NonStop Server for Java 6.0 Programmer's Reference

HP Part Number: 546595-006
Published: August 2013
Edition: J06.04 and all subsequent J-series RVUs and H06.15 and all subsequent H-series RVUs

Printed in the US

Contents

Tables

Examples

About this Manual

This section explains these subsections:

Manual Information

Abstract

This document describes the HP NonStop ™ Server for Java ™, based on Java Platform Standard Edition 6.0, a Java environment that supports compact, concurrent, dynamic, and portable programs for the enterprise server. The NonStop Server for Java uses the HP NonStop operating system to add scalability and program persistence to the Java environment.

Product Version

HP NonStop Server for Java, based on Java Platform Standard Edition 6.0.

Supported Hardware

All HP Integrity NonStop NS-series (TNS/E) servers.

Supported Release Version Updates (RVUs)

This manual supports J06.04 and all subsequent J-series RVUs and H06.15 and all subsequent H-series RVUs, until otherwise indicated by its replacement publications. On J-series platform with IP CLIM configuration, NSJ6.0 is supported only on J06.06 and later RVUs. On H-series platform with IP CLIM configuration, NSJ 6.0 issupported only on H06.17 and later RVUs.

Intended Audience

This *NonStop Server for Java 6.0 Programmer's Reference* is for all Java programmers who want to use Java on HP Integrity NonStop systems.

For programmers new to Java, this document refers to documentation from Oracle, which explains what the J2SE Development Kit (JDK) contains and where to learn more about the language in general. This NonStop Server for Java Programmer's Reference does not teach Java or provide detailed information about the JDK.

For experienced Java programmers who want to access SQL databases with NonStop SQL/MP or NonStop SQL/MX, this manual refers you to the *JDBC Driver for SQL/MP Programmer's Reference* and the *JDBC Driver forSQL/MX Programmer's Reference*, respectively.

For Java programmers who want to determine application performance and behavior, this manual describes how to use the HPeprof and HPROF profilers to obtain runtime information for a NonStop Server for Java 6.0 program and also analyze profile data using the HPjmeter.

For developers new to NonStop systems, this document:

- Explains NonStop system fundamentals as they apply to the NonStop Server for Java 6.0 product.
- Refers to other appropriate NonStop system documentation.

New and Changed Information

Changes added to this revision part number 546595–006 are:

- "Feature Changes" (page 8)
- "Document Changes" (page 8)

Feature Changes

From NSJ 6.0 SPR — T2766H60^ACH and later versions provide the option `-Djava.security.nativeRNG` option. Enabling this option reduces the startup time for the first invocation of `SecureRandom`. This option can be enabled using the command:

```
-Djava.security.nativeRNG=true
```

Document Changes

The document changes are:

- Added section "Java Signal Handlers" (page 69).
- Added section "Change in Loading of `.hotspot_compiler` and `.hotspotrc files`" (page 70).
- Added section "Oracle's Implementation" (page 79).
- Added unsupported information in "Supported and Unsupported Features of NonStop Server for Java 6.0" (page 87).
- Updated the section "Node Agent" (page 90).
- Provided references to the *HPjmeter 4.2 User's Guide* in the appropriate sections.

Changes added to this revision part number 546595-005 are:

- "Feature Changes" (page 8)
- "Documentation Changes" (page 8)

Feature Changes

NonStop Server for Java 6.0 supports the following feature:

- *address = transport-address-for-this-connection* is updated to support a range of port values, to be specified withcommand line options to run a debuggee.

Documentation Changes

The documentation changes are:

- Updated address=transport-address-for-this-connection for "java Command Line Options to Run a Debuggee" (page 55).
- Added new Note in "java Command Line Options to Run a Debuggee" (page 55).
- Added new content to the Introduction of "Application Profiling" (page 76).
- Added new sub-section "Attaching to the JVM Agent of a Running Application" (page 89).

- Updated the content for the section "Managing Node Agents on a NonStop Operating System" (page 89).
- Added new sub-section "Checking for Application Paging Problems" (page 89).

Changes added to this revision part number 546595-004 are:

- "Feature Changes" (page 9)
- "Documentation Changes" (page 9)

Feature Changes

- None

Documentation Changes

The documentation changes are:

- Added the new section under "Implementation Specifics" (page 42):
  - "Using the Guardian Printer" (page 51)

Changes added to this revision part number 546595-003 are:

- "Feature Changes" (page 9)
- "Documentation Changes" (page 9)

Feature Changes

NonStop Server for Java 6.0 supports the following feature:

- Support for the `HeapDumpOnly` option.

  This option is used to observe memory allocation in a running Java application by taking snapshots of the heapover time.

Documentation Changes

The documentation changes are:

- Added information on "HeapDumpOnly option" (page 64) under "Java GC Profiling" (page 64).
- Added new section "ThreadDumpPath Support" (page 52).
- Removed the 'Analyzing Garbage Collection data' section from Appendix B.
- Updated references and links to Java documentation to point to the correct Oracle websites.

Changes added to this revision part number 546595-002 are:

- "Feature Changes" (page 9)
- "Documentation Changes" (page 10)

Feature Changes

NonStop Server for Java 6.0 supports the following features:

- The `JREHOME` environment variable need not be set to launch Java installed in a nonstandard location. The Javacore classes will be picked up based on the location of the Java executable in the Java Development Kit installation.

NOTE: If the `JREHOME` environment variable is set, Java installed in the location pointed by `$JREHOME` directory will take precedence.

- The `Dnsk.java.fastExec=true` option reduces the latency of the Java `Runtime.exe()` API, which isused to spawn a child process.
- The Dynamic saveabend file creation feature helps to create a saveabend file of a running Java process without aborting the Java process. Thus, the Java process continues normal execution after a short pause, during which the saveabend file is created.
- The `-XX:ThreadTimeSlice[=T]` option specifies the time slice for JVM-forced preemptive thread scheduling.
- The `-XX:EnableCompilerSafepoints` option improves the Garbage Collection performance of an application on systems running J-series.

NOTE: The `-XX:EnableCompilerSafepoints` option must not be used if the application uses a selectable segment.

Documentation Changes

The documentation changes are:

- Updated information on the `JREHOME` environment variable in the following sections:
  - "Verifying the Java Installation" (page 32)
  - "ENV" (page 35)
- Added a note on the `JREHOME` variable in the "JREHOME" (page 39) section.
- Updated the "Calling Java Methods from C or C++" (page 45) section.
- Added information on the `XX:ThreadTimeSlice` option in the "Multithreaded Programming" (page 47) section.
- Added a note on the "Preemptive User Threads On NonStop Server For Java" (page 52) feature.
- Added the following sections:
  - "Dynamic Saveabend File Creation" (page 52)
  - "Creating Child Process Using the `-Dnsk.java.fastExec=true` Option" (page 52)
  - "GC Log Rotation" (page 78)
- Updated the procedure required to set up monitoring for a live Java applications in the "Monitoring live Java applications" (page 76) section.
- Added a note on GC log rotation in the "Analyzing Garbage Collection Data" (page 79) section.
- Provided references to the HPjmeter 4.0 User's Guide in the appropriate sections.
- Replaced 'Appendix B: Addendum to HPjmeter 3.1 Users Guide' with 'Appendix B: Addendum to HPjmeter 4.0 Users Guide'.

Changes added to this revision part number 546595-001 are:

- "Feature Changes" (page 11)
- "Documentation Changes" (page 12)

Feature Changes

NonStop Server for Java 6.0 supports the following features:

- Non-Blocking I/O for OSS regular files feature in Java applications. By default, this feature is enabled on NSJ6.0. It allows performing regular I/O operations on multiple OSS files simultaneously, thereby enhancing the performance of standard I/O operations on OSS regular files. To disable this feature, use the `-Dnsk.java.nonblocking=false` option.
- Java Programming Language and Tools API
  - Framework for compiling source files from within an application.
  - Standardized capabilities of the existing annotation processing tool.
- Security and Networking
  - XML digital signatures.
  - Smart Card I/O API.
  - Default cookie manager implementation.
  - Internationalization of domain names and resource identifiers.
  - Programmatic access to network parameters.
- Java Management Extensions (JMX)
  - Improved JMX Monitor API using a thread pool.
  - MBean descriptor support beyond Model MBean.
  - User-defined MBeans for applications.
- Serviceability
  - Continued investment on diagnosing, monitoring, and management.
  - Improved monitor and diagnostics for locks.
  - Improved diagnosing of `java.lang.OutOfMemoryError`
  - Improved thread dumps of running applications.
  - JVM TI and JPDA improvements.
- Includes all -X options supported in earlier versions of Java.
- Supports APIs that are core to the Java SE platform, among them Remote Method Invocation (RMI), Non-Blocking I/O (NIO) APIs and the Collections Framework. For information on Java SE 6 API Specification, see the Java™ Platform, Standard Edition 6 API Specification.
- Uses headless mode. For more information, see the Using Headless Mode in the Java SE Platform.
- Supports JavaTM Platform, Standard Edition 6 JDK. For information on JDK, see the JDK 6 Documentation.

  For information about Java SE 6 features, see Features and Enhancements.

  The NSJ6 HotSpot JVM (NSJ6 JRE) provides the following functionalities:
  - Improved performance.
  - Java Platform Debugger Architecture (JPDA)

- Nonblocking I/O APIs
- `-Xeprof` option
- `-XX:+HeapDump` option
- `-XX:+HeapDumpOnOutOfMemoryError` command line option

Documentation Changes

This is a new manual for NonStop Server for Java 6.0.

Document Organization

This document includes the following sections:

Table 1 Document Structure

Section	Description
"Introduction to NonStop Server for Java 6.0" (page 22)	Explains NonStop software fundamentals as they apply to NonStop Server for Java 6.0, describes associated Java products on the NonStop system, and points out J2SE, JDK highlights.
"Getting Started" (page 30)	Explains the prerequisites for using the NonStop Server for Java 6.0 for readers new to the Java language or HP NonStop Open System Services (OSS) . Includes step-by-step instructions for running a simple program and for verifying the NonStop Server for Java installation.
"Installation and Configuration" (page 37)	Explains installation and configuration requirements, the NonStop Server for Java 6.0 directory structure, how to run Java tools, and how to verify the installation.
"Implementation Specifics" (page 42)	Explains issues and features that are unique to the NonStop Server for Java 6.0 implementation, such as JNI, IEEE floating-point implementation, Java Print Service, multithreading programming, JavaBeans, remote debugging, garbage collectors, and so forth.
"Transactions" (page 72)	Explains how the NonStop Server for Java 6.0 allows to work with transactions.
"Application Profiling" (page 76)	Describes the application profiling environment and how to use HPeprof (that is Xeprof) and HPROF agent. Also, describes the usage of HPjmeter profile data analysis tool.
"Migrating Applications" (page 80)	Explains how to change your applications that run with earlier versions of NonStop Server for Java.
"Supported and Unsupported Features of NonStop Server for Java 6.0" (page 87)	Summarizes the supported and unsupported features of NonStop Server for Java 6.0.
"Addendum to HPjmeter 4.2 User's Guide" (page 88)	Provides instructions for using the HPjmeter tool on the HP NonStop TM platform.

Notation Conventions

Bold Type

Bold type within text indicates terms defined in the Glossary. For example:

abstract class

Computer Type

`Computer type` letters within text indicate keywords, reserved words, command names, class names, and method names; enter these items exactly as shown. For example:

```
myfile.c
```

Italic Computer Type

`Italic computer type` letters in syntax descriptions or text indicate variable items that you supply. For example:

```
pathname
```

[ ] Brackets

Brackets enclose optional syntax items. For example:

```
jdb [options]
```

A group of items enclosed in brackets is a list from which you can choose one item or none. Items are separated by vertical lines. For example:

```
where [threadID|all]
```

{ } Braces

A group of items enclosed in braces is a list from which you must choose one item. For example:

```
-c identity {true|false}
```

| Vertical Line

A vertical line separates alternatives in a list that is enclosed in brackets or braces. For example:

```
where [threadID|all]
```

... Ellipsis

An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times. For example:

```
print {objectID|objectName} ...
```

An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times. For example:

```
dump objectID ...
```

Punctuation

Parentheses, commas, equal signs, and other symbols not previously described must be entered as shown. For example:

```
-D propertyName=newValue
```

Item Spacing

Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or comma. If there is no space between two items, spaces are not permitted. In the following example, spaces are not permitted before or after the period:

```
subvolume-name.filename
```

Line Spacing

If the syntax of a command is too long to fit on a single line, each line that is to be continued on the next line ends with a back slash (\ ) and each continuation line begins with a greater-than symbol (>). For example:

```
/usr/bin/c89 -c -g -I /usr/tandem/java/include \
> -I /usr/tandem/java/include/oss -I . \
> -Wextensions -D_XOPEN_SOURCE_EXTENDED=1 jnative01.c
```

Related Reading

For background information about the features described in this guide, see the following documents:

- "NonStop Server for Java Library" (page 15)
- "NonStop System Computing Documents" (page 15)
- "Oracle Java Documents" (page 19)

NOTE: To find white papers with the NonStop Server for Java documentation, see the HP NonStop Technical Library at Business Support Center (BSC).

NonStop Server for Java Library

In addition to this manual, the NonStop Server for Java library includes:

- *NonStop Server for Java 6.0 Tools Reference*

  Provides a page for each Java tool and links to the Oracle website where the detail information on that site applies.

- *NonStop Server for Java API*

  Provides the API description for these packages:

  - com.tandem.os
  - com.tandem.tmf
  - com.tandem.util

NonStop System Computing Documents

The following NonStop system computing documents are available in the HP NonStop Technical Library at Business Support Center (BSC).

- Additional Java Based Products
  - *JDBC Driver for SQL/MP Programmer's Reference*

    This document describes how to use the JDBC Driver for SQL/MP (JDBC/MP) on NonStop servers .JDBC/MP gives Java applications JDBC access to NonStop SQL databases through SQL/MP. JDBC/MP conforms to the JDBC API from Oracle.

  - *JDBC Driver for SQL/MX Programmer's Reference*

    This document describes how to use the JDBC Driver for SQL/MX (JDBC/MX) on NonStop servers. JDBC/MX gives Java applications JDBC access to NonStop SQL databases through SQL/MX. JDBC/MX conforms to the JDBC API from Oracle.

  - *JToolkit for NonStop Servers Programmer's Reference*

    This documentation describes the JToolkit for NonStop Servers, a set of additional features that work in conjunction with NonStop Server for Java. These features include:

    - Enscribe API for Java, which enables an application programmer to perform operations on Enscribe files.
    - Pathsend API for Java, which provides support for creating Pathway clients by using the Pathsend interface.
    - Pathway API for Java, which provides for creating Pathway servers by using $RECEIVE.
    - Scalable TCP/IP (SIP) , where SIP provides a transparent method of giving scalability and persistence to a network server written in Java.
    - `ddl2java` Tool, which generates Java code based on Data Definition Language (DDL) Dictionary definitions.

  - *WebLogic Server for the HP NonStop Server Platform Guide*

    This manual describes the installation, configuration, and management of the BEA WebLogic Server on HP NonStop servers.

- *C/C++ Programmer's Guide*

  Describes the HP implementation of the C and C++ programming languages for NonStop systems. For a NonStop Server for Java JDBC driver to access a SQL/MP database, the C

compiler for the HP NonStop Open System Services (OSS) environment (c89) must be installed and configured correctly. Also, you might need this document if you use the Java Native Interface (JNI) to communicate between Java and a native library.

- *DLL Programmer's Guide for TNS/E Systems*

  Provides an introduction to the process of creating and using Dynamic-Link Libraries (DLLs) on TNS/Esystems.

- *eld Manual*

  Describes how programmers can use eld, the object file linker to create loadfiles that run on TNS/E systems.

- *iTP Secure WebServer System Administrator's Guide*

  Provides an overview of NSJSP with reference to *NonStop Servlets for JavaServer Pages (NSJSP) System Administrator's Guide.*

- *Kernel-Managed Swap Facility (KMSF) Manual*

  Explains how to use the NSKCOM tool to create additional or larger swap files, which might be necessary before you can execute the Java HotSpot virtual machine.

- *Native Inspect Manual*

  Explains how to use the Native Inspect, which is a system-level command line symbolic debugger that can be used to debug TNS/E native processes and snapshots.

- *NonStop Servlets for JavaServer Pages (NSJSP) System Administrator's Guide*

  Describes how to develop NonStop Servlets for JavaServer Pages and use them with the iTP Secure WebServer.

- ODBC (Open Database Connectivity) Documents

  The following documents describe products that allow programs written for the Microsoft® Open Database Connectivity (ODBC) product to access NonStop SQL/MX and NonStop SQL/MP.

  - *ODBC Server Reference Manual*

    Contains reference information for the NonStop ODBC Server, describes the NonStop ODBC or SQL/MP Server features and the statements that the NonStop ODBC Server supports, and explains how the NonStop ODBC Server accommodates the differences between the ODBC or SQL/MP Server and NonStop SQL/MP.

  - SQL/MX Connectivity Service Manual

    Describes how to install and manage the SQL/MX Connectivity Service, commonly known as MXCS. This product enables applications developed for the Microsoft® Open Database Connectivity (ODBC) application programming interface and other connectivity APIs to use SQL/MX to access NonStop SQL databases on an HP NonStop system.

- *Open System Services Installation Guide*

  Explains how to install the OSS environment.

- *Open System Services Porting Guide*

  Includes information on differences between OSS POSIX Threads and Standard POSIX Threads that are useful for migrating multithreaded native libraries used with NonStop Server for Java 3.1 to NonStop Server for Java 5.

- *Open System Services Programmer's Guide*

  Describes how to write applications in C for the OSS environment and includes information on using standard POSIX threads and TMF transaction jacket routines.

- *Spooler FASTP Network Print Processes Manual*

  Describes the Spooler FASTP network print processes, which provide the ability to use the Spooler subsystem to print documents on a printer attached to a particular type of local area network (LAN) or wide area network (WAN).

- *Spooler Utilities Reference Manual*

  Presents a general introduction to the Spooler subsystem and describes the Spooler utilities—Peruse, Spoolcom, Font, RPSetup — presenting the complete syntax for these utilities.

- SQL/MP Documents

  The NonStop Server for Java includes JDBC drivers that enable Java programs to interact with NonStop SQL/MP databases.

  - *Introduction to SQL/MP*

    Provides an overview of the SQL/MP product.

  - *SQL/MP Reference Manual*

    Explains the SQL/MP language elements, expressions, functions, and statements.

  - *SQL/MP Installation and Management Guide*

    Explains how to plan, install, create, and manage a NonStop SQL/MP database; describes the syntax of installation and management commands; and describes NonStop SQL/MP catalogs and file structures.

  - *SQL/MP Query Guide*

    Explains how to retrieve and modify data from a NonStop SQL/MP database and how to analyze and improve query performance.

  - *SQL/MP Report Writer Guide*

    Explains how to use report writer commands and SQL command interface (SQLCI) options to design and produce reports.

  - *SQL/MP Version Management Guide*

    Explains the rules governing version management for the NonStop SQL software, catalogs, objects, messages, programs, and data structures.

  - *SQL/MP Messages Manual*

    Explains NonStop SQL/MP messages for the conversational interface, the application programming interface (API), and the utilities.

  - *SQL/MP Programming Manual for C*

    Describes the SQL/MP programmatic interface for ANSI C. Also describes embedded SQL statements used in C applications.

  - *SQL/MP Programming Manual for COBOL*

    Describes the SQL/MP programmatic interface for ANSI COBOL. Also describes embedded SQL statements used in COBOL applications.

  - See also *SQL Supplement for H-series RVUs*.

- SQL/MX Documents

  NonStop Server for Java includes JDBC drivers that enable Java programs to interact with NonStop SQL databases with SQL/MX.

  - *SQL Supplement for H-series RVUs*

    This supplement provides information about HP NonStop SQL/MP™ and HP NonStop SQL/MX that is specific to H-series release version updates (RVUs). The currently published SQL/MP and SQL/MX manuals are intended for G-series users. To use G-series manuals, H-series users should review and understand the exceptions noted in this supplement.

  - *SQL/MX Guide to Stored Procedures in Java*

    Describes how to develop and deploy stored procedures in Java (SPJs) in SQL/MX.

  - *SQL/MX Quick Start*

    Describes basic techniques for using SQL in the SQL/MX conversational interface (MXCI). Also includes information about installing the sample database.

  - *SQL/MX Comparison Guide for SQL/MP Users*

    Compares SQL/MP and SQL/MX.

  - *SQL/MX Installation and Management Guide*

    Describes how to install and manage SQL/MX on a NonStop server.

  - *SQL/MX Glossary*

    Explains the terminology used in SQL/MX documentation.

  - *SQL/MX Query Guide*

    Explains query execution plans and how to write optimal queries for SQL/MX.

  - *SQL/MX Reference Manual*

    Describes SQL/MX language elements (such as expressions, predicates, and functions) and the SQL statements that can be run in MXCI or in embedded programs. Also describes MXCI commands and utilities.

  - *SQL/MX Messages Manual*

    Describes SQL/MX messages.

  - *SQL/MX Programming Manual for C and COBOL*

    Describes the SQL/MX programmatic interface for ANSI C and COBOL.

  - *SQL/MX Data Mining Guide*

    Describes the SQL/MX data structures and operations needed for the knowledge-discovery process.

  - *SQL/MX Queuing and Publish/Subscribe Services*

    Describes how SQL/MX integrates transactional queuing and publish/subscribe services into its database infrastructure.

- *TCP/IP Configuration and Management Manual*

  Describes the installation, configuration, and management of the NonStop TCP/IP product (see TCP/IP). For Java's Remote Method Invocation (RMI) application program interface (API) to function correctly, TCP/IP and its component, Domain Name Server (DNS), must be installed and configured correctly.

- *TCP/IPv6 Configuration and Management Manual*

  Describes how to configure and manage the NonStop TCP/IPv6 subsystem on a NonStop S-series server. For IPv6 support, you must use the NonStop TCP/IPv6 subsystem with NonStop Server for Java.

- TMF Documents

  - *TMF Introduction*

    Introduces the concepts of transaction processing and the features of the HP NonStop Transaction Management Facility (TMF) product.

  - *TMF Application Programmer's Guide*

    Explains how to design requester and server modules for execution in the TMF programming environment and describes system procedures that are helpful in examining the content of TMF audit trails.

- *TS/MP Pathsend and Server Programming Manual*

  HP NonStop Transaction Services/MP supports the creation of Pathway servers to access SQL/MP, SQL/MX, or Enscribe databases in an online transaction processing (OLTP) environment. Using the NonStop Server for Java, you can communicate with Pathway servers from Java programs. This document explains how to write Pathway programs, including Pathsend processes.

- *TS/MP System Management Manual*

  Provides information about Pathway applications. You must be familiar with Pathway applications in order to configure Scalable TCP/IP (SIP) applications.

Oracle Java Documents

For Java SE 6 documentation, visit the Oracle website http://www.oracle.com/technetwork/java/javase/overview/index-jsp-136246.html.

The following documents were available on Oracle websites when this document was published, but HP cannot guarantee their continuing availability. If a link to a Oracle document fails, use the Oracle documentation zipped up on the distribution CD.

- JNDI document

  (http://www.oracle.com/technetwork/java/index-jsp-137536.html)

- JDBC documents

  (http://download.oracle.com/javase/6/docs/technotes/guides/jdbc/)

- Java Print Service (JPS) document

  (http://download.oracle.com/javase/6/docs/technotes/guides/jps/index.html)

- Java Transaction API (JTA) document

  (http://www.oracle.com/technetwork/java/javaee/tech/jta-138684.html)

- Java Transaction Service (JTS) document

  (http://www.oracle.com/technetwork/java/javaee/tech/jts-140022.html)

- Java Remote Method Invocation (RMI) document

  (http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html)

Revision History

Part Number	Product Version	Published
546595-001	HP NonStop Server for Java 6.0	March 2010
546595-002	HP NonStop Server for Java 6.0	April 2010
546595-003	HP NonStop Server for Java 6.0	January 2011
546595-004	HP NonStop Server for Java 6.0	April 2011
546595-005	HP NonStop Server for Java 6.0	November 2011
546595-006	HP NonStop Server for Java 6.0	August 2013

HP Encourages your Comments

HP encourages your comments concerning this document. We are committed to providing documentation that meets your needs. Send any errors found, suggestions for improvement, or compliments to **docsfeedback@hp.com.**

Include the document title, part number, and any comment, error found, or suggestion for improvement you have concerning this document.

Printing this Document

To print this document, visit the HP NonStop Technical Library at Business Support Center (BSC). For a list of the sections, see Table 1 (page 12).

NOTE: Some browsers require that you reduce the print size to print all the text displayed on the screen.

