

# Java™ AVK for the Enterprise: Testing J2EE™ Applications for Portability

Technical White Paper  
April 2004



# Table of Contents

<b>Introduction</b> .....	<b>.1</b>
Java Verification Program .....	2
Why Use the Java AVK for the Enterprise? .....	2
Creating Portable Applications: Using Java BluePrints .....	2
How to Use the Java AVK for the Enterprise .....	2
<b>Internals of the Java AVK for the Enterprise</b> .....	<b>.4</b>
Criteria for Verifying EJB Components .....	5
Criteria for Verifying Web Components .....	5
Criteria for Verifying Web Services .....	5
Static Verification .....	5
Dynamic Testing .....	7
Outside the Scope of the Toolkit .....	7
Limitations .....	8
Reporting Tool .....	8
Java AVK for the Enterprise: Part of the Development Cycle .....	9
Resolving Portability Issues .....	10
<b>Summary</b> .....	<b>11</b>

## Chapter 1

# Introduction

The Java™ Application Verification Kit (AVK) for the Enterprise is part of the Java Verification Program. The program is designed to identify enterprise applications — developed using Java 2 Platform, Enterprise Edition (J2EE™) technology — that are intended to be portable across different implementations of the J2EE platform. The Java AVK for the Enterprise tests the applications for portability across J2EE technology-compatible servers. It tests for correct use of the J2EE APIs, scans source code for nonportable APIs, and helps developers avoid inadvertently writing nonportable code.

The J2EE platform extends the portability benefits of the Java programming language to the enterprise, providing a single set of specifications for application servers and an extensive set of compatibility tests. It enables organizations to run applications across different types of J2EE technology-compatible servers. Given the large investments most businesses have in client-server applications, the J2EE platform can offer companies significant savings in both development time and money by reducing the need to install and test across multiple application servers. With J2EE technology's portability features, there is little or no need to rewrite code for different servers.

Regardless of which product is used to develop enterprise applications, organizations need to ensure that their code will easily port from one application server to another. To do this, they need to make sure that the code does not use vendor-specific extensions to the J2EE platform and that it meets the responsibilities for applications stated in the J2EE specifications.

## Java Verification Program

The Java Verification Program goes beyond the Java AVK for the Enterprise toolkit, and includes a list of criteria that must be met to receive the right to use the Java Verified™ logo for the tested application. The list of criteria is set for the JavaServer Pages™ (JSP™), Java Servlet, and Enterprise JavaBeans™ (EJB™) specifications; the Java AVK for the Enterprise toolkit can be used in part to meet these criteria.

The Java Verification Program is an indication that the developer of the application is actively seeking to provide support for multiple J2EE technology-compatible application servers. And it can provide a key market differentiator for developers who have Java Verified applications.

## Why Use the Java AVK for the Enterprise?

The Java AVK for the Enterprise is a toolkit intended to help developers test applications created using J2EE technology for correct use of the J2EE APIs as well as for portability across J2EE technology-compatible servers. The Java AVK for the Enterprise toolkit is targeted at the applications that run on a J2EE technology-based server. Once the application is verified, it is said to be portable or easily portable to any J2EE technology-compatible application server. The Compatibility Test Suite (CTS) is a set of tests and rules that are applied to the application servers. Once an application server meets the CTS requirements, it is considered J2EE technology compatible.

Application developers use the Java AVK for the Enterprise toolkit to identify portability problems in their application. The toolkit provides tools and guidelines for testing a J2EE technology-based application for portability, and sets the standard for J2EE application portability. Tools include the following:

- The Sun Java System Application Server Platform Edition 8 (“the Application Server”), which is the first compatible production-grade implementation of the J2EE 1.4 specification and is used for logging public API invocations at runtime
- The static archive testing tool, which contains hundreds of tests based on assertions from all J2EE specifications
- The code scanning tool, which scans source code for nonportable APIs
- The reporting tool, which summarizes results from the tests run

## Creating Portable Applications: Using Java BluePrints

The Java BluePrints Program provides an extensive set of guidelines, design patterns, and sample applications that serve as models for developers writing portable applications. In particular, the Java Pet Store and Adventure Builder sample applications demonstrate how to use the capabilities of the J2EE platform to develop robust, scalable, portable, and maintainable e-business applications. Sun recommends that anyone implementing a J2EE application study these applications. For more information about Java BluePrints, see <http://java.sun.com/blueprints>.

