Hewlett Packard
Enterprise

NonStop Server for Java 7.0 Programmer's Reference

Part Number: 693949-007
Published: April 2017
Edition: L15.02, J06.15, and H06.26 and all subsequent L-series, J-series, and H-series RVUs

The information contained herein is subject to change without notice. The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

Confidential computer software. Valid license from Hewlett Packard Enterprise required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Links to third-party websites take you outside the Hewlett Packard Enterprise website. Hewlett Packard Enterprise has no control over and is not responsible for information outside the Hewlett Packard Enterprise website.

Acknowledgments

Microsoft®, Windows®, and Windows NT® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Intel®, Intel Itanium®, Pentium®, and Celeron® are trademarks of Intel Corporation in the United States and other countries.

Java® and Oracle® are registered trademarks of Oracle and/or its affiliates.

UNIX® is a registered trademark of The Open Group. Motif, OSF, the OSF logo, OSF1, OSF/Motif, IT DialTone, The Open Group, and Open Software Foundation are trademarks of The Open Group.

OSF MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE OSF MATERIAL PROVIDED HEREIN, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. OSF shall not be liable for errors contained herein or for incidental consequential damages in connection with the furnishing, performance, or use of this material.

© 1990, 1991, 1992, 1993 Open Software Foundation, Inc. The OSF documentation and the OSF software to which it relates are derived in part from materials supplied by the following:© 1987, 1988, 1989 Carnegie-Mellon University. © 1989, 1990, 1991 Digital Equipment Corporation. © 1985, 1988, 1989, 1990 Encore Computer Corporation. © 1988 Free Software Foundation, Inc. © 1987, 1988, 1989, 1990, 1991 Hewlett-Packard Company. © 1985, 1987, 1988, 1989, 1990, 1991, 1992 International Business Machines Corporation. © 1988, 1989 Massachusetts Institute of Technology. © 1988, 1989, 1990 Mentat Inc. © 1988 Microsoft Corporation. © 1987, 1988, 1989, 1990, 1991, 1992 SecureWare, Inc. © 1990, 1991 Siemens Nixdorf Informationssysteme AG. © 1986, 1989, 1996, 1997 Sun Microsystems, Inc. © 1989, 1990, 1991 Transarc Corporation. OSF software and documentation are based in part on the Fourth Berkeley Software Distribution under license from The Regents of the University of California. OSF acknowledges the following individuals and institutions for their role in its development: Kenneth C.R.C. Arnold, Gregory S. Couch, Conrad C. Huang, Ed James, Symmetric Computer Systems, Robert Elz. © 1980, 1981, 1982, 1983, 1985, 1986, 1987, 1988, 1989 Regents of the University of California.

Contents

About this manual

NonStop Server for Java 7.0 Programmer's Reference manual identifies the changes in the Hewlett Packard Enterprise adaptation of the reference Java implementation, emphasizing the differences between the reference implementation and NSJ7. For more information on the standard architecture, see **Oracle Java documentation**.

Manual information

Abstract

This manual describes the HPE Nonstop Server for Java, based on Java Platform Standard Edition 7.0, a Java environment that supports compact, concurrent, dynamic, and portable programs for the enterprise server. The NonStop Server for Java 7.0 (NSJ7) uses the HPE NonStop operating system to add scalability and program persistence to the Java environment.

Product version

NonStop Server for Java 7.0

Supported hardware

All HPE Integrity NonStop NS-series (TNS/E) servers and HPE Integrity NonStop L-series (TNS/X) servers.

Supported release version updates (RVUs)

This manual supports L15.02 and subsequent L-series RVUs, J06.15 and all subsequent J-series RVUs, and H06.26 and all subsequent H-series RVUs until otherwise indicated by its replacement publications.

Intended audience

This *NonStop Server for Java 7.0 Programmer's Reference* manual is for the Java programmers who want to use Java on NonStop systems.

Programmers developing Java applications on NonStop systems must be familiar with:

- NonStop System fundamentals applicable to NSJ7
- The Open System Services (OSS) environment

For more information, see “Related information” (page 12).

This manual does not provide information about the following:

- The Java language and tools. For details about the Java language and tools, see **Oracle Java documentation**.
- Accessing SQL/MP and SQL/MX databases. For information on accessing the NonStop SQL/MP and SQL/MX databases, see *JDBC Driver for SQL/MP Programmer's Reference* and *JDBC Type 4 Driver Programmer's Reference for SQL/MX Release x.x*.

NonStop Server for Java 7.0 Programmer's Reference manual identifies the changes in the Hewlett Packard Enterprise adaptation of the reference Java implementation, emphasizing the differences between the reference implementation and NSJ7. For more information on the standard architecture, see **Oracle Java documentation**.

New and changed information

Changes to 693949-007 manual are as follows:

- Installation instructions are updated.

Changes to 693949-006R manual are as follows:

- Updated Hewlett Packard Enterprise references.

Changes to 693949-006 manual are as follows:

- Removed the RVU support information notes from features section.

Changes to 693949-005 manual are as follows:

- Features pertaining to NSJ7 Update 2 are added in the section “Features” (page 16).
- “Multi-threaded programming” (page 45) is updated to reflect general availability thread time slice option.
- Added a note in the section “FastExec option support” (page 51).
- Added the section “Java `getLocalHost()` caching” (page 51).
- Added the section “Heap layout for G1GC” (page 57).
- Added G1GC to the section “Implementation of garbage collector types” (page 59).
- Added a warning message in the section “Implementation of garbage collector types” (page 59).
- Added code samples for thread time slice feature in the section “Thread scheduling” (page 45).
- Added the section “G1GC collector” (page 61).
- Updated `vproc` information in the section “Verifying the JI installation” (page 86).

Changes to 693949-004 manual are as follows:

- Updated the entire manual with L-series specific information.
- Updated unsupported options for the C3 compiler.
- Updated the migration considerations for NSJ7 applications to L-series.
- Updated the “Java infrastructure” (page 77) section to reflect the L-series updates.

Changes to 693949–002 manual are as follows:

- “Feature changes” (page 9)
- “Document changes” (page 10)

Feature changes

1. SecureRandom Startup enhancement: An improved version of random number generation is introduced, which reduces the time taken for a CPU to generate the first random number in a Java application.
2. JVM data segment allocation: Two command line options are added for better utilization of the virtual memory. One each for 32–bit and 64–bit NSJ7.
3. `VirtualMachine.list()` support: A method that is used to return the list of Java processes in the target machine.
4. Handling large GC files: A command line option is introduced to support GC log file rotation.

Document changes

- Added a note in the section "Installation requirements" (page 23).
- Updated version procedure information in the section "Verifying the Java installation" (page 26).
- Added a section "Handling large GC log files" (page 62).
- Updated the section "JVM data segment for 32–bit JDK7" (page 62).
- Updated the section "JVM data segment for 64–bit JDK7" (page 63).
- Added a section "Java signal handlers" (page 72).
- Added a section "SecureRandom startup enhancement" (page 76).
- Added a section "`VirtualMachine.list()` support" (page 76).
- Client_socket is modified to `client-socket` and server_socket to `server-socket` in the section "Key" (page 82).
- Added a section "Analyzing garbage collection data" (page 114).
- Added a section "Change in loading of `.hotspot_compiler` and `.hotspotrc files`" (page 76).
- Updated the section "Node agent" (page 115).

Changes to 693949–001 manual are as follows:

This is a new manual.

NOTE: This manual is valid for H-series and J-series systems. However, "Garbage collection (GC)" (page 56) implementation enhancements are applicable only for J-series systems.

Document organization

This manual is structured as follows:

Table 1 Organization of chapters

Section	Description
"Introduction to NSJ7" (page 16)	Explains NonStop software fundamentals applicable to NSJ7, describes associated Java products on the NonStop system, lists the J2SE and JDK features supported by NSJ7.
"Installation and configuration" (page 23)	Explains installation and configuration requirements, NSJ7 directory structure, procedures to run Java tools and to verify the installation.
"Getting started" (page 34)	Explains the procedure to run a sample Java program.
"Implementation specifics" (page 38)	Explains the NSJ7 implementation specifics.
"Java infrastructure" (page 77)	Java Infrastructure provides Java abstraction to NonStop FS and TS/MP API.
"Transactions" (page 90)	Explains the transaction behavior in NSJ7.
"Application tuning and profiling" (page 94)	Describes the application profiling environment, HPROF (that is Xeprof), and HPROF agent. Also, describes `HPjmeter` profile data analysis tool.
"Migrating applications" (page 99)	Explains migration details for earlier NSJ applications.

Table 1 Organization of chapters *(continued)*

Section	Description
“Supported and unsupported features of NSJ7” (page 111)	Summarizes the supported and unsupported features of NSJ7.
“Addendum to `HPjmeter` 4.3 user's guide” (page 112)	Provides instructions for using the `HPjmeter` tool on the NonStop system.

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 information

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

- “NonStop server for Java library” (page 13)
- “NonStop system computing documents” (page 13)
- “Oracle Java documents” (page 14)
- Various white papers on NonStop Server for Java.

NonStop server for Java library

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

- *NonStop Server for Java 7.0 Tools Reference*
- *NonStop Server for Java API Reference Pages*

NonStop system computing documents

The following NonStop system computing documents are available in the HPE NonStop Technical Library at **Hewlett Packard Enterprise Support Center (HPESC)**.

- Additional Java Based Products
  - *JDBC Driver for SQL/MP Programmer's Reference*
  - *JDBC Driver for SQL/MX Programmer's Reference*
  - *JToolkit for Java API Reference Pages*
    This documentation describes how to access:
    - *JToolkit for NonStop Server for Java Programmer's Reference and API*
- *C/C++ Programmer's Guide*
- *DLL Programmer's Guide for TNS/E Systems*
- *eld Manual*
- *iTP Secure WebServer System Administrator's Guide*
- *Kernel-Managed Swap Facility (KMSF) Manual*
- *Native Inspect Manual*
- *NonStop Servlets for JavaServer Pages (NSJSP) System Administrator's Guide*
- ODBC (Open Database Connectivity) Documents
  - *ODBC Server Reference Manual*
  - SQL/MX Connectivity Service Manual
- *Open System Services Installation Guide*
- *Open System Services Porting Guide*
- *Open System Services Programmer's Guide*
- *Spooler FASTP Network Print Processes Manual*
- *Spooler Utilities Reference Manual*
- SQL/MP Manuals
  - *Introduction to NonStop SQL/MP*
  - *SQL/MP Reference Manual*
  - *SQL/MP Installation and Management Guide*
  - *SQL/MP Query Guide*
  - *SQL/MP Report Writer Guide*
  - *SQL/MP Version Management Guide*

- *SQL/MP Messages Manual*
- *SQL/MP Programming Manual for C*
- *SQL/MP Programming Manual for COBOL*
- See also *SQL Supplement for H-series RVUs*.

- SQL/MX Manuals

  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*
  - *SQL/MX Guide to Stored Procedures in Java*
  - *SQL/MX Quick Start*
  - *SQL/MX Comparison Guide for SQL/MP Users*
  - *SQL/MX Installation and Management Guide*
  - *SQL/MX Glossary*
  - *SQL/MX Query Guide*
  - *SQL/MX Reference Manual*
  - *SQL/MX Messages Manual*

    Describes SQL/MX messages.
  - *SQL/MX Programming Manual for C and COBOL*
  - *SQL/MX Data Mining Guide*
  - *SQL/MX Queuing and Publish/Subscribe Services*
- *TCP/IP Configuration and Management Manual*
- *TCP/IPv6 Configuration and Management Manual*
- TMF Manuals
  - *TMF Introduction*
  - *TMF Application Programmer's Guide*
- *TS/MP Pathsend and Server Programming Manual*
- *TS/MP System Management Manual*

Oracle Java documents

For Java SE 7 documentation, visit the Oracle web site **Java Platform Standard Edition 7 Documentation**.

The following documents are available on Oracle web site at the time of this manual's publication. However, Hewlett Packard Enterprise cannot guarantee their current availability. If a link to an

Oracle document fails, use the Oracle documentation, which is available in ZIP format in the distribution CD.

- **JNDI document**
- **JDBC documents**
- **Java Print Service (JPS) document**
- **Java Transaction API (JTA) document**
- **Java Transaction Service (JTS) document**
- **Java Remote Method Invocation (RMI) document**

Publishing history

Part number	Product version	Published
693949-003	NonStop Server for Java 7.0	February 2015
693949-004	NonStop Server for Java 7.0	February 2015
693949-005	NonStop Server for Java 7.0	May 2015
693949-006	NonStop Server for Java 7.0	August 2015
693949-006R	NonStop Server for Java 7.0	November 2015
693949-007	NonStop Server for Java 7.0	April 2017

1 Introduction to NSJ7

NonStop Server for Java 7.0 (NSJ7) provides a Java environment that supports compact, concurrent, and dynamic portable programs for the NonStop systems. NSJ7 is supported on the NonStop Open System Services (OSS) environment. NSJ7 uses the NonStop operating system to provide scalability and program persistence to the Java environment. NSJ7 is based on the Java Platform Standard Edition (Java SE) 7.0 reference, which is a Java implementation for Solaris. NSJ7 contains two product components, 32–bit NSJ7 (T2766) and 64–bit NSJ7 (T2866).

NSJ7 is a fully compliant headless JDK. This section discusses the following topics:

Features

Starting with J06.19 RVU, NSJ7 (SPR IDs T2766H70^ACN and T2866H70^ACN) provide the following new features, which are applicable for both 32–bit and 64–bit NSJ7:

- **Support for G1GC**

  This release contains a new Garbage Collector (GC) algorithm called Garbage First Garbage Collector (G1GC) that is supported by Oracle. This version of G1GC is a beta version.

  For more information, see **http://www.oracle.com/technetwork/java/javase/tech/g1-intro-jsp-135488.html**.

- **Thread Pre-emption Changes**

  In earlier releases of NSJ7, thread pre-emption feature was used as a beta version. Starting with J06.19, thread pre-emption option is made available as a general availability function. Using this option, Java user threads can be pre-empted and the JVM daemon threads are not pre-empted.

  `-XX:ThreadTimeSlice` option is enhanced to meet the product quality. The default `timeslice` for one Java thread is 400 ms. The Java thread is pre-empted only when it is executing code from Java code cache and has completed its allotted time slice. This includes execution in both Java interpreted code and Java Hotspot compiled code. This option benefits only a multithreaded Java application, which ensures that there is an equal amount of time slice for all the Java threads. This reduces the starvation of runnable threads waiting for CPU time as other Java thread may not yield the CPU.

  NOTE: The `timeslice` mentioned is the minimum time used by a thread. The actual time used by a thread depends on which mode the thread is executed, privileged mode or native code. If the thread is executed in privilege mode or native code, and when the `timeslice` expires, the thread gets another `timeslice` to run.

- **EMS Templates Installation Changes**

  During the installation of NSJ7 (T2766H70^ACN and T2866H70^ACN), DSM/SCM installs EMS templates for Java. To achieve this, a separate configuration for EMS templates file is distributed along with the PAX file.

- **Java Infrastructure**

NOTE: Currently, the product number, installation path, and configuration path refers to H70 in this manual. If JI is being installed and configured on L-series RVUs, then consider H70 as L70.

Java Infrastructure (JI) is available as a separate product with the T number T2966H70. JI must be installed at the same location as that of NSJ7. By default, JI is installed at the location `/usr/tandem/nssjava/jdk170_h70`. JI contains the following components:

ji.jar
: the jar file is available at the location `$JREHOME/lib`.

libji.so
: the native library is available at location `$JREHOME/lib/oss` (for 32-bit) and `$JREHOME/lib/oss64` (for 64-bit).

Features released in the earlier versions of NSJ7 are as follows:

- **Support for 64–bit Java application**

  Using 64–bit NSJ7, it is possible to develop or port applications that require Java heaps greater than 1276 MB, which is the limit with 32–bit NSJ7. The theoretical maximum Java heap available with 64–bit NSJ7 is 484 GB. For more information, see “Managing generation size” (page 58) and “64–bit process support” (page 69).

  NOTE: 64–bit NSJ7 (T2866) cannot be installed as a standalone product.

- **Support for nonblocking I/O for regular and non-regular OSS files**

  NSJ7 uses the new Pthreads library called POSIX User Thread Model (PUT) library. As a result, NSJ7 provides nonblocking I/O support for both regular and non-regular OSS files. For more information, see “Pthread library changes” (page 49).

- **Support for parallel and CMS GC**

  NSJ7 supports parallel and CMS GC which provide reduced GC pause times.

  64–bit JDK enables you to port or develop large Java applications on NSJ7. If an application uses large Java heap and serial GC, the application pause time due to GC may be high. Hence, to reduce application pause time during garbage collection, NSJ7 provides parallel and concurrent garbage collectors. This implementation differs from other platforms. However, the functionality remains same. The difference is due to the absence of Kernel threads on NonStop systems.

  On other platforms, Kernel threads allow GC threads to run in parallel on multi core systems. On Nonstop, in the absence of Kernel threads only one thread runs at any time, and is managed in the user space by the Pthreads library. As a result, a multi-threaded application on Nonstop cannot utilize the parallelism in execution offered by multi core systems.

  To introduce parallelism in garbage collection, the garbage collector threads are converted to processes, when parallel and CMS garbage collectors and G1GC are enabled.

  For more information, see “Migrating from serial GC to parallel GC” (page 105).

  NOTE:

  - For user thread limitation see *Open System Services Programmer's Guide*.
  - Parallel, CMS GC, and G1GC are not supported on single core machines.

- **NSJ7 continues to provide headless support**

  NSJ7 is a headless JVM that conforms to the Oracle's headless support standards regarding Java Abstract Window Toolkit (AWT) classes and methods. For implementation-specific information, see “Headless support” (page 38).

- **Support for JPDA**

  JPDA consists of three interfaces designed for debuggers in development environments for desktop systems. For more information, see **Oracle Java documentation for JPDA**.

Java HotSpot server virtual machine

NSJ7 implements the HotSpot server compiler and the runtime Java HotSpot virtual machine. The HotSpot Server Java virtual machine provides a fast, reliable technology for the enterprise server environment. For more information, see **The Java HotSpot Server VM** and **Java Virtual Machines**.

Java standard edition development kit (JDK)

NSJ7 consists of the following standard Java components (and the Hewlett Packard Enterprise extensions described else where in this manual):

- Java virtual machine (VM) based on the Java SE Runtime Environment (JRE) 7.0
- Core Packages of the Java™ SE Development Kit (JDK) 7.0
- Standard Java SE 7.0 tools as documented in the *NonStop Server for Java 7.0 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.
- JDK 7.0 API packages such as `java`, `javax`, and `org` packages described in the **Java Platform Standard Edition 7.0 API Specification**.

Table 2 (page 18) lists the packages which NSJ7 supports according to Headless Support.

Table 2 List of packages with headless support on NSJ7

Package	Description
java.awt and AWT-related packages	Contains the classes used 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 these 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, which works in the similar mode on all platforms.

If a call is made to the code that depends on a keyboard, display, mouse, or sound processing, NSJ7 throws a `java.awt.HeadlessException`.

For more information on the core packages of Java SE Development Kit 7.0, see “Oracle Java documents” (page 14).

Java naming and directory interface (JNDI)

The 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 *Oracle Java documentation* **Java Naming and Directory Interface 1.1.1 Specification**.

IPv6 support

The Java SE JRE 7.0 release includes Internet Protocol version (IPv6) support in Java Networking. For more information, see *Oracle Java Documentation* **Networking IPv6 User Guide**.

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

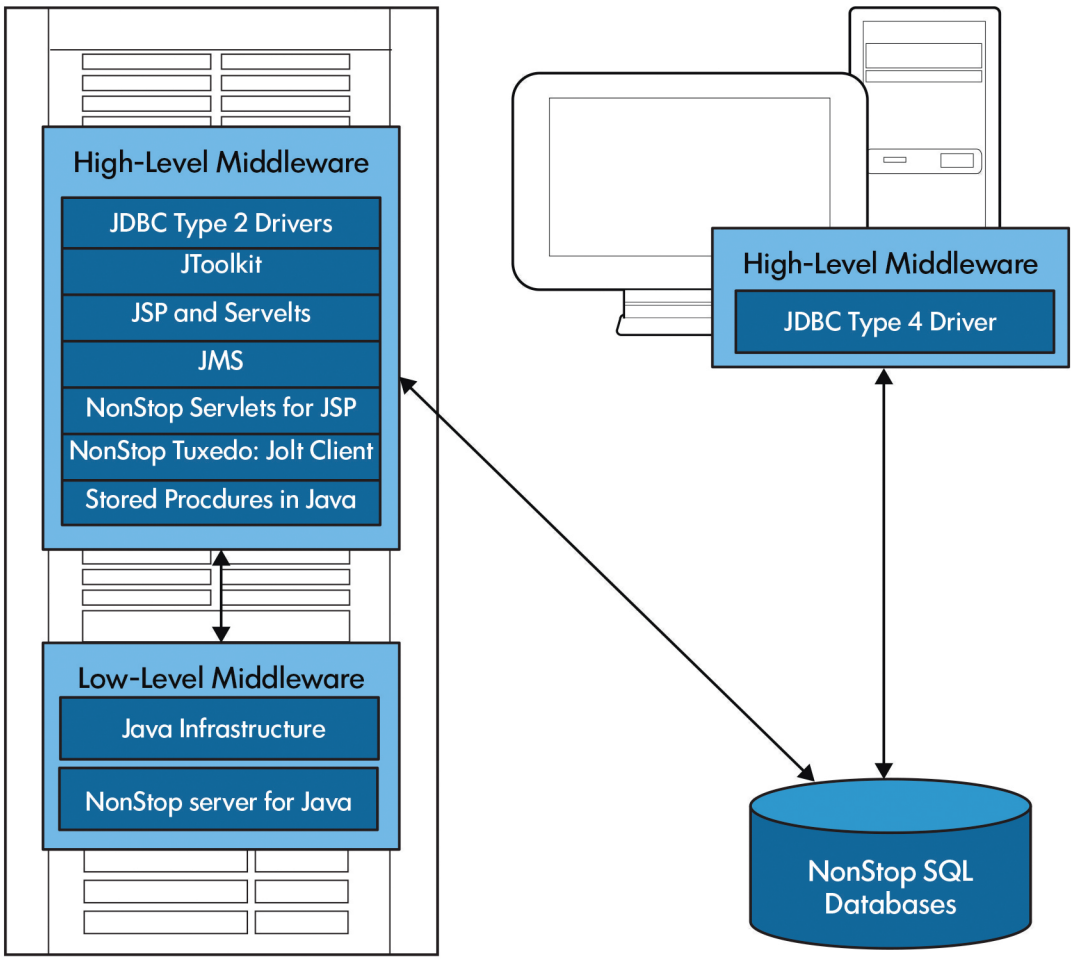
Associated Java based products

Consider a scenario in which you develop a standard Java application and then deploy and run them on NonStop servers. You can develop these applications by using a wide-range of associated Java based products that can use the NSJ7 runtime. The products are as follows:

- “JDBC drivers for NonStop SQL database access” (page 20)
- “JToolkit for NonStop servers” (page 21)
- “NonStop servlets for JavaServer pages” (page 21)
- “NonStop server for Java message service (JMS)” (page 21)
- “NonStop Tuxedo: Jolt client” (page 22)
- “Stored procedures in Java” (page 22)

Figure 1 (page 20) illustrates the high-level middleware products that appear working with NSJ7 and NonStop SQL databases.

Figure 1 Java based products on NonStop systems



JDBC drivers for NonStop SQL database access

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

- Type 2, which is native API driver to use in Java programs running with NonStop Server for Java on a NonStop system. The type 2 driver is installed using SUT.
  - 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 in the database engine. Type 4 driver communicates directly to the database using Java sockets. You can use the HPE 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 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 **Oracle documentation**.

For information on Hewlett Packard Enterprise drivers that are provided to access SQL/MX or SQL/MP, see JDBC driver manuals at **http://www.hpe.com/info/nonstop-docs**.

JToolkit for NonStop servers

The HPE 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. For more information, see *JToolkit for Java API Reference Pages*.