Abbreviations

ANSI.	American National Standards Institute
AWT.	Abstract Window Toolkit
API.	Application Program Interface
ASCII.	American Standard Code for Information Interchange
BDK.	JavaBeans Development Kit
CD.	Compact Disk
CGI.	Common Gateway Interface
COBOL.	Common Business-Oriented Language
CORBA.	Common Object Request Broker Architecture
CPU.	Central Processing Unit
DDL.	Data Definition Language
DNS.	Domain Name Server
DSA.	Digital Signature Algorithm
GUI.	Graphical User Interface
HTML.	Hypertext Markup Language
HTTP.	Hypertext Transfer Protocol
IDL.	Interface Definition Language
IEC.	International Electrotechnical Committee
IEEE.	Institute for Electrical and Electronic Engineers
IETF.	Internet Engineering Task Force

ISO.	International Organization for Standardization
J2SE.	Java 2 Platform Standard Edition
JAAS.	Java Authentication and Authorization Service
JAR.	Java Archive
JCK.	Java Conformance Kit
JDK.	J2SE Development Kit
JFC.	Java Foundation Classes
JDBC.	Java Database Connectivity
JDBC/MP.	JDBC Driver for SQL/MP
JDBC/MX.	JDBC Driver for SQL/MX
JNDI.	Java Naming and Directory Interface
JNI.	Java Native Interface
JPS.	Java Print Service
JRE.	J2SE Runtime Environment
JTA.	Java Transaction API
JTS.	Java Transaction Service
JVM TI.	Java Virtual Machine Tool Interface
LAN.	Local Area Network
NonStop TS/MP.	NonStop Transaction Services/MP
ODBC.	Open Database Connectivity
OLTP.	Online Transaction Processing
OMG.	Object Management Group
OSS.	Open System Services
OTS.	Object Transaction Services
POSIX.	Portable Operating System Interface X
Pthread.	POSIX thread
RDF.	Remote Database Facility
RMI.	Remote Method Invocation
RVU.	Release Version Update
SIP.	Scalable TCP/IP
SPI.	Service Provider Interface
SPJ.	Stored Procedure in Java
SQLJ.	embedded SQL in Java programs
SQL/MP.	Structured Query Language/MP
SQL/MX.	Structured Query Language/MX
TACL.	HP Tandem Advanced Command Language
TCP/IP.	Transmission Control Protocol/Internet Protocol
TMF.	Transaction Management Facility
URL.	Uniform Resource Locator
VM.	Virtual Machine
WWW.	World Wide Web

1 Introduction to NonStop Server for Java 6.0

The HP NonStop Server for Java 6.0 is a Java environment that supports compact, concurrent, dynamic, portable programs for the enterprise server. The NonStop Server for Java 6.0 requires the HP NonStop Open System Services (OSS) environment. The NonStop Server for Java 6.0 uses the HP NonStop operating system to add the NonStop system fundamentals of scalability and program persistence to the Java environment.

NonStop Server for Java 6.0 is based on the Java Platform Standard Edition (Java SE) 6.0 reference Java implementation for Solaris, licensed by HP from Sun Microsystems, Inc. With the introduction of support for Java SE 6.0, the product's informal name became the NonStop Server for Java 6.0.

The NonStop Server for Java 6.0 is a conformant version of the Sun Microsystems Java SE 6.0 because it is a fully compliant headless JDK as defined by passing the 6b version of the Java Conformance Kit (JCK). NonStop Server for Java 6.0 is branded as "Java Compatible".

NonStop Server for Java 6.0 supports the Sun Microsystems enhancement to AWT called "headless support" that allows a Java VM to indicate whether a display, keyboard, sound, or mouse operation can be supported in a graphics environment. Because of the nonvisual nature of NonStop servers, NonStop Server for Java 6.0 is always a headless Java VM. For implementation-specific information, see "Headless Support" (page 42).

The NonStop Server for Java 6.0 supports the Java Platform Debugger Architecture (JPDA), which consists of three interfaces designed for use by debuggers in development environments for desktop systems. This is described in the Sun Microsystems documentation for JPDA (http://docs.oracle.com/javase/6/docs/technotes/guides/jpda/).

This NonStop Server for Java 6.0 Programmer's Reference identifies the changes in the HP adaptation of the reference Java implementation, emphasizing the differences between the reference implementation and the NonStop Server for Java 6.0. For more information on the standard architecture, see the Sun Microsystems documentation.

This section explains these subjects:

Java HotSpot Server Virtual Machine

The NonStop Server for Java 6.0 implements the HotSpot server compiler and the runtime Java HotSpot virtual machine. The HotSpot Server Java VM, provides a fast, reliable technology for the enterprise server environment. For more information about this VM, see The Java HotSpot Server VM (http://docs.oracle.com/javase/6/docs/technotes/guides/vm/index.html).

Also, For more information, see

- White paper, Java HotSpot Virtual Machine, v1.4.1
  (http://java.sun.com/products/hotspot/docs/whitepaper/Java_Hotspot_v1.4.1/Java_HSpot_WP_v1.4.1_1002_1.html).
- Guide, Java Virtual Machines
  (http://docs.oracle.com/javase/6/docs/technotes/guides/vm/index.html)

NOTE: NonStop Server for Java 6.0 does not support the client Java VM, the Deadlock Detection utility, signal chaining, parallel copying collector, mostly concurrent mark-sweep collector, garbage collector ergonomics, and 64-bit operation described in white papers and other information sources. For more information about garbage collection, see "Garbage Collection (GC)" (page 59).

Java Standard Edition Development Kit (JDK) Highlights

The NonStop Server for Java 6.0 consists of the following standard Java components (and the HP extensions described else where in this document):

- Java virtual machine (VM) based on the J ava SE Runtime Environment (JRE) 6.0
- Core Packages of the Java ™ SE Development Kit (JDK) 6.0
- Standard Java SE 6.0 tools as documented in the *NonStop Server for Java 6.0 Tools Reference Pages*. All standard tools are supported, except graphical-user interface (GUI)—such as `appletviewer`, `policytool`, and `jconsole`—and client-side tools—such as `javaws` and `HtmlConverter`. Experimental tools are not supported.

NonStop Server for Java 6.0 supports the JDK 6.0 API packages (`java`, `javax`, and `org` packages) described in the Java Platform Standard Edition 6.0 API Specification (http://download.oracle.com/javase/6/docs/api/index.html).

Because of the nonvisual nature of NonStop servers, the NonStop Server for Java 6.0 supports the following packages according to "Headless Support" (page 42).

Table 2 Headless Support for Visual Feature Packages

Package	Description
java.awt and AWT-related packages	Contains all of the classes for creating user interfaces and for painting graphics and images.
javax.accessibility	Defines a contract between user interface components and technology that provides access to those components.
javax.sound and Sound-related packages	Provides an API for capturing, processing, and playing back audio and MIDI (Musical Instrument Digital Interface) data. This API is supported by a sound engine that provides high-quality audio mixing and MIDI synthesis capabilities for the platform.
javax.swing and Swing-related packages	Provides a set of Java components that, as much as possible, work in the same manner on all platforms.

If code that depends on a keyboard, display, mouse, or sound processing is called, NonStop Server for Java 6.0 throws a java.awt.HeadlessException.

For more information on the Core Packages of Java SE Development Kit 6.0, see "Oracle Java Documents" (page 19).

Java Naming and Directory Interface (JNDI) Highlights

The Java Naming and Directory Interface (JNDI) provides naming and directory functionality to Java programs. It is independent of any specific directory service implementation; therefore, it allows a variety of directories to be accessed in a common way.

The JNDI architecture consists of an Application Programming Interface (API) and a Service Provider Interface (SPI). Java programs use the JNDI API to access a variety of naming and directory services. The JNDI SPI enables a variety of naming and directory services to be plugged in transparently, allowing Java programs that use the JNDI API to access their services.

NonStop Server for Java supports JNDI, which is a standard interface in Java implementations.

For more information about the JNDI, see the Sun Microsystems Java Naming and Directory Interface 1.1.1 Specification (http://docs.oracle.com/cd/E17802_01/products/products/jndi/javadoc/overview-summary.html).

IPv6 Support

The Java SE JRE 6.0 release includes Internet Protocol version (IPv6) support in Java Networking. This support makes Java SE compliant with the following standards (RFCs):

- RFC2373: IPv6 Addressing Architecture
- RFC 2553: BasicSocket Interface Extensions for IPv6
- RFC 2732: Format for Literal IPv6 Addresses in URLs

Since the Java SE JRE does not support raw sockets, RFC 2292 (Advanced Sockets API for IPv6) is not supported in this release.

NOTE: IPv6 support is supplied only if you use the NonStop TCP/IPv6 subsystem with NonStop Server for Java.

Associated Java Based Products

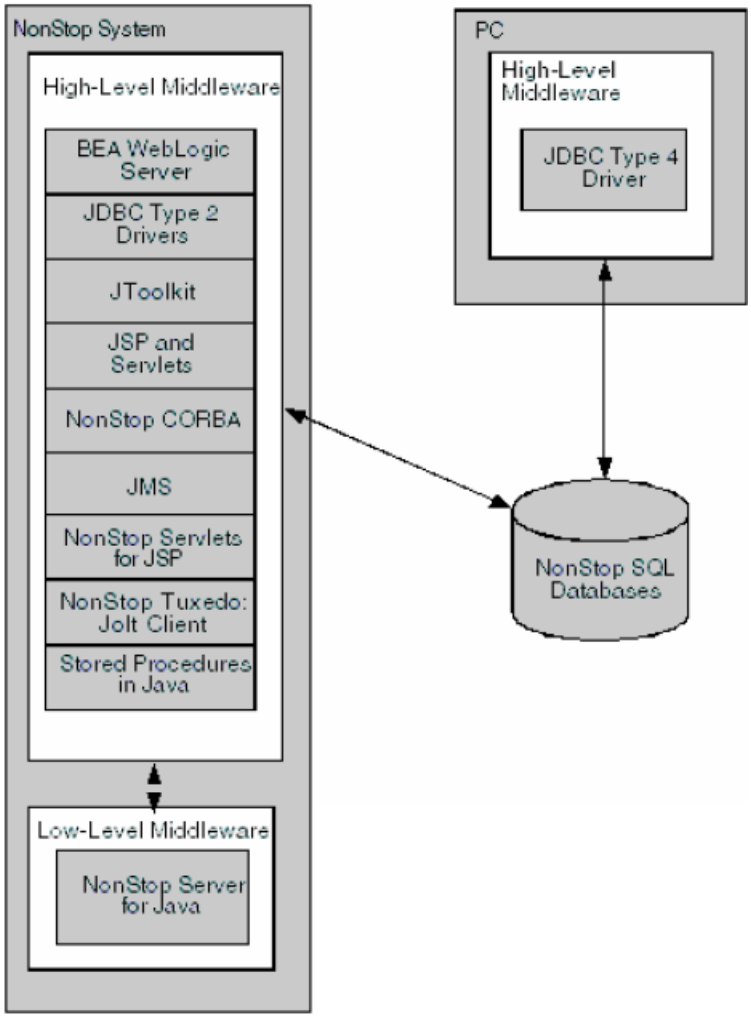
Imagine developing standard Java applications and deploying and running them on NonStop servers. You can develop these applications by using a wide-range of associated Java based products that can use the NonStop Server for Java 6.0 runtime. The products are:

- "BEA WebLogic Server for the HP NonStop Server" (page 25)
- "JDBC Drivers for NonStop SQL Database Access" (page 26)
- "JToolkit for NonStop Servers" (page 26)
- "NonStop CORBA" (page 27)
- "NonStop Servlets for JavaServer Pages" (page 27)
- "NonStop Server for Java Message Service (JMS)" (page 27)
- "NonStop Tuxedo: Jolt Client" (page 28)
- "Stored Procedures in Java" (page 28)

NOTE: For versions of Java products on the Java CD that work with the NonStop Server for Java 6.0 version, see the Readme.txt file on the NonStop Server for Java 6.0 product CD.

The listed high-level middleware products are shown working with NonStop Server for Java 6.0 and NonStop SQL databases.

Java Based Products on NonStop Systems



BEA WebLogic Server for the HP NonStop Server

The BEA WebLogic Server is a standards-based Java 2, Enterprise Edition (J2EE) application server that provides a foundation for building applications and includes:

- Load balancing
- Fault tolerance
- Web services
- Network transparency
- Legacy integration
- Transaction management
- Security
- Multithreading
- Persistence
- Database connectivity
- Resource pooling
- Development, testing, and packaging facilities

The BEA WebLogic Server uses the Java platform for portability to a large number of operating platforms supporting the Java platform. On properly configured NonStop servers, the WebLogic Server runs unchanged like on other platforms.

The BEA WebLogic Server for the HP NonStop Server is an application server that provides a framework for building and managing applications. WebLogic Server simplifies the development, deployment, integration, and management of applications by surrounding the latest J2EE and Web services standards with easy-to-use development and administration tools and powerful clustering, security, integration, and management features. These built-in services alleviate developers' need to create these services manually.

For more information, see BEA WebLogic Products documentation at http://www.hp.com/go/nonstop-doc s.

JDBC Drivers for NonStop SQL Database Access

JDBC drivers implement the JDBC API and provide access to NonStop SQL databases. You can use the JDBC API calls in your Java programs to access SQL tables on NonStop systems. The available drivers and the access they provide are:

- Type 2, which are native API drivers to use in Java programs running with NonStop Server for Java on a NonStop system. The type 2 drivers are included on the NonStop Server for Java distribution CD.
  - JDBC Driver for NonStop SQL/MX (JDBC/MX) for use with SQL/MX
  - JDBC Driver for NonStop SQL/MP (JDBC/MP) for use with SQL/MP
- JDBC Type 4, which uses network protocols built into the database engine. Type 4 drivers talk directly to the database using Java sockets. You can use the HP NonStop JDBC Type 4 Driver in Java programs running on PCs, HP-UX systems, and other platforms for access to NonStop SQL/MX. For the latest list of supported platforms, see the current JDBC Type 4 softdoc, which can be found online by accessing Scout for NonStop Servers.

To obtain detailed information on the standard JDBC API, you can download the JDBC API documentation provided by Sun Microsystems (http://java.sun.com/products/jdbc/download.html).

For information on HP drivers that are provided to access SQL/MX or SQL/MP, see the JDBC driver manuals at http://www.hp.com/go/nonstop-doc s.

JToolkit for NonStop Servers

The HP JToolkit for NonStop Servers includes three APIs as tools for using Java programs to access legacy applications on NonStop servers. JToolkit also includes Scalable TCP/IP (SIP) for developing network servers written in Java. The following paragraphs introduce these tools. For more information on them, see the JToolkit Programmer's Reference at http://www.hp.com/go/nonstop-doc s.

Enscribe API for Java

The Enscribe API for Java allows access to the Enscribe Database Manager, supported by the Guardian file system. This access is typically used to interact with legacy applications.

Pathway API for Java

The Pathway API for Java provides access to a special file called $RECEIVE, which is needed to enable a process to act as a Pathway server. These servers are typically used in legacy applications. Pathway server programs read requests from requester programs and act on those requests. The Guardian requester/server model is described in the *TS/MP Pathsend and Server Programming Manual*.

A process sends a message to another process by opening the recipient process file and writing a message to it. Because a process might not know in advance which processes will send messages to it and in which order, all messages to a process arrive using a single file-system connection. A process receives a message -whether the message is a request from another user process or a system message - by reading from $RECEIVE.

Pathsend API for Java

The NonStop Transaction Services/MP (NonStop TS/MP) product supports the use of Pathway servers to access NonStop SQL or Enscribe databases in an online transaction processing (OLTP) environment. Using the Pathsend API for Java, programs can send requests to these Pathway servers and receive replies from them. Pathway servers can be written in C, COBOL, or Java.

Scalable TCP/IP

Scalable TCP/IP (SIP) for the NonStop Server for Java provides a transparent way to give the NonStop fundamentals of scalability and persistence to a network server (SIP server) written in Java. Existing servers written in Java and their clients can take advantage of SIP without being changed.

NonStop CORBA

HP NonStop CORBA provides the Common Object Request Broker Architecture (CORBA) infrastructure and development environment that enables you to develop distributed object applications and components that run on the NonStop operating system. The NonStop CORBA infrastructure provides the services and tools to help software developers build object-oriented components and distributed object systems using either the C++ or the Java programming language. These systems can be implemented at the application level, the system level, or as middleware software.

Because NonStop CORBA is based on a CORBA standard as defined by the Object Management Group (OMG), application clients and components you develop using NonStop CORBA can interoperate with other CORBA servers running on different platforms.

The NonStop CORBA system architecture combines the flexibility of object technology with the robustness of a transaction-processing monitor. This unique combination provides the availability and scalability required for mission-critical applications. In addition, NonStop CORBA ensures the integrity of its own data stores and offers an object transaction service you can use to maintain a secure environment for your applications. NonStop CORBA gives you a CORBA based, object-oriented development system that provides differentiation in the areas of scalability, availability, and data integrity (transaction protection). Due to tight integration between HP transaction services and transaction monitors, NonStop CORBA contains the true functionality of an object transaction monitor.

For more detailed information on developing NonStop CORBA objects with Java, see the NonStop CORBA programmer's guide for Java for the NonStop CORBA version compatible with your version of NonStop Server for Java 6.0. You can find the NonStop CORBA manuals at http://www.hp.com/go/nonstop-doc s.

NonStop Servlets for JavaServer Pages

NonStop Servlets for JavaServer Pages (NSJSP) are platform-independent server-side programs that programmatically extend the functionality of web-based applications by providing dynamic content from a Web Server to a client browser over the HTTP protocol. NSJSP is an extension of that servlet functionality, primarily supplying a template of static content to be modified with dynamic content from a servlet or another programmable resource.

NSJSP requires the use of the iTP Secure WebServer, which is based on Tomcat. Tomcat implements the Java Servlet and JavaServer Pages specifications. For more information about NSJSP, see the *NonStop Servlets for JavaServer Pages (NSJSP) System Administrator's Guide* at http://www.hp.com/go/nonstop-doc s. For information about the iTP Secure WebServer, seethe iTP WebServer documentation at http://www.hp.com/go/nonstop-doc s.

NonStop Server for Java Message Service (JMS)

NonStop Server for Java Message Service (NSJMS) is the JMS provider that implements Sun Microsystems Java Message Service (JMS) API, on NonStop servers. NSJMS uses the performance

and reliability inherent in SQL/MX products to provide standards-based messaging for local clients running on NonStop servers. NSJMS enables scalability and load distribution through horizontal partitioning and fault-tolerance through process-pair technology.

Features and functions of NSJMS include:

- Implements the JMS API on NonStop systems. Uses the publish and subscribe features of NonStop SQL/MX.
- Uses a Java Naming and Directory Interface (JNDI) environment that allows access to NSJMS connection factories, and queue objects or topic objects.
- Enables use of a persistent, reliable bridge environment to allow interoperability between NSJMS and a locally hosted foreign JMS provider.
- Supports the NSJMS C++ API, which implements a subset of the functionality provided by the Sun JMS API, and is used by C++ client applications running on a NonStop system to interoperate with other JMS clients.
- Uses the NSJMS administrative utility to manage the NSJMS environment. You can invoke the utility through a command line interface or XML interface.

NSJMS conforms to Sun Microsystems published specification, Java Message Service, except as noted in NSJMS documentation. The specification is available on the Sun Microsystems Java Message Service (JMS) web site (http://java.sun.com/products/jms/docs.html). For more information about NSJMS, see the *NonStop JMS User's Manual* at http://www.hp.com/go/nonstop-doc s.

NonStop Tuxedo: Jolt Client

The Jolt product is a Java based interface to the HP NonStop Tuxedo system that extends Tuxedo services to the Internet. Jolt allows you to build client programs and applets that can remotely invoke existing NonStop Tuxedo services allowing application messaging, component management, and distributed transaction processing.

With Jolt, you can leverage existing Tuxedo services and extend your transaction environment to the corporate intranet or world-wide Internet. The key feature of the Jolt architecture is its simplicity. Using Jolt, you can build, deploy, and maintain robust, modular, and scalable electronic commerce systems that operate over the Internet.

The Jolt product includes the JoltBeans toolkit, which provides a JavaBeans compliant interface to Jolt for NonStop Tuxedo. The JoltBeans toolkit contains beans that wrap the existing Jolt class library into reusable bean components such as, the JoltSessionBean or the JoltServiceBean. These beans can be customized easily by giving application specific values to properties and connecting them with other bean components. You can use the JoltBeans toolkit with your Integrated Development Environment (IDE) to create Jolt clients that can access a Tuxedo application.

The Jolt product includes the Jolt Web Application Services Toolkit, which is an extension to the Jolt 1.1 Java class library. The Toolkit allows the Jolt client class library to be used in a Web Server to provide an interface between HTML clients or browsers, and Tuxedo services.

For more detailed information, see TUXEDO product documentation at http://www.hp.com/go/nonstop-doc s.

Stored Procedures in Java

Stored procedures in Java (SPJs) provide an efficient and secure way to implement business logic in an SQL/MX database. They allow you to write portable applications in Java and access an industry-standard SQL database.

A SPJ is a type of user-defined routine (UDR) that operates within a database server. A UDR can be either a stored procedure, which does not return a value directly to the caller, or a user-defined function, which does return a value directly to the caller. (A stored procedure returns a value only to a host variable or dynamic parameter in its parameter list.)

In the SQL/MX database, a SPJ is a Java method contained in a Java class, registered in SQL/MX, and invoked by SQL/MX when an application issues a CALL statement to the method.

For more information on using SPJs, see the *SQL/MX Guide to Stored Procedures in Java* at docs.hp.com.

2 Getting Started

Although this manual assumes that you are familiar with using Java and HP NonStop Open System Services (OSS), this section provides background information for persons not familiar with these products. Additionally, this section explains how to perform common tasks that are characteristic to running Java applications on NonStop systems. The topics are:

Learning About the Prerequisites

If you are not familiar with using Java and OSS, the following topics describe how you can get this background information.

Learning About Java

Many tutorials and books about the Java programming language are available publicly.

The Sun Microsystems web site provides links to many tutorials, including:

- Tutorials page of the Sun Developer Network, http://www.oracle.com/technetwork/java/index-jsp-135888.html
- Java Technology Learning, http://java.sun.com/learning/index.jsp
- Java Standard Edition JDK 6.0 documentation, for example, http://docs.oracle.com/javase/6/docs/index.html

A bookseller may have many Java tutorials and guides. However, the number of books available about Java is too great to list in this manual and new books or new editions are published often.

When choosing a Java tutorial or book, check that the information is appropriate for use with the JDK 6.0 implementation. Also, books and tutorials about how to write graphical user interfaces are not useful because the NonStop Server for Java 6.0 is a headless implementation.

Learning About Open System Services (OSS)

OSS is the open computing interface to the HP NonStop operating system—the operating system for NonStop servers. Java applications run in the OSS environment.

The user interface in the OSS environment is called the "OSS shell". The OSS shell is a program that interprets the commands you enter, runs the programs you ask for, and sends output to your screen. The OSS shell supports the Korn shell (sh), a shell common to UNIX systems.

The default shell prompt is a $ (dollar sign). (This is the default prompt for the Korn shell.) Throughout this manual, a $ is used to represent the OSS shell prompt.

Depending on your programming experience with NonStop systems or UNIX systems, use the following sources to gain the prerequisite knowledge to run Java applications on NonStop systems.

If you are familiar with using the Korn shell, you only need to peruse the Open System Services User's Guide (especially Section 2, OSS File System) for pertinent platform-specific information to get started using Java in the OSS environment.

The *Open System Services User's Guide*

The *Open System Services User's Guide* describes the OSS user environment; the shell, file system, and user commands. Topics of particular interest for beginning users follow.

NOTE: Although this guide is available only for G-series RVUs in the NTL Library at docs.hp.com, J-series and H-series users should read it, too. For H-series users, the definitive discussions are in the *Open System Services Programmer's Guide*.

The OSS File System

Discusses OSS files, directories, subdirectories, and pathnames. Also, describes how files in the OSS file system relate to the Guardian file system (for those familiar with the Guardian file system).

The OSS Shell

Discusses the features and environment of the OSS shell and how they can be used and modified.

OSS Commands and Utilities

Lists and describes user commands and utilities.

Running the OSS Shell

Describes how to run the OSS shell using the osh command.

Creating Files

Describes how to create new files in the OSS environment using the `vi` text editor.

Note that you can avoid using the `vi` text editor by creating the files in another environment and using the File Transfer Protocol (FTP), the `ftp` command, or another utility to put the files in the OSS file system. The HelloWorld ("Tutorial: Running a Simple Program, HelloWorld" (page 32)) sample, shown later in this section, uses this method.

The following sections are a tutorial that introduces many of the frequently used OSS user commands and utilities to perform the particular tasks:

- Creating files
- Managing files
- Managing directories
- Managing access to files and directories
- Managing processes

UNIX Tutorials

Public sources of information—such as, classes and tutorials—on using UNIX systems through the Korn shell are highly applicable to using the OSS environment. You might find these sources a good first step in learning about the file system, commands, and utilities characteristic of the OSS environment.

Verifying the Java Installation

Ask your system administrator where the NonStop Server for Java Server 6.0 software is installed. Knowing that, you can verify the installation and verify your environment. This example assumes that NonStop Server for Java 6.0 is installed in a nonstandard location—the `/home/lee/jdk60` directory:

NOTE: If your NonStop Server for Java 6.0 software is installed in the standard location (`/usr/tandem`), or if you are using NonStop Server for Java 6.0 version T2766H60^ABP or later, skip Step 4 and Step 5 and specify the location values accordingly.

For more information about the operating system requirements, see the T2766H60^ABP softdoc or the Readme.txt file packaged with the NonStop Server for Java 6.1 software distribution CD.

1. Set the PATH environment variable by using the following command at the OSS prompt:
   ```
   $export PATH=/home/lee/jdk60/java/bin:$PATH
   ```
2. Confirm that the path is set correctly by typing the `whence` command:
   ```
   $whence java
   /home/lee/jdk60/java/bin/java
   ```
3. Determine the version of the Java virtual machine (JVM) by typing the vproc command, which displays the product versions of the `java` binary file and any products bound into the `java` binary. For the version of the NonStop Server for Java 6.0 installation, look for the product number `T2766` following a line that begins `Version Procedure:` The displayed version identifier changes with every update of NonStop Server for Java 6.0.