## How to Use the Java AVK for the Enterprise

One way to help ensure portability across J2EE technology-compatible servers is to run the application on all available servers. However, this is not a practical approach from either an economic or schedule point of view. The approach taken with the Java AVK for the Enterprise is to

run the application on at least one J2EE technology-compatible server as well as the Application Server. The Application Server is the first compatible production-grade implementation of the J2EE 1.4 specification. This product provides extensive support for web services, including the industry's most complete support for WS-I Basic Profile. Furthermore, it is also the first application server to provide integrated support for JavaServer™ Faces technology. The Application Server includes various tools to help developers prototype J2EE applications and learn about the J2EE platform and technologies. Although the Application Server does contain proprietary APIs, the Java AVK for the Enterprise code scanning tool searches for those specific APIs and reports all such findings. The Java AVK for the Enterprise requires an application to use the Application Server to confirm that it runs on a correct implementation of the J2EE platform.

The Java AVK for the Enterprise uses a self-verification approach to verifying applications. That means that the developer deploys and runs the application on the Application Server, exercises the application using existing test suites or manual tests, and then generates the results and reviews them for correctness. The Application Server supplied with the Java AVK for the Enterprise has been modified to log API invocations in the application as it runs on the server. It also logs any exceptions that are thrown during execution of the application. The server then generates the results of the test runs. Finally, the reporting tool summarizes the results gathered during the verification process based on the guidelines that have been established for the Java AVK for the Enterprise.

This paper addresses the following areas:

- Internals of the Java AVK for the Enterprise
- Portability issues that are caught by the Java AVK for the Enterprise
- What is outside the scope of the toolkit

## Chapter 2

# Internals of the Java AVK for the Enterprise

The Java AVK for the Enterprise is made up of the following major components:

- Static verification
- Dynamic testing
- Reporting tool
- Portability guidelines

Together, these components look at various parts of the application for incompatibilities from the specifications. Static verification reviews the deployable archive, validating the deployment descriptors and the structure of the archive. Static verification also inspects the application source code for nonportable APIs. Dynamic testing tracks the invocations made on the application and whether the invocations were successful. The reporting tool generates reports from both static verification and dynamic testing based on the portability guidelines described in the *Java AVK for the Enterprise User's Guide*. The portability guidelines are established based on good programming practices.

Currently, the guidelines tracked by the Java AVK for the Enterprise are separate for EJB components, Web components, and Web services. The complete set of guidelines is in the *Java Verification Program Manual*. Below is a summary of those guidelines. J2EE applications may include JSP, servlet, or EJB components.

**Criteria for Verifying EJB Components**

- Pass all the static verification tests (warnings should be examined)
- Successfully exercise the application suite in the Application Server and invoke at least 80% of the EJB and business logic methods (with no system exceptions)

**Criteria for Verifying Web Components**

- Pass all the static verification tests (warnings should be examined)
- Invoke all servlets and JSP pages with no compilation or run-time errors
- Name all the JSP files with the `.jsp` extension or declare those JSP pages in the deployment descriptors
- Provide a servlet mapping for all servlets in the `web.xml` file

