NonStop Server for Java Message Service User’s Manual

Abstract

NonStop Server for Java™ Message Service (NSJMS) is an implementation of Sun Microsystems Java™ Message Service (JMS) API on HP NonStop™ S-series systems and HP Integrity NonStop NS-series systems. NSJMS utilizes the performance and reliability inherent in HP NonStop SQL/MX products to provide standards-based messaging for JMS clients running on HP NonStop servers. It is assumed the reader is already familiar with Sun Microsystems published specification, *Java Message Service*, Version 1.0.2.

As part of U64 program, a new 64-bit dll (`libnsjms_64.so`) is included in addition to the existing 32-bit dll (`libnsjms.so`)

Product Version

NSJMS 3.0

Supported Release Version Updates (RVUs)

This publication supports G06.18 and all subsequent G-series RVUs and H06.03 and all subsequent H-series RVUs until otherwise indicated by its replacement publication.

The U64 program support is present from H06.25.00 and all subsequent H-series RVUs unless otherwise indicated by its replacement publication. **U64 program does not support G-series RVUs.**

Part Number	Published
522356-006	February 2013

Document History

Part Number	Product Version	Published
522356-001	NSJMS 1.0	April 2002
522356-002	NSJMS 2.0	April 2003
522356-003	NSJMS 3.0	November 2003
522356-004	NSJMS 3.0	April 2005
522356-005	NSJMS 3.0	June 2011
522356-006	NSJMS 3.0	February 2013

Legal Notices

The information contained herein is subject to change without notice. The only warranties for HP 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. HP shall not be liable for technical or editorial errors or omissions contained herein.

Export of the information contained in this publication may require authorization from the U.S. Department of Commerce.

Microsoft, Windows, and Windows NT are U.S. registered trademarks of Microsoft Corporation.

Intel, Itanium, Pentium, and Celeron are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

Motif, OSF/1, UNIX, X/Open, and the "X" device are registered trademarks and IT DialTone and The Open Group are trademarks of The Open Group in the U.S. and other countries.

Open Software Foundation, OSF, the OSF logo, OSF/1, OSF/Motif, and Motif are trademarks of the Open Software Foundation, Inc.

> 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.

> This 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.
>
> Printed in the US

NonStop Server for Java Message Service User's Manual

Glossary Index Examples Figures

4. NSJMS and JMS Client Applications (continued)

5. Reliable Messaging Bridge

Examples

Figures

What’s New in This Manual

Manual Information

Abstract

NonStop Server for Java™ Message Service (NSJMS) is an implementation of Sun Microsystems Java™ Message Service (JMS) API on HP NonStop™ S-series systems and HP Integrity NonStop NS-series systems. NSJMS utilizes the performance and reliability inherent in HP NonStop SQL/MX products to provide standards-based messaging for JMS clients running on HP NonStop servers. It is assumed the reader is already familiar with Sun Microsystems published specification, *Java Message Service*, Version 1.0.2.

As part of U64 program, a new 64-bit dll (`libnsjms_64.so`) is included in addition to the existing 32-bit dll (`libnsjms.so`)

Product Version

NSJMS 3.0

Supported Release Version Updates (RVUs)

This publication supports G06.18 and all subsequent G-series RVUs and H06.03 and all subsequent H-series RVUs until otherwise indicated by its replacement publication.

The U64 program support is present from H06.25.00 and all subsequent H-series RVUs unless otherwise indicated by its replacement publication. **U64 program does not support G-series RVUs.**

Part Number	Published
522356-005	June 2011

Document History

Part Number	Product Version	Published
522356-001	NSJMS 1.0	April 2002
522356-002	NSJMS 2.0	April 2003
522356-003	NSJMS 3.0	November 2003
522356-004	NSJMS 3.0	April 2005
522356-006	NSJMS 3.0	February 2013

New and Changed Information

Updated system requirements at different places across the manual.

