five standard implementations:

- `JdbcRowSet` - used to encapsulate a result set or a driver that is implemented to use JDBC technology
- `CachedRowSet` - disconnects from its data source and operates independently except when it is getting data from the data source or writing modified data back to the data source. This makes it a lightweight container for as much data as it can store in memory.

- `FilteredRowSet` - extends `CachedRowSet` and is used to get a subset of data
- `JoinRowSet` - extends `CachedRowSet` and is used to get an SQL `JOIN` of data from multiple `RowSet` objects
- `WebRowSet` - extends `CachedRowSet` and is used for XML data. It describes tabular components in XML using a standardized XML schema.

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/jdbc/index.html.

CORBA, Java IDL and RMI-IIOP

Enhancements to CORBA, Java IDL, and Java RMI-IIOP are discussed in Changes in CORBA Features Between J2SE 1.4.x and 1.5.0. Refer to the Java IDL documentation at http://java.sun.com/j2se/1.5.0/docs/guide/idl/index.html and to the Java RMI-IIOP documentation at http://java.sun.com/j2se/1.5.0/docs/guide/rmi-iiop/index.html.

Java Naming and Directory Interface (JNDI)

JNDI provides the following new features.

- Enhancements to javax.naming.NameClassPair to access the fullname from the directory/naming service
- Support for standard LDAP controls: Manage Referral Control, Paged Results Control and Sort Control
- Support for manipulation of LDAP names.

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/jndi/index.html.

User Interface

Internationalization

- To render multilingual text, using logical fonts, 2D now takes advantage of installed host OS fonts for all supported writing systems. For example, if you run in a Thai locale environment, but have Korean fonts installed, both Thai and Korean are rendered. The J2RE now also automatically detects physical fonts that are installed into its lib/fonts/fallback directory and adds these physical fonts to all logical fonts for 2D rendering.

- AWT now uses the Unicode APIs on Windows 2000/XP. As a result, some of its components can handle text without being restricted by Windows locale settings. For example, AWT text components can accept and display text in the Devanagari writing system regardless of the Windows locale settings.

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/intl/index.html.

Java Sound Technology

- Ports are now available on all platforms (RFE 4782900).
- MIDI device i/o is now available on all platforms (RFE's 4812168, 4782924).
- Optimized direct audio access is implemented on all platforms (RFEs 4908240 and 4908879). It is enabled by default on systems which offer native mixing (i.e. Linux ALSA with hardware mixing, Solaris Mixer enabled, Windows DirectSound).
- The new real-time Sequencer works with all MIDI devices and allows unlimited Transmitters (RFE 4773012).
- The `sound.properties` configuration file allows choice of default devices (RFE 4776511). For details, see `MidiSystem` and `AudioSystem` for details.
- MidiDevices can query connected Receivers and Transmitters (RFE 4931387, methods MidiDevice.getReceiver and MidiDevice.getTransmitter).
- `AudioFormat`, `AudioFileFormat`, and `MidiFileFormat` now have properties that allow further description and qualification of the format (RFEs 4925767 and 4666845).
- A set of ease-of-use methods allow easier retrieval of lines from AudioSystem (RFE 4896221).
- The Sequencer interface is extended with loop methods, for seamless looping of specific portions of a MIDI sequence (RFE 4204105).
- Java Sound no longer prevents the VM from exiting (bug 4735740).

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/sound/index.html.

Java 2D Technology

Added 2D features include expanded Linux and Solaris printer support, new methods for creating fonts from files and streams, and new methods related to VolatileImages and hardware acceleration of images. A number of internal changes to text rendering code greatly improve its robustness, performance, and scalability. Other performance work includes hardware-accelerated rendering using OpenGL (disabled by default).

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/2d/index.html.

Image I/O

The Image I/O system now has readers and writers for BMP and WBMP formats.

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/imageio/index.html.