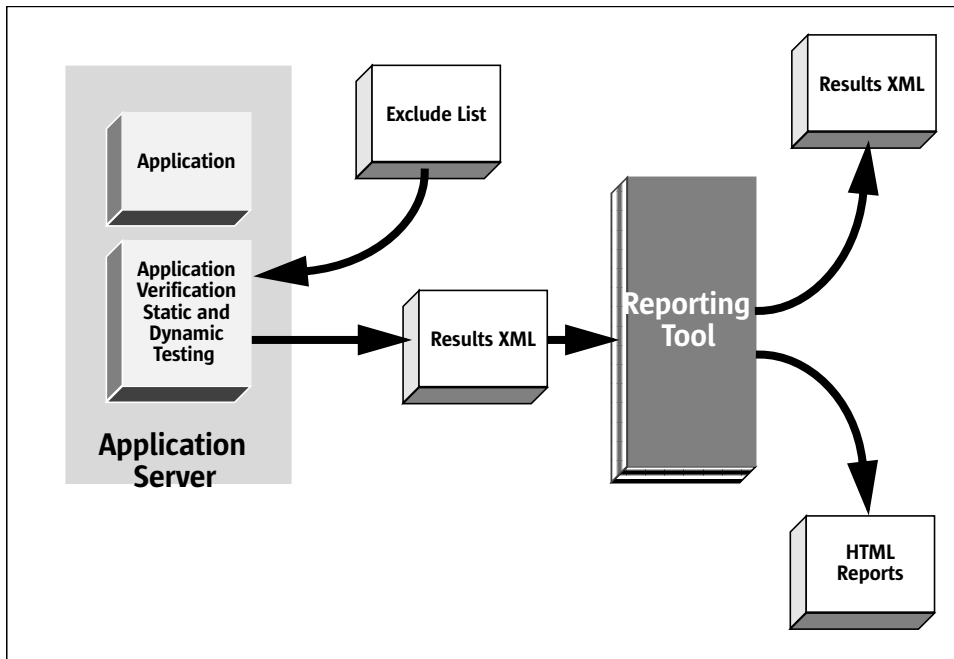
**Criteria for Verifying Web Services**

- Pass all the static verification tests (warnings should be examined)
- Successfully exercise the application suite in the Application Server and invoke at least 80% of the Web service endpoint interface methods (with no system exceptions)

## Static Verification

The first step in verifying an application is static verification, and it is performed by calling two Ant tasks. Static verification looks at both the deployable archive (static archive testing) and the application source code (code scan testing). The tests that make up static archive testing are largely taken from application/container contract sections of the specification. The tests check that the application meets the contracts specified by the specifications that govern that behavior. There are hundreds of tests that are part of static archive testing, and all are based on assertions from the J2EE platform, JavaServer Pages, Java Servlet, J2EE Connector Architecture, and Enterprise JavaBeans specifications.

The static archive tests report a status of Failure or Warning depending on how the assertion is written. If the assertion contains the word “must” or “shall,” failure to meet the assertion is reported as an failure. If the assertion contains the word “can” or “should,” failure to meet the assertion is reported as a warning. All test assertions are normally run against each application. The user may, however, choose to run only a subset of the tests against the application. For example, the user may want to run only the EJB tests or the EJB and Connector tests. The user has the option to run one or more from the following list: EJB, Web, Connector, application client, web services, web services client, and application tests. By default only the Failures and the Warnings are reported, but the user may request that all test results be reported. If the application does not contain a particular J2EE component, such as servlet or EJB components, all tests based on that component are marked as Not Applicable for that application. Finally, if the test passes, it is marked as Passed.



**Figure 2-1:** Java AVK for the Enterprise Tools: High-level View

Static archive testing can be run by invoking the Ant task `ArchiveTest`. A sample of this Ant task is provided in the `samples` directory of the AVK release. This task verifies that the components referenced in the deployment descriptors that are needed for the application exist in the application. For example, all Web welcome files are present, EJB references to another component can be resolved, and tag libraries, servlet filters, and listener classes are also present in the application bundle. Some basic semantic checks are run on the EJB QL that is used in a bean. Additional tests validate for the following:

- Correct packaging of the JAR, WAR, or EAR files
- Method names
- Method signatures
- Matching return values between remote and local interfaces and the bean implementation
- Validation of enumerated values for many of the DTD tags

Code scan testing looks at the J2EE application source code tree for use of nonportable APIs. The `SourceScan` Ant task scans Java and JSP files and for each file reports the nonportable APIs that were found. It reports `PASSED` if nothing was found. The tool looks for methods in known Java packages that are vendor-specific or methods that are in Java packages implementing features that are currently optional in the platform. For example, the tool scans for `com.ibm`, `weblogic`, `com.sun.appserv` or `org.jboss`.

Static verification is a good first step because it can identify problems that may keep the application from deploying or running properly. As time progresses, Sun will continue to update the tool with additional tests from existing specifications as well as to add tests when new specifications become part of the platform.

## Dynamic Testing

The next step in the verification process is dynamic testing, which requires the application to run on the Application Server. There are two parts to dynamic testing:

- Introspection, which compiles the master list of public APIs that must be called
- Instrumentation, which logs the calls to these APIs that are actually made

The list of APIs called and the master list are then used by the reporting tool to generate the coverage numbers.