Updated these for NonStop NS-series system/server:

- NSJMS System Requirements on page 3-1
- NSJMS Installation Procedure on page 3-1
- Item 2 in Before You Begin the Installation on page 3-2
- Run the IPSetup Program on page 3-2
- Step 2 in Use DSM/SCM to Place the Product Files in /usr/tandem on page 3-4
- Steps 2 and 6 in Use DSM/SCM and PINSTALL to Place the Product Files in a User-Specified Installation Directory on page 3-4
- Step 2 in Use COPYOSS to Place the Product Files in /usr/tandem on page 3-5
- Step 1 through Step 5 in Install and Configure NSJMS on page 3-6

About This Manual

Overview

This manual explains how to install, operate, and manage NSJMS on HP NonStop systems.

Who Should Use This Manual

This manual is written for anyone who installs, manages, or monitors NSJMS on NonStop servers. It is assumed the reader is already familiar with Sun Microsystems published specification, *Java Message Service*, Version 1.0.2b, August 27, 2001.

How This Manual Is Organized

Section	Title	This section...
1	Introduction to NSJMS	provides an overview of NSJMS.
2	NSJMS Installation and Configuration for TNS/R	provides installation and configuration details for NSJMS.
3	NSJMS Installation and Configuration for TNS/E	provides installation and configuration details for NSJMS for TNS/E
4	NSJMS and JMS Client Applications	provides details on how NSJMS implements the JMS interfaces when performing JMS client tasks.
5	Reliable Messaging Bridge	provides information on how to configure, run, and monitor, a bridge Pathway configuration.
6	NSJMS Administrative Servlet Installation and Configuration	provides installation and configuration details for the NSJMS administrative servlet.
7	Managing the NSJMS Environment	provides details on managing NSJMS.
A	Error Reporting and Messages	provides details on NSJMS error messages.
	Glossary	provides a glossary of technical terms and abbreviations used throughout the manual.

Related Manuals

Depending on the tasks you are performing, other manuals you might need are:

- *SQL/MX Installation and Management Guide*
- *SQL/MP Installation and Management Guide*
- *SQL/MP Reference Manual*

- *iTP Secure WebServer System Administrator’s Guide*
- *NonStop NS-Series Planning Guide*
- *Open System Services User’s Guide*
- *NonStop Server for Java Programmer’s Reference*

Notation Conventions

Hypertext Links

Blue underline is used to indicate a hypertext link within text. By clicking a passage of text with a blue underline, you are taken to the location described. For example:

> This requirement is described under NSJMS System Requirements on page 2-1.

General Syntax Notation

The following list summarizes the notation conventions for syntax presentation in this manual.

UPPERCASE LETTERS. Uppercase letters indicate keywords and reserved words; enter these items exactly as shown. Items not enclosed in brackets are required. For example:

```
MAXATTACH
```

lowercase italic letters. Lowercase italic letters indicate variable items that you supply. Items not enclosed in brackets are required. For example:

`file-name`

computer type. `Computer type` letters within text indicate C and Open System Services (OSS) keywords and reserved words; enter these items exactly as shown. Items not enclosed in brackets are required. For example:

```
myfile.c
```

italic computer type. *`Italic computer type`* letters within text indicate C and Open System Services (OSS) variable items that you supply. Items not enclosed in brackets are required. For example:

`pathname`

[ ] Brackets. Brackets enclose optional syntax items. For example:

```
TERM [\system-name.]$terminal-name
```

```
INT[ERRUPTS]
```

A group of items enclosed in brackets is a list from which you can choose one item or none. The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
FC [ num  ]
   [ -num ]
   [ text ]
```

```
K [ X | D ] address
```

{ } Braces. A group of items enclosed in braces is a list from which you are required to choose one item. The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LISTOPENS PROCESS { $appl-mgr-name }
                  { $process-name  }
```

```
ALLOWSU { ON | OFF }
```

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

```
INSPECT { OFF | ON | SAVEABEND }
```

… 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:

```
M address [ , new-value ]…
```

```
[ - ] {0|1|2|3|4|5|6|7|8|9}…
```

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

```
"s-char…"
```

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

```
error := NEXTFILENAME ( file-name ) ;
```

```
LISTOPENS SU $process-name.#su-name
```

Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must enter as shown. For example:

```
"[" repetition-constant-list "]"
```

Item Spacing. Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma. For example:

```
CALL STEPMOM ( process-id ) ;
```

If there is no space between two items, spaces are not permitted. In the following example, there are no spaces permitted between the period and any other items:

```
$process-name.#su-name
```

Line Spacing. If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line. This spacing distinguishes items in a continuation line from items in a vertical list of selections. For example:

```
ALTER [ / OUT file-spec / ] LINE

   [ , attribute-spec ]…
```

!i and !o. In procedure calls, the !i notation follows an input parameter (one that passes data to the called procedure); the !o notation follows an output parameter (one that returns data to the calling program). For example:

```
CALL CHECKRESIZESEGMENT (  segment-id                       !i
                         , error        ) ;                 !o
```

!i,o. In procedure calls, the !i,o notation follows an input/output parameter (one that both passes data to the called procedure and returns data to the calling program). For example:

```
error := COMPRESSEDIT ( filenum ) ;                         !i,o
```

!i:i. In procedure calls, the !i:i notation follows an input string parameter that has a corresponding parameter specifying the length of the string in bytes. For example:

```
error := FILENAME_COMPARE_ (  filename1:length           !i:i
                            , filename2:length ) ;       !i:i
```

!o:i. In procedure calls, the !o:i notation follows an output buffer parameter that has a corresponding input parameter specifying the maximum length of the output buffer in bytes. For example:

```
error := FILE_GETINFO_ (  filenum                           !i
                        , [ filename:maxlen ] ) ;           !o:i
```

Notation for Messages

The following list summarizes the notation conventions for the presentation of displayed messages in this manual.

Bold Text. Bold text in an example indicates user input entered at the terminal. For example:

```
ENTER RUN CODE

?123

CODE RECEIVED:      123.00
```

The user must press the Return key after typing the input.

Nonitalic text. Nonitalic letters, numbers, and punctuation indicate text that is displayed or returned exactly as shown. For example:

```
Backup Up.
```

lowercase italic letters. Lowercase italic letters indicate variable items whose values are displayed or returned. For example:

p-register

process-name

[ ] Brackets. Brackets enclose items that are sometimes, but not always, displayed. For example:

```
Event number = number [ Subject = first-subject-value ]
```

A group of items enclosed in brackets is a list of all possible items that can be displayed, of which one or none might actually be displayed. The items in the list might be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
proc-name trapped [ in SQL | in SQL file system ]
```

{ } Braces. A group of items enclosed in braces is a list of all possible items that can be displayed, of which one is actually displayed. The items in the list might be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
obj-type obj-name state changed to state, caused by
{ Object | Operator | Service }
```

```
process-name State changed from old-objstate to objstate
{ Operator Request. }
{ Unknown.          }
```

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

```
Transfer status: { OK | Failed }
```

% Percent Sign. A percent sign precedes a number that is not in decimal notation. The % notation precedes an octal number. The %B notation precedes a binary number. The %H notation precedes a hexadecimal number. For example:

```
%005400
```

```
%B101111
```

```
%H2F
```

```
P=%p-register E=%e-register
```

Change Bar Notation

Change bars are used to indicate substantive differences between this manual and its preceding version. Change bars are vertical rules placed in the right margin of changed portions of text, figures, tables, examples, and so on. Change bars highlight new or revised information. For example:

> The message types specified in the REPORT clause are different in the COBOL environment and the Common Run-Time Environment (CRE).
>
> The CRE has many new message types and some new message type codes for old message types. In the CRE, the message type SYSTEM includes all messages except LOGICAL-CLOSE and LOGICAL-OPEN.

1 Introduction to NSJMS

NSJMS is the JMS provider that implements Sun Microsystems JMS API, version 1.0.2, on NonStop systems. NSJMS uses the performance and reliability inherent in SQL/MX products to provide standards-based messaging for local clients running on NonStop servers. NSJMS enables scalability and load distribution through horizontal partitioning and fault-tolerance through process-pair technology.