AWT

Version 1.5.0 features many AWT enhancements and bug fixes, including some that have often been requested by our customers. Most notably, the new `MouseInfo` class makes it possible to determine the mouse location on the desktop. New `Window` methods make it possible to specify the default location for a newly created window (or frame), appropriate to the platform. Another `Window` enhancement makes it possible to ensure that a window (or frame) is always on top. (This feature does not work for some window managers on Solaris/Linux.) In the area of data transfer, the new `DropTargetDragEvent` API allows the drop target to access transfer data during the drag operation.

AWT http://java.sun.com/j2se/1.5.0/docs/guide/awt/index.html.

Swing

With the 1.4.2 release we provided two new look and feels for Swing: XP and GTK. Rather than taking a break, in 1.5.0 we're providing two more look and feels: Synth, a skinnable look and feel, and Ocean, a new theme for Metal. Beyond look and feels, we've added printing support to `JTable`, which makes it trivial to get a beautiful printed copy of a `JTable`. Lastly, after seven years, we've made `jFrame.add` equivalent to `jFrame.getContentPane().add()`.

Refer to the documentation at Swing http://java.sun.com/j2se/1.5.0/docs/guide/swing/index.html.

Deployment

General Deployment

Pack200, a new hyper-compression format for JAR files defined by JSR 200, can siginificantly reduce the download size of JAR files used in Java Webstart applications and Java Plug-in applets.

For a synopsis of general deployment features and enhancements, refer to http://java.sun.com/j2se/1.5.0/docs/guide/deployment/enhancements-1.5.0.html.

Java Web Start Deployment

For a synopsis of Java Web Start deployment features and enhancements, refer to http://java.sun.com/j2se/1.5.0/docs/guide/javaws/enhancements-1.5.0.html

Tools and Tool Architecture

Java Virtual Machine Tool Interface (JVMTI)

JVMTI is a new native programming interface for use by development and monitoring tools. It provides both a way to inspect the state and to control the execution of applications running in the Java virtual machine (VM). JVMTI is intended to provide a VM interface for the full breadth of tools that need access to VM state, including but not limited to: profiling, debugging, monitoring, thread analysis, and coverage analysis tools.

JVMTI will replace the now deprecated JVMPI and JVMDI in the next major release of J2SE.

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/index.html.

Java Platform Debugger Architecture (JPDA)

JPDA itself has many new features, described in more detail on the JPDA enhancements page.

- A read-only subset of JDI has been defined. This subset can be used on a debuggee in which no debug code can be executed (such as a core file or a process that is hung or was not started in debug mode). The subset allows creation of JDI connectors for use in debugging such debuggees.
- A service provider interface for connectors and transports allows debugger vendors, or even end users, to create their own JDI connectors and transports and plug them into the JPDA reference implementation. For example, a connector could be provided to use SSL to communicate between the debugger and debuggee.
- JDI supports the new language features (generics, enums, and varargs).
- The lowest layer of JPDA, the Java Virtual Machine Debugger Interface (JVMDI), has been deprecated and will be removed in the next major J2SE release. Replacing it is the Java Virtual Machine Tool Interface (JVMTI). This is a more general interface that allows profiling to be done as well as debugging. The current profiling interface, Java Virtual Machine Profiling Interface(JVMPI) is also deprecated and will be removed in the next major release.
- The JPDA reference implementation includes new JDI connectors that allow corefiles and hung processes to be debugged.

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/jpda/index.html.

Java Compiler (javac)

Compiler options include:

- `-source 1.5` - Enable 1.5 specific language features to be used in source files. (`-target 1.5` is implied.)
- `-target 1.5` - Allow javac to use 1.5 specific features in the libraries and virtual machine.
- `-Xlint` - Enable javac to produce warning messages about legal, but suspect and often problematic, program constructs. An example would be declaring a class that implements `Serializable` but does not define a `serialVersionUID`.
- `-d32` - Indicate a 32-bit Solaris or Linux platform.
- `-d64` - Indicate a 64-bit Solaris or Linux platform.

Refer to the man page documentation at http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/javac.html.