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 file called `$RECEIVE`, which is required to enable a process to act as a Pathway server. These servers are typically used in legacy applications. Pathway server programs read requests from requestor programs and act on those requests. The Guardian requestor or server model is described in the *TS/MP 2.5 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 send messages to it and in which order, all messages to a process arrive using a single file system connection. Reading from `$RECEIVE` a process receives a message whether the message is a request from another user process or a system message.

Pathsend API for Java

The 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 are written in C, COBOL, or Java.

Scalable TCP/IP

Scalable TCP/IP (SIP) for the NonStop Server for Java provides a transparent way to provide 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 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 *NonStop Servlets for JavaServer Pages (NSJSP) System Administrator's Guide*. For more information about the iTP Secure WebServer, see *iTP WebServer System Administrator's Guide*.

NonStop server for Java message service (JMS)

NonStop Server for Java Message Service (NSJMS) is the JMS provider that implements Oracle's Java Message Service (JMS) API, on NonStop servers. NSJMS uses the performance and reliability inherent in SQL/MX products to provide standard-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.

The following are the features and functions of NSJMS:

- 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 Oracle 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 Oracle's published specification, Java Message Service, except as noted in NSJMS documentation. The specification is available at **Java Message Service (JMS)**. For more information about NSJMS, see *NonStop JMS User's Manual*.

NonStop Tuxedo: Jolt client

The Jolt product is a Java based interface to the HPE 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 worldwide 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 in 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 information, see TUXEDO product documentation at **http://www.hpe.com/info/nonstop-docs**.

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.

An 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, an 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 *SQL/MX Guide to Stored Procedures in Java*.

2 Installation and configuration

NOTE: Currently, the installation path and configuration path refer to H70 in this manual. If the product is being installed on L-series RVUs, then consider H70 as L70.

Summary of installation and configuration tasks

- Perform the installation requirements task described in “Installation requirements” (page 23).
- Perform the preinstallation tasks described in “Preinstallation tasks” (page 23).
- Perform the installation tasks described in “Installing NSJ7” (page 23).
- Perform the post-installation tasks described in “Post-installation tasks” (page 26) as per your requirement.
- Perform the verification tasks described in “Verifying the Java installation” (page 26) as per your requirement.
- Perform the configuration tasks described in “Configuring NonStop system for NSJ7” (page 27) as per your requirement.
- Perform the verification of the NSJ7 directory structure shown in “NSJ7 directory structure” (page 32).

Installation requirements

- NSJ7 can be installed only on NonStop systems running on H06.26 and J06.15 or later RVUs.
- The software requirements list the earlier compatible versions of the required software for installing NSJ7.
- Read the Softdoc before installing the product as the software requirements are described in NSJ7 product Softdoc.

NOTE: The current version of NSJ7 can coexist with earlier version of NSJ as long as the earlier version of NSJ and the current version of NSJ7 are installed in different directories.

Preinstallation tasks

- If this is the first installation of NSJ7, ignore this step. Back up all the files, including security certificates (`cacerts`) and other user created files in the NSJ7 installation directory for H-series (`/usr/tandem/nssjava/jdk170_h70`).

Preinstallation tasks for installing 64–bit NSJ7

- The installation directory selected for 64–bit NSJ7 (T2866H70) must contain the 32–bit NSJ7 (T2766H70) installation.
- Before proceeding to the 64–bit NSJ7 installation, follow the instructions provided in “Installing NSJ7” (page 23) for 32–bit NSJ7, and complete the installation.

Installing NSJ7

Installation of 32–bit and 64–bit NSJ7 is a three step process and is described in the following topics:

1. “Placing the new software on the NonStop system using DSM/SCM” (page 24)
2. Extracting script and PAX files for 32–bit and 64–bit NSJ7.
3. Installing 32–bit and 64–bit by using NSJ7 installation script.

IMPORTANT: If you intend to use 32–bit NSJ7, installing 32–bit NSJ7 is sufficient. If you intend to use 64–bit features provided by NSJ7, then both 32–bit and 64–bit NSJ7 must be installed.

Placing the new software on the NonStop system using DSM/SCM

NOTE: The following procedure is applicable for both 32–bit and 64–bit NSJ7.

- Receive the SPR from disk or tape.
- Copy the SPR to a new revision of the software configuration you want to update.
- Build and apply the configuration revision.
- Run `ZPHIRNM` to perform the rename step.

DSM/SCM places the `pax` file in `$tsvvol.ZOSSUTL` subvolume, where, `tsvvol` is the disk volume in which the DSM/SCM places the target subvolumes (TSVs).

For more information, see *DSM/SCM User's Guide*.

NOTE: Installation of NSJ7 does not require a system generation, and therefore does not require a cold-load.

Extract script and PAX files of NSJ7 32-bit

1. The DSM/SCM planner interface with the **Manage OSS Files** check box selected. Using this method NSJ7 JDK files are extracted to the default or standard location.

   For example, `/usr/tandem/nssjava/javainstall/$(VPROC_STR)`

   `VPROC_STR`

   is the `VPROC` of the product.

   For example, the `VPROC` of `T2766H70^ADG` is `T2766H70_11MAR2017_jdk170_ADG`.

   The following files are extracted:

   - `NSJavaInstall32` - an installation script to install JDK from the JDK PAX file.
   - `T2766IPAX` - JDK installation PAX file.

2. If **Manage OSS Files** is not selected, DSM/SCM places the PAX file to the `TSV ($tsvvol.ZOSSUTL)`. Then, using PINSTALL you must manually extract the contents of the pax file to the NSJ7 installation directory.

   For more information on PINSTALL, see *DSM/SCM User's Guide*.

   NOTE:

   - DSM/SCM extracts only the installation files, you must run the scripts mentioned in Install NSJ7 32-bit to complete the installation.
   - If the extracted files are deleted, you must re-extract the files.

Extract script and PAX files of NSJ7 64-bit

1. The DSM/SCM planner interface with the **Manage OSS Files** check box selected. Using this method NSJ7 JDK files are extracted to the default or standard location.

   For example, `/usr/tandem/nssjava/javainstall/$(VPROC_STR)`

   `VPROC_STR`

   is the `VPROC` of the product.

   For example, the `VPROC` of `T2866H70^ADG` is `T2866H70_11MAR2017_jdk170_ADG`.

   The following files are extracted:

   - `NSJavaInstall64` - an installation script to install JDK from the JDK PAX file.
   - `T2866IPAX` - JDK installation PAX file.

2. If **Manage OSS Files** is not selected, DSM/SCM places the PAX file to the `TSV ($tsvvol.ZOSSUTL)`. Then, using PINSTALL you must manually extract the contents of the pax file to the NSJ7 installation directory.

   For more information on PINSTALL, see *DSM/SCM User's Guide*.

   NOTE:

   - DSM/SCM extracts only the installation files, you must run the scripts mentioned in Install NSJ7 64-bit to complete the installation.
   - If the extracted files are deleted, you must re-extract the files.

Install NSJ7 32–bit

You can install 32-bit NSJ7 either in the standard location or custom location.

1. Issue the following command to install 32-bit NSJ7 at the standard location:

   ```
   >sh $(SCRIPT_LOCATION)/NSJavaInstall32
   ```

   `$(SCRIPT_LOCATION)`

   is the default location, `/usr/tandem/nssjava/javainstall/$(VPROC_STR)`

   The contents of T2766IPAX are delivered to the OSS directory at:

   `/usr/tandem/nssjava/jdk170_l70`

2. Issue the following command to install 32-bit NSJ7 at the custom location:

   ```
   >sh $(SCRIPT_LOCATION)/NSJavaInstall32 <custom location>
   ```

   `$(SCRIPT_LOCATION)`

   is the default location, `/usr/tandem/nssjava/javainstall/$(VPROC_STR)`

   The contents of T2766IPAX are delivered to the OSS directory at the specified custom location.

Example 1 To install 32-bit NSJ7 in a custom location `/h/myjava`

Issue the following command:

```
>sh $(SCRIPT_LOCATION)/NSJavaInstall32 </h/myjava>
where,
SCRIPT_LOCATION is /usr/tandem/nssjava/javainstall/$(VPROC_STR)
```

The contents of T2766IPAX are delivered to the following location:

`/h/myjava/nssjava/jdk170_l70`

Install NSJ7 64–bit

You can install 64-bit NSJ7 either in the standard location or custom location.

1. Issue the following command to install 64-bit NSJ7 at the standard location:

   ```
   >sh $(SCRIPT_LOCATION)/NSJavaInstall64
   ```

   `$(SCRIPT_LOCATION)`
   is the default location, `/usr/tandem/nssjava/javainstall/$(VPROC_STR)`

   The contents of T2866IPAX are delivered to the OSS directory at:

   `/usr/tandem/nssjava/jdk170_l70`

2. Issue the following command to install 64-bit NSJ7 at the custom location:

   ```
   >sh $(SCRIPT_LOCATION)/NSJavaInstall64 <custom location>
   ```

   `$(SCRIPT_LOCATION)`
   is the default location, `/usr/tandem/nssjava/javainstall/$(VPROC_STR)`

   The contents of T2866IPAX are delivered to the OSS directory at the specified custom location.

Example 2 To install 64-bit NSJ7 in a custom location `/h/myjava`

Issue the following command:

```
>sh $(SCRIPT_LOCATION)/NSJavaInstall64 </h/myjava>
where,
SCRIPT_LOCATION is /usr/tandem/nssjava/javainstall/$(VPROC_STR)
```

The contents of T2866IPAX are delivered to the following location:

`/h/myjava/nssjava/jdk170_l70`

Post-installation tasks

- Installation of NSJ7 does not automatically create the softlink `/usr/tandem/java softlink`. If required, that softlink must be manually created to point to the NSJ7 JDK/JRE installation for H-series (`/usr/tandem/nssjava/jdk170_h70`) after the installation is completed.
  - Ensure to remove the existing softlink.
  - As required, create the softlink `/usr/tandem/java` to point to the NSJ7 home directory as follows:

    For H-series `$ ln –s /usr/tandem/nssjava/jdk170_h70 /usr/tandem/java`

    NOTE: Consider the possibility of `/usr/tandem/java` softlink already being set and used to launch previous versions of NSJ. Redirecting it as shown here, causes all future invocations of `/usr/tandem/java/bin/java` to launch NSJ7 Java. This can lead to failure of applications which are not migrated to NSJ7 or are not intended to be run with NSJ7.

- Restore the files backed up during the pre-installation phase.

Verifying the Java installation

Check with your system administrator for the installed NSJ7 (32–bit and 64–bit) software locations, and verify the installation and the environment. This example assumes that NSJ7 is installed in a nonstandard location for H–series `/home/lee/nssjava/jdk170_h70` directory:

NOTE: For more information about the operating system requirements, see `T2766H70` or `T2866H70` Softdoc file.

1. Set the PATH environment variable by using the following command at the OSS prompt:

For 32–bit NSJ7:

```
$ export PATH=/home/lee/nssjava/jdk170_h70/bin:$PATH
```

For 64–bit NSJ7:

```
$ export PATH=/home/lee/nssjava/jdk170_h70/bin/oss64:$PATH
```

2. Confirm that the path is set correctly by typing the `whence` command:
   For 32–bit NSJ7:

   ```
   whence java
   /home/lee/nssjava/jdk170_h70/bin/java
   ```

   For 64–bit NSJ7:

   ```
   whence java
   /home/lee/nssjava/jdk170_h70/bin/oss64/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 in the `java` binary.

   The version identifier has the following form:

   `T2766Hnn` or `T2866Hnn`

   A `vproc` example is:

   ```
   Version procedure: T2766H70_17APR2015_jdk170_ACN...
   ```

   or

   ```
   Version procedure: T2866H70_17APR2015_jdk170_ACN...
   ```

NOTE: JREHOME shell variable need not be set for NSJ7. However, if this variable is set, NSJ7 will accept it. If this variable is set, then you must ensure that it points to the correct JDK.

Configuring NonStop system for NSJ7

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

- “Creating larger or additional swap file” (page 27)
- “Setting environment variables” (page 28)
- “Configuring TCP/IP and DNS for RMI” (page 30)
- “Memory considerations: Moving QIO to KSEG2” (page 30)

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

Creating larger or additional swap file

Hewlett Packard Enterprise recommends that the swap file size must be equal to the physical memory size, 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.super 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 *Kernel-Managed Swap Facility (KMSF) Manual*.

NOTE: When using the large Java heap feature provided by 64–bit NSJ7, Hewlett Packard Enterprise recommends that a contiguous block of Kernel-managed swap space that is at least greater than the Java heap size be available.

Setting environment variables

The following topics describe 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. Add the absolute path of the `bin` directory to the `PATH` environment variable.

To add the absolute path, use the following command:

For 32–bit NSJ7:

```
export PATH=/install_dir/nssjava/jdk170_h70/bin:$PATH
```

For 64–bit NSJ7:

```
export PATH=/install_dir/nssjava/jdk170_h70/bin/oss64:$PATH
```

where, *`install_dir`* is the directory in which the NSJ7 is installed. By default, for H-series this is `/usr/tandem/nssjava/jdk170_h70`.

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. For more information on how to set the PATH in your startup file, see *Open System Services User's Guide*.

CLASSPATH

The class search path (class path) is the path where the Java run time environment searches for classes and other resource files. The class path notifies the JDK tools and applications to find third-party and user-defined classes. The class path is 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.

NOTE: You can use either `–classpath` or `–cp` option on the command line. Both the options are valid.

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 *NonStop Server for Java 7.0 Tools Reference*.

`classpath1:classpath2`

Class paths to the `.jar`, `.zip`, or `.class` files. Each class path must end with a filename or directory name depending on the specific 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 contains the "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 the default class path, 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 not directories or archives (`.zip` or `.jar` files) are ignored. For example,

Example 3 Setting CLASSPATH in a `java` command

If 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 must set the class path so that it contains `/java/MyClasses`.

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

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

Example 4 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 can use the following command to set and export the `CLASSPATH` environment variable and then run Java.

To run that application, use the following commands:

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

For more information about setting the classpath, see documentation available in Oracle web site (**http://docs.oracle.com/javase/7/docs/technotes/tools/index.html**).

NOTE: When you are referring Oracle documentation for NSJ7, you must follow the instructions provided for `sh` and `ksh` instead of `csh` and `tcsh`. Instructions for the `setenv` and `unsetenv` commands do not apply to NSJ7.

JREHOME

JREHOME shell variable need not be set for NSJ7. However, if this variable is set, NSJ7 accepts it. If this variable is set, then you must ensure that it points to the correct JDK.

For example, `$ export JREHOME=/`*`install_dir`*`/nssjava/jdk170_h70/jre`.

_RLD_LIB_PATH

The `_RLD_LIB_PATH` environment variable specifies the library path for DLLs. You need to use this environment variable if you use user DLLs. You can specify one or more directories as

required. Separate each directory in the list by using a colon (:). Set this environment variable as follows:

```
export _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

NSJ7 is installed in a default directory. The default directory is unique for each release, and is of the following format:

```
/usr/tandem/nssjava/jdk<x>_h<yy>
```

where,

- `x` refers to the version number of Oracle Java update
- `yy` refers to the product version of NonStop Server for Java

For example, for NSJ7, based on Java SE 1.7.0, the installation directory is:

`/usr/tandem/nssjava/jdk170_h70`.

For ease of reference, a symbolic link `/usr/tandem/java` was being created to point to the installation directory as a part of JDK installation in Java versions released prior to NSJ7.

Since, NSJ7 is installed and managed using DSM/SCM, the installation of NSJ7 does not automatically create the symbolic link (soft link) `/usr/tandem/java` to point to the NSJ7 installation directory for H-series (`/usr/tandem/nssjava/jdk170_h70`). You must create the soft link manually if required.

Configuring TCP/IP and DNS for RMI

TCP/IP and its components along with DNS must be configured properly. This has to be done for Remote Method Invocation (RMI) to function.

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/nssjava/jdk170_h70/install` directory for H-series. Execute `javachk` in the same environment as JVM (that is, using the same defines that were used to run the JVM). The `javachk` identifies failing socket routine calls. When you know which calls are failing, you can troubleshoot the problems.

For information about `javachk`, see file `/usr/tandem/nssjava/jdk170_h70/install/README_javachk` for H-series.

Memory considerations: Moving QIO to KSEG2

In NSJ7, the memory for Java heap and JVM memory is allocated from flat segments. The effects of this memory allocation for 32–bit and 64–bit JDKs are as follows:

- For 32–bit application the flat segments are allocated from the end of the user allocatable space. Hence the QIO segment must be moved to KSEG2 region if the memory requirement of the Java process cannot be met with the available user memory. If the application gets out of memory error, QIO segment must be moved to KSEG2 region.
- For 64–bit application the flat segments are allocated from the 64–bit space and hence QIO segment need not be moved to KSEG2 region.

Java server-side applications are typically configured with large Java heap sizes, larger than 128 MB. In addition, the JVM and its native components (for example, 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 is 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 is reserved for the QIO segment, if the process is also a QIO client.

To overcome the problem of losing user 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 default size of the QIO segment is reduced to 256 MB (instead of the default 512 MB in user address space). However, the maximum possible QIO segment size in KSEG2 region is 512 MB and so the QIO segment in KSEG2 region can be increased to 512 MB (same as in user space). For more information about configuring QIO segment, see *QIO Configuration and Management Manual*.

Example: Determining the QIO segment size in use

To determine the QIO segment size usage, 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........... KSEG2
Segment Size........... 268435456
```

```
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`) are moved to KSEG2 based on the value of the segment type.
- The value is KSEG2 if QIO is moved to KSEG2.
- The first SCF output for `\NAVAE1` shows QIO to be in FLAT_UA, which means that QIO is not moved to KSEG2.

NSJ7 directory structure

This section provides information about the directory structure of NSJ7 and it explains the following topics:

- "Directory contents" (page 32)
- "Demonstration programs" (page 33)

Directory contents

For H-series, the directory `/usr/tandem/nssjava/jdk170_h70` contains release documents and subdirectories contains release documents and subdirectories. Table 3 (page 32) lists the subdirectories and describes their contents.

Table 3 Subdirectories of the `/usr/tandem/nssjava/jdk170_h70` and `/usr/tandem/nssjava/jdk170_l70` Directories

Subdirectory	Contents
`/bin`	Executable 32–bit and 64–bit binary files that are present in 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 Java Virtual Machine Interface.
`/install`	The `javachk` file. Only 32–bit version of `javachk` is available.
`/jre`	The root directory of the Java SE Runtime Environment of 32–bit and 64–bit NSJ7. Includes the Java classes supplied by Hewlett Packard Enterprise, core Java classes, and runtime libraries for 32–bit and 64–bit NSJ7. The core Java classes are in the `lib/rt.jar` file.
`/lib`	Includes the classes other than core classes to support the tools and utilities bundled in the JDK software.

NOTE:

- For each 32–bit binary or library found in NSJ JDK/JRE a corresponding 64–bit version of binary or library is provided in NSJ7 JDK/JRE.
- In NSJ7, the `libhpi.so` library is removed and the functionality provided by `libhpi.so` is moved to `libjvm.so` library.

Demonstration programs

The `/demo` directory contains subdirectories, each of which contains a demonstration program and a README file that documents how the demonstration program must be used. Table 4 (page 33) lists the demonstrations programs available in NSJ7:

Table 4 Demonstration programs

Demonstration Program	Description
`/demo/invocation_api/oss`	How to properly build an executable that can create its own Java virtual machine (JVM). Contains source and Makefile to create 32–bit invocation API demo.
`/demo/invocation_api/oss64`	How to properly build an executable that can create its own Java virtual machine (JVM). Contains source and Makefile to create 64–bit invocation API demo.
`/demo/javahjni/oss`	How to create a native library. Contains source and Makefile to create 32–bit `javahjni` demo.
`/demo/javahjni/oss64`	How to create a native library and how to use IEEE floating-point. Contains source and Makefile to create 64–bit `javahjni` demo.

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

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

3 Getting started

Although this manual assumes that you are familiar with using Java and HPE NonStop Open System Services (OSS), this section provides background information for those who are 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:

Tutorial: Running a simple program, HelloWorld

After NSJ7 is installed, use the following steps to create and run the HelloWorld program.

1. **Create a Java source file.**

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

   a. Using an 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 PATH environment variable.**

   Add the directory where the NSJ7 executable file is installed to your PATH environment variable.

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

```
$export PATH=/usr/tandem/nssjava/jdk170_h70/bin/:$PATH
```

   If the NSJ7 is installed in a nonstandard location, `/home/lee`, type:

```
$export PATH=/home/lee/nssjava/jdk170_h70/bin/:$PATH
```

3. **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 must 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`.

4. **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 NSJ7 product. Type the following command on 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 the errors, and perform the steps c and d until class file is created.

5. **Run the program by using the Java tool.**
   a. Go to the directory where your class file is created.
   b. To run the HelloWorld program (also called an application), type the following command at the OSS prompt:

      ```
      $java HelloWorld
      ```

      NOTE: You must 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

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.

Use the following command to specify the CPU where a Java application is to run:

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

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

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

Use the following command to provide a `java` process a process name:

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

For example, the command to provide 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 *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 on using `-name` option with 32–bit JDK and 64–bit JDK, see “Java process name” (page 107).

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

Configuring a Java Pathway serverclass

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 2.5 System Management Manual*.

The serverclass attributes that have specific Java requirements follow. Typically, the attribute settings are available in a configuration file, but the examples show how to set the attributes in the OSS environment.

ARGLIST

The ARGLIST must be in the form:

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

where, `[,arguments]` is an optional, comma-separated list of arguments that are 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: 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. For a `CLASSPATH` example, the following set server command is entered at the PATHCOM prompt.

```
$gtacl -p pathcom \$foo
PATHCOM .....
=set server ENV CLASSPATH=/myclasspath
```

where, `/myclasspath` is the location where all the classes are available.

NOTE: JREHOME shell variable need not be set for NSJ7. However, if this variable is set, NSJ7 will accept it. If this variable is set, you must ensure that it points to the correct JDK.