   The version identifier has the following form:

   ```
   T2766Hnn
   ```

   A `vproc` example is:

   ```
   Version procedure: T2766H60_30Jan2009_jdk60...
   ```
4. To set the `JREHOME` shell variable, type the following command at the OSS prompt:
   ```
   $export JREHOME=/home/lee/jdk60/java/jre
   ```
5. To confirm that your `JREHOME` shell variable is set correctly, type the following `echo` command at the OSS prompt:
   ```
   $echo $JREHOME
   /home/lee/jdk60/java/jre
   ```

Tutorial: Running a Simple Program, HelloWorld

After the NonStop Server for Java 6.0 is installed, follow these steps to create and run the HelloWorld program.

The default OSS prompt is the dollar sign ($). The commands show the prompt, so do not type it. In text showing computer input and output, the input you type is shown in bold.

1. **Create a java Source File.**

   Perform either steps a, b, and c or step d only.

   a. Using your favorite editor, create a file that contains the following source code.
   b. Name the file `HelloWorld.java`.
   c. Place the file in the OSS file space by using FTP.

   ```
   /**
   * The HelloWorld application implements a java class that
   * displays "Hello World!" to the standard output.
   */
   ```

```
class HelloWorld
{
   public static void main(String[] args)
   {
     System.out.println("Hello World!");
   }
}
```

d. Alternatively, at the OSS prompt, use the `cat` command to create the `HelloWorld.java` file and type the contents of the HelloWorld program listed previously.

```
$cat> HelloWorld.java
type-contents-of-the-file
(Ctrl+y)
```

2. **Set the JREHOME Shell Variable.**

   If you know that the NonStop Server for Java 6.0 product is installed on your system in the standard location, you can omit this step. The standard location for the NonStop Server for Java 6.0 installation is:

   ```
   /usr/tandem/
   ```

   If you do not know where the NonStop Server for Java 6.0 product is installed, ask your system administrator.

   The `JREHOME` shell variable must point to the `jre` directory of your installation of the NonStop Server for Java 6.0 product.

   The default value for `JREHOME` is:

   ```
   /usr/tandem/java/jre
   ```

   If the NonStop Server for Java 6.0 is installed in a different location, for example `/test_dir`, type the following command at the OSS prompt set the `JREHOME` shell variable:

   ```
   $export JREHOME=/test_dir/java/jre
   ```

3. **Set the PATH Environment Variable.**

   Add the directory where the NonStop Server for Java 6.0 executable file is installed to your PATH environment variable.

   For the standard installation, type the following command at the OSS prompt:

   ```
   $export PATH=/usr/tandem/java/bin:$PATH
   ```

   If the NonStop Server for Java 6.0 is installed in a nonstandard location, `/test_dir`, type:

   ```
   $export PATH=/test_dir/java/bin:$PATH
   ```

4. **Check Your Path Settings.**

   Optionally, you can check whether your path is set correctly by using the `whence` command. Type:

   ```
   $whence -v java
   ```

   This command should display the fully qualified name of the `java` executable found in your path. If no Java executable is found, the command displays the message, "java not found".

5. **Compile the Java Source Code by Using the javac Tool.**

   a. Ensure you have performed step 3 so that `javac` is in your current path.

   b. At the OSS prompt, change the directory (`cd` command) to where your Java source file is stored.

c. Compile the Java source code by using the Java compiler, `javac`, which is part of the installed NonStop Server for Java 6.0 product. Type the following command at the OSS prompt:

```
$javac HelloWorld.java
```

If compilation is successful, the compiler produces a Java class file called `HelloWorld.class`. Once you have the class file, your program is ready to run.

d. Check to see that the `HelloWorld.class` file has been created by typing:

```
$ls -l HelloWorld.class
```

If the file is not present or if you received an error message, check for typographical errors in your source code. Fix them, and perform sub-steps c and d again until you have a class file.

6. **Run the Program by Using the java Tool.**

a. At the OSS prompt, ensure that the position is in the directory where your `HelloWorld.class` file is stored. For information about changing position, see step 5b.

b. To run the HelloWorld program (also called an application), type the following command at the OSS prompt:

```
$java HelloWorld
```

Note that you should not type `java HelloWorld.class`. All Java classes have the `.class` extension. Typing `.class` at the end causes an error message.

Your Java application displays the message, "Hello World!".

Specifying the CPU and Process Name with Which an Application Runs

You can specify which CPU an application process runs in and its process name by using options of the `run` utility. The `run` utility starts OSS programs with specific attributes.

The format of the command to specify the CPU where a Java application is to run is:

```
run -cpu=cpu_number java class_name
```

For example, the command to run Java in CPU 3 is:

```
$run -cpu=3 java HelloWorld
```

The format of the command to give a `java` process a process name is:

```
run -name=/G/process_name java class_name
```

For example, the command to give Java the process name `$APPJ` is:

```
$run -name=/G/appj java HelloWorld
```

where the /G directory identifies the Guardian fileset. For information about the /G directory, see the *Open System Services User's Guide*.

The following example combines more than one run option in a single command:

```
$run -name=/G/japp -cpu=3 java HelloWorld
```

For more information about the `run(1)` utility, see the *Open System Services Shell and Utilities Reference Manual*.

Configuring a Java Pathway Serverclass

The following is a brief overview of the specific Java requirements for configuring a Java program to run as a Pathway serverclass. Complete information about the topic is available in the *TS/MP System Management Manual*.

The serverclass attributes that have specific Java requirements follow. Typically, the attribute settings would be put in a configuration file, but the examples here show setting them in the OSS environment.

ARGLIST

The ARGLIST should appear as follows:

```
-Xabend,class-name[,arguments]
```

where `[,arguments]` is an optional, comma-separated list of arguments that are to be passed to the named serverclass. For example from the OSS prompt, start a PATHMON (process monitor) named `$foo` and set the ARGLIST at the PATHCOM prompt:

```
$gtacl -p pathcom \$foo
PATHCOM .....
=set serverclass ARGLIST -Xabend,MyClass,3000
```

This is similar to entering java `-Xabend MyClass 3000` at an OSS shell prompt. The `-Xabend` argument to the `java` executable causes `java` to abend instead of exiting with a non-zero exit code. Pathway servers must abend rather than merely stop when a fatal error occurs, so that the PATHMON process can restart them. `MyClass` is the name of the Java class to be invoked, and "3000" is an argument to the `MyClass` class.

Note that, in the OSS environment, the dollar sign (`$`) has special meaning; therefore, the process name `$foo` must be preceded by a backslash (\), the escape character.

PROCESSTYPE

Set this attribute to `OSS` .

ENV

Environment variables are set using the ENV serverclass attribute. For Java use, you must set the `CLASSPATH` environment variable so that the Java runtime can find your classes. If you are either using NonStop Server for Java 6.0 versions earlier than the T2766H60^ABP SPR or running `java` executable in a location other than the standard location of `/usr/tandem`, you must set the `JREHOME` environment variable to the `jre` directory. For a `JREHOME` example, the set server command shown entered at the PATHCOM prompt follows. (However, you would typically type these commands in a configuration file to be used with PATHCOM.)

```
$gtacl -p pathcom \$foo
PATHCOM .....
=set server ENV /home/lee/jdk60/java/jre
```

NOTE: If you are using NonStop Server for Java 6.0 version T2766H60^ABP or later, you need not set the `JREHOME` variable. The standard location for all NonStop Server for Java 6.0 installations is `/usr/tandem` .

PROGRAM

Set the PROGRAM attribute to the `java` executable. The `java` executable is located by default in `/usr/tandem/java/bin/`. For example from the OSS prompt, start a PATHMON (process monitor) named `$foo` and set the PROGRAM attribute at the PATHCOM prompt:

```
$gtacl -p pathcom \$foo
PATHCOM .....
=set server PROGRAM /usr/tandem/java/bin/java
```

3 Installation and Configuration

This section explains these subsections:

- "Installation Requirements" (page 37)
- "Configuration Requirements" (page 37)
- "NonStop Server for Java 6.0 Directory Structure" (page 41)

For information about how to verify a NonStop Server for Java 6.0 installation, see "Verifying the Java Installation" (page 32).

Installation Requirements

Regarding hardware, NonStop Server for Java 6.0 can run on these systems:

All HP Integrity NonStop NS-series servers

The software requirements for the NonStop Server for Java 6.0 are described in the README file and SOFTDOC on the product CD. Read those documents before installing the product. The software requirements list the earliest acceptable versions of the required software. You can substitute later versions of the same products.

HP recommends that NonStop Server for Java 6.0 be installed in the standard location whenever possible as described under "Symbolic Link" (page 40). You can install NonStop Server for Java 6.0 in a non-standard location. To do that kind of installation, use the File Transfer Protocol (FTP) to transfer the file from the CD to the NonStop system. Then follow the directions in the product Softdoc for a nonstandard installation.

You can install NonStop Server for Java 6.0 with earlier versions of NonStop Server for Java on the system.

Configuration Requirements

This subsection explains how to configure your system for the Java SE JDK by understanding the following:

- "Creating Larger or Additional Swap Files" (page 37)
- "Setting Environment Variables" (page 38)
- "Symbolic Link" (page 40)
- "Configuring TCP/IP and DNS for RMI" (page 40)

NOTE: Do not install any JAR or native library files under the NonStop Server for Java installation directory.

Creating Larger or Additional Swap Files

HP recommends a total swap files size of 512 MB (1024 `Extents`, 255 `MaxExtents`) for each processor that runs the

Java virtual machine (JVM). If you plan to run multiple large processes in the same processor, you might need to create additional swap files because processes running in the same processor share the same swap file.

Your system administrator can use the NSKCOM tool to create additional swap files.

To add swap files, you must log on to your NonStop operating system as a super-group user. Then, from the Guardian TACL prompt, run the NSKCOM tool. From the NSKCOM tool, use the help add and help start commands to get more information. For further information, see the *Kernel-Managed Swap Facility (KMSF) Manual*.

Setting Environment Variables

The following subsections describes the variables that define the environment in which Java operates.

PATH

The environment variable `PATH` enables Open System Services (OSS) to find the Java executable files. As a convenience so that you do not have to fully qualify the Java executable, add the absolute path of the `java/bin` directory to the `PATH` environment variable.

To add the absolute path, use this command:

```
export PATH=/install_dir/java/bin:$PATH
```

where `install_dir` is the directory in which the NonStop Server for Java 6.0 is installed. By default, this is `/usr/tandem`.

The `PATH` shell variable must be created in each shell in which you plan to run `java` or one of its tools. For this reason, it is a good idea to set the `PATH` in the `.profile` file in your home directory that is executed each time you logon to an OSS shell. See the *Open System Services User's Guide* for information on how to set the path in your startup file.

Class Path

The class search path (more commonly, “class path”) is the path that the Java runtime environment searches for classes and other resource files. The class path tells the JDK tools and applications where to find third-party and user-defined classes. The class path can be set either by using the `-classpath` option when calling a JDK tool (such as, `java` or `javac`), or by setting the CLASSPATH environment variable.

The preferred method is using the `-classpath` option because you can set that option individually for each application without affecting other applications and without other applications modifying the option's value.

Syntax

`jdkTool` `-classpath` *`classpath1:classpath2...`*

-or-

`export CLASSPATH=`*`classpath1:classpath2...`*

`jdkTool`

A command line tool, such as `java` or `javac` . For the tools list, see the *NonStop Server for Java 6.0 Tools Reference* .

`classpath1:classpath2`

Class paths to the `.jar` , `.zip`, or `.class` files. Each class path should end with a filename or directory name depending on the particular setting.

- For a `.jar` or `.zip` file that contains. class files, the class path ends with the name of the `.jar` or `.zip file`.
- For `.class` files in an unnamed package, the class path ends with the directory name that contains the `.class` files.
- For `.class` files in a named package, the class path ends with the directory name that containsthe "root" package (the first package in the full package name).

Multiple path entries are separated by colons.

The default class path is the current directory. Setting the CLASSPATH variable or using the `-classpath` command line option overrides that default, so if you want to include the current directory in the search path, you must include a dot (.) in the new settings.

Class path entries that are neither directories nor archives (`.zip` or `.jar` files) are ignored.

Example: Setting Class Path in a `java` Command

Suppose you want the Java runtime to find a class named `Cool.class` in the package `utility.myapp`. If the path to that directory is: `/java/MyClasses/utility/myapp`, you would set the class path so that it contains `/java/MyClasses`.

To run that application, you could use the following `java` command:

```
$java -classpath /java/MyClasses utility.myapp.Cool
```

Example: Setting the CLASSPATH Environment Variable

Using the same situation as in the preceding example, except that you want to set the `CLASSPATH` environment variable so that the Java runtime can find the class named `Cool.class`, you would use the following command to set and export the `CLASSPATH` environment variable and then run Java.

To run that application, you could use the following commands:

```
$ export CLASSPATH=/java/MyClasses
$ java utility.myapp.Cool
```

For further information about setting class path, see the documentation provided by Sun Microsystems (http://docs.oracle.com/javase/6/docs/technotes/tools/index.html). Note that when applying that documentation to the NonStop Server for Java 6.0, you must follow the instructions given for `sh` and `ksh` instead of those for `csh` and `tcsh`. Instructions for the `setenv` and `unsetenv` commands do not apply to the NonStop Server for Java 6.0.

JREHOME

The JREHOME shell variable is used by the Java runtime to determine where the core Java classes are located. If you install NonStop Server for Java 6.0 in the standard location, you do not need to set up the JREHOME shell variable because it has the default value of `/usr/tandem/java/jre`.

NOTE: If you are using NonStop Server for Java 6.0 version T2766H60^ABP or later, you need not set the JREHOME variable because the Java classes will be located based on the location of the Java executableof a given JDK installation.

If you install the NonStop Server for Java 6.0 in a location other than the `/usr/tandem/java` directory, you must do the following:

1. Create a shell variable called JREHOME and set it to the location of the `jre` directory. For example, if you installed the NonStop Server for Java 6.0 at `/h/myjava` instead of `/usr/tandem`, do the following to create and set the `JREHOME` variable:

   ```
   $ export JREHOME=/h/myjava/java/jre
   ```

2. You must create the `JREHOME` shell variable in each shell in which you plan to run `java` or one of its tools. For this reason, it is a good idea to put a creation mechanism in the `.profile` file in your home directory that is executed each time you log on to an OSS shell. See the *Open System Services User's Guide* for information on how to set the path in your startup file.

Failure to specify the JREHOME variable may cause the JVM to fail to start, prompting the following Java error message:

```
Can't find class java.lang.NoClassDefFoundError. (Wrong classpath?)
```

Alternatively, the JVM starts successfully but the `jar` files in the directory `/usr/tandem/java/jre` are used instead of the files under the directory where NonStop Server for Java 6.0 is installed. This can produce unpredictable results.

_RLD_LIB_PATH

The `_RLD_LIB_PATH` environment variable specifies the library path for DLLs. You need to use this environment variable only if you use user DLLs. You can specify one or more directories as necessary. Separate each directory in the list by using a colon (:). Set this environment variable as follows:

```
_RLD_LIB_PATH=dll_path[:dll_pathn]...
```

where *dll-path* and *dll-pathn* are the directories where the user DLLs reside.

For example:

```
export _RLD_LIB_PATH=/home/me/mydll
```

Symbolic Link

The link `/usr/tandem/java` is created when NonStop Server for Java 6.0 is installed. It is a symbolic link to the actual JDK directory, which has the form:

```
/usr/tandem/nssjava/jdk60x_hyy
```

where x refers to the version number of the Sun Microsystems update upon which NonStop Server for Java 6.0 is based and `yy` refers to the particular product version update (PVU) of NonStop Server for Java 6.0. For example, for NonStop Server for Java 6.0, based on Java SE 6.0, the symbolic link is `/usr/tandem/nssjava/jdk160_h60`.

The `/usr/tandem/java` is a shorthand way to refer to the latest version of the JDK installed on the system. When you unpax a NonStop Server for Java 6.0 `pax` file, the symbolic link is created or reset to refer to the JDK directory that is being unpaxed, which means that the symbolic link refers to the version of NonStop Server for Java 6 that was last installed. You are not required to use the symbolic link. To make sure you are always using the 6.0 version of the JDK, even if a later version has been installed, you can put the `bin` directory in your PATH, for example:

```
export PATH=/usr/tandem/nssjava/jdk160_h60/bin:$PATH
```

You can also reset the symbolic link yourself by using the `ln` command with the `-s` option. For example, if you install a PVU based on version 1.4.2 of the JDK, but still want version 6.0 to be the one referred to by `/usr/tandem/java`, you can reset the symbolic link instead of unpaxing version 6.0 again:

```
$ cd /usr/tandem
$ rm java
$ ln -s /usr/tandem/nssjava/jdk160_h60 java
```

The symbolic link is always put in the directory where NonStop Server for Java 6.0 is installed, so if you use the `-s` option to specify an alternative installation directory during the unpaxing step, the symbolic link is `install_dir/java` instead of `/usr/tandem/java`. For example, if `nssjava/jdk160_h60` is installed in `/h/myjava`, the symbolic link is `/h/myjava/java`, and this symbolic link points to the directory `/h/myjava/nssjava/jdk160_h60`.

Configuring TCP/IP and DNS for RMI

For Remote Method Invocation (RMI) API to work, TCP/IP and its component, DNS, must be configured correctly. For the correct version of TCP/IP, see the NonStop Server for Java 6.0 Softdoc.

A network administrator usually configures TCP/IP and DNS, but you can determine if an incorrect TCP/IP configuration is causing a JVM problem. To check the TCP/IP configuration, use the Java Checker, javachk, which is available in the `/usr/tandem/java/install` directory. Execute `javachk` in the same environment as the JVM has (that is, using the same defines that were used to run the JVM). The Java Checker will identify failing socket routine calls. When you know which calls are failing, you can fix or work around the problems.

For information about `javachk`, see the file `/usr/tandem/java/install/README_javachk`.

NonStop Server for Java 6.0 Directory Structure

This subsection explains:

- "Directory Contents" (page 41)
- "Demonstration Programs" (page 41)

Directory Contents

The `/usr/tandem/java` directory contains release documents and subdirectories. Table 3 (page 41) lists the subdirectories and describes their contents.

Table 3 Subdirectories of the `/usr/tandem/java` Directory

Subdirectory	Contents
`/bin`	Executable binary files that make up the JDK . These files include tools that are not part of the Java SE Runtime Environment, such as `javac` and `javah` .
`/demo`	Additional subdirectories, each containing a README file and a complete example.
`/include`	C-language header files that support native code programming using the Java Native Interface and the Java Virtual Machine Interface.
`/install`	The `javachk` file.
`/jre`	The root directory of the Java SE Runtime Environment. Includes core classes, classes supplied by HP, and runtime libraries. The core Java classes are in the `lib/rt.jar` file.
`/lib`	Classes other than core classes for support of the tools and utilities in the JDK software.

Demonstration Programs

The `/demo` directory contains subdirectories, each of which contains a demonstration program and a README file that documents how the demonstration program should be used. Demonstration programs provided include the following:

Table 4 Demonstration Programs

Demonstration Program	Illustrates
`invocation_api`	How to properly build an executable that can create its own Java virtual machine (JVM).
`javahjni`	How to create a native library and how to use IEEE floating point.

Many of the demonstration programs require you to edit some files before the demonstration programs can be run, as described in the accompanying README file.

Additional demonstration programs are provided if you install a JDBC Driver for NonStop SQL product. These programs are located with the driver software.

4 Implementation Specifics

This section explains these subjects regarding HP implementations of NonStop Server for Java, based on Java Platform Standard Edition 6.0, for Integrity NonStop systems:

In addition, see "Application Profiling" (page 76).

Headless Support

Because the HP NonStop operating system does not provide support for windowing operations, NonStop Server for Java 6.0 is a headless JVM that conforms to the Sun Microsystems headless support standard regarding Java Abstract Window Toolkit (AWT) classes and methods. For similar reasons, the NonStop Server for Java 6.0 does not support the `AppletViewer` tool.

If your Java programs use classes and methods that require a display, keyboard, sound, or mouse operation, the class or method will throw a `HeadlessException` if invoked when `GraphicsEnvironment.isHeadless` returns true. This value is always true in NonStop Server for Java 6.0.

Classes and methods that support printing, fonts, and imaging are fully supported in a headless JVM.

While the Sun Microsystems documentation for the reference implementation states that you must set the system property `-Djava.awt.headless=true` to run a headless JVM, setting this system property is unnecessary for NonStop Server for Java 6.0.

The following Java SE features are not applicable (and, therefore, not available):

- Class data sharing (CDS), a feature intended to reduce application startup time and footprint, that is available only with a Java HotSpot client VM.
- Server-class machine detection because a server-class machine is always assumed
- Java user-interface features because they are desktop related
  - General deployment features, such as klist
  - Java Web Start Technology
  - Sound Java API

Additional Files

In addition to the standard Java packages, the NonStop Server for Java 6.0 provides these files:

`jtatmf.jar`

File containing classes for the version of the NonStop Java Transaction Agent that uses TMF.

`javachk`

The Java Checker program, which determines whether a problem with the JVM is caused by an incorrect TCP/IP configuration.

Additional Environment Variable

NonStop Server for Java 6.0 has an implementation-specific environment variable that you can use to control the runtime environment. The `JAVA_PTHREAD_MAX_TRANSACTIONS` environment variable specifies the maximum number of TMF transactions allowed per process. The default number allowed is 1000. For more information, see "Controlling Maximum Concurrent Transactions" (page 72).

Java Native Interface (JNI)

The Sun Microsystems Java Native Interface (JNI) standard defines both the C-language APIs that enable Java methods to call C and C++ methods and the way that C and C++ methods can start and interact with a Java Virtual Machine (JVM). The NonStop Server for Java 6.0 supports JNI and the Invocation API with the following modifications:

- Set the _RLD_LIB_PATH environment variable to point the location of the user DLLs.

  ```
  export _RLD_LIB_PATH=dll-path[:dll_pathn]...
  ```

  where: `dll-path` and `dll-pathn` are the directories where the user DLLs reside.

  For example, if the user DLLs are in the directory `/home/mydll`

  ```
  export _RLD_LIB_PATH=/home/mydll
  ```

- Multithreaded native C or C++ routines must use the same Pthread library that the JVM uses, but HP recommends that any multithreaded code be written in Java.
- If native C or C++ routines must invoke Transaction Management Facility (TMF) calls, you must use TMF transaction jacket routines as described in the *Open System Services Programmer's Guide* in the "Application Programming With Standard POSIX Threads" section under the topic "TMF Transaction Jacket Routines." The calls are:
  - `SPT_ABORTTRANSACTION()`
  - `SPT_BEGINTRANSACTION()`

- SPT_ENDTRANSACTION()
- SPT_RESUMETRANSACTION()

NOTE: The *Open System Services Programmer's Guide* states that the maximum concurrent transactions allowed in a process is 100; however, the JVM is an exception where the maximum allowed is 1000 as described under "Controlling Maximum Concurrent Transactions" (page 72).

- When calling a C or C++ routine, where the function passes or returns parameters of type `float` or `double`, NonStop Server for Java 6.0 performs no conversion. All float and double values remain in IEEE floating-point format when crossing the JNI boundary. For more information, see "IEEE Floating-Point Implementation" (page 46).
- When using the `JNI_OnLoad` function, use the following format:

```
jint JNI_OnLoad(JavaVM *vm, void *reserved);
```

- The `JNI_OnUnload` function is supported by NonStop Server for Java 6.0 on NS-series servers, but not supported on S-series servers.

When naming library files, observe the following rules:

- Do not use names that begin with `Tandem`, `tandem` , or `tdm` .
- NonStop Server for Java 6.0 requires that all DLLs be named with a prefix `lib` and a sufix `.so`. So you must name your DLL as follows:

  `libname.so`

  where `(name)` signifies the string that is passed to the `System.loadLibrary()` call. `libname.so`

The remainder of this subsection explains:

- "Calling C or C++ Methods from Java" (page 44)
- "Calling Java Methods from C or C++" (page 45)
- "Linker and Compiler Options" (page 45)

For more information about JNI, see the Sun Microsystems JNI document (http://docs.oracle.com/javase/6/docs/technotes/guides/jni/index.html).

Calling C or C++ Methods from Java

To call C or C++ methods from Java, follow these steps:

1. Compile the Java code.
2. Use `javah` to generate header files. The function declarations listed in the generated header file are those that must be exported by the user-JNI DLL. To export functions, either specify `export$` in the function definition or use the linker option `-export_all`.
3. Compile the C or C++ code. C++ code must be compiled using the following compiler command line options: `-Wversion2` or `-Wversion3`, and `-WIEEE_float`.

   If the native code has large variables on the stack, calling this native code might exceed the default stack space provided for each thread. If the native code exceeds the amount of stack space allocated for it, unpredictable results can occur. To prevent overflowing the available stack space, consider allocating large variables on the heap rather than using the stack. Otherwise, you can increase the default stack size for each thread by specifying the `-Xss` option when starting `java`. This option increases the stack size for every thread. For more

information about the `-Xss` option, see `java` in the *NonStop Server for Java 6.0 Tools Reference Pages*.

4. Create a DLL file (`.so` file type) and specify the linker option `-set float type IEEE_float`. Then set the `_RLD_LIB_PATH` environment variable to point to where the created DLL file resides by using the following command:

   `export _RLD_LIB_PATH=`*`dll-path`*

   where *`dll-path`* is the directory where the user DLL resides. For more information, see "_RLD_LIB_PATH" (page 40).