Javadoc Tool

See What's New in Javadoc 1.5.0 at
http://java.sun.com/j2se/1.5.0/docs/guide/javadoc/whatsnew-1.5.0.html.

Annotation Processing Tool (`apt`)

`apt` is a new command-line utility for annotation processing. It includes a set of reflective APIs and supporting infrastructure to process program annotations.

Refer to the documentation at http://java.sun.com/j2se/1.5.0/docs/guide/apt/index.html.

OS & Hardware Platforms

Supported System Configurations

For more information, refer to http://java.sun.com/j2se/1.5.0/system-configurations.html.

64-Bit AMD Opteron Processors

With J2SE 5, AMD Opteron processors are supported by the server VM on Suse Linux and on Windows 2003.

Compatibility with Previous Releases

This document contains information on the following topics:

-
-
-

The compatibility documents are divided to track incompatibility only between adjacent versions. For example, this 1.5.0 compatibility page details only 1.5.0 incompatibility with 1.4.2 and not previous versions. Therefore, to find how 1.5.0 is incompatible with all versions, you would need to look on all compatibility pages.

The following documents contain information about incompatibilities between adjacent releases.

- Incompatibilities in J2SE 1.4.2 (since 1.4.1) at http://java.sun.com/j2se/1.4.2/compatibility.html
- Incompatibilities in J2SE 1.4.1 (since 1.4.0) at http://java.sun.com/j2se/1.4.1/compatibility.html
- Incompatibilities in J2SE 1.4.0 (since 1.3) at http://java.sun.com/j2se/1.4/compatibility.html
- Incompatibilities in J2SE 1.3 (since 1.2) at http://java.sun.com/j2se/1.3/compatibility.html

See the Java Language Specification Maintenance Page (at http://java.sun.com/docs/books/jls/jls-maintenance.html) for a summary of changes that have been made to the specification of the Java programming language since the publication of the Java Language Specification, Second Edition (at http://java.sun.com/docs/books/jls/index.html).

Binary Compatibility

Version 1.5.0 of the Java 2 Platform Standard Edition 5 is upwards binary-compatible with version 1.4.2 except for the incompatibilities listed below. This means that, except for the noted incompatibilities, class files built with version 1.4.2 compilers will run correctly on version 1.5.0.

Some early bytecode obfuscators produced class files that violated the class file format as given in the virtual machine specification. Such improperly formatted class files will not run on the Java 2 JDK's virtual machine, though some of them may have run on earlier versions of the virtual machine. To remedy this problem, regenerate the class files with a newer obfuscator that produces properly formatted class files.

Source Compatibility

Downward source compatibility is not supported. If source files use new language features or Java 2 Platform APIs, they will not be usable with an earlier version of the Java platform.

In general, the policy is as follows, except for any incompatibilities listed further below:

- Maintenance releases (such as 1.4.1, 1.4.2) do not introduce any new language features or APIs, so they maintain source-compatibility with each other.
- Functionality releases and major releases (such as 1.3.0, 1.4.0, 1.5.0) maintain upwards but not downwards source-compatibility.

Deprecated APIs are interfaces that are supported *only* for backwards compatibility. The `javac` compiler generates a warning message whenever one of these is used, unless the `-nowarn` command-line option is used. It is recommended that programs be modified to eliminate the use of deprecated APIs, though there are no current plans to remove such APIs, with the exception of JVMDI and JVMPI entirely from the system. (Refer to bug 4639363.)

Some APIs in the `sun.*` packages have changed. These APIs are not intended for use by developers. Developers importing from `sun.*` packages do so at their own risk. For more details, see, *Why Developers Should Not Write Programs That Call sun.* Packages* (at http://java.sun.com/products/jdk/faq/faq-sun-packages.html).

Incompatibilities in the Java 2 Platform Standard Edition 5 (since 1.4.2)

J2SE 5 is strongly compatible with previous versions of the Java 2 Platform. Almost all existing programs should run on J2SE 5 without modification. However, there are some minor potential source and binary incompatibilities in the JRE and JDK that involve rare circumstances and corner cases that we are documenting here for completeness.