PROGRAM

Set the PROGRAM attribute to the `java` executable. For H-series, the `java` executable is located by default in `/usr/tandem/nssjava/jdk170_h70/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/nssjava/jdk170_h70/bin
```

4 Implementation specifics

This section explains the various topics pertaining to the implementation specifics of NonStop Server for Java:

See also, “Application tuning and profiling” (page 94).

Headless support

Since NonStop operating system does not provide support for GUI operations, NSJ7 is a headless JVM that conforms to the Oracle's headless support standards regarding Java Abstract Window Toolkit (AWT) classes and methods. For similar reasons, NSJ7 does not support the `AppletViewer` tool.

Class data sharing (CDS) is a feature intended to reduce application startup time and footprint, which is available only with a Java HotSpot client VM. Hence this is not supported in NSJ.

When `GraphicsEnvironment.isHeadless` returns true, and if your Java programs use classes and methods that require a display, keyboard, sound, or mouse operation, the class or method throws a `HeadlessException`. This value is always true in NSJ7.

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

While the Oracle 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 not required for NSJ7.

Additional files

In addition to the standard Java packages, NSJ7 provides the following files:

- `jtatmf.jar`

  File containing classes for the version of NonStop Java Transaction Agent that uses TMF. For more information, see “Transactions” (page 90).

- `javachk`

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

Additional environment variable

NSJ7 contains 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 of transactions allowed are 200. For more information, see “Controlling maximum concurrent transactions” (page 90).

Java native interface (JNI)

The Oracle Java Native Interface (JNI) standard defines the C language APIs that enable Java routines to call C and C++ routines. JNI defines the way that these routines can start and interact with a Java Virtual Machine (JVM). It also allows you to load the JVM into native applications to integrate Java functionality with native applications.

NSJ7 supports JNI and Invocation API with the following modifications:

- Set the _RLD_LIB_PATH environment variable to point to the User DLL location.

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

  where, `dll-path` and `dll-pathn` are the User DLL directories.

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

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

- NSJ7 supports only the POSIX User Thread library (PUT). As a result of this, additional migration considerations arise while migrating threaded native code to NSJ7. There are two kinds of native applications that require changes:
  1. Applications that dynamically load `libjvm.so` and invoke JVM using Invocation APIs. For more information, see “Calling Java methods from C or C++” (page 41).
  2. Applications that use Pthread library to create multi-threaded application. For more information, see “Threading considerations for native code” (page 48).

- If native C or C++ routines invoke Transaction Management Facility (TMF) calls, you must use TMF transaction jacket routines defined in “TMF Transaction Jacket Routines” in the

"Using the POSIX User Thread (PUT) Model Library" section in the *Open System Services Programmer's Guide*. The calls are:

- `PUT_ABORTTRANSACTION()`
- `PUT_BEGINTRANSACTION()`
- `PUT_ENDTRANSACTION()`
- `PUT_RESUMETRANSACTION()`

NOTE: OSS supports a maximum of 1000 concurrent transactions in a process. However, JVM is an exception where the maximum transactions allowed are 200 as described in "Controlling maximum concurrent transactions" (page 90).

- NSJ7 performs no conversion when calling a C or C++ routine where the function passes or returns parameters of type `float` or `double`. All `float` and `double` values remain in IEEE floating-point format when crossing the JNI boundary. For more information, see "Floating-point implementation" (page 43).
- When using the `JNI_OnLoad` function, use the following format:

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

- The `JNI_OnUnload` function is supported.
- Conform to the following rules when naming user DLLs:
  - Do not use names that begin with `Tandem`, `tandem`, or `tdm`.
  - NSJ7 requires that all DLLs be named with a prefix `lib` and a suffix `.so`. Hence you must name your DLL as follows:

    `libname.so`

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

The remainder of this subsection explains:

- "Calling C or C++ routines from Java" (page 40)
- "Calling Java methods from C or C++" (page 41)
- "Linker and compiler options" (page 42)

For more information about JNI, see **Oracle JNI document**.

Calling C or C++ routines from Java

To call C or C++ routines from Java, conform to the following 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 compiler command-line options as explained in "Linker and compiler options" (page 42).

   For example:

```
/usr/bin/c89 -g -I /usr/tandem/nssjava/jdk170_h70/include -I /usr/tandem/nssjava/jdk170_h70/include/oss
-I. -I /G/system/system -Wallow_cplusplus_comments -Wlp64 -Wextensions -D_XOPEN_SOURCE_EXTENDED=1
-Wnowarn=141,209 -Wcall_shared -Wsystype=oss -Wtarget=tns/e -c SystemInfo.c
```

NOTE: If the native code has large variables on the stack, then 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, then it results SIGSTK.

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 7.0 Tools Reference Pages*.

4. Create a DLL file (`.so` file type) and specify the linker option as explained in "Linker and compiler options" (page 42).

   After specifying the linker option, 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 example:

   ```
   /usr/bin/eld TandemWrap.o SystemInfo.o -o libCompanySysInfo.so -set SYSTYPE OSS -set HIGHPIN ON -set
   data_model lp64 -set CPlusPlusDialect neutral -dll -lcre -lcrtl -export_all
   ```

   For more information, see "_RLD_LIB_PATH" (page 29).

NOTE: Using *neutral* data type DLL may not work with 64–bit JDK as JVM pass true 64–bit pointers that is not sign-extended value of a 32–bit pointer.

The `javahjni` demo shows an example of how to create a library file, see Table 4 (page 33) to view the directory location of the demo programs. The demo also shows the conversion between the TNS and IEEE floating-point.

Calling Java methods from C or C++

You can create your own C or C++ program. Follow these guidelines to use the Invocation API to load the JVM in an arbitrary native program:

- NSJ7 supports only the POSIX User Thread library. As a result, the following issues must be considered when migrating to NSJ7:
  - Java application which uses the Invocation API must necessarily link with the POSIX User Thread library. An application must link with either `ZPUTDLL` or `YPUTDLL` depending on whether it is a 32–bit or 64–bit application.

    Set the `_RLD_LIB_PATH` environment variable to the location of `libjvm.so` as shown:

    For 32–bit NSJ7: `$ export _RLD_LIB_PATH=$JREHOME/lib/oss/server/`

    For 64–bit NSJ7: `$ export _RLD_LIB_PATH=$JREHOME/lib/oss64/server/`

- A native application can use the `JNI_CreateJavaVM()` call to become a Java process, and such applications are called JNI applications. The native application can be either multi-threaded or single threaded. If an application calls the `JNI_CreateJavaVM()` on the main thread, then the main thread becomes the Java thread after creating the virtual machine. If an application already contains a VM, another thread can become a Java thread by invoking `AttachCurrentThread()`. In either case, if the calling thread is the main thread and if distributed GC is enabled, the main thread stack must be shared by GC processes. As the main thread stack cannot be shared across GC processes, the distributed GC cannot be enabled when the main thread is the Java thread. Hence, `JNI_CreateJavaVM()` and

`AttachCurrentThread()` return JNI_EUNSUPPORTED error if they are called from the main thread with distributed GC enabled.

NOTE: All the instances of distributed GC in this manual refer to parallel and CMS GC.

- Compile code written in C++ by using the compiler command-line options as explained in “Linker and compiler options” (page 42).
- NSJ7 provides DLLs. Therefore, you can build your own executable and link it to the JVM DLL, `libjvm.so`.

  For more information, see `invocation_api` demo provided with the NSJ7 installation.
- Do not set signal handlers for the following signals:
  - `SIGSEGV`
  - `SIGPIPE`
  - `SIGCHLD`
  - `SIGINT`
  - `SIGQUIT`
  - `SIGTERM`
  - `SIGHUP`
  - `SIGWINCH`
  - `SIGALRM`
  - `SIGSTK`
  - `SIGILL`
  - `SIGTIMEOUT`
- Set the executable to use IEEE floating-point.

NSJ7 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 parameters of type `float` or `double` that are in IEEE floating-point format. Any parameters of type `float` or `double` that are passed to NSJ7 must be in IEEE floating-point format. If such a program requires conversion 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 in the README file in directory `/usr/tandem/nssjava/jdk170_h70/demo/invocation_api`. Directory location of `invocation_api` (for 32–bit and 64–bit NSJ7) demo programs are listed in Table 4 (page 33).

Linker and compiler options

Compiler options

You can use C++ code compiled using either a dialect of version 2 or version 3 for user DLLs because NSJ7 is built with a C++ version neutral flag (the `–set CPlusPlusDialect` neutral option for the linker). To compile using version 2, the following compiler option must be used: `–Wversion2`.

–Wversion3 is the default version for TNS/E native compilers, and hence it is not needed to be defined in the compile command-line.

In addition, 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, must use the IEEE floating-point arithmetic. IEEE floating-point format is the default for TNS/E native compilers, and hence it is not needed to be defined in the compile command-line.

Any compilation units not called directly by the JVM can be compiled without conforming to the IEEE floating-point format. However, the complications that can occur while using such mixed modes are beyond the scope of this manual.

The javahjni demo shows an example of mixed modes. For information on demos, see “Demonstration programs” (page 33).

If you want to call the C/C++ routines from 64–bit java applications, then you must compile the routines using the following complier command-line option:

```
-Wlp64
```

Linker options

IEEE floating-point is the default format for TNS/E native compilers, and hence it is not needed to be specified during linking.

When building 64–bit native libraries, the following additional linker option must also be used:

```
-set data_model lp64
```

How to create your own library of native code

The javahjni demonstration program shows how to create a native library for 32–bit and 64–bit NSJ7. In order to use a native library for 32–bit JDK, it must be a 32–bit library, and for use with 64–bit JDK, it must be 64–bit library.

Directory location of javahjni (for 32–bit and 64–bit NSJ7) demo programs are listed in Table 4 (page 33).

Floating-point implementation

Java uses IEEE floating-point arithmetic for computation. This subsection explains the following topics:

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.

NOTE: In NSJ7, you cannot specify whether your Java classes use TNS format.

Floating-point values

For floating-point values, TNS floating-point representations have larger exponents (hence a larger range) than IEEE floating-point representations, but they are less precise, which is demonstrated in Table 5 (page 44):

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 (hence a smaller range) than IEEE floating-point representations, but they are more precise, which is demonstrated in Table 6 (page 44):

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/

Calling TNS floating-point functions from JNI code

This section describes the procedure to call a TNS floating-point function from an IEEE floating-point function.

Perform the following when using TNS floating-point compiled functions in native code linked in 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 the following:
  - A `float` as one of the 32–bit structures defined in `$SYSTEM.SYSTEM.KFPCONVH` (`NSK_float_ieee32`, `NSK_float_tns32`) or `/usr/include/kfpconv.h`.
  - A `double` as one of the 64–bit structures defined in `$SYSTEM.SYSTEM.KFPCONVH` (`NSK_float_ieee64`, `NSK_float_tns64`) or `/usr/include/kfpconv.h`.
- You can call a native function that accepts or returns a TNS `float` or `double` value if you create an intermediate function. The intermediate function assembles between the IEEE floating-point compiled JNI method that the JVM calls and the native function that accepts or returns the `float` or `double` value.

  Either the JNI method or 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 `$SYSTEM.SYSTEM.KFPCONVH` or `/usr/include/kfpconv.h` file.
  - Calls the TNS compiled native function passing TNS `float` or `double` arguments.
  - Converts any `float` or `double` return value to an IEEE floating-point value, which the JNI caller expects.
  - Returns the float or double in one of the special structures defined in the `$SYSTEM.SYSTEM.KFPCONVH` or `/usr/include/kfpconv.h` file.

For an example, see `javahjni` “Demonstration programs” (page 33).

Multi-threaded programming

The Java virtual machine for the NSJ7 is multi-threaded and uses POSIX User Thread (PUT) library, which conforms to the IEEE POSIX Standard 1003.1, 2004. With PUT library enabled on NonStop systems, CPU intensive thread that does not invoke an I/O, sleep, or wait can never be pre-empted by another thread unless `-XX:ThreadTimeSlice` option is used.

This section includes the following topics:

Thread scheduling

When a Java thread is created, the thread inherits its priority from the parent thread. The priority of the thread varies from `MIN_PRIORITY` to `MAX_PRIORITY`, where the default priority is `NORM_PRIORITY`. After a thread is created, the `setPriority` method can be used to alter the priority of the thread.

Java application developers must be aware of the following consideration while designing or enabling multi-threaded Java applications for Nonstop systems:

- A selfish thread is a thread that executes in a tight loop without giving up control either by explicitly yielding or by invoking a blocking operation.
- Since the thread-scheduling algorithm on Nonstop is non-preemptive and does not time slice, it is possible for such a thread to prevent other threads from getting access to CPU time. Theoretically, such a thread can run forever. However, after a while, the operating system periodically reduces the priority of the process in stages, until its priority reaches a very low value.

For a demonstration of scheduling on a NonStop system, review the results and output of the following programs:

Example 5 Snnipet for ThreadTimeSlice feature

```
import java.io.*;
import java.lang.*;

class LoopThread extends Thread {
   private Thread t;
   private int threadNum;
   private static int tick = 1;
   private static volatile int numThreadsStillRunning = 0;
   private static int maxThreads = -1;
   private static volatile int numThreadsCreated = 0;
   public  static void setMaxThreads(int num) { maxThreads = num; }

   LoopThread( int num){
      if ( num <= 0 || num > maxThreads ) {
         System.out.println("Invalid thread number: " + num);
         System.exit(1);
      }
      threadNum = num;
      System.out.println("Creating " + threadNum );
      numThreadsCreated++;
      numThreadsStillRunning++;
   }
   public void run() {
      if ( threadNum == 1 ) {
         // First thread. Execute until other threads terminates
         while ( numThreadsCreated < maxThreads ||
                 numThreadsStillRunning > 1 ) {
            tick++;
         }
      } else {
         // Sleep some time to guarantee first thread runs
         try {
            Thread.sleep(10);
         } catch ( InterruptedException e ) {
            System.out.println("Sleep interrupted.");
         }
         for (int i=0; i < 10000; i++ ) {
             tick++;
         }
      }
      System.out.println("Thread " + threadNum + " exiting.");
      numThreadsStillRunning--;
   }
}

public class ThreadPreemptionDemo {
   public static void main(String args[])
   {
      int numThreadsToCreate = 2;
      // LoopThread[] threads;

      if ( args.length > 0 ) {
         // User has specified number of threads to create
         numThreadsToCreate = Integer.parseInt(args[0]);
         if ( numThreadsToCreate < 2 ) {
            System.out.println("Invalid argument. Minimum value = 2");
            System.exit(1);
         }
      }
      LoopThread[] threads = new LoopThread[numThreadsToCreate];
      LoopThread.setMaxThreads(numThreadsToCreate);
      for (int i = 0; i < numThreadsToCreate; i++) {
         threads[i] = new LoopThread(i+1);
```

```
            threads[i].start();
        }
        try {
            for (int i = 0; i < numThreadsToCreate; i++) {
                threads[i].join();
            }
        } catch (InterruptedException ex) {
            System.out.println("Exception " + ex);
        }
    }
}
```

On the NonStop system, the execution of threads is not time sliced. When a thread gets a chance to run, it enters the loop and starves the other thread. Hence, the application hangs.

Output when thread time slice is not enabled:

```
java -cp . ThreadPreemptionDemo
Creating 1
Creating 2
```

⚠ **WARNING!** You will observe a hang when thread time slice option is not specified. Therefore, “Creating 2” in the output may or may not be printed.

The Java runtime supports a simple, deterministic, scheduling algorithm known as fixed-priority scheduling. The Java runtime does not time-slice. 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. 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.

NSJ7 (SPR IDs T2766H70^ACN and T2866H70^ACN) release includes a 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 gets 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 pre-emptive user threads, use the following option:

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

where,

T specifies the time in milliseconds.

- T is an optional argument.
- The default value of T is 400 milliseconds.
- The value of T can range between 1 to 32767.

  If the specified value of T is above 32767, the value is time-sliced to 32767.

  If the specified value of T is 0, the value is time-sliced to 400.

Following is the output for the snippet provided in Example 5 (page 46):

Output when thread time slice is enabled

```
java -cp . -XX:ThreadTimeSlice=10 ThreadPreemptionDemo
Creating 1
Creating 2
Thread 2 exiting
Thread 1 exiting
```

Threading considerations for Java code

A thread-aware function blocks only the thread calling that function, rather than blocking the entire process. NSJ7 uses the PUT library in which all I/O are nonblocking.

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 *JToolkit for Java API Reference Pages*.

Thread-aware I/O Support

In NSJ7, I/O operators are always nonblocking for OSS files. Earlier versions of NSJ, provided an option `–Dnsk.java.nonblocking` to switch between the blocking and nonblocking I/O operation. This option is no longer supported on NSJ7.

Threading considerations for native code

NSJ7 supports only the POSIX User Thread library. As a result, the following issues must be considered when migrating to NSJ7:

- The JNI or C/C++ code in a Java application that uses the Pthread API must be ported to use the `_PUT_MODEL_` as described in the *Open System Services Programmer's Guide*.
- The PUT library `Z/YPUTDLL` must be linked statically to the application[1]. This impacts JNI applications that currently (pre-NSJ7) load `ZSPTDLL` dynamically.

In NSJ7, the application must be linked with `Z/YPUTDLL`[2] depending on whether it is a 32–bit or 64–bit application.

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 more careful to ensure that the operations performed in the user threads are thread-safe on the NonStop system. All threaded code preferably written in Java rather than native code.

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

- NSJ7 uses the POSIX User Thread Library from the standard location.

  Your code must include the POSIX User Thread header files in the standard location. You must use the header files from the standard location `/usr/include/`.

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

  The NonStop system does not have an implementation of native threads since threads run at a user level. Even on a multiprocessor NonStop system, all native threads (user threads) in a process are executed in the same processor as the process. If you create user threads whose only purpose is to run a certain task, the thread creation overhead makes the task run 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 discussion of “Thread scheduling” (page 45).

- In a very long-running, CPU-intensive thread, having your native code occasionally invokes the `yield()` method which allows timer and I/O completions to be detected. Invocation of

1. For L15.02, applications must be linked with `X/WPUTDLL`.
2. For L15.02, applications must be linked with `X/WPUTDLL`.

timer callbacks and detection of I/O completions can be severely impacted by long-running threads.

- You must be careful when using thread-aware interfaces.

  The *Open System Services Programmer's Guide* lists thread-aware equivalents of NonStop system-specific interfaces. These interfaces have an explicit `put_` 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).

Pthread library changes

Pthreads library provides the following functionality:

- The PUT library converts all I/O calls as nonblocking calls. If a thread invokes an I/O, the call blocks only the calling thread and not the application. This enables porting of Java applications to NSJ7.
- The PUT library provides features that allow Java applications to be more robust are described as follows:
  - Protected thread stack – In earlier versions of NSJ, the Java thread stack was unprotected and hence the JVM only detected stack overflow caused by Java methods. This was done by stack boundary checking code in interpreted and hotspot compiled code. It was possible for native methods to overflow the stack space erroneously and corrupt the program data. In NSJ7, the PUT library protects the thread stacks by using guard pages. JVM uses this Pthread feature to catch stack overflow from native methods. A stack overflow from native method causes a `SIGSTK` signal.
  - Signal handler uses new stack called alternate signal stack - This enables JVM to execute the signal handler code for `SIGSTK` signal.
- Provides standard POSIX thread API names and calling sequences which adhere to IEEE standard 1003.1, 2004. For more information, see *Open System Services Programmer's Guide*.

For more information on Pthread library, see *Open System Services Porting Guide*.

ThreadDumpPath support

The thread dump lists all threads in the Java process. If any of the distributed GC options are enabled, GC threads are converted into processes. Hence, the GC threads are prefixed with `*` to indicate that it is a process if the application enables distributed GC.

The execution stack trace of all Java threads in a NSJ7 process can be dumped by sending a SIGQUIT signal to the Java process using the following OSS command:

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

NOTE: GC thread stack trace is not printed.

By default, the thread stack dump is written in text format on the `stdout`. NSJ7 provides the ability to redirect this output to a user-defined file using the following Java command:

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

Java print service

The Java Print Service (JPS) implemented in NSJ7 uses the headless version of the **javax.print** API. JPS 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 *Spooler FASTP Network Print Processes Manual*. For information on the Spooler subsystem, see *Spooler Utilities Reference Manual*.

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 in JPS. The following are the JPS classes:

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

For applications using the `java.awt.print.PrinterJob` class, the printer must be postscript enabled. For information on enabling postscript printing, see *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.#p255";
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);
..
..
```

NOTE: In the above example, `$s.#p255` is the name of the configured printer. If there is no printer configured with this name, then print output is routed to Spooler with location `#p255`. To print the document, you need to explicitly reroute the document to the configured printer location. For more information, see *Spooler Utilities Reference Manual*.

Dynamic `saveabend` file creation

NOTE: Dynamic saveabend file creation works only with Java processes. It will not work with GC processes.

Dynamic saveabend file creation helps to create a `saveabend` file of a running Java process without aborting (abending) the Java process, and by issuing a signal to the process. This 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, and low responsiveness without impacting the running Java process.

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

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

Saveabend file generation

If distributed GC is enabled, then the `saveabend` files are generated differently. The Java process or GC process can encounter a problem and terminate abnormally. When one process in the Java process group terminates abnormally, then the other processes detect the abnormal termination and also terminate abnormally.

The first process that terminates abnormally dumps the entire user address space (full dump), and the other processes do the partial dump of the address space. To analyze the problem, load the `saveabend` file independently in `eInspect` for analysis. As the partial snapshot files do not contain all the data, the full dump needs to be used to get the data.

⚠ **CAUTION:** The `saveabend` flag must be turned on for both Java launcher executable file and the `javagc` executable file for creation of `saveabend` files for both the Java and GC processes.

You can collect the snapshot files of the Java process and GC process and send it to Hewlett Packard Enterprise support for analysis. You must collect all the saveabend files.

FastExec option support

For applications that use the `Runtime.exec` method to create a child process, NSJ7 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, thus reduces the time required to create a child process.

NOTE: The same result can be achieved by using `-Djdk.lang.Process.launchMechanism="posix_spawn"` option introduced by Oracle, which is supported from NSJAVA H70^ACN onwards.

Java `getLocalHost()` caching

Java `getLocalHost()` has to make a call to the Kernel API to obtain the desired results, which is a time consuming factor. Java caches results only for five seconds that is not useful for applications which run for a longer time.

Starting with T2766H70^ACN or T2866H70^ACN, option `-Dnsk.java.cachegetLocalHost=true` is used to overcome the unnecessary calls made to the Kernel API, and cache the results of `getLocalHost()` until the application ends.

⚠ **WARNING!** Do not change the IP address of the system while running the application.

Java authentication and authorization service

The Java Authentication and Authorization Service (JAAS) is integrated in NSJ7. JAAS augments the core Java 2 platform which facilities to authenticate and enforce access controls. JAAS 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 plug ability permits applications to remain independent from underlying authentication technologies. New or updated authentication technologies can be plugged in without requiring modifications to the application. Applications enable the authentication process

by instantiating a `LoginContext` object, which in turn references a configuration to determine the authentication technology, or `LoginModule`, that is to be used in performing the authentication. The `LoginModule` interface provides developers the ability to implement different kinds of authentication technologies that can be plugged in an application.

For example, one type of `LoginModule` might perform a username and password based form of authentication. Other `LoginModules` might involve more sophisticated authentication mechanisms.

NSJ7 product includes `LoginModule` interfaces implemented by Oracle, 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 ***JAASLoginModule Developer's Guide***.

JavaBeans

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

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

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

Because of the CLI nature of the NonStop operating system, NSJ7 supports only runtime execution of invisible JavaBeans. The NSJ7 does not support design-time execution or runtime execution that requires a GUI operation. In NSJ7, the Boolean expression `java.beans.Beans isGuiAvailable` returns the value `false`.

For more information about JavaBeans, see **Oracle JavaBeans document**.

Debugging Java programs

This subsection discusses the debugger architecture and how to run the programs involved in debugging Java applications, and also discusses the following topics:

Debugging overview

NSJ7 supports Java Platform Debugger Architecture (JPDA) that provides debugging support for the Java platform. For more information, see Oracle's **Java Platform Debugger Architecture Documentation**.

NSJ7 ships a terminal oriented non-GUI debugging tool. This Java Debugger Tool (JDB) can be used on a NonStop system 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 NSJ7 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. NSJ7 provides a socket transport that uses the standard TCP/IP sockets to communicate between the debugger and debuggee.