The `javahjni` demo shows an example of how to create a library file. This demo also shows converting between the TNS and the IEEE floating point.

Calling Java Methods from C or C++

You can create your own C or C++ program and use the Invocation API to load the JVM into an arbitrary native program. Be sure to follow these guidelines:

- Compile code written in C++ by using the `-Wversion2` or `-Wversion3` compiler, and the `-WIEEE_float` compiler command line options.
- The NonStop Server for Java 6.0 provides DLLs. Therefore, you can build your own executable and link it to the JVM DLL, `libjvm.so`. For details, see the `invocation_api` demo provided with the NonStop Server for Java 6.0 installation.
- Do not set signal handlers for the following signals: `SIGSEGV` , `SIGPIPE` , `SIGCHLD` , `SIGINT` , `SIGQUIT` , `SIGTERM` ,and `SIGHUP` .
- Set the executable to use IEEE floating point.

NonStop Server for Java 6.0 does not support the signal-chaining facility implemented in some other vendors' JVMs.

When a program uses the Invocation API to start a JVM, its function returns are parameters of type `float` or `double` that are in IEEE floating-point format. Any parameters of type float or double that are passed to NonStop Server for Java 6.0 must also be in IEEE floating-point format. If such a program wants to convert between TNS floating-point format and IEEE floating-point format, the *Guardian Procedure Calls Reference Manual* documents a series of procedures with names beginning with NSK_FLOAT_ that can be used to convert `float` and `double` data between the two formats.

To run the Invocation API demo, follow the instructions for the Invocation API demo in the README file in the directory `/usr/tandem/java/demo/invocation_api`.

Linker and Compiler Options

Compiler Options

When you compile C++ source for use with NonStop Server for Java 6.0, you must use the following compiler options to identify which dialect of the C++ compiler is to be used:

`-Wversion2` or-

`-Wversion3`

In addition, for a compilation unit containing JNI code that has any floating-point parameters being passed across the JNI boundary and that is directly called by the JVM, you must use the compiler option:

`-WIEEE_float`

Any compilation units not called directly by the JVM can be compiled without the `–WIEEE_float` option; however, the complications that can occur while using such mixed modes are beyond the

scope of this document. However, the `javahjni` demo shows an example of mixed modes. (For information on demos, see "Demonstration Programs" (page 41).)

Linker Options

When building native libraries, you must use the following linker option:

```
-set floattype IEEE_float
```

How to Create Your Own Library of Native Code

The `javahjni` demonstration program that comes with NonStop Server for Java 6.0 shows how to create a native library.

You can find the `javahjni` demonstration program in the `install-dir/demo/javahjni` directory.

IEEE Floating-Point Implementation

Java uses IEEE floating-point arithmetic.

NOTE: In NonStop Server for Java 6.0, you cannot specify whether your Java classes use TNS format.

Incompatibilities between the IEEE floating point and TNS floating-point representations might cause loss of precision or accuracy when you convert between TNS `float` or `double` and IEEE `float` or `double`.

This subsection explains the following subjects:

- "Floating-Point Values" (page 46)
- "Double-Precision Values" (page 46)
- "How to Call TNS Floating-Point Functions from JNI Code" (page 47)

Floating-Point Values

For floating-point values, TNS floating-point representations have larger exponents (and therefore, a larger range) than IEEE floating-point representations, but they are less precise, as Table 5 (page 46) shows:

Table 5 Floating-Point Ranges

Floating-Point Representation	Minimum Positive Decimal Value	Maximum Decimal Value
TNS	1.7272337e-77F	1.1579208e77F
IEEE	1.40239846e-45F	3.40282347e+38F

Double-Precision Values

For double-precision values, TNS floating-point representations have smaller exponents (and therefore, a smaller range) than IEEE floating-point representations, but they are more precise, as Table 6 (page 46) shows:

Table 6 Double-Precision Ranges

Floating-Point Representation	Minimum Positive Decimal Value	Maximum Decimal Value
TNS	1.7272337110188889e-77	1.15792089237316192e77
IEEE	4.94065645841246544e-324	1.79769313486231570e+308/

How to Call TNS Floating-Point Functions from JNI Code

This topic describes how to call a TNS floating-point function from an IEEE floating-point function.

When using TNS floating-point compiled functions in native code linked into the `java` executable:

- Do not call the Common Runtime Environment (CRE) functions with TNS floating-point values because CRE functions are expecting IEEE floating-point values.
- Do not pass floating-point values (`float` and `double`) across mixed float compilation units. When passing or returning floating-point values between IEEE floating-point compiled functions and TNS floating-point compiled functions, pass or return.
  - A `float` as one of the 32-bit structures defined in `$SYSTEM.SYSTEM.KFPCONVH` (NSK_float_ieee32 or NSK_float_tns32)
  - A `double` as one of the 64-bit structures defined in `$SYSTEM.SYSTEM.KFPCONVH` (NSK_float_ieee64 or NSK_float_tns64)
- You can call a native function that accepts or returns a TNS `float` or `double` value if you create an intermediate function that sits between the IEEE floating-point compiled JNI method that the JVM calls and the native function that accepts or returns a TNS `float` or `double` . Either the JNI method or the intermediate method can be responsible for calling one of the NSK_float_* procedures to convert between IEEE and TNS floating-point formats.

  The intermediate function:
  - Is compiled with TNS floating point.
  - Accepts `float` and `double` arguments as one of the special structures defined in the `$SYSTEM.SYSTEM.KFPCONVH` file.
  - Calls the TNS compiled native function passing TNS `float` or `double` arguments.
  - Converts any `float` or `double` return value to an IEEE floating-point value of the JNI caller expects the value.
  - Returns the float or double in one of the special structures defined in the `$SYSTEM.SYSTEM.KFPCONVH` file.

For an example, see the `javahjni` "Demonstration Programs" (page 41).

Multithreaded Programming

The Java virtual machine for the NonStop Server for Java 6.0 is multithreaded. It uses Standard POSIX Threads, which conforms to IEEE POSIX Standard 1003.lc. Threads are scheduled for execution by the Standard POSIX Threads library, not by the operating system. All threads created within a process share the same process address space in the same CPU. With Standard POSIX Threads on NonStop systems, one thread can never be preempted by another thread if the `–XX:ThreadTimeSlice` option is not used.

The topics in this discussion are:

- "Thread Scheduling" (page 47)
- "Threading Considerations for Java Code" (page 49)
- "Threading Considerations for Native Code" (page 50)

Thread Scheduling

The Java runtime supports a simple, deterministic, scheduling algorithm known as fixed-priority scheduling. By default, the Java runtime does not time-slice. You can enable time slicing by using the `–XX:ThreadTimeSlice` option.

For the NonStop system, the thread-scheduling algorithm is not preemptive; that is, a thread continues to run until it explicitly yields or otherwise causes a yield by invoking a blocking operation on the thread. However, you can assign time-slice to each thread using the `-XX:ThreadTimeSlice` option.

When a thread gives up control, the runnable threads of the highest priority are run in first-in-first-out order. A lower priority thread is run (also in first-in-first-out order) only when no runnable threads of a higher priority are available. Where no runnable user threads are available, an internal NULL thread is run. This NULL thread wakes up and gives control to other threads on events, such as signals, timer completions, and I/O completions.

When a Java thread is created, the thread inherits its priority from the thread that created it. The priority of the thread varies from `MIN_PRIORITY (1)` to `MAX_PRIORITY (10)`, where the default priority is `NORM_PRIORITY (5)`. After a thread is created, the `setPriority` method can be used to alter the priority of the thread.

The Java virtual machine threads use predetermined priorities, some of which are higher than the priority of any of the user threads. By default, the `-XX:+UseThreadPriorities` option is true and any attempt to alter the option has no effect. Although attempts to use `-XX:` options, which affect thread priorities, might be accepted at the command line, these options have no effect when used on the NonStop system.

A selfish thread (a thread that executes in a tight loop without giving up control) could, theoretically, run forever. However, after a while, the operating system will periodically reduce the priority of the process in stages, until its priority reaches a very low value.

Timers are never guaranteed to be exact. Invocation of timer callbacks and detection of I/O completions can be severely impacted by long-running threads.

For a demonstration of scheduling on a NonStop system, review the output of the following program and its results when run:

RaceDemo.java

```
public class RaceDemo {
 private final static int NUMRUNNERS = 2;
 public static void main(String[] args) {

 SelfishRunner[] runners=new SelfishRunner[NUMRUNNERS];
 for (int i = 0; i < NUMRUNNERS; i++) {
  runners[i] = new SelfishRunner(i);
  runners[i].setPriority(2);
 }
 for (int i = 0; i < NUMRUNNERS; i++)
  runners[i].start();
 }
}
```

SelfishRunner.java

```
public class SelfishRunner extends Thread {

 private int tick = 1;
 private int num;

 public SelfishRunner(int num) {
  this.num = num;
 }

 public void run() {
  while (tick < 400000) {
   tick++;
   if ((tick % 50000) == 0)
    System.out.println("Thread #"+num+", tick = "+tick);
  }
```

```
  }
}
```

On the NonStop system, the execution of threads is not time-sliced. Messages from one thread precede those from the other thread as shown below for thread 0 and thread 1.

```
Thread #0, tick = 50000
Thread #0, tick = 100000
Thread #0, tick = 150000
Thread #0, tick = 200000
Thread #0, tick = 250000
Thread #0, tick = 300000
Thread #0, tick = 350000
Thread #0, tick = 400000
Thread #1, tick = 500000
Thread #1, tick = 100000
Thread #1, tick = 150000
```

When the SelfishRunner.java program is run with the `-XX:ThreadTimeSlice` option, time slicing is enabled and the messages from one thread do not precede messages from the other threads. Instead, the messages are displayed in the following manner:

```
Thread #0, tick = 50000
Thread #0, tick = 100000
Thread #0, tick = 150000
Thread #0, tick = 200000
Thread #0, tick = 250000
Thread #1, tick = 50000
Thread #1, tick = 100000
Thread #1, tick = 150000
Thread #1, tick = 200000
Thread #0, tick = 300000
Thread #0, tick = 350000
Thread #0, tick = 400000
```

NOTE: The NonStop Java 6.0 T2766H60^ABP release includes a beta version of the JVM-forced, preemptive thread scheduling feature. This feature also provides an option to specify the time slice for threads. A thread will run for the specified time slice, after which another thread will get dispatched from the ready queue. This helps in force-yielding a thread which consumes large processor time so that the other ready threads also get the processor time to run.

To enable preemptive user threads, use the following option:

```
-XX:ThreadTimeSlice[=T]
```

where,

T specifies the time in milliseconds.

NOTE:

- T is an optional argument.
- The default value of T is 40 milliseconds.
- The value of T can range between 0 to 32767. If the specified value of T is above 32767, the value is time-sliced to 32767.

Threading Considerations for Java Code

A thread-aware function blocks only the thread calling that function, rather than blocking all threads in the process. At the Java program level, the following blocking operations are thread-aware and, therefore, block only the thread performing the operation rather than blocking all threads in Java:

- Socket I/O
- Terminal I/O

- Pipe I/O
- TMF transactions (`com.tandem.tmf` package)

JToolkit provides a thread-aware API for performing I/O to Enscribe files, $RECEIVE, and Pathway servers by using Pathsend procedure calls. For more information, see the *JToolkit Programmer's Reference*.

Thread-Aware I/O Support for OSS regular files

By default, the NonStop Server for Java 6.0 enables Non-Blocking I/O for OSS regular files on SUT versions J06.04 and greater and H06.15 and greater. This means that regular file I/O operations on multiple OSS files can be performed simultaneously by the Java application. Therefore, this feature is useful for multithreaded Java programs that perform regular OSS file I/O operations. To turn the Non-Blocking I/O mode off, use the following java command line option:

```
-Dnsk.java.nonblocking=false.
```

Threading Considerations for Native Code

All threaded code should be written in Java rather than native code. If a user native library linked into Java creates a thread, non thread-aware operations executed in the user library might impact the operation of the Java virtual machine. If the user library creates multiple user threads, the program needs to be all the more careful to ensure that the operations performed in the user threads are thread-safe on the NonStop system.

You need to consider the issues discussed below when using threads in native code linked to a Java program on a NonStop system:

- NonStop Server for Java 6.0 does not use the POSIX threads SRL.

  Instead, NonStop Server for Java 6.0 contains its own version of POSIX threads. Therefore, your code should include the Standard POSIX Threads header files shipped with NonStop Server for Java 6.0. The header files for this version of POSIX threads can be found in the directory:

  `[install-dir]/java/include/oss`

  where `install-dir` is the NonStop Server for Java 6.0 installation directory.

- Creating a thread for a task does not make the task run faster.

  The NonStop system does not have an implementation of native threads; threads run at a user level. Even on a multiprocessor NonStop system, all threads in a process are executed in the same processor as the process. If you create a thread whose only purpose is to run a certain task, the thread-creation overhead makes the task run marginally slower than the same task being performed without creating the thread.

- The thread-scheduling algorithm is not preemptive.

  A thread executes until it explicitly yields. For more information, see the discussion of "Thread Scheduling" (page 47).

- In a very long-running, CPU-intensive thread, having your native code occasionally invoke the `yield()` method allows timer completions and I/O completions to be detected. Invocation of timer callbacks and detection of I/Ocompletions can be severely impacted by long-running threads.
- Be familiar with the issues discussed in the "Application Programming with Standard POSIX Threads" section of the *OSS Programmer's Guide*.

  This section contains information about the jacket routines that make many of the available system calls thread-aware. The interfaces themselves, however, are documented in the *OSS System Calls Reference Manual*.

  To use these jacket routines, you need to add the following define in your native code.

`#define SPT_THREAD_AWARE`

Adding this define in the native C/C++ code, transparently provides you the thread-aware equivalents of many of the interfaces, for example, the Socket interface. Additionally, the interfaces are available to determine if a socket is read-ready (`spt_fd_read_ready`) or write-ready (`spt_fd_write_ready`). The Socket implementation on NonStop systems supports streaming; multiple sends and receives are outstanding at a time.

- Be careful when using thread-aware interfaces.

  The *OSS Programmer's Guide* lists thread-aware equivalents of NonStop system-specific interfaces. These interfaces have an explicit `spt_` prefix.

  For example, when using a thread-aware function, do not attempt to control the set of files that are enabled for completion or directly attempt to wait for a completion on a file registered with pthreads (`FILE_COMPLETE_SET_`, `FILE_COMPLETE_`, `AWAITIO`, or `AWAITIOX` procedure).

Java Print Service (JPS)

The Java Print Service is implemented in NonStop Server for Java 6.0. The Java Print Service allows you to print on printers directly to NonStop systems and to network printers attached to a local area network (LAN) or wide area network (WAN). For information on configuring network printers, see the *Spooler FASTP Network Print Processes Manual*. For information on the Spooler subsystem, see the *Spooler Utilities Reference Manual*.

The Java Print Service implemented into NonStop Server for Java 6.0 uses the headless version of the javax.print API. All printing features and attributes in the JPS classes listed below work when the NonStop spooler and printer support the API. However, the NonStop operating system requirement for sending text and postscript files to separate printers also applies when printing under JPs The JPs classes are:

- javax.print
- javax.print.attribute
- javax.print.attribute.standard
- javax.print.event

NOTE: For applications using the `java.awt.print.PrinterJob` class, the printer should be postscript enabled. For information on enabling postscript printing, see the *Spooler FASTP Network Print Processes Manual*.

Using the Guardian Printer

NonStop Java API accepts the Guardian printer filenames.

The following code fragment shows how to set the Guardian printer filename and print the `print.txt` file.

```
..
String printer = "\$s.#<guardian-printer-name>";
FileInputStream stream = new FileInputStream("print.txt"); // file to print
..
PrintServiceAttributeSet prAttr = new HashPrintServiceAttributeSet();
prAttr.add(new PrinterName(printer, null));
PrintServiceLookup lookup = new UnixPrintServiceLookup();
PrintService[] services = null;
..
services = lookup.getPrintServices(null, prAttr);
..
DocPrintJob job = services[0].createPrintJob();
SimpleDoc doc = new SimpleDoc(stream, DocFlavor.INPUT_STREAM.AUTOSENSE, null);
..
job.print(doc, null);
```

. .
. .

ThreadDumpPath Support

The execution stack trace of all Java threads in a NonStop Java 6.0 process can be dumped by sending a SIGQUIT signal to the Java process, using the following OSS command:

```
$ kill -QUIT <pid>
```

By default, the thread stack dump is written in text format on *stdout*. NSJ 6.0 SPR- T2766H60^ABX introduces the ability to redirect this output to a user-defined file using the following Java command line option:

```
-XX:ThreadDumpPath=<path/filename>
```

Dynamic Saveabend File Creation

The Dynamic Saveabend File Creation feature helps to create a saveabend file of a running Java process, without abending (aborting) the Java process, by issuing a signal to the process. This feature allows the Java process to continue execution even after the abend file is created. The time taken to create the abend file, that is, the application pause time is low (measurable in milliseconds).

The saveabend file enables you to analyze any observed Java runtime problems, such as observed high memory consumption, low responsiveness, without impacting the running Java process.

To create a saveabend file in the working directory of the process, complete the following steps:

1. Export DUMP_CORE=1.
2. Start the Java application.
3. Press Ctrl-break while the process is running.

Creating Child Process Using the `-Dnsk.java.fastExec=true` Option

For applications that use the `Runtime.exec` method to create a child process, NonStop Java 6.0 version T2766H60^ABP and later provides a faster method to create a child process using the `-Dnsk.java.fastExec=true` option. The `Runtime.exec` method forks a child process and overlays its image with the new executable. Forking the child process duplicates the parent process image, which is discarded when the exec is executed. The `-Dnsk.java.fastExec=true` option builds the child process image from the executable. It does not inherit the parent process image, thereby reducing the time required to create a child process.

Preemptive User Threads On NonStop Server For Java

The preemptive user threads feature enables you to specify the time slice for threads and thus helps in thread scheduling on the NSK systems. A thread will run for the specified time slice, after which another thread will get dispatched from the ready queue. This helps in yielding a thread which consumes large processor time and allowing the other ready threads to run.

To enable preemptive user threads, use the `-XX:ThreadTimeSlice[=T]` option.

Syntax:

```
-XX:ThreadTimeSlice[=T]
```

where,

T specifies the time in milliseconds.

NOTE:

- T is an optional argument.
- The default value of T is 40 milliseconds.
- Values of T can range between 0 to 32767. If the specified value of T is above 32767, the value is time-sliced to 32767.

Java Authentication and Authorization Service (JAAS)

The Java Authentication and Authorization Service (JAAS) is integrated into the NonStop Server for Java 6.0. JAAS augments the core Java 2 platform with facilities to authenticate and enforce access controls upon users. JAAS, also, has the ability to enforce access controls based on who runs the code.

JAAS implements a Java version of the standard Pluggable Authentication Module (PAM) framework. This pluggability permits applications to remain independent from underlying authentication technologies. New or updated authentication technologies can be plugged in without requiring modifications to the application itself. Applications enable the authentication process by instantiating a `LoginContext` object, which in turn references a Configuration to determine the authentication technology, or `LoginModule`, to be used in performing the authentication. The `LoginModule` interface gives developers the ability to implement different kinds of authentication technologies that can be plugged in under an application. For example, one type of `LoginModule` may perform a username-password-based form of authentication. Other `LoginModules` may involve more sophisticated authentication mechanisms.

The NonStop Server for Java 6.0 product includes `LoginModule` interfaces implemented by Sun Microsystems, such as `JndiLoginModule` and `KeyStoreLoginModule`, but does not provide a `LoginModule` that interfaces to the Safeguard subsystem on NonStop Systems. You can also develop your own `LoginModule` implementation.

For more information on writing a `LoginModule` implementing an authentication technology, see the *JAASLoginModule Developer's Guide*.

(http://java.sun.com/javase/6/docs/technotes/guides/security/jaas/JAASLMDevGuide.html)

JavaBeans

JavaBeans are reusable software components that can run in both a design environment (inside a s fsfbuilder tool) and a runtime environment.

The design environment is highly visual and requires that JavaBeans provide design information to the programmer and allow the end user to customize its behavior and appearance.

In the runtime environment, JavaBeans might be visible, as in the case of a Graphical User Interface (GUI), or invisible, as in the case of a data feed control.

Because of the nonvisual nature of the NonStop operating system, the NonStop Server for Java 6.0 supports only runtime execution of invisible JavaBeans. The NonStop Server for Java 6.0 does not support design-time execution or runtime execution that requires a GUI operation. For this reason, in the NonStop Server for Java 6.0, the Boolean expression `java.beans.Beans isGuiAvailable` returns the value `false`.

The NonStop Server for Java 6.0 includes the JavaBeans Development Kit (BDK).

For more information about JavaBeans, see the Sun Microsystems JavaBeans document

(http://java.sun.com/javase/6/docs/technotes/guides/beans/index.html).

Debugging Java Programs

This subsection discusses the debugger architecture and how to run the programs involved in debugging Java applications. The topics are:

Debugging Overview

NonStop Server for Java 6.0 supports Java Platform Debugger Architecture (JPDA) that provides debugging support for the Java platform. JPDA consists of a three-layered set of APIs:

JDI

Java Debug Interface—A high-level Java language interface, includes support for remote debugging that is used by debugger applications

JDWP

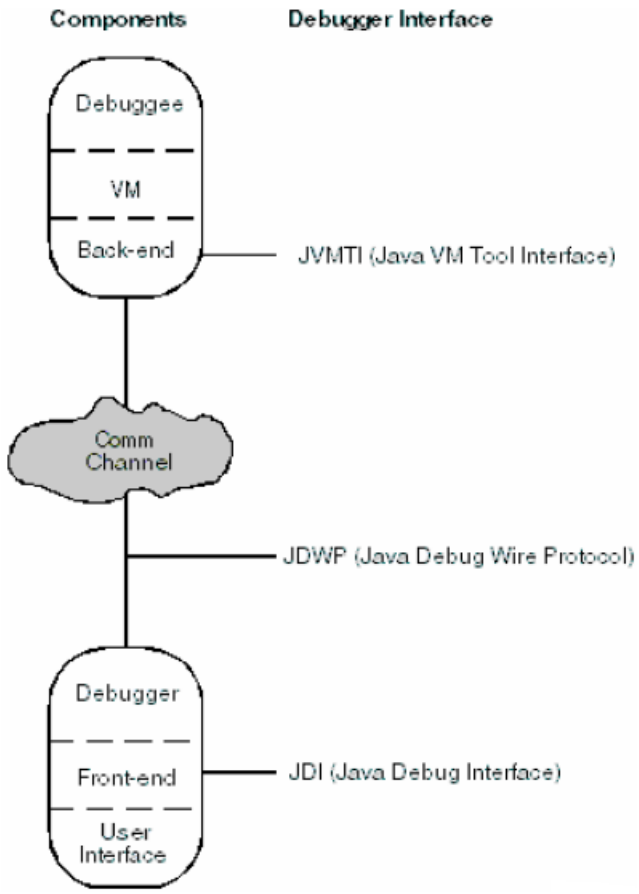
Java Debug Wire Protocol—Defines the format of the request between the debugger and the debuggee (the application that is being debugged)

JVM TI

Java Virtual Machine Tool Interface—A programming interface used by development and monitoring tools that provides both a way to inspect the state and to control the execution of applications running in the Java VM, and thereby, defines the debugging services a VM provides.

The structure of JPDA is shown in the diagram below.

The Java Platform Debugger Architecture Structure



For more details, see the Sun Microsystems Java Platform Debugger Architecture Documentation (http://docs.oracle.com/javase/6/docs/technotes/guides/jpda/architecture.html).

JDI specification does not provide in-process debugging. Hence, a Java based tool called a debugger is used to debug another JVM running the Java application. This JVM, called the debuggee, also contains the back-end of the debugger that is responsible to accept the request from the front-end of the debugger and to respond to these requests.

NonStop Server for Java 6.0 ships a terminal oriented non-GUI debugging tool. This Java Debugger Tool (JDB) can be used on a NonStop system platform to debug the Java applications running on the same NonStop system or on another NonStop system or any other platform.

You can also use the non-GUI debugging jdb and other vendors' GUI debuggers running on Microsoft Windows and other platforms to debug NonStop Server for Java 6.0 applications running on a NonStop system.

Transports

A JPDA transport is a form of inter-process communication used by a debugger application and the debuggee. NonStop Server for Java 6.0 provides a socket transport that uses the standard TCP/IP sockets to communicate between debugger and the debuggee.

NonStop Server for Java 6.0 defaults to socket transport. NonStop Server for Java 6.0 does not support shared memory transport.

java Command Line Options to Run a Debuggee

For remote debugging, you need to start the Java application to be debugged (debuggee) as a server using the following command:

```
java -Xdebug -Xnoagent
```

`-Xrunjdwp:transport=dt_socket,server=y,address=port_no,suspend=y classname arguments`

`-Xdebug`

Enables debugging.

`-Xnoagent`

Disables the old Sun.tools.debug agent. This is the default.

`-Xrunjdwp:sub-options`

`sub-options` are specified in the following format:

`name1[=value1],name2.=[=value2]...`

The *`sub-options`* are:

transport

Name of the transport to use. `DT_socket` is the value for this option. NonStop Server for Java 6.0 defaults to `DT_socket`.