1. Generification - *Generification* is the process of adding generic type parameters and arguments to existing classes and methods in a manner that's consistent with the specifications of those classes. JSR 14 specified the generification of many of the core libraries, in particular the collection classes and the `Class` class. In the 1.5 Beta 2 release, the effect of the core generification was propagated throughout the rest of the platform wherever possible.

   Most source code that uses generified classes, constructors, methods, and fields will continue to compile in 1.5, though some will not. The simplest workaround for code that fails to compile due to the generification changes is to specify `-source 1.4` on the `javac` command line.

   For information about generics and the core generification, see JSR 14 and the generics tutorial (at http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf).

2. Virtual Machine - Previously, the default virtual machine (VM) for Solaris/SPARC was the client VM. However, many Solaris/SPARC boxes are used as servers, on which the server VM is more appropriate for performance reasons. Thus, as of 1.5, server-class Solaris/SPARC machines run the server VM by default. In general, the throughput of the server VM is much better than the client VM, but the startup time is somewhat worse. A *server-class machine* is currently defined to be one with 2 or more processors and 2 or more gigabytes of memory.

   For more information, see Server-Class Machine Detection (at http://java.sun.com/j2se/1.5.0/docs/guide/vm/server-class.html) and Garbage Collection Ergonomics at (http://java.sun.com/j2se/1.5.0/docs/guide/vm/gc-ergonomics.html).

3. Virtual Machine - To reflect the class sharing feature introduced in 1.5, the `java.vm.info` property (which is reflected in the text displayed by `java -version`) now specifies the sharing mode. Any code that parses all the way to the end of the `java.vm.info` property value or the output of `java -version` might need to be changed.

   For more information, see bug 4964160 and Class Data Sharing (at http://java.sun.com/j2se/1.5.0/docs/guide/vm/class-data-sharing.html).

4. Class Loader - Previously, it was possible to specify a non-binary class name to `ClassLoader` methods that take a `String` class name argument. This unintended behaviour was not compliant with the long-standing specification of class names. As of 1.5, parameter checking of these `ClassLoader` methods has been modified to comply with the specification, and any class name that is not a binary name is treated like any other unrecognized class name. Since the APIs that explicitly require or return class names (for example, `Class.forName` or `Class.getName`) use the binary name for reference types, it is unlikely that the typical user would have produced a class name that would have returned a `Class`.

   For more information, see the definition of binary name (at http://java.sun.com/docs/books/jls/second_edition/html/binaryComp.doc.html#59876) in the Java Language Specification, Second Edition (at http://java.sun.com/docs/books/jls/). Also see the evaluation of bug 4986512.

5. Serialization - Changes in compiler-generated synthetics affect the default serial version UID, and therefore can cause serialization incompatibility when that UID is not explicitly overridden.

   For more information, see bug 4786115.

6. Logging - Previously, the `java.util.logging.Level(String name, int value, String resourceBundleName)` constructor allowed a null `name` argument, but the `parse` method did not. In 1.5, the constructor now throws a `NullPointerException` when the name is null. The compatibility risk is mitigated in that you had to subclass `Level` to use this constructor and would get a `NullPointerException` when using a `Level` name of null for subsequent calls, except for simple calls such as `toString`.

   For more information, see bug 4625722.

7. Apache - The `org.apache` classes, which have never been supported J2SE APIs but are used by the `javax.xml` package, have moved in 1.5 to `com.sun.org.apache.package.internal` so that they won't clash with more recent, developer-downloaded versions of the classes. Any applications that depend on the `org.apache` classes being part of the J2SE release must do one of the following to work in 1.5:

   - Code the application so it uses only the supported interfaces that are part of JAXP.
   - Download the `org.apache.xalan` classes from Apache.

   For more information, see bug 4740355.