By default, NSJ7 uses socket transport. NSJ7 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
-agentlib:jdwp=transport
```

`-Xdebug`

Enables debugging.

`-Xnoagent`

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

`-agentlib:jdwp=sub-options`

`sub-options` are specified in the following format:
`name1[=value1],name2=[=value2]...`

The *`sub-options`* are as follows:

`transport`

Indicates the name of the transport. The value is `dt_socket`, and by default, NSJ7 uses `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 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, by using only the interpreter and not the HotSpot compiler.

NOTE: Specifying a range of port numbers for `address` is specific to NonStop.

The subsequent examples show the various ways in which the connection address is specified:

Example 6 When the port number is specified

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

Example 7 When the range of ports are specified and hostname is implicit

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

Example 8 When the machine name and port range are specified

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

Starting the Java debugger (JDB) tool

The Java Debugger (`JDB`) tool can be started so that it communicates with the debuggee by using the `jdb` command as described for the following situations:

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

  ```
  jdb -attach portnum
  ```

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

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

- If you are using JDB from Microsoft Windows or any other platform, use the following command:

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

Additional 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 NSJ7 applications are tested with Eclipse 3.7.2 of the Eclipse Project. For information and software downloads, see **http://www.eclipse.org/**

For more details on command-line options, see **Connection and Invocation Details**

Debugging JNI code

Use the inspect debugger tool available on the NonStop system to debug the native code. This native code is written by application developer, and is used with the Java program. Use Visual Inspect (the preferred debugger) or Native Inspect to debug the native code. For more information, see *Native Inspect Manual*.

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

```
run -debug java java_options
```

To debug native code, wait until the DLL is loaded. Visual Inspect allows you to stop the program after the DLL is loaded so that you can set breakpoints. For information on how to set the breakpoints, see *Native Inspect Manual*.

You can view and debug the native routine to be debugged and other native routines that routine calls. All other scopes above the native routine are either compiled or interpreted Java code, which the inspect debugger has no knowledge about. For more information on debugging, see “Debugging Java process” (page 107).

Using visual inspect to add an event breakpoint on DLL open event

Since 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.

Perform the following to add an event breakpoint on DLL open event:

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

Debugging Java and JNI code

You can use the Native Inspect debugger tool to debug the native code and Java Debugger tool to debug the Java code simultaneously. You need to start the Java debuggee process in a debugger.

For example, type the following command:

```
run -debug java -Xdebug -Xnoagent -agentlib:jdwp=sub-options
```

You can then use the Java Debugger tool to communicate with the debuggee process as explained in “Debugging overview” (page 52).

Deviations in JVM specification options

The compiler specification options for both the `java` and `jdb` tools deviate from standard Java because NSJ7 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 NSJ7.

`-server`

Selects the Java HotSpot Server virtual machine (VM).

NOTE: The `-server` is the default option for NSJ7. Hence, specifying `-server` is optional.

For more information about the `java` tool and additional deviations from standard Java, see “Implementation of garbage collector types” (page 59) and `java` in the *NonStop Java 7.0 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 NSJ7.

`-tserver`

Runs the application in the Java HotSpot server VM.

NOTE: The `-tserver` is the default option for NSJ7. Hence, specifying `-tserver` 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 7.0 Tools Reference Pages*.

Garbage collection (GC)

This section discusses the following implementation-specific topics about garbage collection for NSJ7:

General information on garbage collection

In general, the various GC algorithms, and modeling in NSJ7 are same as those implemented by Oracle in their JVM. Accordingly, you must refer to the Java web site for details about garbage collection. However, some of the information is not applicable to NSJ7. For more relevant information, see the following links on Java web site:

- **Java SE 7 HotSpot Virtual Machine Garbage Collection Tuning**
- **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.

  Although, this paper was written for JDK1.4.1, it contains information related to tuning application behavior using data in verbose GC logs, which is applicable to JDK 7.0.

Heap Layout

The subsequent sections provide an overview about heap layouts managed in Parallel, CMS GC, and G1GC.

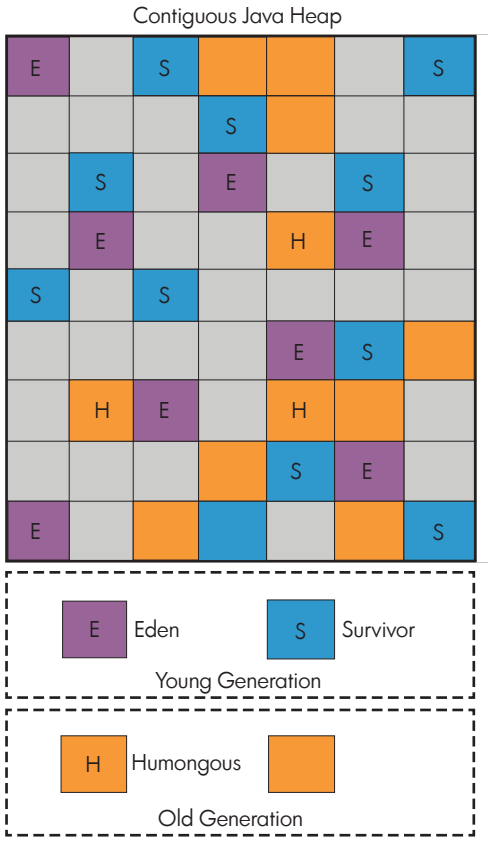
Heap layout for G1GC

Heap layout for G1GC is organized in a different way — G1GC moves away from a contiguous heap wherein the heap layout used to split into young and old generations. G1GC introduces a concept of regions:

- Java heap space is divided into fixed size regions.
- Size of the region can span from 1 MB to 32 MB depending upon the total heap size.
- The main goal of G1GC is to split the total heap into approximately 2048 regions.
- G1GC tries to reclaim regions with least amount of live data and free regions are assigned to new or old generations.

Figure 2 (page 57) illustrates the heap layout for regions:

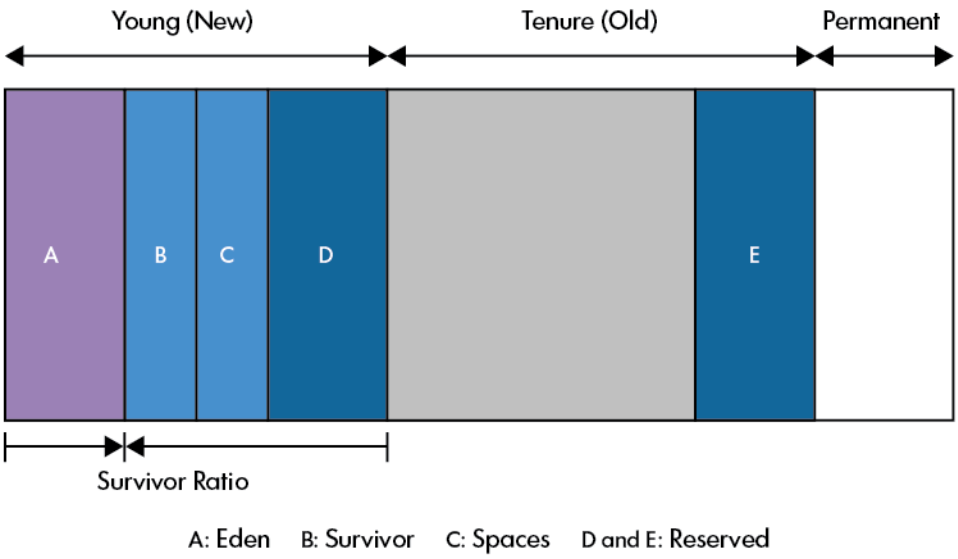
Figure 2 Heap layout for regions



Heap layout for Parallel and CMS GC

In NSJ7, the memory is managed in generations (or memory pools) based on objects at different ages for Java objects. Figure 3 (page 58) illustrates the heap layout for generations.

Figure 3 Layout for generations



The following list describes the various generations:

- 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 for the next GC operation, and which also copies the collection of any live objects in the Eden and 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 the initialization and is used when garbage collection does not have sufficient space to satisfy the allocation request. In the Oracle implementation, the address range for reserved area is reserved but memory space is not allocated until it is used. Whereas, in NSJ7 implementation, the address range for reserved area is reserved and memory space is allocated during the initialization.

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 64 MB for 32–bit NSJ7, and 1024 MB for 64–bit NSJ7. Smaller values lead to shorter but more frequent garbage collections, whereas larger values lead to longer but less frequent garbage collections. For large server applications, Hewlett Packard Enterprise 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 MB for 32–bit NSJ7, and 1024 MB for 64–bit NSJ7.

With 64–bit NSJ7, `-Xmx` can be set to values greater than 1276 MB (that is around 1.24 GB, which is the maximum heap size for 32–bit NSJ7).

NOTE: You must consider the amount of physical memory available on the system while choosing a value for `–Xmx`. If the virtual memory of a process exceeds the physical memory, then memory thrashing occurs and the application performance slows down.

NOTE: Hewlett Packard Enterprise recommends that the application specifies the same value for `–Xms` and `–Xmx`. If only `–Xmx` is specified in the command-line, the default value used for `–Xms` is the same value that is specified for `–Xmx`.

`-XX:MaxPermSize`

Sets the maximum size for the permanent generation. The default value for `MaxPermSize` is 64 MB for both 32–bit and 64–bit NSJ7. 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 maximum size of permanent generation are allocated.

Implementation of garbage collector types

The available garbage collectors in NSJ7 are as follows:

- Serial collector (default collector for young and tenured generations)
- Parallel collector
- Parallel compacting collector
- Concurrent Mark-Sweep collector (or Concurrent low-pause collector)
- G1GC

⚠ **WARNING!** When `PUT_PROTECTED_STACK_DISABLE` is set, the behavior of all implementation of Garbage Collector types is undefined except Serial Collector.

Table 7 Summary of garbage collector implementations

Collector Type	Implementation Status
Serial collector	Default collector for the young and tenured generation
“Parallel collector” (page 59)	Allowed
“Concurrent Low-Pause collector” (page 60)	Allowed

Parallel collector

The parallel collector (or throughput collector) uses the parallel version of the young generation collector. The parallel collector is available only for J-series systems.

NOTE: To enable the appropriate GC, use the `"+"` option. To disable the appropriate GC, use the `"–"` option.

The following are the valid `java` command options, that specify or apply to a parallel collector:

`-XX:+UseParallelGC`

This option enables parallel scavenge for the young generation.

NOTE: The default GC is the serial GC.

`-XX:+UseParNewGC`

This option enables new parallel scavenge for the young generation. The difference between parallel GC and ParNewGC is that `ParNewGC` is optimized for use with CMS GC. This is the default scavenge GC if CMS GC is specified.

`-XX:+UseParallelOldGC`

This option enables parallel GC for the old generation. This option automatically enables parallel scavenge (for young generation).

`-XX:+UseParallelOldGCCompacting`

This option is used to enable parallel old compacting collector for the tenured generation.

`-XX:+UseAdaptiveSizePolicy`

This option enables adaptive size policy for the Java heap. By default, this option is disabled in NSJ7. However, if the parallel GC is enabled, this option is enabled automatically unless it is disabled it in the command-line.

NOTE: `AdaptiveSizePolicy` on NSJ7 considers only the throughput and pause time goal. The footprint goal is not considered.

`-XX:+AggressiveHeap`

Obtains the platform resources and configures the heap layout accordingly, uses parallel collector, and enables `AdaptiveSizePolicy` option.
`-XX:+AggressiveHeap` option is not supported in NSJ7. However, 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 for a parallel collector are supported:
`-XX:MaxGCPauseMillis=nnn`

`-XX:GCTimeRatio=nnn`

These specify performance goals for the application.

`-XX:ParallelGCThreads=<n>`

This option is used to specify the number of parallel GC threads for the parallel garbage collection. The default number of GC threads created is equal to the number of cores present in the processor. If a value greater than the number of cores is specified, the following warning is displayed and the application continues to run:

⚠ **WARNING!** Parallel GC thread count (`<n>`) cannot be greater than the number of cores (`<numcore>`) in a CPU. Using `<numcore>` for Parallel GC count.

Concurrent Low-Pause collector

A concurrent low-pause collector can be selected for tenured generation. This collector does most of the collection concurrently with the application execution. This collector divides the

collection in different phases and operates specific phases concurrently with the application execution, and others in `stop the world` mode.

The following `java` command options that specify a concurrent low-pause collector are valid for NSJ7:

`-XX:+UseConcMarkSweepGC`

This option enables concurrent mark sweep GC for the old generation. Note that the default GC is the serial GC for old generation, and hence this option must be specified to enable concurrent mark sweep GC. By default, if this is selected, `UseParNewGC` is selected for the young generation.

`-Xconcgc`

This option enables concurrent mark sweep GC.

`-XX:+CMSIncrementalMode`

This option enables running CMS GC for the old generation in the incremental mode. Each time the GC is run, only the portion of the tenured generation is collected. `—XX:+UseConcMarkSweepGC` must be specified to use this option.

`-XX:ParallelMarkingThreads=<n>`

This option is used to specify the number of parallel marking threads for the CMS garbage collection. The default value for this option is zero and this value cannot be changed on the command-line.

If a value greater than zero is specified on the command-line, the following warning message is displayed, and the application continues to run.

⚠ **WARNING!**

```
Java HotSpot(TM) Server VM warning:
Setting ConcGCThreads/
ParallelMarkingThreads/
ParallelCMSThreads greater than 1 is not allowed on this platform.
Setting it equal to default value that is zero.
```

`-XX:ParallelCMSThreads=<n>`

This option is used to specify the number of parallel CMS threads for the CMS garbage collection. The default value of this option is zero and this value cannot be changed on the command-line.

If a value greater than zero is specified on the command-line, the following warning message is displayed, and the application continues to run.

⚠ **WARNING!**

```
Java HotSpot(TM) Server VM warning:
Setting ConcGCThreads/
ParallelMarkingThreads/
ParallelCMSThreads greater than 1 is not allowed on this platform.

Setting it equal to default value that is zero.
```

G1GC collector

From NSJ7 SPR T2766H70^ACN or T2866H70^ACN, Oracle command line options support G1GC. The options are as follows:

`-XX:+UseG1GC`

This option enables G1GC. This option is enabled only on J06.19 RVU only. If it is used on L15.02 and H-series RVU, JVM displays an error message and exits.

`-XX:G1ReservePercent=<n>`

This option specifies the size of the heap to be reserved for reducing the possibility of promotion failure. The default value is 10 percent.

`-XX:G1HeapRegionSize=<n>`

This option specifies the size of the individual region. The minimum size of the region is 1 MB and the maximum size is 32 MB. The default size is determined ergonomically.

`-XX:ConcGCThreads=<n>`

This option specifies the number of threads that are to be used for concurrent phase of the garbage collector. The allowed value is 1.

Handling large GC log files

From NSJ7 SPR T2766H70^ACJ or T2866H70^ACJ 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.

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

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

Other CMS/Parallel GC options

In earlier versions of NSJ, JVM does not throw an error message if some of the parallel and CMS GC related options are used. JVM ignores these options since parallel and CMS collectors are not available. However, with NSJ7, these options are accepted by JVM since the above-mentioned collectors are available. For a complete list of the options, see **Oracle Java documentation**.

Memory management considerations

This section provides information about memory management considerations for both 32–bit and 64–bit JDK7.

JVM data segment for 32–bit JDK7

In 32–bit NSJ7, JDK/JRE allocates one or more flat segments (up to a maximum of six starting from segment ID 1069 to 1074) for its internal dynamic memory requirement. These segments are referred to as JVM data segments.

In earlier releases, during JVM initialization, the first JVM heap segment (1069) is allocated with a segment size of 256 MB. When this segment is filled, the next available segment is allocated with a size of 256 MB or less, depending on the virtual memory available during the time of allocation. This process continues until all six segments or the virtual memory is filled. Since the first segment size is 256 MB and there is no option to change the size, complete 256 MB is reserved for JVM irrespective of whether the JVM uses the complete virtual memory (256 MB). This space cannot be used for other purposes such as Java heap, native application heap, and

so on. Therefore, the maximum Java heap size available for an application is reduced by 256 MB from 1.3 GB on NSJ6 and in NSJ7 it is reduced to 1084 MB.

NSJ7 SPR T2766H70^ACJ provides a feature to overcome the limitation of earlier NSJ7 JDK/JRE as explained in the preceded section. This feature frees up 192 MB of virtual memory for the process to use Java heap, native application heap, and so on compared to earlier version of NSJ7. Additionally, a command line option, `MaxJVMHeapSize`, is provided to customize the JVM heap size depending on the application requirement. The syntax for this option is as follows:

```
–XX: MaxJVMHeapSize=<size>
```

```
where, size can vary from 64 MB to 1.5 GB.
```

Default value of `MaxJVMHeapSize` is 1.5 GB.

Using this command line option, application can customize the amount of virtual memory utilized by JVM heap segments. Also from NSJ7 SPR T2766H70^ACJ, the first segment size is always 64 MB. The remaining segments sizes vary depending on the `MaxJVMHeapSize`. If the `MaxJVMHeapsize` is 1.5 GB, which is the default value, the remaining five segments varies from 128 MB to 512 MB. It must be noted that in earlier release, the maximum size of the JVM heap segment is always 256 MB. However, from NSJ7 SPR T2766H70^ACJ, the maximum size of the JVM heap segment is not a fixed value and it depends upon the `MaxJVMHeapSize`.

JVM always rounds-off the value of the segment size to the next higher multiple of 4 MB. It must be noted that JVM allocates the complete memory specified in `MaxJVMHeapSize` only on availability of memory. Hence, `MaxJVMHeapSize` may not be honored always.

Sample Invocation:

```
java –XX:MaxJVMHeapSize=256M -version
```

In the sample command invocation, the first JVM segment (segment ID 1069) size is 64 MB. Then the remaining 192 MB memory is divided (calculated dynamically) among the next five segments whose segment ID range is 1070–1074.

JVM data segment for 64–bit JDK7

64–bit NSJ7 JDK/JRE allocates one extensible flat segment for its internal dynamic memory requirement. The initial size of this segment is 512 MB, and the maximum size is 12 GB. This segment is referred as JVM data segment.

When the allocated portion of the segment is filled, and if the allocated size of the segment has not reached the maximum value of 12 GB, then the segment is resized by increasing its size by multiples of 512 MB. To avoid segment resize during the application runtime NSJ7 SPR T2866H70^ACJ provides an option `InitialJVMHeapSize`, that can be used to specify the maximum JVM heap size required by the application. If this option is specified during JVM initialization, the JVM heap segment is resized with the value specified and it avoids the resize of the JVM heap segment later when the application runs. It must be noted that the value specified with the command line option is a suggestive value. If the application needs more virtual memory, the segment can be resized until it reaches 12 GB. The syntax for this option is as follows:

```
–XX:InitialJVMHeapSize=<size>
```

```
where, size can vary from 512 MB to 12 GB.
```

Default value of `InitialJVMHeapSize` is 512 MB.

Sample Invocation:

```
java –d64 XX:InitialJVMHeapSize=4G <className>
```

In the sample command invocation, during JVM initialization, the JVM Heap segment is resized to 4 GB.

NOTE: Minimum size of the JVM heap segment cannot be less than 512 MB.

Java heap size with 32–bit JDK7

As a result of the JVM data segment(s) allocation as described in “JVM data segment for 32–bit JDK7” (page 62), maximum Java heap available for a 32–bit Java application is 1276 MB.

With earlier versions of NSJ7, the maximum Java heap that can be allocated is 1084 MB.

Java heap size with 64–bit JDK7

The default initial and maximum Java heap size for 64–bit NSJ7 is 1 GB. The theoretical maximum value of Java heap that can be specified is 484 GB. However, the actual maximum Java heap size is limited by the available Kernel-managed swap space.

Native heap size with 32–bit JDK7

As a result of the JVM data segment(s) allocation, the size of the native heap available for JNI components is lesser by the sum of the sizes of the JVM data segments and size of Java heap.

Native heap size with 64–bit JDK7

The native heap size is limited to 12 GB. Java does not have command-line option to modify the value of the native heap size. If application requires more than 12 GB native heap size, then it can be modified by using the `eld -change heap_max` option on Java launcher program.

Java garbage collector tuning for application performance

NSJ7 incorporates the HotSpot VM. This section discusses the options available for tuning the JVM, suggests flags and values to tune the Java HotSpot VM, and points to the HotSpot tuning information on the Internet. GC tuning is assisted by GC profile data which can be collected and analyzed as explained in the subsequent sections.

Since GC occurs when generations fill up, the amount of available total heap memory has direct impact on the performance. The parameters that affect the heap are as follows:

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

`-Xmx`: The maximum memory allocation pool

The maximum value for Nonstop system depends on the location of QIO segment when using 32–bit NSJ7. If the segment is moved to KSEG2, the maximum value can be as high as 1276 MB, otherwise, the value might stay around 350 MB. For information on QIO segment location, see “Memory considerations: Moving QIO to KSEG2” (page 30). Also, see “Java heap size with 32–bit JDK7” (page 64).

When using 64–bit NSJ7, the theoretical maximum value is 484 GB. However, the actual limit is constrained by the size of physical memory and swap space available from the Kernel-managed swap space, and the number of processes contending for these resources.

For more information, see “Java heap size with 64–bit JDK7” (page 64).

For larger server applications, the default values of the two options listed are usually inadequate. In general, you must assign as much memory to the JVM.

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 as follows:

`-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 Figure 3 (page 58)).

For example,

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

These options notify the JVM to set:

- The initial size of the heap to 512 MB.
- The maximum heap size to 512 MB.
- The new generation to 256 MB (128 MB belongs to Eden and 2x64 MB survivor).
- The old generation to 256 MB.

For more information about all these parameters, see **Java HotSpot VM Options**.

For more information about tuning garbage collection with the Java SE 7 HotSpot virtual machine, and for general performance tuning information, see the following documentation:

- **Java SE 7 HotSpot Virtual Machine Garbage Collection Tuning**.
- **The article Tuning Java I/O Performance**.
- The article, **Turbo-charging Java HotSpot Virtual Machine, v1.4.x to Improve the Performance and Scalability of Application Servers**.
- **Java Performance Documentation**.

NOTE:

- The following options do not apply either for the NSJ7 or NonStop system:

  `-XX:-UseBoundThreads`

  `-XX:+UseAltSigs`

  `-XX:+AllowUserSignalHandlers`

  `-XX:+MaxFDLimit`

  `-XX:-UseLWPSynchronization`

  `-XX:PreBlockSpin=nn`

  `-XX:-UseMPSS`

  `-XX:+UseSpinning`

javagc

`javagc` is a executable file that is used by Java process to launch the GC process on J-series systems. The GC process is a multi-threaded application. It interacts with Java application using the standard NonStop Kernel IPC mechanism. The GC processes are launched on the same processor on which the Java process is running. The maximum number for parallel GC processes is equal to the number of cores in the processor. If CMS GC is enabled, the number of `ParNewGC` processes created will be equal to the number of cores. In addition to these processes, there will be one CMS GC task process. These processes are unnamed processes.

GC processes are bound to separate IPUs for improved performance. The IPU number to which a GC process is bound is the ordinal number of that GC process. For example, if parallel GC is enabled on a 4-core CPU, 4 GC processes are launched. The first GC process created binds to IPU 0, second to IPU 1, third to IPU 2, and fourth to IPU 3.

You must not invoke `javagc` executable file directly, and if it is invoked directly, the process displays the error message:

Internal Error: Cannot invoke directly!!!

NOTE: There are two versions of `javagc` available in NSJ7, one for 32–bit and the other for 64–bit.

GC profiling

NSJ7 supports an Hewlett Packard Enterprise proprietary option, `-Xverbosegc` to capture the Java application's GC activity. The output of this option can be used to view and analyze the detailed statistics in an offline mode with `HPjmeter`. The following `-Xverbosegc` option generates detailed information about the spaces within the Java heap before and after garbage collection:

```
-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 7.0 Tools Reference Pages*.

For more information on application profiling, see “Application tuning and profiling” (page 94).

GC log rotation

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. NSJ7 allows you to control the size and number of the GC log files. The GC log records are written in the specified number of GC log files in a round-robin method. This allows GC data to be archived easily and thus 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 along with `-Xverbosegc`, `-Xloggc`, or zero-preparation `verbosegc`:

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

where,

`M` is a non-negative integer that specifies the number of rotating GC log records per file. A value of 0 specifies unlimited number of GC log records per file.

NOTE: Each GC log record corresponds to a GC event.

`N` is a non-negative integer that specifies the maximum number of rotating GC log files. A value of 0 specifies 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). For example, `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.

Example

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

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

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

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

–XX:+HeapDump and _JAVA_HEAPDUMP environment variable

The -XX:+HeapDump option can be used to examine 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 before starting the Java application.

For example, export _JAVA_HEAPDUMP=1

With the -XX:+HeapDump option enabled, each time the process is sent a SIGQUIT signal, the JVM 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 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, the following three different HeapDump options are available: -XX:+HeapDumpOnCtrlBreak, -XX:+HeapDumpOnOutOfMemoryError, and -XX:+HeapDumpOnly.

Table 8 (page 67) lists the heap dump options.

Table 8 HeapDump options

Option	Trigger	Format	Filename
-XX:+HeapDump	SIGQUIT	ASCII; Set the _JAVA_BINARY_HEAPDUMP environment variable to obtain 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 in a filename with the naming convention java_<pid>.hprof.<millitime>.

If the 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`
- `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`

  It enables dumping the Java heap when the Java process encounters a `java.lang.OutOfMemoryError` exception. The heap dump filename defaults to `java_pid<pid>.hprof` in the current working directory. The option `–XX:HeapDumpPath=file` can be used to specify the heap dump filename 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`

  It enables heap dumps using the SIGWINCH signal (signal 12) with the help of `–XX:+HeapDumpOnly` option or `_JAVA_HEAPDUMP_ONLY` environment variable. This interface is provided to separate the generation of thread and trace information triggered through SIGQUIT from the heap dump information. If the `–XX:+HeapDumpOnly` option is specified or `_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 in the following 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 `_JAVA_BINARY_HEAPDUMP` environment variable. This environment variable can also be used with the `–XX:+HeapDump` option to generate `hprof` binary format with 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 must be taken after the application has finished initializing and has run for a short period.