*Introspection* uses the Java Reflection API to compile the list of all application public EJB APIs. This list of APIs is then pruned to the correct list of APIs that will be required to be invoked. For example, finder methods, setters, and getters for the CMP and CMR fields are not required to be called because the container actually invokes them. The Web components are generated by looking at the file name suffixes (all `.jsp` files are assumed to be JSP pages) as well as by looking at the deployment descriptors for mapping of JSP files and servlets. Finally, there is an exclude list that contains the APIs that should be excluded from the master list. Currently `ejbRemove` is on the exclude list. Any JSP that is included in another JSP using the `include` directive is also automatically excluded from the final list, because it will automatically be called when the parent JSP is called. The exclude list is maintained by Sun Microsystems, and end users are not allowed to modify the list. At the end of this step, we have the complete list of all APIs in the application to be invoked.

*Instrumentation* logs the calls to the public APIs that were actually made. The Application Server has been modified to log all invocations of the EJB public APIs in the application. In addition, it logs whether those invocations were successful. At this point the testing process can determine the use of proprietary APIs. All invocations through the EJB container and the Web container are logged, including the method signature. If an exception is thrown, the exception stack trace is also logged. Currently, the instrumentation tracks public EJB methods, the service method for JSP files and servlets, and the `onMessage` method for message-driven beans. The developer can discover which APIs were called and whether the calls were successful by looking at the reports generated by the reporting tool.

The master list and the instrumentation list are both generated when the server is shut down. The instrumentation list is always generated in a unique file name. All files generated by the instrumentation and introspection runs need to be cleared each time the application is redeployed or a new application is deployed. Use the command `reporttool -clear_results` to do this.

## Outside the Scope of the Toolkit

The Java AVK for the Enterprise checks the application by looking at the EAR file or deployable unit that contains class files and deployment descriptors. The toolkit scans source code looking for use of vendor-specific APIs; however, it does so by scanning for known Java package names. In order to find every use of proprietary server vendor code, the developer would need to exercise all branches of all methods in the application. When vendor-specific code is executed on the Application Server, an exception is thrown. The exception is then trapped by the instrumentation code and reported as an error along with the information about the component that was invoked when the exception was thrown. It is possible for an application that contains vendor-specific code to pass static and dynamic testing if the vendor-specific APIs do not use the known Java package names or if the methods containing vendor-specific code are not exercised at runtime.

## Limitations

There is limited checking for various components in this release of the Java AVK for the Enterprise. Currently, there are static tests for Connectors; however, there is no instrumentation code for runtime support. There is no instrumentation for servlet filters or listeners. One area where portability problems may occur is in the SQL for entity beans that use bean-managed persistence. There are no checks on that SQL, and only a subset of the Java Message Service (JMS) API is instrumented. Finally, there are no tests for Section 25.1.2 of the EJB specification.

## Reporting Tool

The reporting tool summarizes the results from both the static and dynamic runs. The reporting tool also uses the guidelines in the *User's Guide* to determine the status of the coverage numbers. Currently, the guidelines state that for dynamic testing, 80% of the EJB public APIs must be called, and 100% of the Web components must be called. If less than 80% of the EJB methods are called, or if less than 100% of the Web components are called, it is reported as a failure. The reporting tool reads in the results from the instrumentation runs. If there is more than one file for instrumentation results, then all results are merged. The tool compares the master list from the introspection to the final list from the instrumentation, tracking which EJB methods or Web components were called and which were not, and whether any exceptions were thrown. Finally, using XSLT and XSL style sheets, the final reports are generated.

Figure 2-2: Suite Summary

Java AVK for the Enterprise - Suite Summary	
<a href="#">Click to See Static Archive Tests Summary</a>	
<a href="#">Click to See Code Scanning Tests Summary</a>	
EJB Method Coverage	
EAR File	Status
<a href="#">BankEAR</a>	Passed 100% of methods called
<a href="#">ActivitySupplierEAR</a>	Passed 100% of methods called
<a href="#">LodgingSupplierEAR</a>	Passed 100% of methods called
<a href="#">AirlineSupplierEAR</a>	Passed 100% of methods called
<a href="#">OrderProcessingCenterEAR</a>	Passed 90% of methods called
Web Component Coverage	
Status	
<a href="#">Passed</a>	100% of web components called

The results are organized so that the developer can quickly get an idea of the coverage numbers for an application. The EJB and Web component results are summarized on the summary page. To obtain a breakdown of the dynamic results for each, follow the links under the EJB and Web component tables. To obtain a breakdown of the static results, click either the Static Archive

Tests Summary button for archive test results or the Code Scanning Tests Summary button for code scan results.