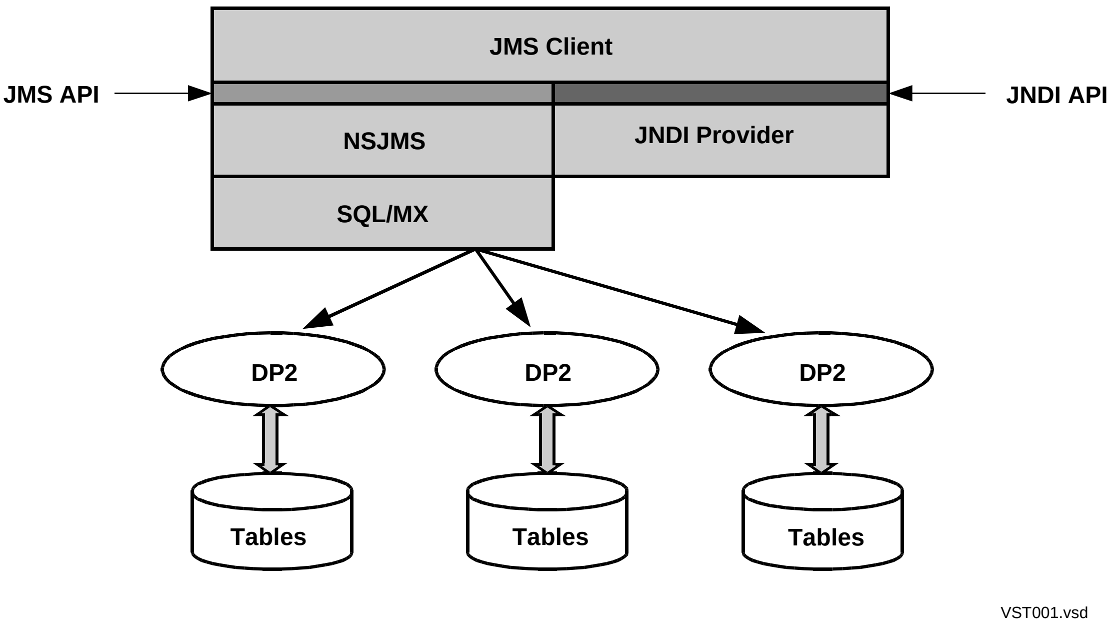
Features and Functions of NSJMS

- Implements the JMS API on NonStop systems.
- Uses the publish and subscribe features of HP 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++ AP,I which implements a subset of the functionality provided by the Sun JMS API, and is used by C++ client applications running on a NonStop system to interoperate with other JMS clients.
- Uses the NSJMS administrative utility to manage the NSJMS environment. You can invoke the utility through a command-line interface or XML interface.
- Sample programs are provided to demonstrate the use of NSJMS.

NSJMS Client Messaging Components

The JMS API provides a generic set of interfaces that enable JMS clients to exchange messages. NSJMS is the implementation of the interfaces contained in the JMS API on a NonStop system. The components that enable NSJMS client messaging are shown in Figure 1-1 on page 1-2. Using NSJMS as a JMS provider is described in Section 4, NSJMS and JMS Client Applications.

Figure 1-1. NSJMS Client Messaging on a NonStop System



JMS Client

A JMS client is a user-written application that uses NSJMS to exchange messages with other JMS client applications.

JMS API

The Java JMS API provides a generic set of interfaces that enables JMS clients to exchange messages.

JNDI API

The JNDI API provides a generic set of interfaces that enables JMS clients to access naming and directory services.

NSJMS

NSJMS is the JMS provider that provides an implementation of the JMS API on a NonStop system.

JNDI Provider

The JNDI provider implements the naming and directory services.