The second snapshot is 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 collecting snapshots, view them in `HPjmeter`, and perform the following:

- Compare the files using `File>Compare`.

  You must be able to find out the type of objects that are accumulating in the Java heap.
- Select a type of objects by using `Mark to Find`.
- Go back and view one of the snapshots.
- Go to `Metric>Call Graph Tree` and do a `Find`. You must be able to see the context of the object retention.

TIP: When creating heap dumps, running the application with smaller heap sizes results in smaller heap dump files. Smaller heap dump files enable `HPjmeter` analysis to use lesser memory.

ZapInitialHeap option

`ZapInitialHeap` is a Hewlett Packard Enterprise specific option to help improve the speed of garbage collection. By default, this option is set to false.

When this option is set to true, it forces the JVM to map all the virtual memory pages of the Java object heap during the initialization (JVM load) phase. This action significantly reduces the number of page faults that might occur during GC in the later stages of execution, thereby speeding up memory access during GC.

However, using this option increases the application start up time, since the initialization of JVM is slower.

Hewlett Packard Enterprise recommends that a pilot run be conducted before deploying this option in production environment. The objective of the pilot run is to ensure that the increase in the application start-up time is tolerable, and that there is a visible reduction in the application stalls (pause times) during GC. For more information, see “Application tuning and profiling” (page 94).

64–bit process support

NSJ7 allows creation of 64–bit Java processes. A 64–bit Java process can have a large Java heap; the heap size can be greater than 1276 MB, which is the limit for 32–bit Java process. The theoretical maximum Java heap size for a 64–bit Java process is 484 GB. However, the actual limit is determined by the swap space configured, physical memory of the system, and number of processes contending for these resources.

In 64–bit mode, the Java runtime environment performs marginally lower than it does in 32–bit mode. This is due to the increase in the pointer sizes (8 bytes in 64–bit mode, and 4 bytes in 32–bit mode), which results in higher memory usage for the same set of operations compared to 32–bit mode, and hence lower performance.

The following sections describe how a 64–bit Java process can be launched:

- “`-d64` option” (page 69)
- “Using 64–bit `java` launcher” (page 70)

`-d64` option

This option can be used to launch a 64–bit Java process. To use this option, 64–bit JDK must be installed along with the 32–bit JDK.

To launch a 64–bit Java process use the following steps:

1. Set the PATH environment variable to point to the 32–bit launcher using the following command at the OSS prompt:

   ```
   $ export PATH=/home/lee/nssjava/jdk170_h70/bin/:$PATH
   ```

2. Run the application using the 32–bit launcher and –d64 option:

   ```
   java -d64 <application_classname>
   ```

Launching a 64–bit Java process using –d64 option has the following limitations:

- 64–bit process cannot not be launched as a named process. For more information, see “Java process name” (page 107).
- 64–bit process cannot be launched inside the Native Inspect (eInspect) debugger, see “Debugging Java process” (page 107).

  To overcome these limitations, you must launch the 64–bit Java process as described in “Using 64–bit java launcher” (page 70).

  If –d64 option is used with any previous NonStop Java versions, the following error message is displayed on stdout and the application exits:

  ```
  Running a 64-bit JVM is not supported on this platform.
  ```

  If -d64 option is used with 32–bit version of NSJ7, and 64–bit version of NSJ7 is not installed, then the following error message is displayed:

  ```
  This Java instance does not support a 64-bit JVM. Please Install the
  64-bit JDK.
  ```

NOTE: If application uses user library then _RLD_LIB_PATH must point to a 64–bit User Library.

Using 64–bit java launcher

To launch a 64–bit Java process use the following steps:

1. Set the PATH environment variable to point to the 64–bit launcher using the following command at the OSS prompt:

   ```
   $ export PATH=/home/lee/nssjava/jdk170_h70/bin/oss64:$PATH
   ```

2. Run the application using the 64–bit launcher:

   ```
   java <application_classname>
   ```

NOTE: If the 64–bit Java launcher is launched with -d64 option, the option is ignored and it does not performs an exec to create new 64–bit Java process. This behavior differs from the behavior on other platforms in which an exec is run to create new 64–bit Java process.

32–bit process support

It is possible to launch a 32–bit java process using the 64–bit launcher.

To launch a 32–bit Java process use the following steps:

1. Set the PATH environment variable to point to the 64–bit launcher using the following command at the OSS prompt:

   ```
   $ export PATH=/home/lee/nssjava/jdk170_h70/bin/oss64:$PATH
   ```

2. Run the 32–bit process using the 64–bit launcher with the command-line option –d32.

   ```
   java -d32 <application_classname>
   ```

Launching a 32–bit Java process using –d32 option has the following limitations:

- 32–bit process cannot not be launched as a named process. For more information, see “Java process name” (page 107).
- 32–bit process cannot be launched inside the Native Inspect (eInspect) debugger, see “Debugging Java process” (page 107).

To overcome these limitations, you must launch the 32–bit Java process as described:

For H-series `$ export PATH=/home/lee/nssjava/jdk170_h70/bin:$PATH`

`java <application_classname>`

NOTE: If the 32–bit Java launcher is launched with –d32 option on NonStop Java, the option is ignored and it does not performs an exec to create a new 32–bit Java process. This behavior differs from the behavior on other platforms in which an exec is run to create the new 32–bit Java process.

The following table summarizes the effect of using –d32 and –d64 options.

NSJ7 JDK launcher type	Command-line option	Launched Java process
32–bit Launcher	-d32	32–bit Java process (option ignored)
	-d64	64–bit Java process
	Neither –d32 nor –d64 is specified.	32–bit Java process
64–bit Launcher	-d32	32–bit Java process
	-d64	64–bit Java process (option ignored)
	Neither –d32 nor –d64 is specified.	64–bit Java process

Large heap support

If a Java application requires large heap (greater than 1276 MB), 64–bit JDK must be used. After the installation, you can run the application with heap sizes greater than 1276 MB using –Xms and –Xmx command-line options. The following are the examples which invoke an application with 24 GB of Java heap size:

Example 9 JAVA_HOME pointing to 32–bit JDK

```
java –d64 –Xms24G –Xmx24G <other command-line options> <application>
```

Example 10 JAVA_HOME pointing to 64–bit JDK

```
java –Xms24G –Xmx24G <other command-line options> <application>
```

NOTE: The virtual space of a process must not exceed the size of the physical memory. If the physical memory size is 32 GB, the maximum Java heap size is around 28 GB (2 GB for JVM memory requirement, and 2 GB for C heap, thread stack, and 28 GB for Java heap).

Version command-line option

When `java -version` option is specified on 64–bit JDK, it reports that the JVM version is a 64–bit VM.

```
Java HotSpot(TM) 64-Bit Server VM
```

When `java -version` option is specified on 32–bit JDK, the following message is displayed.

```
Java HotSpot(TM) Server VM
```

Posting signals to GC process

You can post signals to a Java application. In NSJ7, if parallel and CMS GC is enabled, GC processes are created to perform the garbage collection. The GC process installs signal handlers for the synchronous signals and blocks the asynchronous signals. Hence, posting asynchronous signals to GC process does not have any effect. When a synchronous signal occurs, the GC process signal handler creates a error log file and terminates.

NOTE: Java process continues to process the synchronous and asynchronous signals.

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 9 (page 72) 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 9 Reserved Signals List

Reserved Signals (Always)	Reserved Signals (depending upon command line options)[1]
SIGFPE	“SIGWINCH” (page 72)
SIGILL	“SIGALRM” (page 72)
SIGSEGV	“SIGUSR2” (page 73)
SIGQUIT	“SIGHUP” (page 73)
SIGUSR1	“SIGINT” (page 73)
SIGSTK	“SIGTERM” (page 73)

[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 disabled.

`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 (Hewlett Packard Enterprise 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.

Unhandled exception

If Distributed Garbage Collector (DGC) is enabled and a Java or its GC processes encounter unhandled exception, the process which encounters the exception terminates. The other processes in the process group detect the termination of the process and exit abnormally.

Error file

When a fatal error occurs, one or more error log file are created depending upon the type of GC used by the application.

If the application uses Serial GC and a fatal error occurs, only one error file is created and it is in the form `hs_err_pid<pid>.log`

where,

`<pid>` is the process ID of the Java process. This is the same format of the error file created in earlier version of NSJ.

If the application uses Parallel or Concurrent GC and a fatal error occurs, more than one fatal error files are created and their filename format is of the form `hs_err_pid<Java pid>_<Java pid | GC pid>.log`

where,

<Java pid> is the process ID of the Java process and <GC pid> is the process ID of the GC process.

The first process ID is always the Java process ID. The second process ID is the process ID of the process that creates the error file. If the error file is created by the Java process, the second process ID is the Java process ID and if created by the GC process, it is the GC process ID. The number of error files created is equal to the number of processes in the Java process group.

When a fatal error occurs, the process that encounters the fatal error terminates. The header of the error log file generated by this process contains information about the fatal error. This could be an operating system signal or a JVM internal error. If Parallel or Concurrent GC is used, other processes in the Java process group also terminate and create error log files as mentioned earlier. The error log file contains additional information about the process that encountered the fatal error and the list of processes that forms the Java group.

Thus, the process that encountered the fatal error can be identified by looking at the headers of all the error log files generated.

Example 11 The following is the first few lines of the error log file when a Java process with parallel or concurrent GC terminates abnormally

```
#
# A fatal error has been detected by the Java Runtime Environment:
#
# SIGSEGV (0xb) at pc=7ed16ab4, pid=1979711505, tid=10 # # JRE version: 7.0 # Java VM: Java HotSpot(TM) Server
 VM (21.1-b02-svcnedccadmin:2012jul25-11:47 mixed mode nsk oss-ia64 ) # Problematic frame:
# C  [(DLL libjvm.so)+0x0]  invoke__17PSParallelCompactSFb + 0x1C0 (DLL libjvm.so)+0x0 #
```

This example indicates that the hs_err_pid1979711505_1979711505.log file was created by a Java process with PID 1979711505, and that it terminated due to SIGSEGV.

Example 12 The following is the first few lines of the error log file when a GC process terminated after detecting the abnormal termination of another process in the Java process group

```
#
# A fatal error has been detected by the Java Runtime Environment:
#
#  Internal Error (0xe0000010), pid=2097152069, tid=4 #  Error: Process 3,1140 in the Java process group has
terminated abnormally. Initiating termination of all the other processes in the process group # # JRE version:
 7.0 # Java VM: Java HotSpot(TM) Server VM (21.1-b02-svcnedccadmin:2012jul25-11:47 mixed mode nsk oss-ia64 ) #
 Failed to write core dump.
#
#  Please report this error to HP customer support.
#
---------- P R O C E S S   G R O U P-------------
# The processes in the Java process group are:
# PID (CPU, PIN):
# 620757051 (3,1138)
# 1006633021 (3,1141)
# 2097152069 (3,1139)
# 1291845689 (3,1144)
# 1979711505 (3,1140)
```

This example indicates that GC process with PID 2097152069 created the file hs_err_pid1979711505_2097152069.log. The log indicates that the GC process had terminated due to another process in the Java process group terminating abnormally. The Java PID corresponding to this GC process is 1979711505. There are four GC processes in the Java group and their PIDs are 620757051, 1006633021, 2097152069, and 1291845689.

ErrorFile with %p option

The product flag –XX:ErrorFile=file can be used to specify where the error log file is created.

Where, *file* represents the full path for the file location. The file variable can contain the substring %p.

If the application uses Serial GC and a fatal error occurs, the substring %p is expanded to process ID of the Java process.

If the application uses Parallel or CMS GC and a fatal error occurs, the substring `%p` is expanded to `<Java PID>_<[Java PID| GC PID]>`.

Where, `<Java PID>` is the process ID of the Java process ID and `<GC PID>` is the process ID of the GC process.

The first process ID is always the Java process ID. The second process ID is the process ID of the process that creates the error file. If the error file is created by the Java process, the second process ID is the Java process ID and if created by the GC process, it is the GC process ID. The number of error files created is equal to the number of processes in the Java process group. This is illustrated by the following examples:

Example 13 When the filename is specified as `errorlog%p`

```
java –XX:ErrorFile=errorLog%p SampleJava
```

where, `errorLog` is the filename of the error file.

If DGC is enabled, multiple files are created with the name format:

```
errorLog<Java PID>_<[Java PID| GC PID]>
```

where,

- `<Java PID>` is the Java process ID
- `<GC PID>` is the process ID of the GC process

For error log file created by the Java process, both PIDs are Java process ID and for error log file created by the GC processes, the first one is the Java process ID and the second one is the GC process ID.

Example 14 When the filename is specified as `error%p.log`

```
java –XX:ErrorFile=error%p.log SampleJava
```

where, `error.log` the filename of the error file.

If DGC is enabled, multiple files are created with the name format:

```
error<Java PID>_<[Java PID| GC PID]>.log
```

where,

- `<Java PID>` is the Java process ID
- `<GC PID>` is the process ID of the GC process

For error log file created by the Java process, both PIDs are Java process ID and for error log file created by the GC processes, the first one is the Java process ID and the second one is the GC process ID.

System property

The system property `sun.arch.data.model` is available in NSJ7 and provides information on the data model used. For 32–bit NSJ7, it returns the value `"32"`, and for 64–bit NSJ7, it returns the value `"64"`.

The system property `os.arch` shall return `x86_64N` for 32-bit Java process and for 64-bit Java process.

LogVMOutput option

If the `-XX:+LogVMOutput` option is used, the output produced by JVM is redirected to the `./hotspot.log` file or file specified in `–XX:LogFile=<file>` option. If any of the DGC options are specified in the command-line, then the file contains the output produced by the Java process alone. However, the output produced by the GC processes are not logged to this file.

UseCompressedOops

For 64–bit Java applications which use less than 32 GB of Java heap, you can specify `–XX:+UseCompressedOops` option to enable JRE to use 32–bit object pointers from a base 64–bit pointer. This technique improves the data access latency and applications performance. However, in NSJ7, this option is provided only as an experimental feature. Hewlett Packard Enterprise recommends that you must not deploy this option in a production environment.

SecureRandom startup enhancement

From NSJ7 SPR T2766H70^ACJ or T2866H70^ACJ, random number generation has been improved. When this option (precisely, `java.security.nativeRNG`) is enabled, it reduces the processing time taken to generate the first random number in a Java application.

By default, this option is not enabled. Use the following option to enable the same:

```
–Djava.security.nativeRNG=true
```

NOTE: This option can be used in NonStop Server for Java 6.0 also.

`VirtualMachine.list()` support

On earlier versions of NSJ, the list method of VirtualMachine API (attach API) used to return an empty value on NonStop system. However, on other platforms the list method used to return the list of Java process running on it. From SPR T2766H70^ACJ and T2866H70^ACJ and later versions, list method returns a list of Java process running on the NonStop system. Following are the changes in output compared to other platforms:

- The name of the Java process is the binary name for applications using Invocation API.
- For pure Java application, the name of the Java process is the class name or the `jar` name available in the command line.
- Usage of `–XX:+PerfDisableSharedMem` or `–XX:-UsePerfData` is ignored on NonStop.
- Lists only the Java process using T2766H60^ACH or later, T2766H70 or T2866H70 or later SPRs.

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

From NSJ 7.0 SPR — T2766H70^ACJ and T2866H70^ACJ or later SPRs, 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 Java infrastructure

Java - WORA

Applications written in Java, as is well known, are platform agnostic. The platform specific features are implemented by the Java Virtual Machine (JVM) thus insulating applications from platform idiosyncrasies. Thus Java applications exhibit similar behavior even when run on different software platforms. This is a unique feature of Java and is popularly called Write Once Run Anywhere (WORA).

NonStop TS/MP: A scalable middleware platform on NonStop

NonStop provides a middleware platform in the form of NonStop TS/MP that provides applications, NonStop characteristic features such as scalability and fault tolerance. NonStop TS/MP provides all the necessary infrastructure that allows you to develop and deploy mission-critical online transaction processing (OLTP) applications on NonStop servers. This helps you to concentrate on implementing business logic, without being concerned with common application services such as load balancing, communications, I/O, and fault tolerance. These common services are provided by NonStop TS/MP, comprising a run time execution environment and a set of APIs. Applications hosted on NonStop TS/MP must use NonStop File System (FS) and Pathsend API to inherit the unique features of NonStop TS/MP. These APIs provide function calls that are used by the client and server applications for Inter Process Communication (IPC).

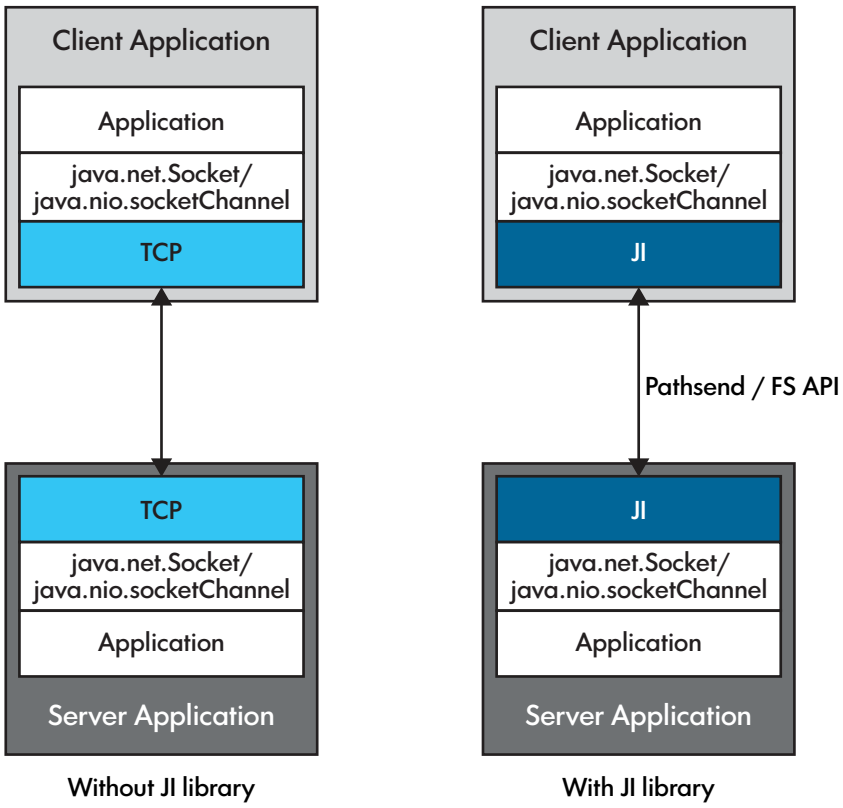
Java infrastructure

Java Infrastructure provides Java abstraction to NonStop FS and TS/MP API that would enable programmers to use `Socket` class in the `java.net` package or the `SocketChannel` in the `java.nio` package to write NonStop TS/MP or standalone server applications without having to use any FS or Pathsend API. With this a programmer can use plain sockets or any package that use sockets for communication to write server applications and host the application on NonStop TS/MP and inherit the unique features of NonStop TS/MP.

Such programs can be developed and tested on a desktop environment and deployed on NonStop TS/MP to inherit the platform features. The subsequent sections help you to understand the functionality of JI in detail. However, the actual operations related to connections in JI are discussed in the section “Establishing a connection” (page 87).

Figure 4 (page 78) illustrates client server application depicting TCP/IP and JI as two different communication channels for client server communication.

Figure 4 Client server application with TCP/IP and JI as communication channels



Sample code snippet in Table 10 (page 78) illustrates a simple client application that connects to a server, sends data and in turn receives response. On the server side, the server code listens for a connection and when the connection is established, it waits for the data and then responds back.

The sample code snippet and the subsequent sections help you to understand the functionality illustrated in Figure 1 (page 20).

Table 10 Sample code snippet

Sample client code	Sample server code

`.....`
`.....`
`(2) Socket sock = new`
`Socket("nonstop.server.com",8070);`
`.....`
`OutputStream os = sock.getOutputStream();`
`InputStream is = sock.getInputStream();`
`.....`
`(3) os.write();`
`.....`
`(6) is.read();`
`.....`
`sock.close();`
`.....`
	`.....`
	`.....`
	`ServerSocket ssock = new`
	`ServerSocket(8070);`
	`.....`
	`(1) Socket sock = ssock.accept();`
	`.....`
	`InputStream is = sock.getInputStream();`
	`(4) is.read();`
	`.....`
	`OutputStream os = sock.getOutputStream();`
	`(5) os.write();`
	`.....`

The following table describes the behavior of the relevant calls when JI is enabled. For more information on enabling JI, see "Enabling JI" (page 81). The server is configured to run as a TS/MP server class (JISVC is an assumed server class name), whereas the client is configured

to act as a Pathsend client. The configuration details are explained in the section “Mapping file” (page 82).

Sequence	Calls	Description
(1)	`ssock.accept();` This call is initiated on the server side.	The server starts listening on $RECEIVE for any new connections. For more information on connections, see “Establishing a connection” (page 87).
(2)	`new Socket(“nonstop.server.com”,8070);` This call is initiated on the client side.	This call internally results in establishing a connection to the server class JISVC using the Pathsend dialog API. This happens because as per JI configuration, the client needs to act as a Pathsend client.
(3)	`os.write();` This call is initiated on the client side.	This call internally results data being sent to the server using Pathsend dialog API. This happens because as per JI configuration, the client needs to act as a Pathsend client.
(4)	`is.read();` This call is initiated on the server side.	This call results in data being read from $RECEIVE on the connection that is established.
(5)	`os.write();` This call is initiated on the server side.	This results in the reply being sent to the client.
(6)	`is.read();` This call is initiated on the client side.	This results in the reply data being read.

Architecture

This section explains how the socket related classes in `java.net` package and the channel related classes in `java.nio.channel` package demonstrate NonStop IPC behavior.

NOTE: In the subsequent sections:

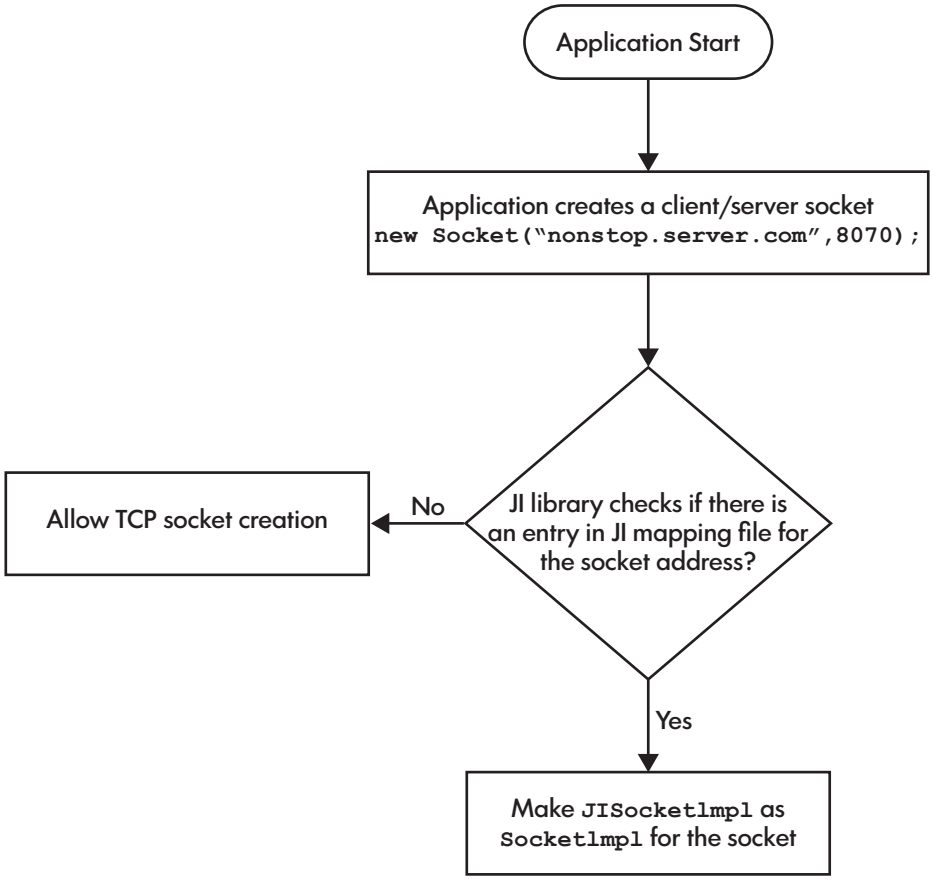
- Reference to socket includes an NIO socket channel and any reference to server socket includes an NIO server socket channel.
- Assume that JI is enabled. For more information on enabling JI, see “Enabling JI” (page 81).