`server=y`

`y` means listen for a debugger application

`address=` transport-address-for-this-connection

The transport address is the port number in which the debuggee is listening on for the debugger or a range of port value from which the debuggee selects the first available port to listen for the debugger.

The following syntax is used:

`address=[<name>:]<port> | <start port>-<end port>`

where,

`<name>` is the host name.

`<port>` is the socket port number.

`<start port>` is the starting port number for a range of ports.

`<end port>` is the ending port number for a range of ports.

`suspend=y`

Suspends the debuggee just before the main class loads.

Optionally, you can specify the `-Xint` argument to specify, by using only the interpreter and not the HotSpot compiler.

NOTE: Specifying a range of port numbers for `address` is available from `SPR T2766H60^ACC` or later. This option is specific to NonStop. The following examples show the various ways in which the connection address is specified:

Example 1 Example 1:

```
java -Xdebug -Xnoagent
-Xrunjdwp:transport=dt_socket,server=y,address=4000,
suspend=y classname arguments
```

The port number is specified in this example.

Example 2 Example 2:

```
java -Xdebug -Xnoagent
-Xrunjdwp:transport=dt_socket,server=y,address=4000-4050,
suspend=y classname arguments
```

The range of ports is specified in this example and the hostname is implicit.

Example 3 Example 3:

```
java -Xdebug -Xnoagent
-Xrunjdwp:transport=dt_socket,server=y,address=someMachine:4000-4050,
suspend=y classname arguments
```

The machine name and the port range is specified in this example.

Starting the Java Debugger (jdb) Tool

Now, the Java Debugger (`jdb`) tool can be started to communicate with the debuggee by using the `jdb` command as described for various situations.

- If you are using JDB on the same NonStop system where the debuggee runs, use

  ```
  jdb -attach portnum
  ```

- If you are using JDB on a different NonStop system from where the debuggee runs, use

  ```
  jdb -attach host-name:portnum
  ```

- If you are using JDB from Microsoft Windows or any other platform, use

  ```
  jdb -connect com.sun.jdi.SocketAttach:hostname=hostname,port=portnum
  ```

Further Information

If you are using a GUI debugger, refer to the vendors' documentation to configure the debugger to communicate with the debuggee.

Remote debugging of NonStop Server for Java 6.0 applications has been tested with Eclipse 3.4 of the Eclipse Project. For information and software downloads, see the website at http://www.eclipse.org/.

For more details on command line options, see Connection and Invocation Details (http://docs.oracle.com/javase/6/docs/technotes/guides/jpda/conninv.html).

Debugging JNI Code

To debug native code that the application writers wrote and linked with the Java program, use the inspect debugger tool available on the NonStop system. Use Visual Inspect (the preferred debugger) or Native Inspect; for further information, see the *Native Inspect Manual*.

You can use the following command to start `java` under an inspect debugger:

```
run -debug java java_options
```

To debug native code, load the DLL first. Visual Inspect lets you stop the program after the DLL is loaded so you can set breakpoints.

You can see and debug only the native routine to be debugged and other native routines that routine calls. All other scopes above the native routine are compiled or interpreted Java code, which the inspect debugger has no knowledge about.

Using Visual Inspect To Add an Event Breakpoint on DLL Open Event

Because Visual Inspect does not support deferred breakpoints, you need to ensure that a DLL is loaded before setting a breakpoint. Visual Inspect supports the DLL Open event breakpoint that suspends the program just after a DLL is loaded but before initialization routines are invoked.

To add an Event Breakpoint on DLL Open event:

1. In Visual Inspect, choose **View —> Breakpoints** in Application or Program Control view.
2. Click the **Event** tab.
3. Click **Add Breakpoint** and select **DLL Open** from **Event Name** drop-down menu.
4. Click **OK** .

Debugging Java and JNI Code

You can use the Native Inspect debugger tool to debug the native code and the Java Debugger tool to debug the Java code at the same time. You need to start the Java debuggee process under a debugger. For example, type the following command.

```
run -debug java -Xdebug -Xnoagent -Xrunjdwp:sub-options
```

Then, you can use the Java Debugger tool to communicate with the debuggee process as explained under Debugging Overview"Debugging Overview" (page 54).

Deviations in JVM Specification Options

The compiler specification options for both the `java` and `jdb` tools deviate from standard Java because NonStop Server for Java 6.0 implements only the HotSpot server VM and does not implement a client VM. Accordingly, the options that specify running the client VM are not valid.

java: Java Application Launcher Command Line Option Deviations

`-client`

Selects the Java HotSpot Client virtual machine (VM).

NOTE: The `-client` option is not valid with NonStop Server for Java 6.0.

`-server`

Selects the Java HotSpot Server virtual machine (VM).

NOTE: `-server` the default option for NonStop Server for Java 6.0; therefore, specifying `-server` is optional.

For more information about the `java` tool and additional deviations from standard Java, see "Implementation of Garbage Collector Types" (page 61) and `java` in the *NonStop Java Tools Reference Pages*.

jdb: Java Debugger Command Line Option Deviations

`-tclient`

Runs the application in the Java HotSpot client VM.

NOTE: The `-tclient` option is not valid with NonStop Server for Java 6.0.

`-tserv`

Runs the application in the Java HotSpot server VM.

NOTE: `-tserv` is the default option for NonStop Server for Java 6.0; therefore, specifying `-tserv` is optional.

For more information about `jdb` and how to start a Java program so that `jdb` can attach to it, see `jdb` in the NonStop Java 6.0 Tools Reference Pages.

Garbage Collection (GC)

This subsection discusses implementation-specific information about garbage collection for NonStop Server for Java 6.0. The topics are:

General Information on Garbage Collection

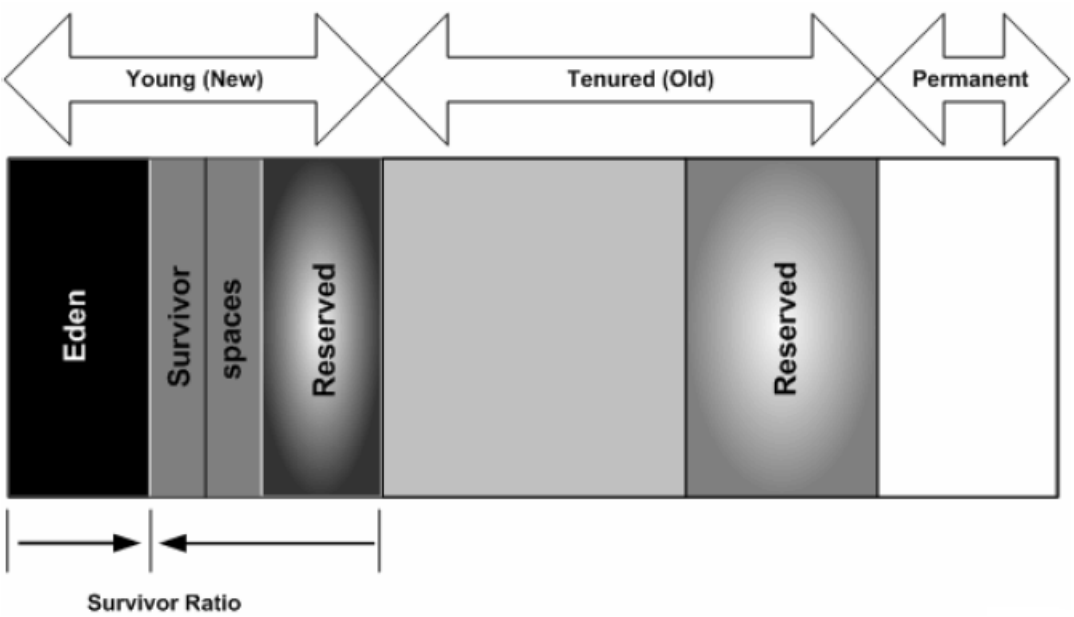
In general, garbage collectors, the various GC algorithms, and modeling in the NonStop Server for Java 6.0 are the same as those implemented by Sun Microsystems in their JVM. Accordingly, you should refer to the SunMicrosystems web site for details about garbage collection. But keep in mind that some of the information is not applicable to NonStop Server for Java 6.0. Links to pertinent information (but not all the information) on the SunMicrosystems web site are:

- Java SE 6 HotSpot Virtual Machine Garbage Collection Tuning
  http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html)
- Turbo-charging the Java HotSpot Virtual Machine, v1.4.x to Improve the Performance and Scalability of Application Servers by Alka Gupta and Michael Doyle
  (http://java.sun.com/developer/technicalArticles/Programming/turbo/)
- Improving Java Application Performance and Scalability by Reducing Garbage Collection Times and Sizing Memory Using JDK 1.4.1 by Nagendra Nagarajayya and J. Steven Mayer
  (http://developers.sun.com/mobility/midp/articles/garbagecollection2/)

Heap Layout

In NonStop Server for Java 6.0, the memory is managed in generations (or memory pools) based on objects at different ages for Java objects. "Layout for Generations" (page 60) is illustrated and described below.

Layout for Generations



The generations are:

- Young (also called new) generation—The JVM allocates objects in the young generation pool. Minor garbage collection happens when this young generation is full and the JVM is unable to allocate new objects. The young generation also includes two survivor spaces. One survivor space is empty at any time and serves as a destination of the next GC operation, which copies the collection of any live objects in Eden and the other survivor space. Objects are copied between survivor spaces in this way until they are old enough to be tenured—copied to the tenured generation.
- Tenured (also called old) generation—The JVM moves objects that survived minor garbage collections from the young generation to the old generation.
- Permanent generation—Class objects and metadata objects are allocated in permanent generation.

The young and tenured generations each have an area called "Reserved," which is allocated at initialization and used when garbage collection does not have free sufficient space to satisfy the allocation request. In a Sun Microsystems implementation, the address range for this area is reserved but memory space is not allocated until it is used.

Managing Generation Size

Several `java` command options allow you to manage the initial size and maximum size of the combined young and tenured generations.

`-Xms`

Sets the initial size for the combined young and tenured generation. The default initial size is 3058 kilobytes (K). Smaller values lead to shorter but more frequent garbage collections, larger values lead to longer but less frequent garbage collections. For large server applications, HP recommends that the initial size be equal to the maximum size.

`-Xmx`

Sets the maximum size for the combined young and tenured generation. The default maximum size is 64 megabytes (MB).

`-XX:MaxPermSize`

Sets the maximum size for the permanent generation. The default value for `MaxPermSize` is 32 MB. The initial size of the permanent generation `PermSize` option is ignored.

NOTE: At initialization time, the maximum size of the combined young and tenured generations and the maximum size of permanent generation are allocated.

Implementation of Garbage Collector Types

The default garbage collectors in NonStop Server for Java 6.0 are:

- Copying collector for the young generation
- Mark-sweep collector for the tenured generation

NonStop Server for Java 6.0 uses these garbage collectors because they are the most efficient on the NonStop system. A summary of garbage collector implementations appears in Table 7 (page 61). Paragraphs that follow discuss various implementations to help you understand performance issues.

Table 7 Summary of Garbage Collector Implementations

Collector Type	Implementation Status
Copying collector	Default collector for the young generation
Mark-sweep collector	Default collector for the tenured generation
"Parallel Collector" (page 61)	Disabled
"Concurrent Low-Pause Collector" (page 62)	Disabled
"Incremental Low-Pause Collector" (page 62)	Allowed

Parallel Collector

The parallel collector (or throughput collector) uses the parallel version of the young generation collector. The parallel collector is disabled in NonStop Server for Java 6.0 because this collector applies only for Symmetric Multiprocessing (SMP) type machines. NonStop systems are not SMP type machines.

The following `java` command options that specify a parallel collector or apply to a parallel collector are not valid for NonStop Server for Java 6.0:

`-XX:+UseParallelGC`

Specifies a parallel garbage collector. If you specify this option, the JVM exits with the error: `-XX:+UseParallelGC` option is not supported on this platform.

`-XX:+UseParNewGC`

Specifies a parallel garbage collection. This option is disabled. If you specify this option, the JVM exits with the error: `-XX:+UseParNewGC` option is not supported on this platform.

`-XX:+UseAdaptiveSizePolicy`

Specifies an adaptive size policy. This option applies only for a parallel collector and, therefore, is disabled. If you specify this option, the JVM exits with the error: `-XX:+UseAdaptiveSizePolicy` option is not supported on this platform.

`-XX:+AggressiveHeap`

Obtains the platform resources and configures the heap layout accordingly, uses parallel collector, and enables `AdaptiveSizePolicy` option. This option applies

only for a parallel collector and, therefore, is disabled. If you specify this option, the JVM exits with the error: `-XX:+AggressiveHeap` option is not supported on this platform.

`-XX:GCHeapFreeLimit=space-`*`limit`*

Specifies the lower limit on the amount of space freed during a garbage collection in percentage of the maximum heap.

`-XX:GCTimeLimit=time-`*`limit`*

Specifies the upper limit on the amount of time spent in garbage collection in percent of total time.

Also, the following flags of the Garbage Collector Ergonomics technology for a parallel collector are not supported. These specify performance goals for the application.

```
-XX:MaxGCPauseMillis=nnn
-XX:GCTimeRatio=nnn
```

Concurrent Low-Pause Collector

A concurrent low-pause collector is for tenured generation. This collector does most of the collection concurrently with the application execution. This collector divides the collection into different phases and does some phases concurrently with the application execution and others in "stop the world" mode. This collection technique does not perform well as the default collector when the phases of the collection cannot run concurrently with the application thread. This garbage collector, therefore, is disabled in NonStop Server for Java 6.0.

The following `java` command options that specify a concurrent low-pause collector are not valid for NonStop Server for Java 6.0:

`-XX:+UseConcMarkSweepGC`

Specifies using the concurrent mark-sweep garbage collector. If you specify this option, the JVMexits with the error: `-XX:+UseConcMarkSweepGC` option is not supported on this platform.

`-Xconcgc`

Enables a concurrent mark-sweep garbage collector. This option is disabled. If you specify this option, the JVM exits with the error: `-Xconcgc` option is not supported on this platform.

Incremental Low-Pause Collector

By careful book-keeping, the incremental low-pause collector collects just a portion of the tenured generation at each minor collection. The collector tries to spread the large pause of a major collection over many minor collections. In overall throughput, this collector is slower than the default tenured generation collector.

To use the incremental low-pause collector, specify the following `java` command option:

`-Xincgc`

Specifies using the incremental low-pause garbage collector.

Java Garbage Collector Tuning for Application Performance

The NonStop Server for Java 6.0 incorporates the HotSpot VM. This topic discusses the options available for tuning the JVM, suggests flags and values to tune the Java HotSpot VM, and points to HotSpot tuning information on the Internet. GC tuning is assisted by GC profile data which can be collected and analyzed as explained in the next section.

Since GC takes place when generations fill up, the amount of available total heap memory has direct impact on the performance. The parameters that affect the heap are listed below.

`-Xms` : the startup size of memory allocation pool (the GC heap)

`-Xmx` : the maximum memory allocation pool

The maximum value for the NonStop system depends on the location of the QIO segment. If the segment is moved to KSEG2, the maximum value can be as high as 900 MB, otherwise, the maximum value may stay around 350 MB. For information on QIO segment location, see "Memory Considerations: Moving QIO to KSEG2" (page 66). Also, for more information on the `-Xms` and `-Xmx` options, see "Managing Generation Size" (page 60).

For large server applications, the default values of the two options listed are usually not adequate. In general, you should grant as much memory to the JVM as possible.

Another important factor that affects performance is the proportion of the heap that is assigned to the young generation. The parameters for the young-generation heap are listed below:

- `-XX:NewRatio=nn` The ratio between the young and old.
- `-XX:NewSize=nn` The lower size bound.
- `-XX:MaxNewSize=nn` The upper size bound.
- `-XX:SurvivorRatio=nn` Tune the survivor spaces (illustrated in the "Layout for Generations" (page 60)).

For example:

```
java -Xms512m -Xmx512m -XX:NewSize=256m -XX:MaxNewSize=256m \
-XX:SurvivorRatio=2 class
```

These options inform the JVM to set the initial size of the heap to 512 MB, the maximum heap to 512 MB, the new generation to 256 MB (128 MB belongs to Eden and 2x64 MB survivor) and the old generation to 256 MB.

For details about all these parameters, see Java HotSpot VM Options

(http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html)

For details about tuning garbage collection with the Java SE 6 HotSpot virtual machine and for general performance tuning information, see:

- Java SE 6 HotSpot Virtual Machine Garbage Collection Tuning

  (http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html)
- The article Tuning Java I/O Performance

  (http://java.sun.com/developer/technicalArticles/Programming/PerfTuning/)
- The article, Turbo-charging Java HotSpot Virtual Machine, v1.4.x to Improve the Performance and Scalability of Application Servers
  (http://java.sun.com/developer/technicalArticles/Programming/turbo/)
- Java Performance Documentation (http://java.sun.com/docs/performance/)

NOTE:

- As described earlier, the NonStop Server for Java 6.0 does not support the throughput collector (specified with the option `-XX:+UseParallelGC`) or the concurrent low pause collector (specified with the option `-XX:+UseConcMarkSweepGC`).
- Also, the following options do not apply either for the NonStop Server for Java 6.0 or the NonStopsystem:

  `-XX:+UseBoundThreads` — Option to bind user level threads; Solaris specific

  `-XX:+UseAltSigs` — Solaris specific.

  `-XX:+UseV8InstrsOnly` — Only for Sparc (Solaris)

  `-XX:-AllowUserSignalHandlers` — Solaris specific

  `-XX:AltStackSize=nn` — Solaris specific

  `-XX:+MaxFDLimit` — Solaris specific

  `-XX:-UseBoundThreads` — Solaris specific

  `-XX:-UseLWPSynchronization` — Solaris specific

  `-XX:PreBlockSpin=nn` — Only for Linux

  `-XX:-UseISM` — Solaris specific

  `-XX:-UseMPSS` — Solaris specific

  `-XX:-UseSpinning` — Only for Linux

Java GC Profiling

The NonStop Server for Java 6.0 supports an HP proprietary option, `-Xverbosegc` to capture the java application's GC activity. The output of this tool can be used to view and analyze the detailed statistics in an offline mode with HPjmeter. The `-Xverbosegc` option prints out detailed information about the spaces within the Java Heap before and after the garbage collection.

The syntax :

```
-Xverbosegc[:help]|[0|1][:file=[stdout|stderr|<filename>]]
```

Fore more information on the syntax options of `-Xverbosegc`, see `java` in the *NonStop Server for Java 6.0 Tools Reference Pages*.

HeapDumpOnly option

The -XX:+HeapDump option can be used to observe memory allocation in a running Java application by taking snapshots of the heap over time.

Another way to get heap dumps is to use the `_JAVA_HEAPDUMP` environment variable; setting this environment variable enables you to take memory snapshots without making any modifications to the Java command line. To enable this functionality, either use the command line option or set the environment variable (for example, export `_JAVA_HEAPDUMP=1`) before starting the Java application.

The HeapDumpOnly log contains only the heap dump and not the thread stack trace dump. If required, you can use the `-Xrunhprof:heap=dump` option to produce a log that contains both the heap dump and thread stack trace dump.

With the -XX:+HeapDump option enabled, each time the process is sent a SIGQUIT signal, the Java VM produces a snapshot of the Java heap in hprof ASCII format:

```
java_<pid>_<date>_<time>_heapDump.hprof.txt.
```

If you set the `_JAVA_HEAPDUMP_ONLY` option, heap dumps are triggered by SIGWINCH instead of SIGQUIT. Only the heap dump is produced; that is, the thread and trace dump of the application

to stdout is suppressed. Setting the `_JAVA_BINARY_HEAPDUMP` environment variable along with `_JAVA_HEAPDUMP_ONLY` produces a binary format heap dump and instead of ASCII, the SIGWINCH is sent to the process.

NOTE: Before producing the heap dump, JVM performs a full GC.

Other HeapDump Options

In addition to -XX:+HeapDump, there are three other HeapDump options available: `-XX:+HeapDumpOnCtrlBreak`, `-XX:+HeapDumpOnOutOfMemoryError`, and `-XX:+HeapDumpOnly`. The following table lists the heap dump options.

Option	Trigger	Format	Filename
-XX:+HeapDump	SIGQUIT	ASCII	Set the `_JAVA_BINARY_HEAPDUMP` environment variable to get binary `java_<pid>_<date>_<time>_heapDump.hprof.txt`
-XX:+HeapDumpOnCtrlBreak	SIGQUIT	Binary	`java_<pid>.hprof.<millitime>`
-XX:+HeapDumpOnOutOfMemoryError	Out of Memory	Binary	`java_<pid>.hprof` or the file specified by -XX:HeapDumpPath=file
-XX:+HeapDumpOnly	SIGWINCH	ASCII	Set the `_JAVA_BINARY_HEAPDUMP` environment variable to get binary `java_<pid>_<date>_<time>_heapDump.hprof.txt`

The heap dump options are described as follows:

`-XX:+HeapDumpOnCtrlBreak`

It enables taking snapshots of the Java heap when a SIGQUIT signal is sent to the Java process, without using the JVMTI-based -Xrunhprof:heap=dump option. This option is similar to the -XX:+HeapDump option, except the output format, which is in binary hprof format and the output is placed into a filename with the following naming convention: `java_<pid>.hprof.<millitime>`.

If the HP environment variable `_JAVA_HEAPDUMP` is set and the `-XX:+HeapDumpOnCtrlBreak` option is specified, both hprof ASCII and binary dump files are created when a SIGQUIT is sent to the process. For example, the following files are created: java_27298.hprof.1152743593943 and java_27298_060712_153313_heapDump.hprof.txt. If `JAVA_BINARY_HEAPDUMP` is set and the -Xrunhprof:heap=dump command is run, both hprof ASCII and binary files are produced for this option.

`-XX:+HeapDumpOnOutOfMemoryError`

This option enables dumping the Java heap when the Java process encounters a `java.lang.OutOfMemoryError` exception. . The heap dump file name defaults to `java_pid<pid>.hprof` in the current working directory. The option -XX:HeapDumpPath=file can be used to specify the heap dump file name or a directory where the heap dump file must be created. The only heap dump format generated by the -XX:+HeapDumpOnOutOfMemoryError option is the hprof binary format.

NOTE: The -XX:+HeapDumpOnOutOfMemoryError option does not work with the low-pause collector (option -XX:+UseConcMarkSweepGC).

`-XX:+HeapDumpOnly`

The -XX:+HeapDumpOnly option or the `_JAVA_HEAPDUMP_ONLY` environment variable can be used to enable heap dumps using the SIGWINCH signal (signal 12). This interface is provided to separate the generation of thread and trace information triggered via SIGQUIT from the heap

dump information. If the -XX:+HeapDumpOnly option is specified or the `_JAVA_HEAPDUMP_ONLY` environment variable is set, the heap dump functionality is triggered by sending SIGWINCH to the process. The printing of thread and trace information to stdout is suppressed.

The heap dump is written to a file with the following filename format:

`java_<pid>_<date>_<time>_heapDump.hprof.txt.`

The default output format is ASCII. The output format can be changed to hprof binary format by setting the `_JAVA_BINARY_HEAPDUMP` environment variable. This environment variable can also be used with the -XX:+HeapDump option to generate hprof binary format with the SIGQUIT signal.

Using Heap Dumps to Monitor Memory Usage

By creating a series of heap dump snapshots, you can see how the number and size of objects varies over time. It is a good idea to collect at least three snapshots. The first one serves as a baseline. It should be taken after the application has finished initializing and has been running for a short period. The second snapshot can be taken after the residual heap size has grown significantly. You can monitor the residual heap size using -Xverbosegc and HPjmeter. Take the last snapshot just before the heap has grown to a point where it causes problems resulting in the application spending the majority of its time doing full GCs. If you take other snapshots, spread them out evenly based on residual heap size throughout the running of the application. The leak is easier to track down if the difference in size between heap dumps is large.

After you have collected the snapshots, read them into HPjmeter (run with -Xverbosegc to monitor memory usage).

When creating heap dumps, running the application with smaller heap sizes will result in smaller heap dump files. Smaller heap dump files enable HPjmeter analysis to use less memory. Read two files in HPjmeter and compare them using the File->Compare option. You should be able to find out the types of objects that are accumulating in the Java heap. Select a type using the Mark to Find option and go back to a view of one of the snapshots. Go to the Metric->Call Graph Tree option and do a Find. You should be able to see the context of the object retention.

JVM Tuning Tools

PrintGCStats is a tool for mining “verbose:gc” logs that can aid analyzing and tuning garbage collection. You can download this tool from the following location:
http://java.sun.com/developer/technicalArticles/Programming/turbo/PrintGCStats.zip

Additionally, for a discussion of profiling tools, see “Application Profiling” (page 76) section.

Tuning Application Performance

The topics are

Memory Considerations: Moving QIO to KSEG2

Java server-side applications are typically configured with large Java heap sizes, in excess of 128 MB. In addition, the JVM and its native components (for example, NonStop Servlets for JavaServer Pages (NSJSP) transport library, JDBC Driver for SQL/MP, JDBC Driver for SQL/MX, SQL/MX call-level interface, and potentially any custom-user JNI code) allocate memory for their own use. Thus, the overall heap space required by a JVM process can be considerably higher than the configured Java heap space.

When a process uses the parallel TCP/IP transport provider for socket operations (like the iTP Secure WebServer httpd daemon Server process instance), the process becomes a QIO client. For NonStop™ Server QIO shared-memory segments, when a QIO client process makes a socket request, a portion of the process address space is reserved for the QIO segment. This reserved space limits the maximum usable heap size for the JVM process.