The static archive test summary results are grouped by specification. The application module table lists the results of the tests that came from the J2EE platform specification; the Web module table lists the results of the tests that came from the JavaServer Pages and Java Servlet specifications; the EJB module table lists the results of the tests that came from the EJB specification; and so on. Each table has up to four categories: Failure, Warning, Not Applicable, and Passed. By default only the Failures and Warnings are reported. Click the Failures link to see each test assertion, including a description of the test that the application did not pass. The guidelines state that all failures must be corrected. The Not Applicable category lists the tests that did not apply to the application. For example, if the application did not contain JSP pages, then all tests from the JavaServer Pages specification would be marked Not Applicable. The guidelines also recommend that all warnings be corrected.

The code scan test summary results are listed by Java and JSP files. All Java and JSP files found are listed along with a status from the scan. All nonportable APIs found are listed for each file.

The dynamic test results are separated for EJB and Web components, and Web services are included within those components, depending on how they were written. That is, if a Web service is implemented as an EJB endpoint, it is listed with the EJB components. The Web components summary page lists all the JSP pages and servlet files that are part of the application and whether each JSP page or servlet was called. In addition, if a JSP page or servlet was called and there was an exception, a link is created to a page that has the stack trace of the exception.

The EJB components are divided by application or EAR file. The guidelines state that 80% of all public EJB methods must be invoked for each EAR file. The summary page shows the percentage for each EAR file. The next level shows the coverage numbers for each bean within each EAR file. And the last page shows the methods for each bean, including signatures, that were called and the methods that were not called. Again, if the method was called and an exception was thrown, the stack trace of the exception is displayed on an exception page.

The developer can quickly and easily see which EJB methods and Web components were not called. At that time, he or she can, if necessary, go back and run tests to invoke those components. If an exception was thrown, the developer has the name of the method or component that caused the problem. Additional information may be available in the server log files.

## Java AVK for the Enterprise: Part of the Development Cycle

The Java AVK for the Enterprise can be incorporated in the development process for J2EE technology-based applications. The ideal way to promote and maintain portability in the application being developed is to add verification to the development process itself. The Java AVK for the Enterprise command-line tools can become part of the regular build cycle. By invoking command-line tools such as the Ant tasks, the Application Server, the deploytool, and the reporting tool, you can test the application for portability. It is up to the application developer to provide a set of automated tests that exercise the application once it is deployed on the Application Server. By running the application through the verification process on a regular basis, you can identify portability problems early in the development cycle, soon after they are introduced.

In the case where the application is already developed, possibly by a third party, the best first step in identifying portability problems is to run the application through code scanning tests, if source code is available, and static archive tests. The Ant tasks are provided for this purpose. This

step will identify problems in portability and will also identify potential issues that may keep the application from deploying or running properly on the Application Server. The next challenge is to run the application on the server and achieve the coverage numbers that are recommended by the toolkit. This is made easier with the information provided by the reports. The developer can quickly identify which APIs have not been covered in the initial testing, thereby pinpointing where further testing is required.

## Resolving Portability Issues

Use of server-specific features can make applications nonportable. However, the J2EE specifications do allow vendors to provide value-added features in their servers. In order to maintain portability and still take advantage of these features, the Java AVK for the Enterprise guidelines allow code branching. The guideline states that the application must provide the same feature on all supported application servers. Therefore, an application can use a feature that is specific to an application server as long as the feature is supported in a portable way for other supported application servers.

The EJB specification states that a bean cannot write directly to the file system. It is reasonable to provide logging within the EJB component, but there is no defined behavior for a logging mechanism in the J2EE platform. Typically, you discover this problem at runtime, when the EJB attempts to write to the file system and the container throws a security violation exception. The ideal solution would be to write a wrapper for logging. In the wrapper, the code can use a servlet that is acting as a logging server or a message-driven bean to which it sends messages that are logged. Further, the wrapper code can use a generic logging mechanism such as `log4j`. Several application servers do provide logging mechanisms, so the application can take advantage of that feature by testing if a server-specific logging class exists. In this way the application can take advantage of logging features provided by specific application servers, if available, and if not, provide the same functionality using nonproprietary code.