`Socket` or `ServerSocket` in `java.net` package

NSJ defines an abstract class `java.net.SocketImpl`, which is a common superclass of all classes that implement sockets. Concrete implementation of this class is used to create both client and server sockets. While NSJ provides implementation of this class to create TCP sockets, JI provides an implementation (`JISocketImpl`) to create NonStop IPC sockets. Based on the mapping file, NSJ decides if a `Socket` has to be plain TCP or a NonStop IPC socket. For more information on mapping file, see “Mapping file” (page 82). In case of a NonStop IPC socket, NSJ creates the socket with `JISocketImpl` as the class implementing `java.net.SocketImpl`.

Figure 5 (page 80) illustrates the creation of socket object.

Figure 5 Creating a `java.net.socket`



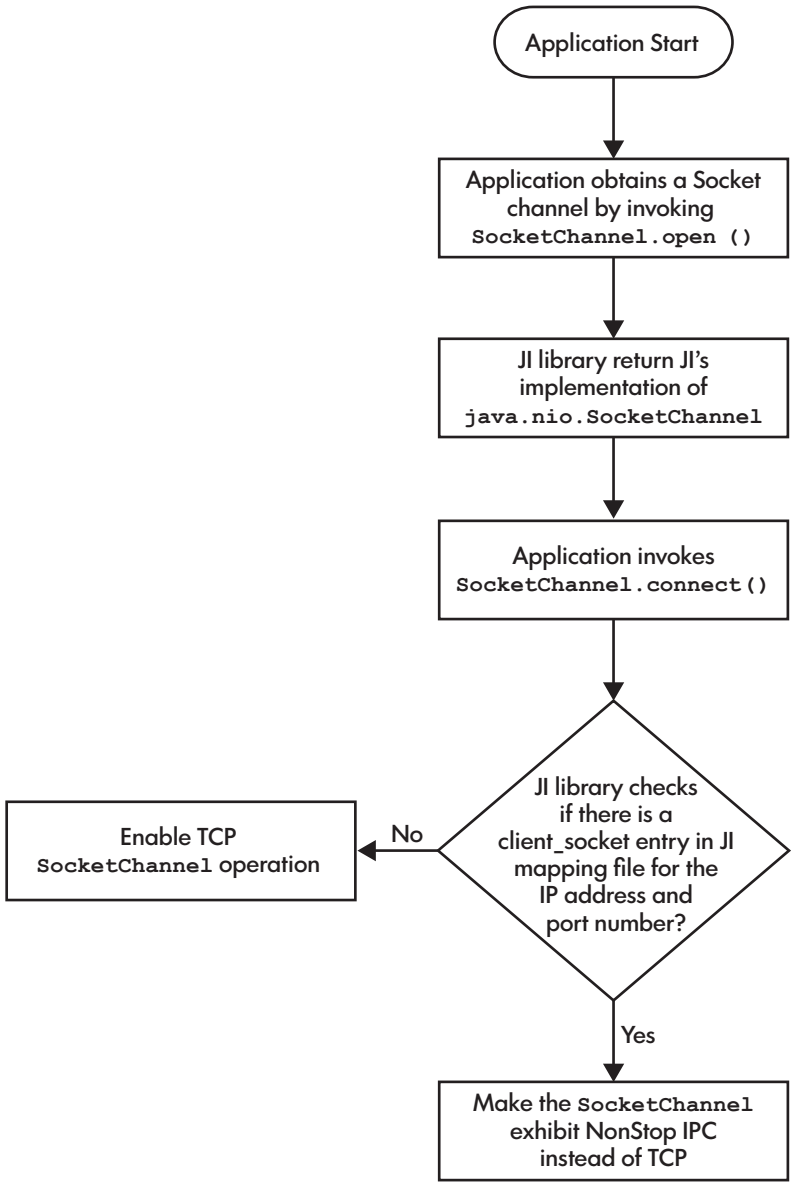
`SelectableChannel`s in `java.nio.channels`package

NSJ defines a service provider interface (spi) package (`java.nio.channels.spi`) in NIO. This package defines an abstract class `SelectorProvider`. A concrete implementation of this class is used by the JVM to obtain instances of `SelectableChannel`s and `Selector`. JI provides a `SelectorProvider` implementation, that extends the default `SelectorProvider` implementation of NSJ and overrides the methods `openSelector()`, `openServerSocketChannel()`, and `openSocketChannel()`. These methods return JI implementation of `java.nio.channels.Selector`, `java.nio.channels.ServerSocketChannel`, and `java.nio.channels.SocketChannel`. However, the JI implementation classes have the ability to exhibit normal TCP behavior and NonStop IPC behavior.

NOTE: A given instance of a channel can either exhibit TCP behavior or NonStop IPC behavior.

Figure 6 (page 81) shows the flow in creating a `java.nio.SocketChannel`.

Figure 6 Creating a `java.nio.SocketChannel`



Selector

The `Selector` implementation provided by JI library can act as a multiplexer of not only TCP channels, but also of NonStop IPC channels.

NOTE: The programmer does not have to contend with any new API or change in the way the API is used. This functionality remains transparent to the programmers.

Enabling JI

JI comprises `ji.jar` and `libji.so`. The files are present in `$JAVA_HOME/jre/lib`, `$JAVA_HOME/jre/lib/oss`, and `$JAVA_HOME/jre/lib/oss64` folders.

If the environment variable `JI_ENABLE` is set to true, the JI functionality is enabled and the `ji.jar` file is loaded by the `bootclassloader`. For example, `> export JI_ENABLE=true` code helps you to enable JI at the OSS prompt.

By default, the JI functionality is disabled. Enabling JI does not make sockets or the channels ready to use NonStop IPC for communication. The mapping file defines the enabling of socket or a channel to use NonStop IPC.

Mapping file

When the JI library is loaded, it searches for a mapping file to determine what sockets and channels need to use NonStop IPC. The mapping file contains key value pairs delimited by *<space>*.

The JI library searches for the mapping file in the following sequence:

1. Java System Property: The JI library searches for a system property by name `ji.mapping.file`. For example,

   ```
   java -Dji.mapping.file=/usr/home/ji.prop
   ```

2. Environment variable `JI_MAPPING_FILE`For example, `export JI_MAPPING_FILE=/usr/home/ji.prop`

The value can either be an absolute path to the mapping file or it can be a relative path. The path is relative to the current working directory (CWD).

Key

The key in the mapping file contains three parts and is of the format `{client_socket|server_socket}-<host name>:<port number>`

where,

`client_socket|server_socket`
indicates if the entry is for a client socket or a server socket.

If the key begins with the string `client_socket`, the JI library assumes that the mapping is for a client socket.

If the key begins with the string `server_socket`, the JI library assumes the mapping is for a server socket.

`hostname`
In case of a `client_socket`, the host name indicates the destination address where the socket is intended to connect.

In case of a `server_socket`, the host name indicates the address on which the socket is intended to bind.

The value of the host name is the output of `address.getAddress().getHostName()`, where address is an instance of the class `java.net.InetSocketAddress`.

`port number`
In case of a `client_socket`, this is the port number the client socket connects to.

In case of a `server_socket`, this is the port on which the server sockets bind.

Value

The value in the mapping file contains information on where (a standalone process or a TS/MP server class) the client socket must connect to, and the conditions that the server socket evaluates to accept new connections and modes of communication. While the key in the mapping has a common format for both client and server sockets, the format of the *value* differs. The value is a set of *`name=value`* pairs delimited by a `:` (colon).

Table 11 (page 83) lists all the attributes relevant for a client socket. Some attributes are relevant for Pathsend IPC, while some others are relevant for File System IPC. All attribute names and values are case sensitive.

Table 11 Attributes relevant for a client socket

Attribute name	Default value	Mandatory	Relevant for Pathsend IPC	Relevant for File system IPC
node Name of the NonStop node where the server process is running. For example, `node=GANESH`	None	No	Yes	Yes
process_name Name of the server process. The value must be mentioned without $. For example, `process_name=SERV`	None	Yes	Not applicable	Yes
open_qualifier This is an alphanumeric string of length seven characters. The qualifier is used to uniquely identify a connection. The server identifies and accepts a connection based on the value of the qualifier. Hewlett Packard Enterprise recommends you to accept the default value if you are not sure which value to use. For example, `open_qualifier=CLI ; open_qualifier=JMX ; open_qualifier=A1234`. For more information on qualifier, see *File Name and Process Identifiers* section in *Guardian Procedure Calls Reference Manual*.	The value of the port with a prefix of the letter `Q`. For example, `open_qualifier=Q7654`	No	Not applicable	Yes
mode This attribute indicates the mode of communication. The values can be either REQUEST_RESPONSE or BI_DIRECTIONAL. For more information on modes, see “Modes of communication” (page 87)	BI_DIRECTIONAL	No	Yes	Yes
pathmon[1] This is the name of the Pathmon which has the serverclass configured. The name of the Pathmon is without the $. For example, `pathmon=PMON ; pathmon=PROD`.	None	Yes (If the domain attribute is not mentioned)	Yes	Not applicable
serverclass This is the name of the serverclass a Pathsend IPC socket communicates with. For example, `serverclass=APP-SERVER ; serverclass=LOGIN-SRV`.	None	Yes	Yes	Not applicable

Table 11 Attributes relevant for a client socket *(continued)*

Attribute name	Default value	Mandatory	Relevant for Pathsend IPC	Relevant for File system IPC
pathsend_qualifier The value of this attribute is used by the server to accept the connection from the Pathsend IPC socket.	The value of the port with a prefix of letter Q. For example, pathsend_qualifier=Q6547.	No	Yes	Not applicable
domain[1] The value is the domain name in case the server class is configured in a TS/MP domain.	None	Yes (If the Pathmon attribute is not mentioned)	Yes	Not applicable

[1] Either Pathmon or the domain must be mentioned. Both cannot be mentioned.

Table 12 (page 84) lists all attributes that are relevant to a server socket.

Table 12 Attributes relevant to server socket

Attribute name	Default value	Mandatory
pathsend_qualifier The server socket accepts new connections based on the value of the pathsend_qualifier. Hewlett Packard Enterprise recommends you to accept the default value if you are not sure which value to use. For more information, see “Establishing a connection” (page 87).	The value of the port with a prefix of letter Q. For example, pathsend_qualifier=Q6547.	No
open_qualifier The server socket accepts new connections based on the value of the open_qualifier. Hewlett Packard Enterprise recommends you to accept the default value if you are not sure which value to use. For more information, see “Establishing a connection” (page 87).	The value of the port with a prefix of letter Q. For example, open_qualifier=Q6547.	No

Examples

Example 15 `client_socket-lab.ind.hp.com:9080 process_name=MRSS`

This means any client socket trying to connect to *lab.ind.hp.com* and port *9080* internally connects to the process by name MRSS running on the same node as the client and establishes a BI-DIRECTIONAL communication.

Example 16 `server_socket-0.0.0.0:9080`

This means a server socket that is listening for connections on any TCP interface (*0.0.0.0*) and port *9080* internally listens for file opens on $RECEIVE with an open qualifier of Q9080 or for any new dialog begin with a Pathsend qualifier of *Q9080*. For more information on how a connection is established, see “Establishing a connection” (page 87).

Example 17 `client_socket-prod.com:8083 pathmon=PROD:serverclass=LOGINSVC`

This means any client socket trying to connect to *prod.com* and port *8083* internally establishes a connection with an instance of the serverclass LOGINSVC in the Pathmon PROD. The connection established is a BI-DIRECTIONAL communication. The Pathmon must be running in the same node as the client.

Installing Java infrastructure

This section provides information about installing and configuring JI.

Installation requirements

- JI can be installed only on NonStop systems running on L15.02 or later RVUs.
- The software requirements list the earlier compatible versions of the required software for installing JI.
- Read the Softdoc before installing the product as the software requirements are described in JI product Softdoc.

Preinstallation tasks

The installation directory selected for JI (T2966L70) must contain the 32–bit and 64–bit NSJ7.

Installing JI

Installation of 32–bit and 64–bit JI is a two step process and is described in the following topics:

1. “Placing the new software on the NonStop system using DSM/SCM” (page 85)
2. “Copying the contents of the PAX file to exact location” (page 86)

IMPORTANT: If you intend to use 32–bit JI, installing 32–bit JI is sufficient. If you intend to use 64–bit features provided by JI, then both 32–bit and 64–bit JI must be installed.

Placing the new software on the NonStop system using DSM/SCM

NOTE: The following procedure is applicable for both 32–bit and 64–bit JI.

- Receive the SPR from disk or tape.
- Copy the SPR to a new revision of the software configuration you want to update.
- Build and apply the configuration revision.
- Run `ZPHIRNM` to perform the rename step.

DSM/SCM places the `pax` file in `$tsvvol.ZOSSUTL` subvolume, where, `tsvvol` is the disk volume in which the DSM/SCM places the target subvolumes (TSVs).

For more information, see *DSM/SCM User's Guide*.

NOTE: Installation of JI does not require a system generation, and therefore does not require a cold-load.

Copying the contents of the PAX file to exact location

This is achieved by using one of the following steps:

a. The DSM/SCM planner interface, with the **Manage OSS Files** check box selected.

Using this method JI is installed in the default or standard location for both 32–bit and 64–bit JI.

`/usr/tandem/nssjava/jdk170_h70`

b. If **Manage OSS Files** is not selected, DSM/SCM places the PAX file to the TSV (`$tsvvol.ZOSSUTL`). Then, PINSTALL must be used to manually extract the contents of the `pax` file to the JI installation directory for both 32–bit and 64–bit JI.

For more information on PINSTALL, see *DSM/SCM User's Guide*.

Using PINSTALL, it is possible to install 32–bit and 64–bit JI either in the standard location or in a directory of your choice. The following steps help you to achieve the same:

i. To install JI in the standard location:

Type the following command in the TACL prompt:

```
TACL> PINSTALL -rvf /G/tsvvol/zossutl/T2966PAX
```

PINSTALL delivers the contents of T2966PAX to the OSS directory located at:

`/usr/tandem/nssjava/jdk170_h70`

NOTE: The user ID under which the PINSTALL command is issued must have write access to `/usr/tandem`.

ii. The following steps help you to install JI in a directory of your choice:

Use PINSTALL with the `–s` option and type the following command in the TACL prompt:

```
TACL>
PINSTALL -s:/usr/tandem:<install-dir>:-rvf /G/tsvvol/zossutl/T2966PAX
```

PINSTALL delivers the contents of T2966PAX to the OSS directory located at:

`<install-dir>/nssjava/jdk170_l70`

Example 18 To install the product in the location `/h/myjava`

Type the following command in the TACL prompt:

```
TACL>
PINSTALL -s:/usr/tandem:/h/myjava: -rvf /G/tsvvol/zossutl/T2966PAX
```

PINSTALL delivers the contents of T2966PAX to the location:

`/h/myjava/nssjava/jdk170_l70`

Verifying the JI installation

Check with your system administrator for the installed JI (32–bit and 64–bit) software locations, and verify the installation and the environment. This example assumes that JI is installed in a nonstandard location for H–series `/home/lee/nssjava/jdk170_l70` directory:

NOTE: For more information about the operating system requirements, see T2966H70 Softdoc file.

1. Determine the version of the Java Infrastructure (JI) by typing the `vproc` command, which displays the product versions of the JI library file and any products bound in the JI library.

   The version identifier has the following form:

   ```
   T2966Hnn
   ```

   A `vproc` example displayed on H-series and J-series RVUs:

   ```
   Version procedure: T2966H70_29APR2015_jdk170_AAA...
   ```

   A `vproc` example displayed on L-series RVUs:

   ```
   Version procedure: T2966L70_20DEC2014_jdk170_ACN...
   ```

NOTE: JREHOME shell variable need not be set for NSJ7. However, if this variable is set, NSJ7 will accept it. If this variable is set, then you must ensure that it points to the correct JDK.

Establishing a connection

As already explained, a NonStop IPC client socket sends data to the server either using FS API or Pathsend API. But before sending the data, the client needs to establish a connection either by explicitly opening the server process or by establishing connection through a Pathsend dialog.

Evidently, a server socket must accept only relevant connections. A server program can have multiple server sockets that are configured as NonStop IPC server sockets. In this case each server socket accesses $RECEIVE and accepts connections that are relevant to itself.

Since all server sockets read from the same $RECEIVE, there must be a mechanism for the client socket to identify itself to enable the server socket to accept the connection. The `open_qualifier` and `pathsend_qualifier` attributes help in establishing and verifying the socket identity.

When a client socket has to open a server process the value of `open_qualifier` attribute is used as *`qualifier-1`* while constructing the process name. For more information on *`qualifier-1`*, see *Process File Names and Named Processes* in *Appendix D* in *Guardian Procedure Calls and Reference Manual*.

When a client socket has to establish a connection with a TS/MP serverclass, the `pathsend_qualifier` is sent to the server as the message in the dialog begin call.

A server socket listening for connections looks for the values of *`qualifier-1`*, in case of new opens, and the message in a dialog begin call, in case of new dialog connections, to identify and validate new connections.

This section explained how the client and server create a channel for communication by exchanging identities using qualifiers. Subsequent sections explain why just a single channel is not sufficient and one more channel of communication needs to be established.

Modes of communication

Traditional IPC on NonStop allows for a request-response mode of communication. Here a client sends a message and the server responds to the message. The server cannot send unsolicited messages to the client. This mode of communication is called REQUEST-RESPONSE.

In case of TCP communication, once a connection is established, data can be freely exchanged between the participating sockets. This mode of communication is referred as BI-DIRECTIONAL communication.

A NonStop IPC socket supports both the modes of communication with the default being BI-DIRECTIONAL.

BI-DIRECTIONAL mode

To support BI-DIRECTIONAL communication, the client establishes a second channel of communication by opening the server process at the time of establishing a connection. At any given time the second channel is in a state where the server can send data to the client. This is achieved internally by the JI library by sending a dummy message to the server such that the server can send data to the client in the form of a reply to the dummy message.

NOTE: The two channels are referred by the terms C2S (Client to Server) and S2C (Server to Client).

Figure 7 (page 88) illustrates the events involved in establishing a connection with a named process in BI-DIRECTIONAL mode.

Figure 7 Events in establishing a connection with a named process in BI-DIRECTIONAL mode

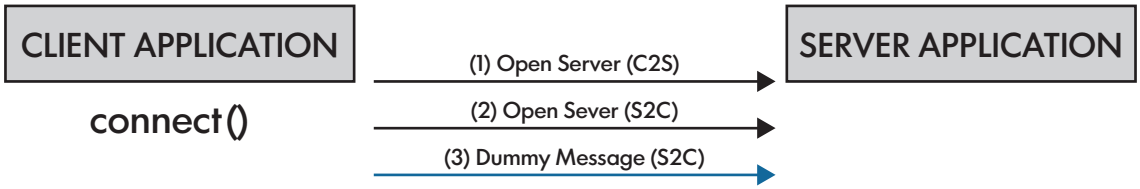
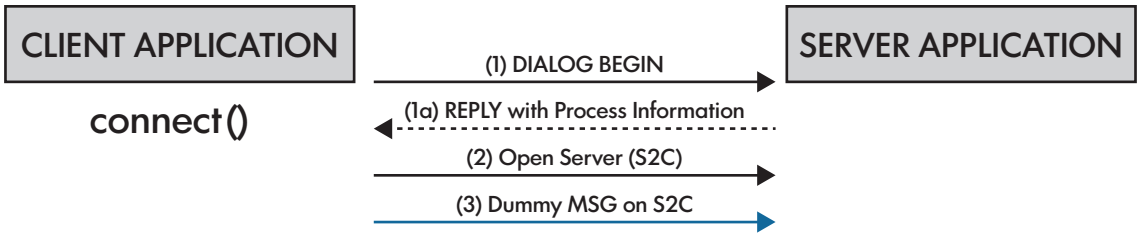


Figure 8 (page 88) illustrates the events involved in establishing a connection with a TS/MP server class in BI-DIRECTIONAL mode.

Figure 8 Events in establishing a connection with a TS/MP server class in BI-DIRECTIONAL mode



It has to be noted that the second channel of communication is established by opening the server process. In case where server process is a TS/MP server class the information necessary to open the server process is sent to the client during establishing the first channel.

Request-response mode

The request response mode is straightforward and requires only one channel C2S. A client is configured to work in this mode mainly when communicating with non JI server. Likewise, a server socket is configured to work in this mode when communicating with a non JI client.

Communicating with non JI components

At times it becomes necessary for a JI client to communicate with legacy server or a legacy application to communicate with a JI server. In such a situation, it is not possible to alter the legacy application and JI component must be able to communicate with the legacy component. The subsequent subsections discussion explain how JI components can interact with legacy components.

JI client with a legacy TS/MP serverclass for context-free communication

The client socket in the mapping file needs to be configured with the attribute `pathsend_qualifier` set to `**CONTEXT-FREE**`.

For example, `pathsend_qualifier=**CONTEXT-FREE**`.

Internally no operations are performed during the connect operation. When the client application invokes the `flush` method on the output stream of the socket, or invokes the `write` method on the NIO channel, data is sent to the server through a `SERVERCLASS_SEND_` call. The response from the server can be read through the input stream of the socket or the read method of the NIO channel.

It has to be noted that the amount of data that can be sent to the server depends on the internal buffer size of the socket. If the TS/MP version supports large data transfer then the maximum data that can be sent to the server is 2 MB, else the limit for data transfer is 32 KB.

JI client with a legacy TS/MP serverclass for dialog based communication

The client socket in the mapping file needs to be configured with the attribute `pathsend_qualifier` set to `**CONTEXT-SENSITIVE**`.

For example, `pathsend_qualifier=**CONTEXT-SENSITIVE**`.

Internally no operations are performed during the connect operation. The client application can write any amount of data through the output stream or the write method of the NIO channel. As and when the internal buffer of the socket gets full, data is sent to the serverclass once through the invocation of dialog begin call and subsequently through the dialog send calls. All invocations of the dialog API until the client application explicitly invokes a `flush` has the `MAX_REPLY_LEN` attribute set to 0. This is an indication to the server that the message transfer is incomplete. Upon invocation of the `flush`, remaining data is sent to the server with the `MAX_REPLY_LEN` attribute of the dialog API set to a positive value greater than 0. This value is equal to the receive buffer size of the socket. The server can use the file system code FECONTINUE (70) to indicate incomplete data transfer. The JI library keeps the dialog open if the file system code received is 70. It ends the dialog if the code is 0, and aborts the dialog if the file system code is other than 0 or 70.

Legacy client with a JI server

When a legacy client wants to communicate with a JI server, the server socket configuration in the mapping file must have `pathsend_qualifier` set to `**ANY-DIALOG**`, and `open_qualifier` set to `**ANY-QUALIFIER**`.

For example, `pathsend_qualifier=**ANY-DIALOG**` and `open_qualifier=**ANY-QUALIFIER**`.

A server socket with this configuration creates a new socket for any open file, context free message, or dialog message that are not handled by any other server socket definitions. There can be only one server socket defined with these unique qualifier values.

In case the client wants to transfer large amount of data (greater than what is permitted per invocation of the API call that is used for transfer) it can perform so by setting the `MAX_REPLY_LEN` attribute of the API call to 0. When the JI library notices this value, it assumes that more data transfer can be expected.

6 Transactions

NSJ7 allows you work with transactions in the following ways:

- 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 the following topics:

- “Controlling maximum concurrent transactions” (page 90)
- “`Current` Class methods” (page 90)
- “Java transaction API” (page 91)

If you use JNI and transactions, see “Java native interface (JNI)” (page 39). When you use JNI, the information available in “Controlling maximum concurrent transactions” (page 90) applies.