The size of the QIO segment is determined by the configured physical memory for a processor (CPU) on the NonStop system. For a processor on a NonStop system configured with 4 GB of physical memory, 512 MB are used for the QIO segment. In addition, 512 MB of the user address space are reserved for the QIO segment, if the process is also a QIO client.

To overcome the problem of losing address space to QIO, you can configure the QIO segment so that the segment is moved to a special region of the privileged address space, called the KSEG2 region. Moving the QIO segment to the KSEG2 region means that the maximum size of the QIO segment is restricted to 128 MB (instead of the default512 MB). Before moving QIO to KSEG2, check that the current maximum QIO segment size in use is within the 128 MB limit (allowing for future requirements that may increase the QIO segment size). If the maximum QIO segment size is within the limit, you can move QIO to KSEG2.

Example: Determining the QIO Segment Size in Use

To determine the QIO segment size in use, use the following SCF command from the TACL prompt:

```
TACL> scf
SCF - T9082G02 - (30APR03) (01APR03) - 03/19/2004 02:20:31 System \NAVAE1
(C) 1986 Tandem (C) 2003 Hewlett Packard Development Company, L.P.
(Invoking \NAVAE1.$DATA11.RAMR.SCFCSTM)
1-> status segment $ZM00, detail.

QIO Detailed Status SEGMENT \NAVAE1.$ZM00

State.................. DEFINED
Segment State.......... STARTED
Segment Type........... FLAT_UA
Segment Size........... 536870912
MDs in Use............. 1258
Max MDs Used........... 1589
Last Fail Size......... 0
Current Pool Size...... 16774788 Initial Pool Size...... 16776992
Max Pool Size.......... 16776992 Min Pool Size.......... 16776992
Current Pool Alloc..... 5039616 Max Pool Alloc.......... 5128320
Current Pool Frags..... 12 Max Pool Frags............... 18
```

The maximum pool size used in this case is 16 MB, which is well below the 128 MB limit, so QIO can be moved to KSEG2.

Example: QIO Segment Moved to KSEG2

The following SCF output shows the QIO segment as moved to KSEG2.

```
TACL> scf
SCF - T9082G02 - (30APR03) (01APR03) - 03/19/2004 02:20:00 System \GOBLIN
(C) 1986 Tandem (C) 2003 Hewlett Packard Development Company, L.P.
(Invoking \GOBLIN.$DATA11.RAMR.SCFCSTM)
1-> status segment $ZM00, detail

QIO Detailed Status SEGMENT \GOBLIN.$ZM00

State.................. DEFINED
Segment State.......... STARTED
Segment Type........... RESIDENT
Segment Size........... 134217728
MDs in Use............. 1248
Max MDs Used........... 2357
Last Fail Size......... 0
Current Pool Size...... 16774788 Initial Pool Size...... 16776992
Max Pool Size.......... 16776992 Min Pool Size.......... 16776992
```

```
Current Pool Alloc..... 4516992 Max Pool Alloc.......... 4715520
Current Pool Frags..... 375 Max Pool Frags.............. 382
```

The QIO segments on this system (\GOBLIN) have been moved to KSEG2 based on the value of the segment type.

The value is RESIDENT if QIO is moved to KSEG2.

The first SCF output for \NAVAE1 shows QIO to be in FLAT_UA, which means that QIO has not been moved to KSEG2.

Determining the Heap Manager

The C Runtime Heap manager (T1269) offers a substantial performance boost over the older heap manager (T8431). While this performance boost might not affect any pure Java code, the JVM contains native (C and C++) code; therefore, using the T1269 heap manager will boost the JVM performance.

To find the C Runtime Heap Manager being used on your NonStop system, use the `vproc` command output from the ZCRESRL library; at the TACL prompt, type.

```
TACL> vproc $SYSTEM.sysnn.ZCRESRL
```

Where *sysnn* is the system number.

Determining the Heap Setting

You should set the Java heap to a value appropriate for your application. For most application-server environments, the heap size is set high to optimize performance. This size can be more than 256 MB (after QIO has been moved to KSEG2).

To study the frequency and length of the JVM Garbage Collection operation, use the `-verbose:gc (-Xverbosegc)` option in the JVM arguments. Then use this data to tune the heap usage of the JVM based on your application requirements.

NOTE: The JVM allocates the maximum required heap for the JVM usage at startup, so the swap space considerations for the JVM process are the maximum Java heap space specified in addition to other JVM memory requirements and the memory requirements for all native components.

To identify the swap usage of a process or the swap requirements for a CPU, use the NSKCOM utility. For example, to identify the swap usage of all the processes or a particular process, enter the following commands at the OSS prompt. Sample NSKCOM output follows the command line.

```
$ gtacl –p nskcom

NSKCOM - T5838G05 BASE (22JUL02) - Nov 5 2002
Copyright 1995 Compaq Computer Corporation

$SYSTEM.SYSTEM.ZSYSCFG
KMS.SWAPFILE = 0 $DATA03.SWAP.CPU0A STOP
KMS.SWAPFILE = 0 $DATA03.SWAP.CPU0B STOP
   .
   .
   .
KMS.SWAPFILE = 5 $SWAP45.SWAP4.CPU05
NSK-status swap-usage 1,423 ,detail

SYSTEM : \GOBLIN LOGTIME : March 19, 2004 02:50:29
TYPE : (E=Extensible S=Shared O=Owner)
(CPU Page size is 16384 Bytes)
```

Process		Pri	User-ID	Program File Name
$JSV1	1,423	155	103,43	$DATA11.ZYQ0000T.Z00001VH

```
Z00001VHKMSF-BACKED SEGMENTS: (Process Space Guarantee = 1904KB)
-----------------------------------------------------------------------------
SEG-ID TYPE                SIZE   RESERVATION
                    KBYTE  PAGES  KBYTE
-----------------------------------------------------------------------------
2101                   8B       1   16
2100                   8B    1   16
Heap+Global+SRL+Stack        330MB  21187  331MB
------------------------------------------------------------------------------
TOTAL                   21187  331MBFILE
331MBFILE-BACKED SEGMENTS: None
______________________________________________________________________________
NSK-
```

In the preceding output, the JVM process (`$JSV1`) uses 331 MB of swap space and has two segments allocated. This JVM process was started with a heap size of 256 MB, which shows that apart from the Java heap, the process requires about 75 MB for process-specific and application-specific data.

NOTE: To get an applicable sample of the swap usage for a particular JVM process, check this swap usage at steady state, where all the native components used by the JVM have been fully initialized and are running.

Related Tuning Guides

For related tuning guides, *BEA WebLogic Server on HP NonStop Server and Tuning Guide for iTP Secure WebServer and NonStop Servlets for JavaServer Pages (NSJSP) on HP NonStop Servers*, see http://www.hp.com/go/nonstop-docs.

Java Signal Handlers

A Java program installs signal handlers for the signals that the current application interacts with. However, there is one restriction for an application when it installs handler for the signals. The restriction is that the application cannot install handlers for the signals that are used by the Java Virtual Machine (reserved signals). If the application attempts to install handlers for the reserved signals, `java.lang.IllegalArgumentException` is thrown. There are two types of reserved signals for which a user cannot install handlers:

1. Signals for which the user cannot install handlers *always*.
2. Signals for which the user can install handlers, if the application does not enable specific command line options.

Table 8 (page 69) provides information about the reserved signals for which signal handlers cannot be installed, and also about the reserved signals which can be installed by enabling or disabling some options from command line.

Table 8 Reserved Signals List

Reserved Signals (Always)	Reserved Signals (depending upon command line options)[1]
SIGFPE	"SIGWINCH" (page 70)
SIGILL	"SIGALRM" (page 70)
SIGSEGV	"SIGUSR2" (page 70)
SIGQUIT	"SIGHUP" (page 70)
SIGUSR1	"SIGINT" (page 70)
SIGSTK	"SIGTERM" (page 70)

[1] Click on the specific signal to obtain information about how to install signal handlers for the corresponding signals.

SIGWINCH

`HeapDumpOnly` feature uses SIGWINCH signal. Hence, the application installs signal handler for this signal, provided `HeapDumpOnly` option is not enabled.

`HeapDumpOnly` option is either enabled by using `XX:+HeapDumpOnly` Java command line option or by setting 1 for the environment variable (`_JAVA_HEAPDUMP_ONLY`).

NOTE: By default, `HeapDumpOnly` option is false, hence, the application can install signal handler if it does not explicitly enable `HeapDumpOnly` option.

SIGALRM

Zero preparation verbose garbage collection feature uses SIGALRM signal. By default, `ZeroPrepVerboseGC` is enabled.

`ZeroPrepVerboseGC` must be disabled to use SIGALARM for other purposes.

SIGUSR2

Zero preparation profiling feature (HP specific feature) uses this signal. The profiling can be enabled or disabled for the entire runtime of the application or for a selected duration. The details are as follows:

- If `–Xeprof::off` is specified in the Java command line, the zero preparation profiling feature is disabled for the entire duration for which the application runs, and hence the application can install signal handler for SIGUSR2 signal.
- If `–Xeprof` is specified in the Java command line, the zero preparation profiling feature is enabled for the entire duration for which the application runs, and hence the application cannot install signal handler for SIGUSR2 signal.
- An alternate signal for zero preparation profiling feature can be specified by using:

  `Xeprof:time_on=<SIGUSR1|SIGUSR>,time_slice=<SIGUSR1|SIGUSR2>.`

  If SIGUSR2 is not specified as the signal for zero profiling feature, then application can install signal handler for SIGUSR2.

SIGHUP

JVM uses this signal to support shut down hook if `–Xrs` is not specified in the command line option (reduce signal usage). If `–Xrs` is specified in the command line option, the application can install signal handlers for SIGHUP signal.

SIGINT

JVM uses this signal to support shut down hook if `–Xrs` is not specified in the command line option (reduce signal usage). If `–Xrs` is specified in the command line option, the application can install signal handlers for SIGINT signal.

SIGTERM

JVM uses this signal to support shut down hook if `–Xrs` is not specified in the command line option (reduce signal usage). If `–Xrs` is specified in the command line option, the application can install signal handlers for SIGTERM signal.

Change in Loading of `.hotspot_compiler` and `.hotspotrc` files

From NSJ 6.0 SPR — T2766H60^ACH and later versions, the default implicit loading of the *`.hotspot_compiler`* and *`.hotspotrc`* files from the current working directory is changed.

These files are no longer loaded by default. For existing deployments that rely on *.hotspot_compiler* (for example, to exclude a method from hotspot compilation), and *.hotspotrc*, an unsupported behavioral option is provided to simulate the old loading behavior. The following command line options support old behavior:

—XX:Flags=.hotspotrc
    reverts to old behavior for *.hotspotrc*.

—XX:CompileCommandFile=.hotspot_compiler
    reverts to old behavior for the *.hotspot_compiler*.

5 Transactions

The NonStop Server for Java 6.0 lets you work with transactions in several ways. You can:

- Use the `Current` class methods to define transactions across transaction services, such as transactions that include JDBC calls.
- Use the Java Transaction API (JTA) .

This section explains these topics:

If you use JNI and transactions, see "Java Native Interface (JNI)" (page 43). When you use JNI, the information under "Controlling Maximum Concurrent Transactions" (page 72) applies.

Controlling Maximum Concurrent Transactions

NonStop Server for Java 6.0 application processes can start, by default, a maximum of 1000 concurrent transactions in each process. By setting the `JAVA_PTHREAD_MAX_TRANSACTIONS` environment variable, you can limit the maximum number of TMF transactions allowed per process to less than 1000. The syntax follows:

JAVA_PTHREAD_MAX_TRANSACTIONS environment variable

Specifies the maximum number of TMF transactions allowed per process.

Allowed values are 100 through 1000. The default value of 1000 is used when:

- The variable is not set.
- The variable is set to a value less than 100 or to a value greater than 1000.

For example, to specify 200 transactions per process, use the following command.

```
export JAVA_PTHREAD_MAX_TRANSACTIONS=200
```

NOTE: The maximum number of concurrent transactions allowed is 1000.

`Current` Class Methods

The `Current` class is based on version 0.5 of the Java Transaction Services (JTS) specification.

The following table describes the `Current` class methods. For the API specification, see the `tandem.com.tmf` package description in the *NonStop Server for Java API Reference*

Table 9 Current Class Methods

Method	Description
`begin`	Starts a new transaction and associates it with the calling thread .
`commit`	Commits the transaction associated with the calling thread.
`get_control`	Gets a `Control` object representing the transaction associated with the calling thread.
`get_status`	Gets the status of the transaction associated with the calling thread.
`get_transaction_name`	Gets a descriptive name of the transaction associated with the calling thread.
`resume`	Sets or resumes association of a transaction with the calling thread.
`rollback`	Rolls back the transaction associated with the calling thread.

Table 9 Current Class Methods *(continued)*

Method	Description
`rollback_only`	Marks the transaction associated with the calling thread so that the only possible outcome is to roll back the transaction.
`set_timeout`	Modifies the time-out value associated with transactions started by subsequent invocations of the `begin` method.
`suspend`	Suspends the association of the calling thread with a transaction context.

The following code fragment shows how to use the `begin()` and `commit()` methods of the `Current` class:

```
import com.tandem.tmf.Current;

Current tx = new Current();

// start a new transaction in the current thread
tx.begin();
// ... access Pathway server

// commit current transaction (JDBC and Pathway)
tx.commit(true);
```

For more information on the `Current` class, see the `tandem.com.tmf` package description in the *NonStop Server for Java API Reference*.

Java Transaction API (JTA)

NonStop Server for Java 6.0 supports transactions by means of the NonStop Server for Java Transaction API, which is an implementation of the Sun Microsystems JTA Version 1.0. NonStop Server for Java Transaction API implements parts of the Sun Microsystems JTA package, `javax.transaction`. NonStop Server for Java 6.0 includes the NonStop Server for Java Transaction API package `com.tandem.jta`.

The NonStop Server for Java Transaction API provides a standard interface for transactions on both homogeneous NonStop systems by means of TMF and heterogeneous CORBA systems by means of JTS.

The version of NonStop Server for Java Transaction API that uses TMF is called NonStop Server for Java Transaction API-TMF; the version of NonStop Server for Java Transaction API that uses JTS is called NonStop Server for Java Transaction API-JTS. NonStop Server for Java Transaction API-TMF and NonStop Server for Java Transaction API-JTS have identical interfaces. You can specify TMF or JTS when you use `JTAFactory.getUserTransaction` to get a reference to `javax.transaction.UserTransaction`. (See "Examples" (page 74)). The default is TMF.

NonStop Server for Java Transaction API-TMF is intended for applications other than CORBA applications. NonStop Server for Java Transaction API-JTA is intended for CORBA applications. If you use NonStop Server for Java Transaction API-JTS for non-CORBA applications, results are unpredictable.

This subsection explains the following subjects:

For more information about JTA, see the Sun Microsystems JTA document (http://java.sun.com/products/jta/index.html).

javax.transaction Interfaces

The Sun Microsystems JTA package, javax.transaction, defines the following interfaces:

- Status
- Synchronization
- Transaction
- TransactionManager
- UserTransaction

NonStop Server for Java Transaction API supports all of the preceding interfaces, but only UserTransaction is available to client programs.

UserTransaction allows the client to control transaction boundaries programmatically. UserTransaction methods do the following:

- Begin transaction
- Commit transaction
- Obtain transaction status
- Mark transaction for rollback
- Rollback transaction
- Set timeout for transaction

javax.transaction Exceptions

The Sun Microsystems JTA package, javax.transaction, defines the following exceptions. NonStop Server for Java Transaction API supports all of them.

- HeuristicCommitException
- HeuristicMixedException
- HeuristicRollbackException
- InvalidTransactionException
- NotSupportedException
- TransactionRequiredException
- TransactionRolledbackException
- SystemException

Examples

The following examples are identical except that:

- The first example uses "NonStop Server for Java Transaction API-TMF by Default" (page 74)
- The second example requests "NonStop Server for Java Transaction API-TMF by Request" (page 75)
- The third example requests "NonStop Server for Java Transaction API-JTS" (page 75)

NonStop Server for Java Transaction API-TMF by Default

The following code gets a reference to UserTransaction based on TMF (by default). It then starts and ends a transaction.

```
import javax.transaction.UserTransaction;
import com.tandem.jta.JTAFactory;
```

```
// Get a reference to UserTransaction based on TMF (by default).
UserTransaction utx = JTAFactory.getUserTransaction();

// Start transaction
utx.begin();

// Do work
...

// Commit transaction
utx.commit();
```

NonStop Server for Java Transaction API-TMF by Request

The following code gets a reference to `UserTransaction` based on TMF (which it requests). It then starts and ends atransaction.

```
import javax.transaction.UserTransaction;
import com.tandem.jta.JTAFactory;

// Get a reference to UserTransaction based on TMF (by request).
UserTransaction utx = JTAFactory.getUserTransaction(JTAFactory.TMF);

// Start transaction
utx.begin();

// Do work
...//

Commit transaction
utx.commit();
```

NonStop Server for Java Transaction API-JTS

The following code gets a reference to `UserTransaction` based on JTS. It then starts and ends a transaction.

```
import javax.transaction.UserTransaction;
import com.tandem.jta.JTAFactory;

// Get a reference to UserTransaction based on JTS.
UserTransaction utx = JTAFactory.getUserTransaction(JTAFactory.JTS);

// Start transaction
utx.begin();// Do work...

// Commit transaction
utx.commit();
```

NOTE: NonStop Server for Java Transaction API-TMF is intended for applications other than CORBA applications. NonStop Server for Java Transaction API-JTA is intended for CORBA applications. If you use NonStop Server for Java Transaction API-JTS for non-CORBA applications, results are unpredictable.

6 Application Profiling

Application profiling and monitoring are supported through HPjmeter tool, which works in combination with NSJ options, `-Xeprof` and `-agentlib:hprof` as explained in this chapter.

The NonStop Server for Java 6.0 supports the profiling of live Java applications. The HPjmeter console is a GUI tool that runs on Java platforms that support GUI, such as HP-UX, Linux, and Windows. It is used to perform the following profiling activities:

Monitoring live Java applications

The Java application must be prepared for live mode of profiling by running HPjmeter agents. The HPjmeter agents are shipped on the NSJ 6.0 product CD as a new product T0866H31.

To set up monitoring for a live Java application:

1. Install the T0866H31 PAX file in the default installation directory: `/usr/tandem/hpjmeter`
2. Run a node agent:
   a. export JMETER_HOME=/usr/tandem/hpjmeter/
   b. $ JMETER_HOME/bin/nodeagent –port *port_number*
3. Launch the Java application using the HPjmeter JVM agent:
   a. export `_RLD_LIB_PATH=$JMETER_HOME/lib/oss/`
   b. export `JAVA_HOME=/usr/tandem/java`
   c. `java -agentlib:jmeter[=options] <application>`
4. Start the HPjmeter console from a local installation on your client workstation (HP-UX, Windows, or Linux). To download HPjmeter consoles for these platforms, visit http://www.hp.com/go/hpjmeter.
5. Connect to the Node Agent from the Console.

NOTE:

- For information on monitoring capabilities and tips , see the *HPjmeter 4.2 User's Guide* available at www.hp.com/go/hpjmeter.
- The instructions for using the HPjmeter tool on the NonStop platform are provided in "Addendum to HPjmeter 4.2 User's Guide" (page 88).

Collecting profile data for analysis

There are three ways to collect application profile data for offline analysis using the HPjmeter console:

1. The eprof profiler (start Java application with `-Xeprof` option).
2. Zero-preparation profiling (start and stop profile data collection by sending signal to the running Java application—this uses the Xeprof profiler, internally, to profile the application dynamically).
3. The HPROF profiler (start Java application `-agentlib:hprof`).

NOTE:

- For information on analyzing profile data, see the *HPjmeter 4.2 User's Guide* available at www.hp.com/go/hpjmeter.
- The instructions for using the HPjmeter tool on the NonStop platform are provided in "Addendum to HPjmeter 4.2 User's Guide" (page 88).

–Xeprof

The –Xeprof option generates profile data for HPjmeter. –Xeprof focuses primarily on performance problems that characterize large server applications. It collects method clock and CPU times, method call count and call graph, and lock contention statistics.

This option creates profile data file with a file extension .eprof. This file can be opened in the HPjmeter console and the collected metrics can be viewed.

For more information on this option, see *NonStop Server for Java 6.0 Tools Reference*.

Zero Preparation Profiling

Profiling can be started from the command line by sending a signal to the Java process indicating JVM to start eprof. Engaging zero preparation profiling may have a short term impact on application performance as the JVM adjusts to the demands of performing dynamic measurements.

To collect profiling data without interrupting your application, perform the following from the command line:

1. Confirm that no –Xeprof option has been specified on the command line.
2. Find the process ID of the running Java application.
3. Start the profiling interval — send a signal to the JVM by typing the following command: `kill -USR2 pid`

   The following message is displayed.

   ```
   eprof: starting profiling
   ```

   Allow the profiling to continue for a desired length of time.
4. Stop the profiling interval by sending the same signal to the JVM:

   ```
   kill -USR2 pid
   ```

   The following message is displayed.

   ```
   eprof: terminating profiling
   ```

   Writing profile data to `./filename.eprof`.
5. You can now open the saved file in the HPjmeter console and view the collected metrics.

HPROF Profiler

HPROF is a profiling agent that is based on a profiling interface called JVMTI. By using HPROF, you can get information about your application's CPU usage, heap allocation, and threads. This agent creates profile data files that can be interpreted after the program terminates. Currently HPjmeter can read text and binary files.

To run your application with profiling, use the following command:

```
$ java ... -agentlib:hprof[=options] ApplicationClassName
```

For more information on HPROF, see *HPjmeter 4.2 User's Guide* available at www.hp.com/go/ and the Oracle documentation at http://docs.oracle.com/javase/7/docs/technotes/

NOTE: The following HPROF option is not supported on NSJ 6.0 :

```
-agentlib:hprof=cpu=samples
```

Obtaining Garbage Collection Data for Analysis

Garbage collection data can be collected in either one of the following two ways:

1. Data collection with –Xverbosegc.

   Launch the Java application using the –Xverbosegc option. For more information on the option, see *NonStop Server for Java 6.0 Tools Reference*.

2. Data collection with Zero preparation.

   Data collection can be started from the command line by sending a signal to the Java process to indicate JVM to start GC data collection.

   To collect GC data without interrupting an already running application, perform the following from the command line:

   a. Confirm that –Xverbosegc or –Xloggc option is not specified.
   b. Locate the process ID of the running Java application.
   c. Start the profiling interval. Send a signal to the JVM by typing the following command:

      ```
      kill -ALRM pid or kill -14 pid
      ```

   d. The GC data is written to a file named `java_pid.vgc` in the current directory of the JVM process.

      Allow the profiling to continue for a desired length of time.

   e. Stop the data collection interval by sending the same signal to the JVM:

      ```
      kill -ALRM pid
      ```

   f. You can now open the saved file in the HPjmeter console and view the collected metrics.

GC Log Rotation

HP's Implementation

When GC logging is enabled using the `-Xverbosegc` or `-Xloggc` option, by default, the GC data is written to a single log file of unlimited size. Starting with the NonStop Java 6.0 T2766H60^ABP release, NonStop Java supports controlling the size and number of the GC log files. The GC log records are written into the specified number of GC log files in a round-robin fashion. This allows GC data to be archived easily and helps to limit the amount of disk space consumed by the GC log files. Log rotation is also supported when using zero-preparation `-Xverbosegc`.

To enable log rotation, use the following option together with `-Xverbosegc`, `-Xloggc`, or zero-preparation `Xverbosegc`:

```
-XX:GCLogLimits=M,N
```

where,

`M` is a non-negative integer that specifies the number of rotating GC log records per file.

NOTE: Each GC log record corresponds to a GC event. A value of 0 specifies an unlimited number of GC log records per file.

`N` is a non-negative integer that specifies the maximum number of rotating GC log files. A value of 0 specifies an unlimited number of files.

You must use both `M` and `N` when you use the `-XX:GCLogLimits=M,N` option. If this option is not specified, the default behavior is to write a single GC log file with unlimited size.

When rotation is in effect, a sequence number is appended to the GC filename (0 through N-1). (Examples of file names are: `filename.0`, `filename.1`, and `filename.2`).

With log rotation, when the specified maximum number of files (N) is reached, logging cycles back to the first file in the sequence `(filename.0)`, thereby overwriting the old GC data with new data. If the maximum number of files (N) is never reached, then no log rotation occurs.

Examples:

To rotate between two log files, each with a maximum of 100,000 GC records, use:

```
-XX:GCLogLimits=100000,2
```

To maintain an unlimited number of smaller files, each with a maximum of 1,000 GC records, use:

```
-XX:GCLogLimits=1000,0
```

Oracle's Implementation

From NSJ 6.0 SPR — T2766H60^ACH and later versions, Oracle command line options support GC log file rotation. The options are as follows:

- `–XX:+UseGCLogFileRotation –XX:NumberOfGCLogFiles=<num_of_files>`
- `–XX:GCLogFileSize=<logsize>`

NOTE: `–Xverbosegc` supports the listed options.

HP's GC log rotation option, `–XX:GCLogLimits`, remains unchanged and still supports both `–Xloggc` and `–Xverbosegc` options.

NOTE: HP's `LogRotation` option `–XX:GCLogLimits=M,N` overrides Javasoft's `LogRotation` options. If HP's `–XX:GCLogLimits` is specified, GC logs rotate to new log file after `N` records.