8. JAXP - The J2SE 1.4 platform included JAXP 1.1 (Crimson). The J2SE 1.5 platform includes JAXP 1.3 (Xerces). Crimson and Xerces are not simply different versions of the same codebase. Instead, they are entirely different implementations of the JAXP standard. So, while they both conform to the JAXP standard, there are some subtle differences between them.

   Although Crimson was small and fast, it was ultimately less functional than Xerces (an open-source implementation hosted at Apache). In addition, the JAXP standard has evolved from 1.1 to 1.3. These two factors combine to create compatibility issues.

   For details, see the JAXP Compatibility Guide for 1.5 at http://java.sun.com/j2se/1.5.0/docs/guide/xml/jaxp/JAXP-Compatibility_150.html.

9. JAXP - The J2SE 1.4 platform supported the DOM Level 2 API. The J2SE 1.5 platform supports the DOM Level 3 family of APIs. New methods have been added to DOM Level 3 interfaces, so some existing applications using DOM Level 2 will not be able to compile with the new interfaces.

   Many DOM Level 2 applications will run if DOM Level 3 is substituted for DOM Level 2 in the class path; however, a small number will encounter a `NoSuchMethodException`. Therefore, some applications will not have binary compatibility.

   For details, see the JAXP Compatibility Guide for 1.5 at http://java.sun.com/j2se/1.5.0/docs/guide/xml/jaxp/JAXP-Compatibility_150.html.

10. JAXP - The J2SE 1.4 platform supported the SAX 2.0 API. The J2SE 1.5 platform supports SAX 2.0.2. In general, SAX 2.0.2 is a bug-fix release, with no API changes. However, a few clarifications done as part of SAX 2.0.2 release are possible compatibility issues:
    - `ErrorHandler`, `EntityResolver`, `ContentHandler`, and `DTDHandler` can now be set to null by applications. SAX 2.0 required the XML processor to throw `java.lang.NullPointerException` in this case. This change is relevant to the XML processor because most parsers react to null by restoring the default settings.
    - `DefaultHandler` is a default implementation class for various handlers including `EntityResolver`. The `resolveEntity` method implementation in `DefaultHandler` is now declared as `throws IOException, SAXException`. Previously it could throw only `SAXException`.
    - The addition of `java.io.IOException` to the list of exceptions thrown by the `resolveEntity` method is a source-incompatible change. Specifically, code that invokes `resolveEntity` might compile successfully with SAX 2.0 but fail compilation with SAX 2.0.2 because it needs to handle `IOException` along with `SAXException`.

    For details, see the JAXP Compatibility Guide for 1.5 at http://java.sun.com/j2se/1.5.0/docs/guide/xml/jaxp/JAXP-Compatibility_150.html.
11. JAXP - Previously, Xalan was the default transformer. Since the Apache community has agreed to make XSLTC the default processor for developing XSLT 2.0, XSLTC is the default transformer as of 1.5. Compatibility risks include:
    - Xalan has bugs that XSLTC does not, and vice-versa. Application code that has taken Xalan bugs into account is likely to fail.
    - XSLTC does not support all the extensions that Xalan does. These extensions are beyond the definition of the JAXP and XSLT specifications. For those users impacted by this, the work around of downloading the Xalan classes from Apache is still available. Also, going forward we expect to be supporting more and more extensions in XSLTC.

    For more information, see the JAXP Compatibility Guide for 1.5 at http://java.sun.com/j2se/1.5.0/docs/guide/xml/jaxp/JAXP-Compatibility_150.html.
12. 2D - Previously, passing a null `Image` parameter to a `Graphics.drawImage` method resulted in a `NullPointerException`. As of 1.5, it doesn't. The new behavior allows applications that worked with the Microsoft VM to work with the standard VM. Any applications that depend on the `NullPointerException` need to be changed so that they'll work in 1.5.
13. AWT - Previously, only containers that were focus cycle roots could provide a focus traversal policy. As of 1.5, any container can provide a focus traversal policy; the new `FocusTraversalPolicyProvider` property of `Container` indicates whether it does.