Controlling maximum concurrent transactions

NSJ7 application processes, by default, can start a maximum of 200 concurrent transactions in each process. By setting the `JAVA_PTHREAD_MAX_TRANSACTIONS` environment variable, you can control the maximum number of TMF transactions allowed per process to less than 1000. The following is the syntax to set the maximum transactions allowed per process:

```
JAVA_PTHREAD_MAX_TRANSACTIONS environment variable
```

Specifies the maximum number of TMF transactions allowed per process.

Permissible values are 100 through 1000. The default value of 200 is used when the variable is:

- Not set.
- 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
```

`Current` Class methods

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

Table 13 (page 90) describes the `Current` class methods.

Table 13 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.
`rollback_only`	Marks the transaction associated with the calling thread so that the only possible outcome is to roll back the transaction.

Table 13 Current Class methods *(continued)*

Method	Description
`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 `com.tandem.tmf` package description in the *JToolkit for Java API Reference Pages*.

Java transaction API

NSJ7 supports transactions by means of the NonStop Server for Java Transaction API (JTA), which is an implementation of the Oracle JTA Version 1.0. NonStop Server for Java Transaction API implements parts of the Oracle JTA package (`javax.transaction`). NSJ7 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 and the version of NonStop Server for Java Transaction API that uses JTS is called NonStop Server for Java Transaction API-JTS. Both 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 92)). The default is TMF.

NonStop Server for Java Transaction API-TMF is intended for applications other than CORBA applications.

This subsection explains the following topics:

- “`javax.transaction` interfaces” (page 91)
- “javax.transaction exceptions” (page 92)
- “Examples” (page 92)

For more information about Oracle JTA, see **Oracle JTA document**.

`javax.transaction` interfaces

Oracle JTA package (`javax.transaction`), defines the following interfaces:

- `Status`
- `Synchronization`
- `Transaction`

- `TransactionManager`
- `UserTransaction`

NSJ Transaction API supports all of the preceding interfaces, but only `UserTransaction` is available for client programs.

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

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

javax.transaction exceptions

Oracle JTA package (`javax.transaction`), defines the following exceptions.

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

NSJ Transaction API supports all of the preceding exceptions.

Examples

The following examples are similar except that:

- The first example uses Example 19 (page 93).
- The second example requests Example 20 (page 93).

Example 19 By default, NSJ transaction API-TMF

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();
```

Example 20 By request, NSJ transaction API-TMF

The following code gets a reference to `UserTransaction` based on TMF (which it requests). 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 request).
UserTransaction utx = JTAFactory.getUserTransaction(JTAFactory.TMF);

// Start transaction
utx.begin();

// Do work
...//

Commit transaction
utx.commit();
```

7 Application tuning and profiling

This chapter provides information about application tuning and profiling, and it includes the following topics:

Profiling application performance

Application profiling and monitoring are supported through HPjmeter tool, which works in combination with NSJ options, `–Xeprof` and `–agentlib:hprof`.

`HPjmeter` tool contains support for both 32–bit and 64–bit NSJ7.

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” (page 94).
- Analyzing garbage collection data.
- Analyzing profiling data.

Monitoring live Java applications

`HPjmeter` contains two components namely `nodeagent` and `jvmagent`. These components obtain the data from Java and GC processes and consolidate them as a single process entity for `HPjmeter` client. Only a 32–bit version of `nodeagent` is available.

Starting with HPjmeter 4.3, three versions of `jvmagent` are available. The following list describes the available `jvmagent` versions. `/usr/tandem/hpjmeter` refers to the installation location of HPjmeter:

- 32–bit `jvmagent` to be used with 32–bit NSJ7 applications. This is available at `$JMETER_HOME/lib/oss32`.
- 64–bit `jvmagent` to be used with 64–bit NSJ7 applications. This is available at `$JMETER_HOME/lib/oss64`.
- 32–bit `jvmagent` to be used with pre-NSJ7 applications. This is available at `$JMETER_HOME/lib/oss`.

NOTE:

1. When distributed GC is enabled, HPjmeter interacts only with the Java processes and not GC processes. The Java process provides all the necessary data that HPjmeter requires.
2. When HPjmeter is used to monitor Java process in real time, it only lists the Java process and does not list the corresponding GC processes.
3. The threads in the GC process are not part of the thread count as shown by **Threads Histogram** tab.

The Java application must be prepared for live mode of profiling by running `HPjmeter` agents.

The following procedure provides the set up for monitoring a live Java application:

1. Install the T0866H31 PAX file in the default installation directory: `/usr/tandem/hpjmeter`
2. Run the 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=`

1) `$JMETER_HOME/lib/oss`, to monitor pre—NSJ7 Java process.
2) `$JMETER_HOME/lib/oss32`, to monitor 32–bit NSJ7 Java process.
3) `$JMETER_HOME/lib/oss64`, to monitor 64–bit NSJ7 Java process.

b. export `JAVA_HOME=/usr/tandem/nssjava/jdk170_h70`

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.hpe.com/downloads/hpjmeter**.
5. Connect to the `nodeagent` from the `HPjmeter` console.

NOTE:

- For information on monitoring capabilities and tips, see *HPjmeter 4.3 User's Guide* available at **http://www.hpe.com/info/hpux-hpjmeter-docs** .
- The instructions for using the `HPjmeter` tool on the NonStop systems are provided in "Addendum to `HPjmeter` 4.3 user's guide" (page 112).

Collecting profile data for analysis

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

1. To enable eprof profiler start the Java application with `-Xeprof` option.
2. Zero-preparation profiling — start and stop the 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 the Java application `-agentlib:hprof`).

NOTE:

- For information on analyzing profile data, see *HPjmeter 4.3 User's Guide* available at **http://www.hpe.com/info/hpux-hpjmeter-docs** .
- The instructions for using the `HPjmeter` tool on the NonStop systems are provided in "Addendum to `HPjmeter` 4.3 user's guide" (page 112).

`-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 7.0 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.

The 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 4.3 User's Guide* available at **http://www.hpe.com/info/hpux-hpjmeter-docs** and the Oracle documentation at **http://docs.oracle.com/javase/7/docs/technotes/samples/hprof.html**.

NOTE: The following HPROF option is not supported on NSJ7:

```
–agentlib:hprof=cpu=samples
```

–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. To take full advantage of HPjmeter functionality, you can gather profiling data using –Xeprof for performance tuning and –agentlib:hprof for memory tuning when you run your application. For a comparison of the features of Xeprof and HPROF, before using either of the profiles, see Table 5-4 in the *HPjmeter 4.3 User's Guide*.

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 7.0 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:

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

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

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.

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

```
kill -ALRM pid
```

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

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:

- NSJ7 supports rotational GC logging in multiple GC log files to help control the GC log file size. For information on GC log rotation, see “GC log rotation” (page 66).
- For information on analyzing GC profile data, see *HPjmeter 4.3 User’s Guide* available at **http://www.hpe.com/info/hpux-hpjmeter-docs** .
- The instructions for using the `HPjmeter` tool on the NonStop systems are provided in “Addendum to `HPjmeter` 4.3 user's guide” (page 112).

Other profiling options

TACL command `status` can be used to obtain the detailed status of application, including the process time of a Java process. Also, PSTATE tool can be used to get details about the Java process.

In NSJ7, if Parallel or CMS GC is enabled, a Java application consists of a Java process group that includes the Java process and GC processes. If parallel or CMS GC is enabled, run the TACL command status and PSTATE tool individually for the Java and the GC processes. To obtain all the GC processes for a Java process, `nsjps` can be invoked with `–gc` option. For information about `nsjps`, see *NonStop Server for Java 7.0 Tools Reference*.

Also, the C heap usage reported by PSTATE tool for an application running with NSJ7 will be lesser than C heap usage reported when the same application is run with NSJ6. This is because most of the JVM allocations and the Java heap are moved to flat segments in NSJ7.

Tuning application performance

This section contains the following topics for tuning application performance:

- “Determining the heap size setting” (page 97)

Determining the heap size setting

You must set the Java heap to a value appropriate for your application for optimal performance. This section discusses the guidelines for setting the heap size.

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: JVM allocates the maximum Java heap at start-up. Hence, 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:

```
$ gtacl -p nskcom

NSKCOM - T5838H02 BASE (15AUG12) - Jun 13 2012
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 1,525 ,detail

SYSTEM : \SYSTEM1 LOGTIME : June 26, 2012 13:57:14
TYPE : (E=Extensible S=Shared)
(CPU Page size is 16384 Bytes)
```

Following is the output for the preceded commands:

Process	Pri	User-ID	Program File Name
1,525	1	200,13	$JAVDV01.ZYQ00003.Z000Q3GF

```
KMSF-BACKED SEGMENTS: (Process Space Guarantee =   --  )
-------------------------------------------------------------------------------
SEG-ID TYPE                 SIZE        RESERVATION
                            KBYTE     PAGES    KBYTE
-------------------------------------------------------------------------------
Heap+Global+SRL+Stack       69 MB     1917     29 MB
1068                        16        1        16
1069                        64 MB     16384    64 MB
1065                        64 MB     4096     64 MB
1067                        1088 MB   69636    1088 MB
2100                        8B        1        16
2101                        8B        1        16
--------------------------------------------------------------------------------
TOTAL                                 92034    1246 MB

FILE-BACKED SEGMENTS: None
________________________________________________________________________________
NSK-
```

In the preceding output, the JVM process uses 1246 MB of swap space and has six segments allocated. The JVM process was started with a heap size of 1024 MB, which shows that apart from the Java heap, the process requires about 222 MB of 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 are fully initialized and running.

For related tuning guides, see *Tuning Guide for iTP Secure WebServer* and *NonStop Servlets for JavaServer Pages (NSJSP) on HPE NonStop Servers*.

8 Migrating applications

This section describes the changes required to migrate applications that use earlier versions of NSJ. The following lists the various terminologies to be noted:

- NSJ4 refers to the product based on J2SE SDK 1.4.x
- NSJ5.1 refers to the product based on J2SE JDK 1.5.x
- NSJ6 refers to the product based on Java SE JDK 1.6.x
- NSJ7 refers to the product based on Java SE JDK 1.7.x

This section also explains the following topics:

Summary of migration changes

See Table 14 (page 99) for the topics that apply to migrating from particular versions.

Table 14 Summary of migration changes for NonStop server for Java versions

Migration Topic	Version 2 of NonStop Server for Java 4 (T2766V20 on TNS/R)	NonStop Server for Java 4 (T2766H10 on TNS/E)	NonStop Server for Java 5.1 (T2766H50)	NonStop Server for Java 6.0 (T2766H60)
"Installation changes" (page 100)	Applicable	Applicable	Applicable	Applicable
"Public library directory" (page 100)	Applicable	N/A	N/A	N/A
"Java based JAR file locations" (page 100)	N/A	N/A	N/A	N/A
"Dynamic link libraries" (page 101)	Applicable	N/A	N/A	N/A
"Makefile to link native libraries" (page 101)	Applicable	N/A	N/A	N/A
"Compiling C++ native code" (page 101)	Applicable	N/A	N/A	N/A
"Floating-point support" (page 102)	N/A	N/A	N/A	N/A
"Using AWT classes" (page 103)	N/A	N/A	N/A	N/A
"POSIX threads" (page 103)	Applicable	Applicable	Applicable	Applicable
"Directories of binary files moved" (page 103)	N/A	N/A	N/A	N/A
"JAAS enhancement" (page 104)	N/A	N/A	N/A	N/A
"Miscellaneous changes for migration to TNS/E" (page 104)	Applicable	N/A	N/A	N/A

Table 14 Summary of migration changes for NonStop server for Java versions *(continued)*

Migration Topic	Version 2 of NonStop Server for Java 4 (T2766V20 on TNS/R)	NonStop Server for Java 4 (T2766H10 on TNS/E)	NonStop Server for Java 5.1 (T2766H50)	NonStop Server for Java 6.0 (T2766H60)
"Java stack size" (page 104)	N/A	N/A	N/A	Applicable
"Default Java heap size and stack size" (page 106)	Applicable	Applicable	Applicable	Applicable
"JNI application consideration" (page 105)	Applicable	Applicable	Applicable	Applicable
"Dynamic snapshot" (page 105)	N/A	N/A	N/A	Applicable

For information about earlier Java version changes, see release notes at the Oracle Java web site for the particular version of Java. For information about changes in NSJ7, see "Supported and unsupported features of NSJ7" (page 111).

Installation changes

32–bit NSJ7

The 32–bit version of NSJ7 can be installed using DSM/SCM. The product number for this JDK is T2766.

64–bit NSJ7

The 64–bit version of NSJ7 can be installed using DSM/SCM. The product number for this JDK is T2866.

NOTE: 32–bit JDK must be installed prior to 64–bit JDK installation, otherwise, the 64–bit JDK application does not run.

For more information on Installing NSJ7, see "Installation and configuration" (page 23).

Public library directory

The public library directory does not apply to NSJ versions 4, 5, 5.1, 6.0, or 7.0 hosted on TNS/E because DLLs are used on TNS/E. For information about migrating native libraries, see "Dynamic link libraries" (page 101).

Java based JAR file locations

If you are using NSJ3.1.x or earlier versions, guidelines have changed for placing JAR files both "For Java based products" (page 100) and "User-provided JAR files" (page 101).

For Java based products

Before the first version of the NSJ4 (based on J2SE SDK 1.4.0), no guidelines were provided for where Java based products must 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 might 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. Therefore, starting with first version of NSJ4, Hewlett Packard Enterprise

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, JAR files were installed in (`/usr/tandem/java/jre/lib/ext`) because you did not have to include such JAR files in the `CLASSPATH`. Starting with first version of NSJ4 (based on J2SESDK 1.4.0), Hewlett Packard Enterprise recommends that you to not install user-provided JAR files in any directory of versions 1 and 2 of NSJ4 tree. You must 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 NSJ4. However, you have to place the JAR files in the `CLASSPATH`.

Dynamic link libraries

On the TNS/E and TNS/X systems, NSJ 4, 5, 5.1, 6.0, and 7.0 support DLLs. All NonStop Server for Java applications migrating from TNS/R to TNS/E or TNS/X must convert native libraries to DLLs.

Consider the following issues when migrating applications to use DLLs with NSJ7:

- All the Java libraries are built as DLLs.
- When using the JNI code, use DLLs instead of static libraries. For more information, see “Java native interface (JNI)” (page 39). A public library directory does not apply for Java applications on the TNS/E and TNS/X systems.
- All DLLs must be in the files that have specific naming requirements.
- On TNS/E and TNS/X, the `-Dcompaq.liblist` option is not supported.
- The customer Makefile no longer exists in NSJ versions 4, 5, 5.1, 6.0, and 7.0 on TNS/E because DLL support precludes the need to bind user native code in the Java executable.
- The `_RLD_LIB_PATH` environment variable, used only on the TNS/E or TNS/X system, specifies the library path for user DLLs. For more information, see “_RLD_LIB_PATH” (page 29).
- The invocation API uses the JVM as a DLL; therefore, if you use this API, you do not need to statically link Java to your programs.

Makefile to link native libraries

The customer Makefile no longer exists for NSJ versions 4, 5, 5.1, 6.0, and 7.0 on TNS/E or TNS/X because DLL support precludes the need to bind user native code to the Java executable. For more information about migrating native libraries, see “Dynamic link libraries” (page 101).

Compiling C++ native code

For TNS/E and TNS/X, C++ code is compiled using either version 2 or version 3 for user DLLs because the NSJ versions 4, 5, 5.1, 6.0, and 7.0 on TNS/E or TNS/X is built with a C++ version neutral flag (the `-setCPlusPlusDialect neutral` option for the linker).

If you are migrating NSJ 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. For more information about this pragma and the warnings it can produce, see *C/C++ Programmers Guide*.

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, build your own executable and link that executable against the JVM DLL. For migration considerations information, see "Calling Java methods from C or C++" (page 41). For a demo, see invocation API demo provided by NSJ7 in *install_dir*`/demo/invocation_api`.

For more information, see "Linker and compiler options" (page 42).

Floating-point support

By default, NSJ 3.1.x and earlier versions converted any floating-point value that crossed the JNI boundary to a TNS float. This default can be overridden by supplying a line in the file `TandemVMClassFP.properties`. If a particular class required IEEE floating-point values passed to its JNI code instead of TNS float values; a property was added to the file (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`, indicating that they wanted IEEE floating-point values passed to their JNI code or `TANDEM_FP`, indicating 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. Similarly, any float value passed to Java must be an IEEE float value. Table 15 (page 102) illustrates the NSJ7 application's floating-point usage compared to earlier versions.

Table 15 Summary of floating-point support

	NSJ 2.x	NSJ 3.x	NSJ 4.x	NSJ 5.x	NSJ 6.0	NSJ 7.0
Java floating-point usage	IEEE float	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	IEEE float
JNI calling convention	Tandem float	Either IEEE or Tandem float	IEEE float	IEEE float	IEEE float	IEEE float
Java compiler flag	Tandem float	IEEE float	IEEE float	IEEE float	IEEE float	IEEE float
Java linker flag	Tandem float	Tandem float	IEEE float	IEEE float	IEEE float	IEEE float

Since NSJ 3.1.x and earlier versions set the linker flag for the process to TNS float, any use of the C runtime library used routines that handled TNS floats. For NSJ versions 4, 5, 5.1, 6.0, and 7.0 versions, the linker flags described in "Linker and compiler options" (page 42) are used to specify IEEE floating-point. Accordingly, the C runtime library uses routines that handle IEEE floating-point.

For NSJ versions 4, 5, 5.1, 6.0, and 7.0 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 “Floating-point implementation” (page 43).

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`.

NonStop operating system does not provide support for windowing operations, NonStop Server for Java 3.1.x or earlier versions 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 throws an `UnsupportedClassException`.

NSJ7 supports the Oracle's 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.

Oracle implemented headless support by supplying two new methods in the `GraphicsEnvironment` class: is `Headless` and is `HeadlessInstance`. In addition, Oracle created a new exception `java.awt.HeadlessException`. `HeadlessException`, which is 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 NSJ versions 4, 5, 5.1, 6.0, and 7.0.

For more information, see “Headless support” (page 38).

POSIX threads

NSJ 3.1.x and earlier versions used OSS POSIX Threads (product number T5819) that conformed to an earlier standard for POSIX threads. NSJ versions 4, 5, 5.1, and 6.0 use the Standard POSIX User Threads (product number T1248) library referred as the SPT model library. Whereas, NSJ version 7 uses POSIX User Threads (product number T1280) referred as PUT model library.

The Pthreads function provided by the PUT model library comply with the IEEE Std 1003.1, 2004, POSIX System Application Program Interface. Therefore any native code that makes Pthread calls might have to change. For more information, see chapter *Using the POSIX User Thread (PUT) Model Library* in *Open System Services Programmer's Guide*.

Additionally, you must change any JNI code that made calls to routine beginning with `cma..` supplied with T5819 or `spt...` supplied with T1248 to use the Wrapper or Development Toolkit routines (`put..`) supplied with T1280.

For information about changes needed to migrate threaded applications to NSJ7, see “Threading considerations for native code” (page 48).

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 NSJ7 installation `bin` directory.

In NSJ 3.1.x or earlier versions, the `bin` and `jre/bin` directories contained a shell script that ran the real executable located in `bin/oss/posix_threads`. In NSJ7 version, the bin directory contains the real executable, no shell script wrapper, and no `bin/oss/posix_threads` directory are present. The `jre/bin` directory contains the executables in the `bin` directory.

JAAS enhancement

In NSJ 3.x and earlier versions, the JAAS was an optional package (extension). It is integrated with NSJ7. JAAS augments the core Java 2 platform with APIs that allow authenticating and enforcing access controls on 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 NSJ7, 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” (page 51).

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 usage depends on the following system type:

- On TNS/R systems, use the following command:

  `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 the following command:

  `JNI_OnLoad`

On TNS/R systems, the `JNI_OnUnload` function is not supported by versions of NSJ available for this platform.

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). For more information, see “Debugging Java process” (page 107).

`dlfcn.h` file

All applications migrating from TNS/R that use the `dlfcn.h` file require code changes. On TNS/E, NSJ versions 4, 5, 5.1, 6.0, and 7.0 do not use their own special version of `dlfcn.h`. Use the file that exists in the `include` directory (`/usr/include`) of the system.

Java stack size

JVM reserves a portion of the Java stack for its runtime operations. This area of the stack is referred as `StackShadowArea`, and its size can be set using the option `-XX:StackShadowPages`. The size used by this area is measured in units of page size. On NonStop systems, the page size is 16K bytes. For 32–bit NSJ7, the value of `StackShadowPages` is 2, and for 64–bit NSJ7, the value is 3.

In NSJ7, the `StackShadowArea` is increased compared to earlier versions of NSJ. Thus, the available space for Java methods is decreased by 16K bytes. When you migrate from earlier versions of NSJ to NSJ7, if the application encounters the error

`java.lang.StackOverflowError`, then increase the value of stack size by using `-Xss` or `ThreadStackSize` options.

JNI application consideration

JNI applications cannot enable parallel and CMS GC if the JVM is loaded in the main thread. For more information, see “Calling Java methods from C or C++” (page 41).

Dynamic snapshot

In NSJ7, the dynamic snapshot support is limited to capturing the snapshot of the Java process only. For more information, see “Saveabend file generation” (page 51).

Migrating from serial GC to parallel GC

NSJ7 supports Parallel and CMS garbage collectors. Parallel and CMS GC provide reduced application pause time during GC when compared to Serial GC.

In NSJ7, when the parallel and CMS GC is enabled using command line option, one process is launched for each GC thread. The Java and corresponding GC processes are called as process group and they perform the following:

- Run on a same logical processor.
- Run simultaneously on different IPU(s) of the same processor.
- Provide a single process image to the Java application.
- Share the virtual address space.
- Communicate among themselves using Guardian IPC mechanism.

In Java process group, whenever a process terminates, the other processes in the group also terminate. For more information, see “`javagc`” (page 65).

Application start-up time overhead

Application start-up time results higher while using Parallel and CMS GC since GC processes must be launched and virtual memory must be shared by the GC processes.

With `–XX:ParallelGCThreads=4`, the overhead in startup time due to parallel GC was found to be approximately 310 milliseconds. If your application cannot tolerate the overhead in startup time due to the distributed GC, do not use the distributed GC options.

Swap space consideration

If parallel and CMS GC is enabled, then each GC thread has a corresponding GC process. The GC process has the following virtual memory overhead:

- Main user stack (256 KB)
- Main RSE stack (256 KB)
- C heap (initial size, around 32 KB)
- One user thread stack (256 KB), for 32–bit NSJ7
- One user thread stack (512 KB), for 64–bit NSJ7
- One user RSE thread stack (256 KB), for 32–bit NSJ7
- One user RSE thread stack (512 KB), for 64–bit NSJ7
- Public DLLs instance data (around 1 MB)

In total, around 3 MB of additional swap space is required for one GC process. Considering that there are four parallel GC processes, the total overhead is around 12 MB swap space per Java process.

Resident space consideration

If parallel and CMS GC is enabled, then each GC thread has a corresponding GC process. The GC process has the following resident memory overhead:

- PFS (256 KB)
- Privileged user stack (64 KB)
- Privileged user RSE stack (64 KB)

In total, around 384 KB of additional resident memory space is required for one GC process. Considering that there are four parallel GC processes, the total overhead is around 1.5 MB resident memories per Java process.

Using _RLD_FIRST_LIB_PATH

NSJ7 does not support loading of a public DLL from a non-standard private location by using `_RLD_FIRST_LIB_PATH` environment variable. This limitation is due to the fact that NSJ7 creates a group of processes that should load the public DLLs at the same virtual memory address order, so that the parallel garbage collectors are functioned appropriately. This is unlike earlier versions of NSJ where the entire Java application is run as a single instance of JRE/JVM process.

For Invocation API, the `_RLD_FIRST_LIB_PATH` needs to be set as follows:

- For 32-bit Java process: export `_RLD_LIB_PATH=/usr/tandem/nssjava/jdk170_x70/jre/lib/oss`
- For 64-bit Java process: export `_RLD_LIB_PATH=/usr/tandem/nssjava/jdk170_x70/jre/lib/oss64`

However, Hewlett Packard Enterprise recommends that you must not load public libraries from private locations.

Migrating to TNS/X

The L-series application environment is similar to the H- and J-series application environment. For more information about the tasks required to migrate NonStop Server for Java 7 applications to L-series systems, see *Migrating NonStop Server for Java 7 Applications* section in *L-series Application Migration Guide*.

Other considerations

Default Java heap size and stack size

For 64–bit NSJ7, the default heap and stack sizes are as follows:

- The default stack size is 1024 KB.
- The initial heap size is 1 GB and the default maximum heap size is 1 GB.

For more information, see “Java heap size with 32–bit JDK7” (page 64) and “Java heap size with 64–bit JDK7” (page 64).

NOTE: Hewlett Packard Enterprise recommends that the application specifies the same value for `–Xms` and `–Xmx`. If only `–Xmx` is specified in the command-line, the default value used for `–Xms` is the same value that is specified for `–Xmx`.

For more information, see “Memory management considerations” (page 62).

Java process name

A Java process can be started as a named process by specifying –name option with the run command. In NSJ7, there is a restriction in starting a named Java process and it is explained in the following subsections.

Using 32–bit JDK

A 32–bit Java application can be launched with NSJ7 by one of the following methods. However, only the **Method 1** can be used to launch 32–bit JDK as a named process.

Method 1: Using 32–bit JDK