Analyzing Garbage Collection Data

After completing the data file collection, perform the following steps:

1. Transfer the data file to an HPjmeter console compatible platform (HP-UX, Windows, or Linux).
2. Run the HPjmeter console and open the data file.

NOTE:

- Starting with NonStop Java 6.0 T2766H60^ABP release, NonStop Java supports rotational GC logging into multiple GC log files to help control the GC log file size. For information on GC log rotation, see "GC Log Rotation" (page 78).
- For information on analyzing GC profile data , see the *HPjmeter 4.2 User's Guide* available at www.hp.com/go/hpjmeter.
- The instructions for using the HPjmeter tool on the NonStop platform are provided in "Addendum to HPjmeter 4.2 User's Guide" (page 88).

-Xeprof versus -agentlib:hprof (HPROF)

Xeprof is a superior profiling tool compared to HPROF in terms of the richness of data. However, Xeprof has a little higher performance impact on an application than HPROF. For a comparison of the features of Xeprof and HPROF, before using either of the profiles, see Table 5-4 in the *HPjmeter 4.2 User's Guide*.

7 Migrating Applications

This appendix describes the changes required to migrate applications that use earlier versions of the NonStop Server for Java. Note the terminology:

- NonStop Server for Java 4 refers to the product based on J2SE SDK 1.4.x
- NonStop Server for Java 5.1 refers to the product based on J2SE JDK 5.0
- NonStop Server for Java 6.0 refers to the product based on Java SE JDK 6.0

See theTable 10 (page 80) for the topics that apply to migrating from particular versions.

Table 10 Summary of Migration Changes for NonStop Server for Java Versions

Migration Topic	Version 2 of NonStop Server for Java 4 (T2766V20 onTNS/R)	NonStop Server for Java 4 (T2766H10 onTNS/E)	NonStop Server for Java 5.1 (T2766H50)	NonStop Server for Java 6.0 (T2766H60)
"Installation Changes" (page 81)	Applicable	Applicable	Applicable	Applicable
"Public Library Directory" (page 81)	Applicable	N/A	N/A	N/A
"Java Based JAR File Locations" (page 81)	N/A	N/A	N/A	N/A
"Dynamic Link Libraries (DLLs)" (page 82)	Applicable	N/A	N/A	N/A
"Makefile to Link Native Libraries" (page 82)	Applicable	N/A	N/A	N/A
"Compiling C++ Native Code with the -Wversion3 Option" (page 82)	Applicable	N/A	N/A	N/A
"Floating-Point Support" (page 83)	N/A	N/A	N/A	N/A
"Using AWT Classes" (page 83)	N/A	N/A	N/A	N/A
"POSIX Threads" (page 84)	N/A	N/A	N/A	N/A
"Directories of Binary Files Moved" (page 84)	N/A	N/A	N/A	N/A
"Character Handling" (page 84)	Applicable	Applicable	Applicable	Applicable
"BigDecimalFormat Class" (page 84)	Applicable	Applicable	N/A	N/A
"JAAS Enhancement" (page 85)	N/A	N/A	N/A	N/A
"Miscellaneous Changes for Migration to TNS/E" (page 85)	Applicable	N/A	N/A	N/A

For information about earlier Java version changes, see the release notes at the Sun Microsystems web site for the particular version of Java. For information about changes in NonStop Server for

Java 6.0, see "Supported and Unsupported Features of NonStop Server for Java 6.0" (page 87)and "New and Changed Information" (page 8).

Installation Changes

The standard location for the NonStop Server for Java 6.0 is a location of the form:

`/usr/tandem/nssjava/jdk160_h60`

where `jdk60` refers to the version number of the Sun Microsystems update upon which NonStop Server for Java 6.0 is based. The number `h60` identifies the particular NonStop Server for Java 6.0 > PVU.

When the PAX file is extracted, a symbolic link is created at the standard location: `/usr/tandem/java`.

If you want to install NonStop Server for Java 6.0 in a nonstandard location, you can do so as with earlier releases. To do that kind of installation, use the File Transfer Protocol (FTP) to transfer the file from the CD to the NonStop system. Then follow the directions in the product Softdoc for a nonstandard installation.

NOTE: HP recommends that NonStop Server for Java 6.0 be installed in the standard location whenever possible.

Public Library Directory

The public library directory does not apply to NonStop Server for Java 4, 5, 5.1, or 6.0 hosted on TNS/E because DLLs are used on TNS/E. For information about migrating native libraries, see "Dynamic Link Libraries (DLLs)" (page 82).

Java Based JAR File Locations

For users of NonStop Server for Java 3.1.x or earlier versions, guidelines have changed for placing Java Archive files (JAR or `.jar`) both "For Java Based Products" (page 81) and "User-Provided JAR Files" (page 81).

For Java Based Products

Before version 1 of the NonStop Server for Java 4 (based on J2SE SDK 1.4.0), no guidelines were provided for where Java based products should install their JAR files. Many of these products installed their JAR files in the `/usr/tandem/java/jre/lib/ext` directory. Occasionally, the installation of a Java based product would overwrite a JAR file required by another Java based product, possibly causing a version mismatch.

In addition, Java based products had to be reinstalled whenever NonStop Server for Java issued a new product version.

For these reasons, beginning with version 1 of the NonStop Server for Java 4, HP recommends that the JAR files associated with Java based products remain in a product-specific directory.

When you follow this recommendation, you must include the JAR files of the Java based product in either your `CLASSPATH` environment variable setting or the `-classpath (-cp)` command line argument.

User-Provided JAR Files

Previously, many users also installed JAR files in `/usr/tandem/java/jre/lib/ext` because they did not have to include such JAR files in their `CLASSPATH`. Beginning with version 1 of the NonStop Server for Java 4 (based on J2SESDK 1.4.0), HP recommends you do not install user-provided JAR files in any directory of versions 1 and 2 of the NonStop Server for Java 4 tree. You should leave the JAR files in user-specific directories. If you follow this recommendation, you

will not have to reinstall user-provided JAR files for new product releases of NonStop Server for Java 4. You, however, have to place the JAR files in your `CLASSPATH`.

Dynamic Link Libraries (DLLs)

On the TNS/E platform, NonStop Server for Java 4 , 5, 5.1, and 6.0 support Dynamic-Link Libraries (DLLs). All NonStop Server for Java applications migrating from TNS/R to TNS/E must convert native libraries to DLLs.

Consider these issues when migrating applications to use DLLs with the NonStop Server for Java 6.0:

- All the Java libraries are built as DLLs.
- When using the JNI code, use DLLs instead of static libraries. For further information, see "Java Native Interface (JNI)" (page 43). A public library directory does not apply for Java applications on the TNS/E platform.
- All DLLs must be in the files that have specific naming requirements. For further information see DLL names.
- On TNS/E, the `-Dcompaq.liblist` option is not supported.
- The customer Makefile no longer exists in the NonStop Server for Java 4 , 5, 5.1, and 6.0 on TNS/E because DLL support precludes the need to bind user native code into the java executable.
- The `_RLD_LIB_PATH` environment variable, used only on the TNS/E platform, specifies the library path for user DLLs. For further information, see "_RLD_LIB_PATH" (page 40) (TNS/E Only).
- The invocation API uses the JVM as a DLL; therefore, if you use this API, you do not need to statically link Java into your programs.

Makefile to Link Native Libraries

The customer Makefile no longer exists for NonStop Server for Java 4 , 5, 5.1, and 6.0 on TNS/E because DLL support precludes the need to bind user native code into the java executable. For information about migrating native libraries, see "Dynamic Link Libraries (DLLs)" (page 82).

Compiling C++ Native Code with the -Wversion3 Option

For TNS/E, you can use C++ code compiled using either a dialect of version 2 or version 3 for user DLLs because the NonStop Server for Java 4 , 5, 5.1, and 6.0 on TNS/E is built with a C++ version neutral flag (the `-setCPlusPlusDialect neutral` option for the linker).

If you are migrating NonStop Server for Java applications based on JDK 1.3.x or earlier, you might need to change your source code. Whether your native code needs source-code changes depends on whether the code uses C++ features that have changed in version 3. To identify required source-code changes, run a migration check on your source code on TNS/R by invoking the version 2 compiler and using the pragma `MIGRATION_CHECK`. Running this migration check causes the compiler to issue a warning when a class or member function is present that has changed or become obsolete for version 3. See the *C/C++ Programmers Guide* for more information about this pragma and the warnings it can produce.

Note that the `VERSION3` directive specifies the use of the Standard C++ Library ISO/IEC version 3 and the C++ Standard headers. `VERSION3` enforces the ISO/IEC IS 14882:1998 standard for C++. The ISO C++ standard is identical to the ANSI C++ standard.

For invocation API users, you build your own executable and link that executable against the JVM DLL. For a demo, see the invocation API demo provided by NonStop Server for Java 6.0 in `install_dir/demo/invocation_api`.

For more information, see "Linker and Compiler Options" (page 45).

Floating-Point Support

By default, NonStop Server for Java 3.1.x and earlier versions converted any floating-point value that crossed the Java Native Interface (JNI) boundary to a TNS float. This default could be overridden by supplying a line in the file `TandemVMClassFP.properties`. If a particular class needed IEEE floating-point values passed to its JNI code instead of TNS float values; users added a property (with the name of the class being the name of the property) to this file. Users also set the value of the property to `IEEE_FP` to indicate that they wanted IEEE floating-point values passed to their JNI code or `TANDEM_FP` to indicate that they wanted TNS floating-point values passed to their JNI code.

A user program cannot specify the floating-point type by using the `TandemVMClassFP.properties` file. Thus, any user-program or Java based product with JNI code that obtains floating-point values from Java must call the `NSK_FLOAT_*` Guardian routines to convert these values to TNS floats. Likewise, any float value passed to Java must be an IEEE float value. Table 11 (page 83) illustrates the NonStop Server for Java 6.0 application's floating-point usage compared to earlier versions.

Table 11 Summary of Floating Point Support

	NSJ 2.x	NSJ 3.x	NSJ 4.x	NSJ 5.x	NSJ 6.0
Java floating-point usage	IEEE float	IEEE float	IEEE float	IEEE float	IEEE float
JNI code floating point	Either IEEE or Tandem float	Either IEEE or Tandem float	IEEE float	IEEE float	IEEE float
JNI calling convention	Tandem float	Either IEEE or Tandem float	IEEE float	IEEE float	IEEE float
Java compiler flag	Tandem float	IEEE float	IEEE float	IEEE float	IEEE float
Java linker flag	Tandem float	Tandem float	IEEE float	IEEE float	IEEE float

Since NonStop Server for Java 3.1.x and earlier set the linker flag for the process to TNS float, any use of the C runtime library used routines that handled TNS floats. For NonStop Server for Java 4 , 5, 5.1, and 6.0 versions, the linker flags described under "Linker and Compiler Options" (page 45) are used to specify IEEE floating point. Accordingly, the C runtime library uses routines that handle IEEE floating point.

For NonStop Server for Java 4 , 5, 5.1, and 6.0 versions, any C runtime library calls such as `sprintf` or `sscanf`, made from JNI code, assumes IEEE float values and calling conventions. For example, assume that JNI code, written for a previous version of Java, converts a TNS floating-point value to a string, which is then passed to Java. To migrate the program, you must change the JNI code to convert the TNS floating-point value to an IEEE floating-point value and then call `sprintf` to convert the floating-point value to a string.

For more information, see "IEEE Floating-Point Implementation" (page 46).

Using AWT Classes

If your Java programs use AWT classes with NonStop Server for Java 3.1.x or earlier versions, change your program code to catch a `HeadlessException` rather than an `UnsupportedClassException`.

Because the NonStop operating system does not provide support for windowing operations, previous versions of NonStop Server for Java supported only those Abstract Windowing Toolkit (AWT) classes and methods that did not require a display, keyboard, sound, or mouse operation. Any class or method that required such an operation threw an `UnsupportedClassException`.

NonStop Server for Java 6.0 supports the Sun Microsystems enhancement to AWT called "headless support" that allows a JVM to indicate whether a display, keyboard, sound, or mouse operation can be supported in a graphics environment.

Sun implemented headless support by supplying two new methods in the `GraphicsEnvironment` class: is `Headless` and is `HeadlessInstance`. In addition, Sun created a new exception `java.awt.HeadlessException`. `HeadlessException` will be thrown by any class or method that requires a display, keyboard, sound, or mouse operation if such a class or method is invoked when `GraphicsEnvironment.isHeadless` returns true. Classes and methods that support printing, fonts, and imaging are fully supported in a headless JVM and are fully supported by NonStop Server for Java 4, 5, 5.1, and 6.0.

For further information, see "Headless Support" (page 42) in the HP Implementation specifics section.

POSIX Threads

NonStop Server for Java 3.1.x and earlier versions used OSS POSIX Threads (product number T5819) that conformed to an earlier standard (Draft 4) for POSIX threads. NonStop Server for Java 4, 5, 5.1, and 6.0 use Standard POSIX Threads (product number T1248), which conforms to IEEE POSIX Standard 1003.lc.

The POSIX threads calls in T1248 have changed to conform to the standard; therefore, any native code for NonStop Server for Java 3.1.x applications that makes POSIX threads calls might have to change to use the standard pthread routine. For more information, see Appendix D in the *Open System Services Porting Guide*, which contains a list of differences between the POSIX thread routines in T5819 and the routines in T1248.

Additionally, you must change any JNI code that made calls to routines beginning with `cma…` to use the Wrapper or Development Toolkit routines (`spt…`.) supplied with T1248.

NOTE: Any user-developed code that makes such POSIX threads calls must change.

Directories of Binary Files Moved

If your NonStop Server for Java programs have references to `bin/oss/posix_threads` in Pathway configuration files or elsewhere, you must change them to use the NonStop Server for Java 6.0 installation `bin` directory.

In NonStop Server for Java 3.1.x or earlier versions, the `bin` and `jre/bin` directories contained a shell script that ran the real executable that was located in `bin/oss/posix_threads`. In the NonStop Server for Java 6.0 version, the bin directory contains the real executable, no shell script wrapper, and no `bin/oss/posix_threads` directory. The `jre/bin` directory contains the executables in the `bin` directory.

Character Handling

With NonStop Server for Java 6.0, character handling is based on version 4.0 of the Unicode standard. This new basis 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 Java SE 6.0. Support for supplementary characters has been specified by the Java Specification Request (JSR) 204 expert group and implemented throughout the Java SE 6.0. See the article Supplementary Characters in the Java Platform and the `Character` class documentation for more information.

BigDecimalFormat Class

In JDK 6.0, the `DecimalFormat` class was enhanced to format and parse `BigDecimal` and `BigInteger` values without loss of precision. The formatting of these values is enhanced

automatically. To enable parsing into `BigDecimal`, you need to use the `setParseBigDecimal` method.

JAAS Enhancement

In NonStop Server for Java 3.x and earlier versions, the Java Authentication and Authorization Service (JAAS) was an optional package (extension). JAAS is integrated into the NonStop Server for Java 6.0. JAAS augments the core Java 2 platform with APIs that allow authenticating users and enforcing access controls upon users. Traditionally, Java 2 provided code-source-based access controls (access controls based on where the code originated and who signed the code). However, Java 2 lacked the ability to additionally enforce access controls based on who runs the code. In NonStop Server for Java 6.0, JAAS provides a framework that augments the Java 2 security architecture with this additional capability.

For more information on JAAS, see "Java Authentication and Authorization Service (JAAS)" (page 53).

Miscellaneous Changes for Migration to TNS/E

JNI_OnLoad and JNI_OnUnLoad Functions

All applications migrating from TNS/R systems must change the `JNI_OnLoad` function. The format you use depends on the system type:

- On TNS/R systems, you use the following name:

  `JNI_OnLoad_`*`libname`*

  where *`libname`* is the name of the library that your program passed to the `System.loadLibrary` function.

- On TNS/E systems, use:

  `JNI_OnLoad`

  On TNS/R systems, the `JNI_OnUnload` function is not supported by NonStop Server for Java 6.0. On TNS/E systems, the `JNI_OnUnload` function is supported.

Debugger

Visual Inspect is the preferred debugger for applications on TNS/E. For debugging native code, you can also use Native Inspect (`$System.SYSnnn.EINSPECT` command).

Default Heap and Stack Sizes

The default heap and stack sizes have changed for TNS/E hosted applications.

- On TNS/E systems, the default stack size is 512 KB; the minimum stack size is 64 KB.
- On TNS/E systems, the initial heap size is 18.25 MB; the default maximum heap size is 64 MB.

dlfcn.h File

All applications migrating from TNS/R that use the dlfcn.h file require code changes. On TNS/E the NonStop Server for Java 4 , 5, 5.1, and 6.0 do not use their own special version of dlfcn.h. Use the file that exists in the include directory (/usr/include) of the system.

A Supported and Unsupported Features of NonStop Server for Java 6.0

NonStop Server for Java 6.0 includes all the features of NonStop Server for Java 5.1 and it is based on Java SE 6.0.

For information about Java SE 6.0 new features, see New Features and Enhancements Java SE 6.0

(http://docs.oracle.com/javase/1.5.0/docs/relnotes/features.html).

Java SE 6.0 Features not Implemented in NonStop Server for Java 6.0

Java SE 6.0 features that do not apply to a server-side HotSpot VM are not implemented in NonStop Server for Java 6.0.

For information about Java SE 6.0 features that are not implemented, see the following implementation-specific topics:

- "Headless Support" (page 42)
- "Java Standard Edition Development Kit (JDK) Highlights" (page 23)
- "Parallel Collector" (page 61) (including garbage collector ergonomics)
- UseBiasedLocking (JVM internal locking optimization, useful for SMP architectures)
- `–XX:+TieredCompilation` option is not supported

B Addendum to HPjmeter 4.2 User's Guide

This appendix provides instructions for using the HPjmeter tool on NonStop system. It is based on the *HP-UX HPjmeter 4.2 User's Guide*, available at the following web address: www.hp.com/go/hpjmeter/.

The following sections correspond one-to-one to those in the HP-UX HPjmeter 4.2 User's Guide.

NOTE: The information provided here must be used as additional or substitute to the information provided in the corresponding sections of the *HP-UX HPjmeter 4.2 User's Guide*.

Completing Installation of HPjmeter

Agent Requirements

Table 12 (page 88) lists the agent requirements.

Table 12 Agent Requirements

Operating system and architecture	• NonStop Operating System H06.15 and later release version updates (RVUs) on NonStop Integrity servers. • NonStop Operating System J06.04 and later RVUs on HP Integrity NonStop BladeSystem • NonStop Server for Java 6.0 and later software product revisions (SPRs)

NOTE:

1. The HPjconfig tool is not supported on a NonStop operating system.
2. Zero-preparation profiling is supported on NSJ 6.0 and later versions only.

File Locations

Starting with HPjmeter 4.2, you can find the directory structure of HPjmeter available at the location `/usr/tandem/hpjmeter` in the following way:

Directory	Description
`./bin`	Consists of top level nodeagent binary, `spt` version of nodeagent at `./spt/nodeagent` and `put` version of nodeagent at `./put/nodeagent`.
`./lib`	Consists of JVM agent library to support NSJ6 versions (`./oss`), 32–bit NSJ7 (`./oss32`), and 64–bit NSJ7 (`./oss64`).
`./demo`	Consists of demo directory.
`./doc`	Consists of HPjmeter document.
`./var`	Consists of `./log` and `./fifos` folder.

The default installation path on the NonStop operating system is `/usr/tandem/hpjmeter`.

Configuring your Application to Use HPjmeter Command Line Options

Preparing to run Java

Complete the following steps to prepare the Java application to run with the JVM agent:

- On NSJ6 and later SPRs, you must set the _RLD_LIB_PATH as follows:

  `$JMETER_HOME/lib/oss`

  where,

  `JMETER_HOME` is set to `/usr/tandem/hpjmeter`.

Example 4 Using `-agentlib` to run the JVM agent

```
$ java -Xms256m -Xmx512m -agentlib:jmeter myapp
```

Example 5 Setting `-Xbootclasspath`

```
$ java -agentlib:jmeter -Xbootclasspath/a:$JMETER_HOME/lib/agent.jar
```

NOTE: `-Xrunjmeter` is not supported on NSJ 6.0.

Attaching to the JVM Agent of a Running Application

The dynamic attach feature is not supported on NSJ6.

Monitoring Applications

Managing Node Agents

Managing Node Agents on a NonStop Operating System

In a NonStop Operating System, the HPjmeter installation process does not automatically start the node agent as a daemon process. You must manually start the node agent as a daemon process. The top level binary nodeagent available at `/usr/tandem/hpjmeter/bin` starts the `spt` or `put` version of nodeagent based on the underlying SUT version.

Diagnosing Errors when Monitoring Running Applications

Checking for Application Paging Problems

HPGlancePlus tool is not available in NonStop. Therefore, references of HPGlancePlus available in *HPjmeter 4.2 User Guide* are not applicable for NonStop.

Profiling Applications

For a complete list of profiling options and instructions on how to use profile options, see "Application Profiling" (page 76). This section lists the options that are not supported on Nonstop and describes the difference in behavior of some options.

Collecting Profile Data

Profiling with —Xeprof

Table 13 Supported –Xeprof options

`time_on`	On NonStop operating systems, the time interval between the start and stop signals must be more than 5 minutes.
`time_slice`	This is because the Java thread switching process is slow due to the non-preemptive nature of the NonStop operating system. As a result, the asynchronous signals might be lost if they are posted to the NonStop OS Java process at very short intervals.
`times=quick|thorough`	This option is not supported on NonStop operating systems.

Profiling with Zero Preparation

This feature is supported on NSJ 6.0 and later versions only.

Profiling with –agentlib:hprof

Table 14 (page 90) lists the supported `–agentlib:hprof` options.

Table 14 Supported —agentlib:hprof options

`cpu=samples|times|old`	`cpu=samples` is not supported on NonStop operating systems.

Troubleshooting

Identifying Version Numbers

To identify the version number for HPjmeter components, run the following command:

- JVM agent for Integrity NonStop:

  ```
  $ vproc $JMETER_HOME/lib/oss/libjmeter.so
  ```

Installation

On NonStop operating systems, only the agents are installed by default at the following location:

`/usr/tandem/hpjmeter`

The scripts found in `/usr/tandem/hpjmeter/bin`, use the standard `/usr/tandem/java` installation as the reference JDK installation. There is no special installer for NonStop operating systems that modifies the scripts, as is done by the HP-UX installer.

Node Agent

FIFOs are used for communication between JVM and node agents. On NonStop operating systems, FIFOs are located in the following directory:

`/usr/tandem/hpjmeter/var/fifos`

From HPjmeter 4.2, the node agent at location `$JMETER_HOME/bin` cannot be started as a named process. If the node agent needs to start as a named process, following requirements must be considered:

- If the SUT version is lower than H06.26/J06.15, start the node agent at location `$JMETER_HOME/bin/spt` as a named process.

  For example, `run -name=/G/nod1 /usr/tandem/hpjmeter/bin/spt/nodeagent`

- If the SUT version is H06.26/J06.15 or higher, start the node agent at location `$JMETER_HOME/bin/put` as a named process.

  For example, `run -name=/G/nod1 /usr/tandem/hpjmeter/bin/put/nodeagent`

Quick References

Determining which HPjmeter Features are Available with a Specific JVM Version

For HPjmeter features available in JVM version see, “Table A-2 HPjmeter Features Available by JVM Version” in *HPjmeter 4.2 User's Guide*.

NOTE:

- HPjmeter features are supported on NSJ 6.0 and later releases.
- All instances of PROF must be read as ALRM.
- All instances of signal 21 must be read as 14.

Glossary

A

abstract class In Java, a class designed only as a parent from which subclasses can be derived, which is not itself suitable for instantiation. An abstract class is often used to "abstract out" incomplete sets of features, which can then be shared by a group of sibling subclasses that add different variations of the missing pieces.

Abstract Window Toolkit (AWT) The package that implements graphical user interfaces for Java. For more information, see the Sun Microsystems AWT Home Page (http://java.sun.com/javase/6/docs/technotes/guides/awt/index.html).

American National Standards Institute (ANSI) The United States government body responsible for approving US standards in many areas, including computers and communications. ANSI is a member of ISO . ANSI sells ANSI and ISO (international) standards.

American Standard Code for Information Interchange (ASCII) The predominant character set encoding of present-day computers. ASCII uses 7 bits for each character. It does not include accented letters or any other letter forms not used in English (such as the German sharp-S or the Norwegian ae-ligature). Compare to Unicode.

ANSI See American National Standards Institute (ANSI).

API See application program interface (API).

application program
1. A software program written for or by a user for a specific purpose.
2. A computer program that performs a data processing function rather than a control function.

application program interface (API) A set of functions or procedures that are called by an application program to communicate with other software components.

ASCII See American Standard Code for Information Interchange (ASCII).

AWT See Abstract Window Toolkit (AWT).

B

BDK See JavaBeans Development Kit (BDK).

branded A Java virtual machine that Sun Microsystems, Inc. has certified as conformant.

browser A program that allows you to read hypertext . The browser gives some means of viewing the contents of nodes and of navigating from one node to another. Internet Explorer, Netscape Navigator, NCSA Mosaic, Lynx, and W3 are examples for browsers for the WWW . They act as clients to remote servers.

bytecode The code that javac, the Java compiler, produces.

C

C language A widely used, general-purpose programming language developed by Dennis Ritchie of Bell Labs in the late 1960s. C is the primary language used to develop programs in UNIX environments.

C++ language A derivative of the C language that has been adapted for use in developing object-oriented programs.