## Chapter 3

# Summary

The J2EE platform provides developers with the ability to write applications that will run on compatible J2EE application servers. The Java AVK for the Enterprise helps developers verify that their applications are portable to these servers. It includes both static tests, which test that the application meets the behavior defined in the specifications for applications and scan source code for nonportable APIs, and dynamic tests, which log API invocations and whether system exceptions are thrown indicating the possible use of proprietary APIs. The Java AVK for the Enterprise toolkit provides a set of guidelines that the reporting tool uses when presenting the results of test runs. Using the toolkit as part of the development process can help identify portability issues early in the development cycle. Uses of proprietary APIs are discovered early, and the guidelines provide a way to take advantage of server optimizations while maintaining portability.

For more information on the Java Verification Program, see <http://java.sun.com/j2ee/verified>.

**SUN™** Copyright 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054 U.S.A. All rights reserved.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Sun, Sun Microsystems, the Sun logo, Solaris, Java, Enterprise JavaBeans, EJB, JavaServer Pages, JSP, Java Verified, and J2EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Products covered by and information contained in this service manual are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

**SUN™** Copyright 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, Californie 95054 États-Unis. Tous droits réservés.

Droits du gouvernement américain, utilisateurs gouvernementaux - logiciel commercial. Les utilisateurs gouvernementaux sont soumis au contrat de licence standard de Sun Microsystems, Inc., ainsi qu'aux dispositions en vigueur de la FAR [ (Federal Acquisition Regulations) ] et des suppléments à celles-ci.

Cette distribution peut comprendre des composants développés par des tierces parties.

Sun, Sun Microsystems, le logo Sun, Java, Enterprise JavaBeans, EJB, JavaServer Pages, JSP, Java Verified, et J2EE sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux États-Unis et dans d'autres pays.

Les produits qui font l'objet de ce manuel d'entretien et les informations qu'il contient sont régis par la législation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes biologiques et chimiques ou du nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des États-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régi par la législation américaine en matière de contrôle des exportations ("U.S. Commerce Department's Table of Denial Orders" et la liste de ressortissants spécifiquement désignés ("U.S. Treasury Department of Specially Designated Nationals and Blocked Persons"), sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.



Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA Phone 800 786-7638 or +1 512 434-1577 Web [sun.com](http://sun.com)



**Sun Worldwide Sales Offices:** Africa (North, West and Central) +33-13-067-4680, Argentina +5411-4317-5600, Australia +61-2-9844-5000, Austria +43-1-60563-0, Belgium +32-2-704-8000, Brazil +55-11-5187-2100, Canada +905-477-6745, Chile +56-2-3724500, Colombia +571-629-2323, Commonwealth of Independent States +7-502-935-8411, Czech Republic +420-2-3300-9311, Denmark +45 4556 5000, Egypt +202-570-9442, Estonia +372-6-308-900, Finland +358-9-525-561, France +33-134-03-00-00, Germany +49-89-46008-0, Greece +30-1-618-8111, Hungary +36-1-489-8900, Iceland +354-563-3010, India-Bangalore +91-80-2298989/2295454; New Delhi +91-11-6106000; Mumbai +91-22-697-8111, Ireland +353-1-8055-666, Israel +972-9-9710500, Italy +39-02-641511, Japan +81-3-5717-5000, Kazakhstan +7-3272-466774, Korea +822-2193-5114, Latvia +371-750-3700, Lithuania +370-729-8468, Luxembourg +352-49 11 33 1, Malaysia +603-21161888, Mexico +52-5-258-6100, The Netherlands +00-31-33-45-15-000, New Zealand-Auckland +64-9-976-6800; Wellington +64-4-462-0780, Norway +47 23 36 96 00, People's Republic of China-Beijing +86-10-6803-5588; Chengdu +86-28-619-9333; Guangzhou +86-20-8755-5900; Shanghai +86-21-6466-1228; Hong Kong +852-2202-6688, Poland +48-22-8747800, Portugal +351-21-4134000, Russia +7-502-935-8411, Singapore +65-6438-1888, Slovak Republic +421-2-4342-9485, South Africa +27 11 256-6300, Spain +34-91-596-9900, Sweden +46-8-631-10-00, Switzerland-German 41-1-908-90-00; French 41-22-999-0444, Taiwan +886-2-8732-9933, Thailand +662-344-6888, Turkey +90-212-335-22-00, United Arab Emirates +9714-3366333, United Kingdom +44-1-276-20444, United States +1-800-555-9SUN or +1-650-960-1300, Venezuela +58-2-905-3800 03/03 FE1960-0