```
$ export PATH=/usr/tandem/nssjava/jdk170_h70/bin:$PATH
$ run –name /G/JAVA1/ java HelloWorld
```

Method 2: Using 64–bit JDK

```
$ export PATH=/usr/tandem/nssjava/jdk170_h70/bin/oss64:$PATH
$ run –name /G/JAVA1/ java –d32 HelloWorld
```

NOTE: The 64–bit Java process created first in the **Method 2** is a named process having name $JAVA1. This process uses execvp() system call to create the 32–bit Java process. A process created using execvp() does not inherit the name of the calling process, and hence the resultant 32–bit process is unnamed.

Using 64–bit JDK

A 64–bit Java application can be launched with NSJ7 by one of the following methods. However, only the **Method 2** can be used to launch 64–bit JDK as a named process.

Method 1: Using 32–bit JDK

```
$ export PATH=/usr/tandem/nssjava/jdk170_h70/bin:$PATH
$ run –name /G/JAVA1/ java –d64 HelloWorld
```

Method 2: Using 64–bit JDK

```
$ export PATH=/usr/tandem/nssjava/jdk170_h70/bin/oss64:$PATH
$ run –name /G/JAVA1/ java HelloWorld
```

NOTE: The 32–bit Java process created first in the **Method 1** is a named process having name $JAVA1. This process uses execvp() system call to create the 64–bit Java process. A process created using execvp() does not inherit the name of the calling process, and hence the resultant 64–bit process is unnamed.

Debugging Java process

A Java process can be automatically launched inside the Native Inspect debugger using –debug command-line option with run command.

Debugging using 32–bit JDK

A 32–bit Java application can be launched inside the Native Inspect (eInspect) debugger by one of the following two methods. However, only the **Method 1** can be used to launch the process under debug.

Method 1: Using 32–bit JDK

```
$ export PATH=/usr/tandem/nssjava/jdk170_h70/bin:$PATH
$ run –debug java HelloWorld
```

Method 2: Using 64–bit JDK

```
$ export PATH=/usr/tandem/nssjava/jdk170_h70/bin/oss64:$PATH
$ run -debug java -d32 HelloWorld
```

NOTE: The 64–bit Java process created first in the **Method 2** is under einspect. As this process uses `execvp()` system call to create the 32–bit Java process and a process created using `execvp()` does not inherit the PIN of the calling process, eInspect cannot continue to debug the process. If you want to debug it, a new debugger needs to be attached to the process.

Debugging using 64–bit JDK

A 64–bit Java application can be launched inside the Native Inspect (eInspect) debugger by one of the following two methods. However, only the **Method 2** can be used to launch the process under debug.

Method 1: Using 32–bit JDK

```
$ export PATH=/usr/tandem/nssjava/jdk170_h70/bin:$PATH
$ run -debug java -d64 HelloWorld
```

Method 2: Using 64–bit JDK

```
$ export PATH=/usr/tandem/nssjava/jdk170_h70/bin/oss64:$PATH
$ run -debug java HelloWorld
```

NOTE: The 32–bit Java process created first in the **Method 1** is under einspect. As this process uses `execvp()` system call to create the 64–bit Java process and a process created using `execvp()` does not inherit the PIN of the calling process, eInspect cannot continue to debug the process. If you want to debug it, a new debugger needs to be attached to the process.

9 Support and other resources

Accessing Hewlett Packard Enterprise Support

- For live assistance, go to the Contact Hewlett Packard Enterprise Worldwide website:

  www.hpe.com/assistance

- To access documentation and support services, go to the Hewlett Packard Enterprise Support Center website:

  www.hpe.com/support/hpesc

Information to collect

- Technical support registration number (if applicable)
- Product name, model or version, and serial number
- Operating system name and version
- Firmware version
- Error messages
- Product-specific reports and logs
- Add-on products or components
- Third-party products or components

Accessing updates

- Some software products provide a mechanism for accessing software updates through the product interface. Review your product documentation to identify the recommended software update method.
- To download product updates, go to either of the following:
  - Hewlett Packard Enterprise Support Center **Get connected with updates** page:

    www.hpe.com/support/e-updates

  - Software Depot website:

    www.hpe.com/support/softwaredepot

- To view and update your entitlements, and to link your contracts and warranties with your profile, go to the Hewlett Packard Enterprise Support Center **More Information on Access to Support Materials** page:

  www.hpe.com/support/AccessToSupportMaterials

IMPORTANT: Access to some updates might require product entitlement when accessed through the Hewlett Packard Enterprise Support Center. You must have an HP Passport set up with relevant entitlements.

Websites

Website	Link
Hewlett Packard Enterprise Information Library	**www.hpe.com/info/enterprise/docs**
Hewlett Packard Enterprise Support Center	**www.hpe.com/support/hpesc**

Website	Link
Contact Hewlett Packard Enterprise Worldwide	**www.hpe.com/assistance**
Subscription Service/Support Alerts	**www.hpe.com/support/e-updates**
Software Depot	**www.hpe.com/support/softwaredepot**
Customer Self Repair	**www.hpe.com/support/selfrepair**
Insight Remote Support	**www.hpe.com/info/insightremotesupport/docs**
Serviceguard Solutions for HP-UX	**www.hpe.com/info/hpux-serviceguard-docs**
Single Point of Connectivity Knowledge (SPOCK) Storage compatibility matrix	**www.hpe.com/storage/spock**
Storage white papers and analyst reports	**www.hpe.com/storage/whitepapers**

Customer self repair

Hewlett Packard Enterprise customer self repair (CSR) programs allow you to repair your product. If a CSR part needs to be replaced, it will be shipped directly to you so that you can install it at your convenience. Some parts do not qualify for CSR. Your Hewlett Packard Enterprise authorized service provider will determine whether a repair can be accomplished by CSR.

For more information about CSR, contact your local service provider or go to the CSR website:

www.hpe.com/support/selfrepair

Remote support

Remote support is available with supported devices as part of your warranty or contractual support agreement. It provides intelligent event diagnosis, and automatic, secure submission of hardware event notifications to Hewlett Packard Enterprise, which will initiate a fast and accurate resolution based on your product's service level. Hewlett Packard Enterprise strongly recommends that you register your device for remote support.

For more information and device support details, go to the following website:

www.hpe.com/info/insightremotesupport/docs

Documentation feedback

Hewlett Packard Enterprise is committed to providing documentation that meets your needs. To help us improve the documentation, send any errors, suggestions, or comments to Documentation Feedback (**docsfeedback@hpe.com**). When submitting your feedback, include the document title, part number, edition, and publication date located on the front cover of the document. For online help content, include the product name, product version, help edition, and publication date located on the legal notices page.

A Supported and unsupported features of NSJ7

NSJ7 includes all the features of NSJ6 and it is based on Java SE 7.0.

For information about Java SE 7.0 features, see **New Features and Enhancements Java SE 7.0**

Java SE 7.0 features not implemented in NSJ7

Java SE 7.0 features that do not apply to a server-side HotSpot VM are not implemented in NSJ7.

For information about Java SE 7.0 features that are not implemented, see following implementation-specific topics:

- “Headless support” (page 38)
- “Java standard edition development kit (JDK)” (page 18)
- UseBiasedLocking (JVM internal locking optimization, useful for SMP architectures)
- `–XX:+TieredCompilation` option is not supported
- SDP (Socket Direct Protocol)
- SCTP (Stream Control Transmission Protocol)
- JDBC 4.1. NSJ7 supports only JDBC4.0
- NSJ7 does not support Parallel and CMS GC for single core systems (H-series systems)
- `–XX:+UseLargePages` option is not supported

B Addendum to HPjmeter 4.3 user's guide

This appendix provides instructions for using the HPjmeter tool on NonStop system. It is based on the *HP-UX HPjmeter 4.3 User's Guide*, available at the following web address:

http://www.hpe.com/info/hpux-hpjmeter-docs .

This version of HPjmeter supports 32–bit and 64–bit NSJ7.

The following sections correspond to those in the *HP-UX HPjmeter 4.3 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 *HPjmeter 4.3 User's Guide*.

Completing installation of HPjmeter

Agent requirements

Table 16 (page 112) lists the agent requirements.

Table 16 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 HPE Integrity NonStop BladeSystem • NSJ6 and later software product revisions (SPRs)

NOTE: The HPjconfig tool is not supported on a NonStop operating system.

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 pre-NSJ7 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 NSJ7 and later SPRs, you must set the `_RLD_LIB_PATH` as follows:

  For 32–bit NSJ7:

  `$JMETER_HOME/lib/oss32`

  For 64–bit NSJ7:

  `$JMETER_HOME/lib/oss64`

  For versions earlier than NSJ7:

  `$JMETER_HOME/lib/oss`

  where,

  `JMETER_HOME` is set to `/usr/tandem/hpjmeter`.

Example 21 Using `-agentlib` to run the JVM agent

```
$ java -Xms256m -Xmx512m -agentlib:jmeter myapp
```

Example 22 Setting `-Xbootclasspath`

```
$ java -agentlib:jmeter -Xbootclasspath/a:$JMETER_HOME/lib/agent.jar
myapp
```

NOTE: `-Xrunjmeter` is not supported on NSJ7.

Attaching to the JVM Agent of a running application

The dynamic attach feature is supported on NSJ7 from HPjmeter 4.3 or later versions. This feature lists the Java process using SPR from T2766H60^ACH and T2766 or T2866H70.

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 4.3 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 tuning and profiling” (page 94). 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 17 Supported –Xeprof options

time_slice	On NonStop Operating Systems, the time interval between the start and stop signals must be more than five minutes. This is because the Java thread switching process is slow due to the nonpreemptive nature of the NonStop Operating System. As a result, the asynchronous signals might be lost if they are posted to the NonStop Operating System 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 NSJ7.

Profiling with -agentlib:hprof

Table 18 (page 114) lists the supported -agentlib:hprof options.

Table 18 Supported -agentlib:hprof options

cpu=samples|times|old	cpu=samples is not supported on NonStop Operating Systems.

Analyzing garbage collection data

NonStop Java does not support G1 collector, therefore information available for G1 collector in this section is not applicable for NonStop Java.

Using visualizer functions

Using monitoring displays

Monitor memory and/or heap activity menu

Process resident memory

Not supported on NonStop system.

Process virtual memory

Not supported on NonStop system.

Data region resident memory

Not supported on NonStop system.

Data region virtual memory

Not supported on NonStop system.

Monitor JVM and/or system activity menu

System call rate

Not supported on NonStop system.

Troubleshooting

Identifying version numbers

Run the following commands to identify the version number for HPjmeter components:

- nodeagent for NonStop

  ```
  vproc $JMETER_HOME/bin/nodeagent
  ```

- JVM agent for pre-NSJ

  ```
  vproc $JMETER_HOME/lib/oss/libjmeter.so
  ```

- JVM agent for 32–bit NSJ7

  ```
  vproc $JMETER_HOME/lib/oss32/libjmeter.so
  ```

- JVM agent for 64–bit NSJ7

  ```
  vproc $JMETER_HOME/lib/oss64/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` for pre-NSJ7 versions. If NSJ7 is installed on the system and `/usr/tandem/java` is not available, then it uses `/usr/tandem/nssjava/jdk170_h70` for H-Series. There is no specific installer for NonStop Operating Systems that modifies the scripts, as performed 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`

On NonStop `HPjmeter`, a specific option `-ipaddress` is provided as follows:

Example Usage: `$JMETER_HOME/bin/nodeagent -ipaddress` *<IPv4 address>*

This option is used to specify the IP address on which the `nodeagent` can run. It supports only IPv4 address format.

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 4.3 User's Guide*. The features listed in Table A-2 for Java 6.x is also applicable for Java 7.x.

NOTE: This version of HPjmeter is supported only for NSJ6 and NSJ7.

NOTE:

- All instances of PROF in *HPjmeter 4.3 User's Guide* must be read as ALRM.
- All instances of signal 21 must be read as 14.
- The dynamic attach feature is not supported on NSJ7.

C Warranty and regulatory information

For important safety, environmental, and regulatory information, see *Safety and Compliance Information for Server, Storage, Power, Networking, and Rack Products*, available at **www.hpe.com/support/Safety-Compliance-EnterpriseProducts**.

Warranty information

HPE ProLiant and x86 Servers and Options

www.hpe.com/support/ProLiantServers-Warranties

HPE Enterprise Servers

www.hpe.com/support/EnterpriseServers-Warranties

HPE Storage Products

www.hpe.com/support/Storage-Warranties

HPE Networking Products

www.hpe.com/support/Networking-Warranties

Regulatory information

Belarus Kazakhstan Russia marking

Manufacturer and Local Representative Information

Manufacturer information:

- Hewlett Packard Enterprise Company, 3000 Hanover Street, Palo Alto, CA 94304 U.S.

Local representative information Russian:

- **Russia**:

  ООО «Хьюлетт Паккард Энтерпрайз», Российская Федерация, 125171, г. Москва, Ленинградское шоссе, 16А, стр.3, Телефон/факс: +7 495 797 35 00

- **Belarus**:

  ИООО «Хьюлетт-Паккард Бел», Республика Беларусь, 220030, г. Минск, ул. Интернациональная, 36-1, Телефон/факс: +375 17 392 28 20

- **Kazakhstan**:

  ТОО «Хьюлетт-Паккард (К)», Республика Казахстан, 050040, г. Алматы, Бостандыкский район, проспект Аль-Фараби, 77/7, Телефон/факс: + 7 727 355 35 52

Local representative information Kazakh:

- **Russia**:

  ЖШС "Хьюлетт Паккард Энтерпрайз", Ресей Федерациясы, 125171,
  Мәскеу, Ленинград тас жолы, 16А блок 3, Телефон/факс: +7 495 797 35 00

- **Belarus**:

  «HEWLETT-PACKARD Bel» ЖШС, Беларусь Республикасы, 220030, Минск қ.,
  Интернациональная көшесі, 36/1, Телефон/факс: +375 17 392 28 20

- **Kazakhstan**:

  ЖШС «Хьюлетт-Паккард (К)», Қазақстан Республикасы, 050040, Алматы к.,
  Бостандық ауданы, Әл-Фараби даңғ ылы, 77/7, Телефон/факс: +7 727 355 35 52

Manufacturing date:

The manufacturing date is defined by the serial number.

CCSYWWZZZZ (serial number format for this product)

Valid date formats include:

- YWW, where Y indicates the year counting from within each new decade, with 2000 as the starting point; for example, 238: 2 for 2002 and 38 for the week of September 9. In addition, 2010 is indicated by 0, 2011 by 1, 2012 by 2, 2013 by 3, and so forth.
- YYWW, where YY indicates the year, using a base year of 2000; for example, 0238: 02 for 2002 and 38 for the week of September 9.

Turkey RoHS material content declaration

Türkiye Cumhuriyeti: EEE Yönetmeliğine Uygundur

Ukraine RoHS material content declaration

Обладнання відповідає вимогам Технічного регламенту щодо обмеження використання деяких небезпечних речовин в електричному та електронному обладнанні, затвердженого постановою Кабінету Міністрів України від 3 грудня 2008 № 1057

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 Oracle **AWT Home Page**.

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

Branded — A Java virtual machine that Oracle 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 requestor.

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 NSJ7 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. However, this is not supported in L15.02. The following C3 compiler specific JVM options cannot be used with Java for L15.02:

- `-XX:+UseC3`
- `-XX:+C3Verbose`
- `-XX:+CompilerOpts`
- `-XX:+TimeCompilerFull`
- `-XX:TimeCompilerLog`
- `-XX:+CaliperOptoOutput`
- `-XX:+OfflineLoadProf`

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.

Concurrent Mark Sweep Garbage Collector (CMS GC) Concurrent Mark Sweep (CMS) is one of HotSpot JVM low pause garbage collectors. CMS can do most of its work for reclaiming memory concurrently with application (without stopping it).

Conformant A Java implementation is conformant if it passes all the tests for Java SE 7.

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 JVM 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.
Distributed Garbage Collector (DGC)	This refers to the NSJ7 implementation of parallel and CMS Garbage collectors where the garbage collection is performed by GC processes running on the same CPU on which the Java process is running.
DNS	See Domain Name Server (DNS).
Domain Name Server (DNS)	A Hewlett Packard Enterprise 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. NSJ7 has these driver implementations: JDBC Driver for SQL/MP(JDBC/MP) and JDBC Driver for SQL/MX (JDBC/MX).

E-F

Enscribe	Hewlett Packard Enterprise 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 HPE 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

G1 GC	The G1 collector is a server-style garbage collector, targeted for multi-processor machines with large memories. It meets garbage collection (GC) pause time goals with high probability, while achieving high throughput.
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, and 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 Hewlett Packard Enterprise product's command interpreter. Compare to OSS.
GUI	See graphical user interface (GUI).

H

Hotspot Virtual Machine	See Java Hotspot virtual machine.
HPE JDBC Driver for SQL/MP (JDBC/MP)	The product that provides access to SQL/MP and conforms to the JDBC API.

HPE JDBC Driver for SQL/MX (JDBC/MX) The product that provides access to SQL/MX and conforms to the JDBC API.

HPE NonStop Operating System The operating system for NonStop systems.

HPE 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.

HPE NonStop Server for Java, based on Java Standard Edition 7.0 The formal name of the NonStop Server for Java product whose Java HotSpot virtual machine conforms to the Java SE 7.0. See also NonStop Server for Java 7.0.

HPE NonStop SQL/MP (SQL/MP) HPE NonStop Structured Query Language/MP, the Hewlett Packard Enterprise relational database management system for NonStop servers.

HPE NonStop SQL/MX (SQL/MX) HPE NonStop Structured Query Language/MX, the Hewlett Packard Enterprise next-generation relational database management system for business-critical applications on NonStop servers.

HPE NonStop System Hewlett Packard Enterprise computers (hardware and software) that support the NonStop operating system.

HPE NonStop Technical Library The browser-based interface to NonStop computing technical information.

HPE NonStop Transaction Management Facility (TMF) A Hewlett Packard Enterprise 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.

HPE NonStop TS/MP (TS/MP) A Hewlett Packard Enterprise product that supports the creation of Pathway servers to access NonStop SQL/MP or Enscribe databases in an online transaction processing (OLTP) environment.

HPE 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 manuals (nodes) containing cross-references or links that, with the aid of an interactive browser, allow a reader to move easily from one manual 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. Hewlett Packard Enterprise 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 in 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.

IPU Instruction Processing Unit is responsible for organizing for program instructions to be fetched from memory, and executed, in an appropriate order.

IPv6 See Internet Protocol version 6 (IPv6).

ISO See International Organization for Standardization (ISO).

iTP Secure WebServer The Hewlett Packard Enterprise web server with which the NonStop Server for Java integrates using servlets.

J

`jar` The Java Archive tool, which combines multiple files in 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 to be conformant with the Oracle's 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 Heap	The contiguous block of memory from which JVM manages allocation of Java objects.
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 Oracle **Java IDL documentation**.
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 7)	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 Oracle JDK 7.0 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 7.0 (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 7.0 (Java SE).
Java Transaction API (JTA)	The Oracle 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 Oracle **JTA document**.
Java Transaction Service (JTS)	The transaction API, modeled on OMG's OTS. NSJ7 includes an implementation of the `jts.Current` interface.
Java Virtual Machine (JVM)	The process that loads, links, verifies, and interprets Java bytecode. NSJ7 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 JVM, 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 Oracle **JavaBeans document**.
JavaBeans Development Kit (BDK)	A set of tools for creating JavaBeans that is included with NSJ7.
`javac`	The Java compiler, which compiles Java source code in byte code. 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 HPE JDBC Driver for SQL/MP (JDBC/MP).

JDBC/MX See HPE 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. NSJ7 includes an implementation of `jts.Current`.

JVM See Java virtual machine (JVM).

JVM Heap JVM needs to allocate memory for its operation. This is allocated from C-heap (pre-NSJ7) on required basis. This memory is not garbage collected.

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 in the algorithm used to decode an encrypted message 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 in 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 7.0 (NSJ7) The informal name for the NonStop Server for Java products based on the Java Platform Standard Edition 7.0 product. This product is a Java environment that supports compact, concurrent, dynamic, and portable programs for the enterprise server.

NonStop Technical Library The browser-based interface to NonStop computing technical information. NonStop Technical Library replaces HPE Total Information Manager (TIM).

NSJ7 JDK containing both 32–bit and 64–bit NSJ based on Oracle JDK 7.0.

nsjps NonStop Java Virtual Machine Process Status tool. For more information, see *HPE NonStop for Java 7.0 Tools Reference Pages*.

NSK See HPE NonStop operating system.

NSKCOM A program management tool for swap space.

O

Object Management Group (OMG) The standards body that defined CORBA.

Object Serialization Oracle 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, in 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).

Old Generation (or Tenured Generation) In generational garbage collection, old or tenure generation is one of the generation in the Java Heap. The objects which are survived for threshold times (in scavenge or young generation) is promoted to old generation.

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 requestor 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.

Parallel GC — Parallel Garbage Collector, `Stop-the-world` GC where the GC runs in parallel using more than one thread.

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 (requestor) process that uses the Pathsend interface to communicate with a server process. A Pathsend process can be either a standard requestor, which initiates application requests, or a nested server, which is configured as a server class but acts as a requestor by making requests to other servers. Also called a Pathsend requestor.

Pathway — A group of software tools for developing and monitoring OLTP programs that use the client / server model. Servers are grouped in 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 — A Hewlett Packard Enterprise 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 Hewlett Packard Enterprise 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.

Requestor 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.

Scavenge Young generation garbage collection is also know as scavenge.

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 Oracle. 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 HPE NonStop SQL/MP.

SQL/MX See HPE 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 Oracle has defined and published an API standard. Some of the Standard Extensions might migrate in 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 HPE Tandem Advanced Command Language (TACL).

TCP/IP
See Transmission Control Protocol/Internet Protocol (TCP/IP).

Technical Documentation
Hewlett Packard Enterprise's technical documentation is found at **http://www.hpe.com/support/hpesc**.

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 Oracle Java SE 7.0.

`throw`
Java keyword used to raise an exception.

`throws`
Java keyword used to define the exceptions that a method can raise.

TMF
See HPE NonStop Transaction Management Facility (TMF)

TNS/E
The hardware platform based on the Intel® Itanium® architecture and the HPE 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 HPE NonStop operating system, and the software specific to that platform. Code might be PIC (position independent code) or non-PIC.

TNS/X
The hardware platform based on the Intel® Itanium® architecture and the HPE NonStop operating system, and the software specific to that platform.

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 Hewlett Packard Enterprise 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 HPE 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).
Young Generation	In generational garbage collection, young generation is one of the generation in the Java Heap from which memory for the new objects are allocated.

Index