CGI See Common Gateway Interface (CGI).

class path The directories where a Java virtual machine and other Java programs that are located in the `/usr/tandem/java/bin` directory search for class libraries (such as classes.zip). You can set the class path explicitly or with the `CLASSPATH` environment variable.

client A software process, hardware device, or combination of the two that requests services from a server . Often, the client is a process residing on a programmable workstation and is the part of a program that provides the user interface . The workstation client might also perform other portions of the program logic. Also called a requester.

command The operation demanded by an operator or program; a demand for action by, or information from, a subsystem. A command is typically conveyed as an interprocess message from a program to a subsystem.

Common Gateway Interface (CGI) The World Wide Web standard interface for servers , often written in C . The NonStop Server for Java 6.0 supports CGI-like use of Java using servlets with iTP Secure WebServer . See also Pathway CGI.

Common Object Request Broker Architecture (CORBA) The OMG standard that allows objects that adhere to it to interact over a network regardless of the types of machines or operating systems on which they reside. Java interoperates with this standard using Java IDL and JTS .

Compiler (C3) A JVM Hotspot Compiler back-end for application performance.

concurrency A condition in which two or more transactions act on the same record in a database at the same time. To process a transaction, a program must assume that its input from the database is consistent, regardless of any concurrent changes being made to the database. TMF manages concurrent transactions through concurrency control .

concurrency control Protection of a database record from concurrent access by more than one process. TMF imposes this control by dynamically locking and unlocking affected records to ensure that only one transaction at a time accesses those records.

conformant A Java implementation is conformant if it passes all the tests for Java SE 6.

CORBA See Common Object Request Broker Architecture (CORBA).

Core Packages The required set of APIs in a Java platform edition which must be supported in any and all compatible implementations.

D

Data Control Language (DCL) The set of data control statements within the SQL language.

Data Definition Language (DDL) A language that defines all attributes and properties of a database, especially record layouts, field definitions, key fields, field locations, file locations, and storage strategy.

DCL See Data Control Language (DCL).

DDL See Data Definition Language (DDL).

debuggee A process being debugged. The process consists of the application being debugged and the Java VM running the application.

debugger A program designed to use the Java Debugging Interface (JDI) and connect to the (debuggee) so that a programmer can step through the debuggee process, examine the data, and monitor conditions such as the values of variables.

deserialization, object The reverse of Object Serialization.

digital signature A way of automatically identifying the sender of an electronic message or document, without the possibility of alteration.

DNS See Domain Name Server (DNS).

Domain Name Server (DNS) An HP product, part of TCP/IP , that provides the facilities for the maintenance and automated distribution of network resource name information. DNS permits decentralized administration of resource names and specifies redundancy of servers to provide a reliable query service for users.

driver A class in JDBC that implements a connection to a particular database management system such as NonStop SQL/MX . The NonStop Server for Java 6.0 has these driver implementations: JDBC Driver for SQL/MP(JDBC/MP) and JDBC Driver for SQL/MX (JDBC/MX) .

E-F

Enscribe HP database management software that provides a record-at-a-time interface between servers and a database. As an integral part of the operating system distributed across two or more processors, Enscribe helps ensure data integrity if a processor module, I/O channel, or disk drive

fails. Files on a NonStop system can be Enscribe files, SQL/MP tables, or SQL/MX tables. Enscribe files can be either structured or unstructured.

exception An event during program execution that prevents the program from continuing normally; generally, an error. Java methods raise exceptions using the throw keyword and handle exceptions using try, catch, and finally blocks.

Expand The HP NonStop operating system network that extends the concept of fault tolerance to networks of geographically distributed NonStop systems. If the network is properly designed, communication paths are constantly available even if there is a single line failure or component failure.

expandability See scalability.

fault tolerance The ability of a computer system to continue processing during and after a single fault (the failure of a system component) without the loss of data or function.

G

garbage collection The process that reclaims dynamically allocated storage during program execution. The term usually refers to automatic periodic storage reclamation by the garbage collector (part of the runtime system), as opposed to explicit code to free specific blocks of memory.

graphical user interface (GUI) Software that provides user control using a graphic display format. GUI software provides a bit-mapped, icon-oriented, windowing, graphical environment.

Guardian An environment available for interactive and programmatic use with the NonStop operating system . Processes that run in the Guardian environment use the Guardian system procedure calls as their API . Interactive users of the Guardian environment use the TACL or another HP product's command interpreter. Compare to OSS.

GUI See graphical user interface (GUI).

H

Hotspot virtual machine See Java Hotspot virtual machine.

HP JDBC Driver for SQL/MP (JDBC/MP) The product that provides access to SQL/MP and conforms to the JDBC API.

HP JDBC Driver for SQL/MX (JDBC/MX) The product that provides access to SQL/MX and conforms to the JDBC API.

HP NonStop operating system The operating system for NonStop systems.

HP NonStop Server for Java Transaction API An implementation of Java Transaction API (JTA) . One version of the NonStop Server for Java Transaction API uses JTS and another uses TMF.

HP NonStop Server for Java, based on Java Standard Edition 6.0 The formal name of the NonStop Server for Java product whose Java HotSpot virtual machine conforms to the Java SE 6.0. See also NonStop Server for Java 6.0.

HP NonStop SQL/MP (SQL/MP) HP NonStop Structured Query Language/MP, the HP relational database management system for NonStop servers.

HP NonStop SQL/MX (SQL/MX) HP NonStop Structured Query Language/MX, the HP next-generation relational database management system for business-critical applications on NonStop servers.

HP NonStop system HP computers (hardware and software) that support the NonStop operating system.

HP NonStop Technical Library The browser-based interface to NonStop computing technical information.

HP NonStop Transaction Management Facility (TMF)
An HP product that provides transaction protection, database consistency, and database recovery. SQL statements issued through a JDBC driver against a NonStop SQL database call procedures in the TMF subsystem.

HP NonStop TS/MP (TS/MP)
An HP product that supports the creation of Pathway servers to access NonStop SQL/MP or Enscribe databases in an online transaction processing (OLTP) environment.

HP Tandem Advanced Command Language (TACL)
The command interpreter for the operating system, which also functions as a programming language, allowing users to define aliases, macros, and function keys.

HTML
See Hypertext Markup Language (HTML).

HTTP
See Hypertext Transfer Protocol (HTTP).

hyperlink
A reference (link) from a point in one hypertext document to a point in another document or another point in the same document. A browser usually displays a hyperlink in a different color, font, or style. When the user activates the link (usually by clicking on it with the mouse), the browser displays the target of the link.

hypertext
A collection of documents (nodes) containing cross-references or links that, with the aid of an interactive browser, allow a reader to move easily from one document to another.

Hypertext Mark-up Language (HTML)
A hypertext document format used on the World Wide Web.

Hypertext Transfer Protocol (HTTP)
The client - server TCP/IP protocol used on the World Wide Web for the exchange of HTML documents.

I

IEC
See International Electrotechnical Commission (IEC).

IEEE
Institute for Electrical and Electronic Engineers (IEEE).

inlining
Replacing a method call with the code for the called method, eliminating the call.

interactive
Question-and-answer exchange between a user and a computer system.

interface
In general, the point of communication or interconnection between one person, program, or device and another, or a set of rules for that interaction. See also API .

International Electrotechnical Commission (IEC)
A standardization body at the same level as ISO.

International Organization for Standardization (ISO)
A voluntary, nontreaty organization founded in 1946, responsible for creating international standards in many areas, including computers and communications. Its members are the national standards organizations of 89 countries, including ANSI.

Internet
The network of many thousands of interconnected networks that use the TCP/IP networking communications protocol . It provides e-mail, file transfer, news, remote login, and access to thousands of databases. The Internet includes three kinds of networks:

- High-speed backbone networks such as NSFNET and MILNET
- Mid-level networks such as corporate and university networks
- Stub networks such as individual LANs

Internet Protocol version 6 (IPv6)
IP specifies the format of packets and the addressing scheme. The current version of IP is IPv4. IPv6 is a new version of IP designed to allow the Internet to grow steadily, both in terms of number of hosts connected and the total amount of data traffic transmitted. (IP is pronounced eye-pea)

interoperability
1. The ability to communicate, execute programs, or transfer data between dissimilar environments, including among systems from multiple vendors or with multiple versions of

operating systems from the same vendor. HP documents often use the term connectivity in this context, while other vendors use *connectivity* to mean hardware compatibility.

2. Within a NonStop system node , the ability to use the features or facilities of one environment from another. For example, the `gtacl` command in the OSS environment allows an interactive user to start and use a Guardian tool in the Guardian environment.

interpreter The component of a Java virtual machine that interprets bytecode into native machine code.

Invocation API The C-language API that starts a Java virtual machine and invokes methods on it. The Invocation API is a subset of the JNI .

IPv6 See Internet Protocol version 6 (IPv6).

ISO See International Organization for Standardization (ISO).

iTP Secure WebServer The HP web server with which the NonStop Server for Java integrates using servlets.

J

`jar` The Java Archive tool, which combines multiple files into a single Java Archive (JAR) file. Also, the command to run the Java Archive Tool.

JAR file A Java Archive file, produced by the Java Archive Tool, jar.

`java` The Java application launcher, which launches an application by starting a Java runtime environment, loading a specified class, and invoking that class's `main` method.

Java 2 Platform, Enterprise Edition (J2EE platform) An environment for developing and deploying enterprise applications. The J2EE platform consists of a set of services, application programming interfaces (APIs) and protocols that provide the functionality for developing multi-tiered, Web-based applications.

Java Conformance Kit (JCK) The collection of conformance tests that any vendor's JDK must pass in order to be conformant with the Sun Microsystems specification.

Java Database Connectivity (JDBC) An industry standard for database-independent connectivity between the Java platform and relational databases such as NonStop SQL/MP or NonStop SQL/MX . JDBC provides a call-level API for SQL-based database access.

Java HotSpot virtual machine The Java virtual machine implementation designed to produce maximum program-execution speed for applications running in a server environment. This virtual machine features an adaptive compiler that dynamically optimizes the performance of running applications.

Java IDL See Java Interface Development Language (Java IDL)

Java Interface Development Language (Java IDL) The library that supports CORBA and Java interoperability . For more information, see the Sun Microsystems Java IDL documentation (http://java.sun.com/products/jdk/idl/index.html).

Java Naming and Directory Interface (JNDI) A standard extension to the Java platform, which provides Java technology-enabled application programs with a unified interface to multiple naming and directory services.

Java Native Interface (JNI) The C-language interface used by C functions called by Java classes. Includes an Invocation API that invokes a Java virtual machine from a C program.

Java Platform Standard Edition (Java SE 6) The core Java technology platform, which provides a complete environment for applications development on desktops and servers and for deployment in embedded environments. For more information, see the Sun Microsystems JDK 6.0 Documentation .

Java runtime See Java SE Runtime Environment.

Java SE Development Kit (JDK) The development kit delivered with the Java SE platform. Contrast with Java SE Runtime Environment (JRE). See also, Java Platform Standard Edition 6.0 (Java SE)

Java SE Runtime Environment (JRE) The Java virtual machine and the Core Packages. This is the standard Java environment that the java command invokes. Contrast with Java SE Development Kit (JDK). See also, Java Platform Standard Edition 6.0 (Java SE).

Java Transaction API (JTA) The Sun Microsystems product that specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the

application server, and the transactional applications. For more information, see the Sun Microsystems JTA document (http://java.sun.com/products/jta/index.html).

Java Transaction Service (JTS) The transaction API , modeled on OMG's OTS . The NonStop Server for Java 6.0 includes an implementation of the jts.Current interface .

Java virtual machine (JVM) The process that loads, links, verifies, and interprets Java bytecode . The NonStop Server for Java 6.0 implements the Java HotSpot Server virtual machine .

Java Virtual Machine Tool Interface (JVM TI) A programming interface used by development and monitoring tools. It is used to inspect the state and to control the execution of applications running in the Java VM, thereby defining the debugging services a VM provides.

JavaBeans A platform-neutral component architecture supported by Java, intended for developing or assembling software for multiple hardware and operating system environments. For more information, see the Sun Microsystems JavaBeans document (http://java.sun.com/javase/6/docs/technotes/guides/beans/index.html).

JavaBeans Development Kit (BDK) A set of tools for creating JavaBeans that is included with the NonStop Server for Java 6.0.

`javac` The Java compiler, which compiles Java source code into bytecode. Also, the command to run the Java compiler.

`javachk` The Java Checker, which determines whether a problem with the Java virtual machine is due to an incorrect TCP/IP configuration. Also, the command to run the Java checker.

`javadoc` The Java API documentation generator, which generates API documentation in HTML format from Java source code. Also, the command to run the Java API documentation generator.

`javah` The C header and Stub file generator, which generates C header files and C source files from a Java class, providing the connections that allow Java and C code to interact. Also, the command to run the C header and stub file generator.

`javap` The Java class file disassembler, which disassembles compiled Java files and prints a representation of the Java bytecode . Also, the command to run the Java class file disassembler.

JCK See Java Conformance Kit (JCK).

`jdb` The Java Debugger, which helps you find and fix errors in Java programs. Also, the command to run the Java Debugger . `jdb` uses the Java Debugger API.

JDBC See Java Database Connectivity (JDBC).

JDBC/MP See HP JDBC Driver for SQL/MP (JDBC/MP).

JDBC/MX See HP JDBC Driver for SQL/MX (JDBC/MX).

JDK See Java SE Development Kit (JDK).

JNDI See Java Naming and Directory Interface (JNDI).

JNI See Java Native Interface (JNI).

`jre` The Java runtime environment, which executes Java bytecode. See also Java SE Runtime Environment (JRE).

JTA See Java Transaction API (JTA).

JTS See Java Transaction Service (JTS).

`jts.Current` A JTS interface that lets you define transaction boundaries. The NonStop Server for Java 6.0 includes an implementation of `jts.Current`.

JVM See Java virtual machine (JVM).

JVM TI See Java Virtual Machine Tool Interface (JVM TI).

K-M

key

1. A value used to identify a record in a database, derived by applying a fixed function to the record. The key is often simply one of the fields (a column if the database is considered as

a table with records being rows). Alternatively, the key can be obtained by applying a function to one or more of the fields.

2. A value that must be fed into the algorithm used to decode an encrypted message in order to reproduce the original plain text. Some encryption schemes use the same (secret) key to encrypt and decrypt a message, but public key encryption uses a private (secret) key and a public key that is known by all parties.

LAN See local area network (LAN).

local area network (LAN) A data communications network that is geographically limited (typically to a radius of 1 kilometer), allowing easy interconnection of terminals, microprocessors, and computers within adjacent buildings. Ethernet is an example of a LAN.

macro A sequence of commands that can contain dummy arguments. When the macro runs, actual parameters are substituted for the dummy arguments.

N

native In the context of Java programming, something written in a language other than Java (such as C or C++) for a specific platform.

native method A non-Java routine (written in a language such as C or C++) that is called by a Java class.

`native2ascii` The Native-to-ASCII converter, which converts a file with native-encoded characters into one with Unicode-encoded characters. Also, the command to run the Native-to-ASCII converter.

node

1. An addressable device attached to a computer network.
2. A hypertext document.

NonStop Server for Java 6.0 The informal name for the NonStop Server for Java products based on the Java Platform Standard Edition 6.0 product. This product is a Java environment that supports compact, concurrent, dynamic, and portable programs for the enterprise server. See also, HP NonStop Server for Java, based on the Java Platform Standard Edition 6.0 .

NonStop Technical Library The browser-based interface to NonStop computing technical information. NonStop Technical Library replaces HP Total Information Manager (TIM).

NSK See HP NonStop operating system.

NSKCOM A program management tool for swap space.

O

Object Management Group (OMG) The standards body that defined CORBA.

Object Serialization A Sun Microsystems procedure that extends the core Java Input/Output classes with support for objects by supporting the following:

- The encoding of objects, and the objects reachable from them, into a stream of bytes.
- The complementary reconstruction of the object graph from the stream.

Object Serialization is used for lightweight persistence and for communication by means of sockets or RMI. The default encoding of objects protects private and transient data, and supports the evolution of the classes. A class can implement its own external encoding and is then solely responsible for the external format.

Object Transaction Service (OTS) The transaction service standard adopted by the OMG and used as the model for JTS.

ODBC See Open Database Connectivity (ODBC).

OLTP See online transaction processing (OLTP).

OMG See Object Management Group (OMG).

online transaction processing (OLTP) A method of processing transactions in which entered transactions are immediately applied to the database. The information in the databases is always current with the state of company and is readily available to all users through online screens and printed reports. The transactions are

processed while the requester waits, as opposed to queued or batched transactions, which are processed at a later time. Online transaction processing can be used for many different kinds of business tasks, such as order processing, inventory control, accounting functions, and banking operations.

Open Database Connectivity (ODBC) The standard Microsoft product for accessing databases.

Open System Services (OSS) An environment available for interactive and programmatic use with the NonStop operating system . Processes that run in the OSS environment use the OSS API . Interactive users of the OSS environment use the OSS shell for their command interpreter . Compare to Guardian.

OSS See Open System Services (OSS).

OTS See Object Transaction Service (OTS).

P

package A collection of related classes; for example, JDBC.

Pathsend API The application program interface to a Pathway system that enables a Pathsend process to communicate with a server process.

Pathsend process A client (requester) process that uses the Pathsend interface to communicate with a server process. A Pathsend process can be either a standard requester, which initiates application requests, or a nested server, which is configured as a server class but acts as a requester by making requests to other servers. Also called a Pathsend requester.

Pathway A group of software tools for developing and monitoring OLTP programs that use the client / server model. Servers are grouped into server classes to perform the requested processing. On NonStop systems, this group of tools is packaged as two separate products: TS/MP and Pathway/TS .

Pathway CGI An extension to iTP Secure WebServer that provides CGI -like access to Pathway server classes. Extended in the NonStop Server for Java so that Java servlets can be invoked from a `ServletServerClass`, a special Pathway CGI server.

Pathway/TS An HP product that provides tools for developing and interpreting screen programs to support OLTP programs in the Guardian environment on NonStop servers . Pathway/TS screen programs communicate with terminals and intelligent devices. Pathway/TS requires the services of the TS/MP product.

persistence

1. A property of a programming language where created objects and variables continue to exist and retain their values between runs of the program.
2. The capability of continuing in existence, such as a program running as a process.

portability The ability to transfer programs from one platform to another without reprogramming. A characteristic of open systems. Portability implies use of standard programming languages such as C .

Portable Operating System Interface X (POSIX) A family of interrelated interface standards defined by ANSI and IEEE . Each POSIX interface is separately defined in a numbered ANSI/IEEE standard or draft standard. The standards deal with issues of portability, interoperability, and uniformity of user interfaces.

POSIX See Portable Operating System Interface X (POSIX).

private key An encryption key that is not known to all parties.

protocol A set of formal rules for transmitting data, especially across a network. Low-level protocols define electrical and physical standards, bit-ordering, byte-ordering, and the transmission, error detection, and error correction of the bit stream. High-level protocols define data formatting, including the syntax of messages, the terminal-to-computer dialogue, character sets, sequencing of messages, and so on.

Pthread A POSIX thread.

public key An encryption key that is known to all parties.

pure Java Java that relies only on the Core Packages, meaning that it can run anywhere.

R

RDF See Remote Duplicate Database Facility (RDF).

Remote Duplicate Database Facility (RDF) The HP software product that does the following:

- Assists in disaster recovery for OLTP production databases.
- Monitors database updates audited by the TMF subsystem on a primary system and applies those updates o a copy of the database on a remote system.

Remote Method Invocation (RMI) The Java package used for homogeneous distributed objects in an all-Java environment.

requester See client.

RMI See Remote Method Invocation (RMI).

`rmic` The Java RMI stub compiler, which generates stubs and skeletons for remote objects.

`rmicregistry` The Java Remote Object Registry, which starts a remote object registry on the specified port on the current host.

S

scalability The ability to increase the size and processing power of an online transaction processing system by adding processors and devices to a system, systems to a network, and so on, and to do so easily and transparently without bringing systems down. Sometimes called expandability.

Scalable TCP/IP (SIP) A NonStop Server for Java feature that transparently provides a way to give scalability and persistence to a network server written in Java.

serialization See Object Serialization.

serialized object An object that has undergone object serialization.

`serialver` The Serial Version Command, which returns the `serialVersionUID` of one or more classes. Also, the command to run the Serial Version Command.

server

1. An implementation of a system used as a stand-alone system or as a node in an Expand network.
2. The hardware component of a computer system designed to provide services in response to requests received from clients across a network. For example, NonStop system servers provide transaction processing, database access, and other services.
3. A process or program that provides services to a client. Servers are designed to receive request messages from clients; perform the desired operations, such as database inquiries or updates, security verifications, numerical calculations, or data routing to other computer systems; and return reply messages to the clients.

servlet A server -side Java program that any World Wide Web browser can access. It inherits scalability and persistence from the Pathway CGI server that manages it.

The Java class named servlets executes in server environments such as World Wide Web servers. The Servlet API is defined in a draft standard by Sun Microsystems. The `servlets` class is not in the Core Packages for the JDK.

shell The command interpreter used to pass commands to an operating system; the part of the operating system that is an interface to the outside world.

SIP See Scalable TCP/IP (SIP).

skeleton In RMI , the complement of the stub . Together, skeletons and stubs form the interface between the RMI services and the code that calls and implements remote objects.

socket A logical connection between two application programs across a TCP/IP network.

SQL/MP See HP NonStop SQL/MP.

SQL/MX See HP NonStop SQL/MX.

SQLJ Also referred to SQLJ Part 0 the "Database Language SQL—Part 10: Object Language Bindings (SQL/OLB) part of the ANSI SQL-2002 standard that allows static SQL statements to be embedded directly in a Java program.

Standard Extension API An API outside the Core Packages for which Sun Microsystems, Inc. has defined and published an API standard. Some of the Standard Extensions might migrate into the Core Packages. Examples of standard extensions are servlets and JTS .

stored procedure A procedure registered with NonStop SQL/MX and invoked by NonStop SQL/MX during execution of a CALL statement. Stored procedures are especially important for client/server database systems because storing the procedure on the server side means that it is available to all clients. And when the procedure is modified, all clients automatically get the new version.

stored procedure in Java (SPJ) A stored procedure whose body is a static Java method.

stub

1. A dummy procedure used when linking a program with a runtime library. The stub need not contain any code. Its only purpose is to prevent "undefined label" errors at link time.
2. A local procedure in a remote procedure call (RPC). A client calls the stub to perform a task, not necessarily aware that the RPC is involved. The stub transmits parameters over the network to the server and returns results to the caller.

T

TACL See HP Tandem Advanced Command Language (TACL).

TCP/IP See Transmission Control Protocol/Internet Protocol (TCP/IP).

Technical Documentation HP's Technical documentation is found at docs.hp.com.

thread A task that is separately dispatched and that represents a sequential flow of control within a process.

`threads` The nonnative thread package that is shipped with Sun Microsystems Java SE 6.0.

`throw` Java keyword used to raise an exception.

`throws` Java keyword used to define the exceptions that a method can raise.

TMF See HP NonStop Transaction Management Facility (TMF)

TNS/E The hardware platform based on the Intel® Itanium® architecture and the HP NonStop operating system, and the software specific to that platform. All code is PIC (position independent code).

TNS/R The hardware platform based on the MIPS™ architecture and the HP NonStop operating system, and the software specific to that platform. Code might be PIC (position independent code) or non-PIC.

transaction A user-defined action that a client program (usually running on a workstation) requests from a server .

Transaction Management Facility (TMF) A set of HP software products for NonStop systems that assures database integrity by preventing incomplete updates to a database. It can continuously save the changes that are made to a database (in real time) and back out these changes when necessary. It can also take online "snapshot" backups of the database and restore the database from these backups.

Transmission Control Protocol/Internet Protocol (TCP/IP) One of the most widely available nonvendor-specific protocols , designed to support large, heterogeneous networks of systems.

TS/MP See HP NonStop TS/MP.

U-Z

-Xeprof Java application profile collection option.

-Xeverbosegc Java application's Garbage Collector (GC) activity profile collection option.

Unicode A character-coding scheme designed to be an extension of ASCII . By using 16 bits for each character (rather than ASCII's 7), Unicode can represent almost every character of every language and many symbols (such as "&") in an internationally standard way, eliminating the complexity of incompatible extended character sets and code pages. Unicode's first 128 codes correspond to those of standard ASCII.

uniform resource locator (URL) A draft standard for specifying an object on a network (such as a file, a newsgroup, or, with JDBC, a database). URLs are used extensively on the World Wide Web . HTML documents use them to specify the targets of hyperlinks .

URL See uniform resource locator (URL).

virtual machine (VM) A self-contained operating environment that behaves as if it is a separate computer. See also Java virtual machine and Java Hotspot virtual machine .

VM See virtual machine (VM).

World Wide Web (WWW) An Internet client - server hypertext distributed information retrieval system that originated from the CERN High-Energy Physics laboratories in Geneva, Switzerland. On the WWW everything (documents, menus, indexes) is represented to the user as a hypertext object in HTML format. Hypertext links refer to other documents by their URLs. These can refer to local or remote resources accessible by FTP, Gopher, Telnet, or news, as well as those available by means of the HTTP protocol used to transfer hypertext documents. The client program (known as a browser) runs on the user's computer and provides two basic navigation operations: to follow a link or to send a query to a server.

wrapper A shell script that sets up the proper execution environment and then executes the binary file that corresponds to the shell's name.

WWW See World Wide Web (WWW).