    The focus traversal policies provided with the Java platform have been changed in 1.5 to accommodate focus traversal policy providers. Specifically, when a policy encounters a focus traversal policy provider during forward (backward) traversal, it should not treat its components as belonging to the provided focus cycle root but should use the focus traversal policy of focus traversal policy provider to get next (previous) component. If the returned

component is the same as the first (last) component returned by the focus traversal policy of the focus traversal policy provider, then invoking the policy should get the next (previous) component in the cycle after (before) the focus traversal policy provider. Calculation of first and last components in focus cycle roots should use the focus traversal policies of focus traversal policy providers when necessary (when a first or last component is itself a `Container` and a focus traversal policy provider).

Because this change doesn't require any new methods in focus traversal policies, third-party focus traversal policies will continue to work, although they will not support the notion of providers.

If you have written a focus traversal policy and wish to support providers, you need to make changes similar to the ones made to the platform-provided policies in 1.5.

For more information, see the Focus Traversal Policy Providers section at http://java.sun.com/j2se/1.5.0/docs/api/java/awt/doc-files/FocusSpec.html #FocusTraversalPolicyProviders of the focus specification, and the AWT Focus Subsystem at http://java.sun.com/j2se/1.5.0/docs/api/java/awt/doc-files/FocusSpec.html.

14. Drag and Drop - Previously, the only drag and drop (DnD) protocol supported on X11 was the Motif DnD protocol. In 1.5, the XDND protocol is also supported, and the Motif DnD protocol has been reimplemented to not depend on the Motif library. It's possible that regressions might be caused by the difference between the new Motif DnD protocol implementation and one provided by the Motif library. However, since the Motif library's implementation is buggy, it's believed that the new implementation is at least as high in quality, as well as better supported.

    For more information, see bug 4638443.

15. Swing - Buttons with a customized background color might require code changes to be rendered as intended with the 1.5 Java look and feel theme, Ocean. The reason is that Ocean draws a gradient on buttons, by default. If you don't want the gradient, either set the `contentAreaFilled` property to true or set the background to a `Color` that is not a `UIResource`. In most cases this is as simple as: `button.setBackground(Color.RED);` If, for some reason, you are picking up a `UIResource` you can create a new `Color` that is not a `UIResource` like this: `button.setBackground(new Color(oldColor));`

    For more information, see bug 4908404.

16. Swing - In `JTree` and `JList` it has always been the case that the user manipulates the lead index with the keyboard. For example, if the lead is on row four in a `JList` and you press the up key, this moves the lead to row three and selects the item there. With these components, then, the lead is considered the focused index. They pass information to their renderers indicating whether or not to draw the focus indicator for a given index, and this is based on whether that index is the lead.

    Prior to 1.5, `JTable` was doing the opposite and using the anchor index in the same manner that `JTree` and `JList` use the lead. A request to correct this was made as RFE number 4759422 and eventually fixed as part of 4303294. Now `JTable` is consistent with `JList` and `JTree`. This could affect developers that assumed the previous behavior. For example,

consider an application that needs information on what is being shown as the focused cell in a `JTable`, and it assumes that to be the anchor. While this would be correct pre-1.5, it could now result in determining one index to be focused, when in reality some other index is displaying the focus rectangle.

For more information, see bugs 4759422 and 4303294.

17. JVMDI - As of 1.5 the Java Virtual Machine Debug Interface (JVMDI) is deprecated. *JVMDI will be removed in the next major release.* Any new development should use JVMTI. Existing tools should begin moving to JVMTI.

    For more information, see the JVMTI documentation (at http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/index.html).

18. JVMPI - As of 1.5 the Java Virtual Machine Profiling Interface (JVMPI) is deprecated. JVMPI will be removed in the next major release. Any new development should use JVMTI. Existing tools should begin moving to JVMTI.

    For more information, see the JVMTI documentation at http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/index.html.