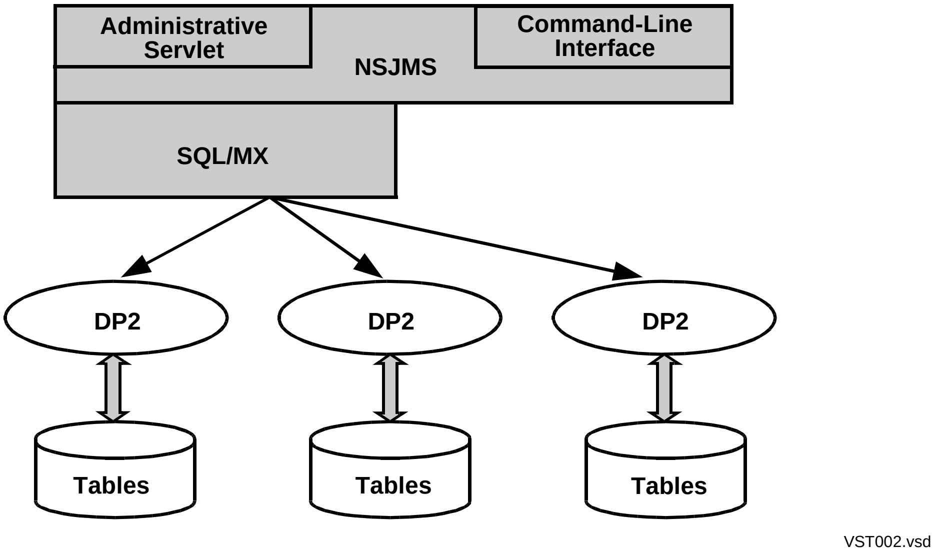
SQL/MX

SQL/MX is the HP relational database management system that provides access to large distributed databases. JMS clients use SQL/MX to access messages and information stored in NSJMS SQL databases.

Administrative Utility Components

The administrative utility is a Java-based tool used to manage the NSJMS database. The component parts that access and maintain the NSJMS database are shown in Figure 1-2. The administrative utility is described in detail in Section 7, Managing the NSJMS Environment.

Figure 1-2. NSJMS Administrative Utility



Administrative Servlet

The optional administrative servlet provides a programmatic interface for the administrative utility to access and manage the NSJMS database.

Command-Line Interface

The command-line interface provides a local command-line tool for the administrative utility to access and manage the NSJMS objects.

NSJMS

NSJMS is the JMS provider that provides an implementation of the JMS API on a NonStop system.

SQL/MX

SQL/MX is the HP relational database management system that provides access to large distributed databases. JMS clients use SQL/MX to access messages and information stored in NSJMS SQL databases.

JNDI API

The JNDI API provides a generic set of interfaces that enables the administrative utility to access naming and directory services.

JNDI Provider

The JNDI provider implements the naming and directory services.

Supported Platform

This publication supports H06.25.00 and all subsequent H-series RVUs unless otherwise indicated by its replacement publication. This program does not support G-series RVUs.

Compliance Information

NSJMS conforms to Sun Microsystems published specification, *Java Message Service*, Version 1.0.2, except as noted. The specification is available on the Sun Microsystems Java Message Service (JMS) Web site (http://java.sun.com/products/jms/docs.html).

- JMS Application Server facilities refers to server-side facilities in a JMS implementation that could be used by an expert application. These features are optional interfaces (ConnectionConsumer, ServerSession and SeverSessionPool) and are not available in the NSJMS implementation.
- A JMS provider can provide Java Transaction API (JTA)-aware connections and sessions by using the XAConnectionFactory, XAConnection, XASession, and so on. On the operating system, the transaction integration of NSJMS and other products is provided by HP NonStop Transaction Management Facility (TMF). Although these XA interfaces are not implemented in NSJMS, you can use JTA to start or commit transactions which involve NSJMS.
- The JMS specification defines the priority value as 0 (zero), the lowest priority, and nine, the highest. Priority does not affect delivery of messages in the NSJMS implementation.

2

NSJMS Installation and Configuration for TNS/R

NSJMS System Requirements

Hardware

HP NonStop S-series server

Software

When Running NonStop Server for Java (T0083V31)

△ **Caution.** Every time you install a new version of the HP NonStop Server for Java (NSJ), you must run the NSJ Makefile to bind the NSJMS libraries with the Java Virtual Machine (JVM). See the *NonStop Server for Java Programmer's Reference* for additional information.

HP NonStop operating system (T9050G06), G06.18 or later RVU

NonStop OSS (T8620D40)

NonStop Server for Java (T0083V31 PVU AAP or later)

NonStop SQL/MX (T1050G08 PVU AAI or later)

NonStop SQL/MX EXE (T1051G08 PVU AAI or later)

NonStop SQL/MP (T9095G07)

JDBC/MX (T1225V20)

JDBC/MP (T1227V10 PVU AAA)

Java Naming and Directory Interface (JNDI) 1.2.1 FS Context Service Provider

▲ **WARNING.** The JNDI 1.2.1 FS Context Service Provider from Sun Microsystems Java Naming and Directory Interface (JNDI) Web site (http://java.sun.com/products/jndi/index.html) must be downloaded and extracted on your workstation. These files are required for NSJMS.

When Running NonStop Server for Java (T2766V10)

△ **Caution.** Every time you install a new version of the NonStop Server for Java (NSJ), you must run the NSJ Makefile to bind the NSJMS libraries with the Java Virtual Machine (JVM). For more information, see the *NonStop Server for Java Programmer's Reference*.

HP NonStop operating system (T9050G06), G06.20 or later RVU

NonStop OSS (T8620D40)

NonStop Server for Java (T2766V10 or later)

NonStop SQL/MX (T1050G08 PVU AAI or later)

NonStop SQL/MX EXE (T1051G08 PVU AAI or later)

NonStop SQL/MP (T9095G07)

JDBC/MX (T1225V30)

JDBC/MP (T1227V30)

Java Naming and Directory Interface (JNDI) 1.2.1 FS Context Service Provider

▲ **WARNING.** The JNDI 1.2.1 FS Context Service Provider from Sun Microsystems Java Naming and Directory Interface (JNDI) Web site (http://java.sun.com/products/jndi/index.html) must be downloaded and extracted on your workstation. These files are required for NSJMS.

Minimum Disk Space

1 Megabyte

NSJMS Installation Procedure

This subsection describes what you need to do before installing NSJMS and explains how to install and configure NSJMS on a NonStop S-series system. It also includes instructions for creating and testing sample programs.

The following installation instructions are correct as of the time this manual was published; however, the README.TXT file or Softdoc supersedes the information here.

Before You Begin the Installation

- Check that you have downloaded and extracted `FS Context Service Provider` on to your workstation from the Sun Microsystems' JNDI web site (http://java.sun.com/products/jndi/index.html).
- Review the `README` file on the product CD to ensure you have the correct version for all products installed on your NonStop S-series system.
- Check that your site meets the minimum hardware and software requirements for the installation utility and also any product-specific installation requirements. (See the `README` on the product CD).
- Review the `USRGUIDE.PDF` file in the `NSK_SW` subdirectory on the product CD. This file contains the *IPSetup User's Guide* which provides instructions for using IPSetup, a utility that installs Independent Products.

- Determine whether you will use DSM/SCM to move files to Installation Subvolumes (ISVs) after files are placed on your workstation. For additional information about using DSM/SCM, see the *DSM/SCM User's Guide*.

Note. Using DSM/SCM is optional for G06.18 or later RVUs, but is recommended.

Run the IPSetup Program

Use the IPSetup program to place the NSJMS components on your host NonStop S-series server. If TCP/IP and FTP are unavailable, or if you have problems with automatic file placement, see the *IPSetup User's Guide* (the `USRGUIDE.PDF` file in the `NSK_SW` subdirectory on the product CD) for information about how to manually place NonStop Kernel files.

△ **Caution.** Every time you use the Install product to install a release of the operating system software, you must reinstall your Independent Product. Installing a release using the Install product can result in overwriting this Independent Product with older versions of software. This problem does not occur if you are using DSM/SCM to install releases.

To Run IPSetup

Exit all other Windows applications before placing NSJMS software on the host system.

1. Open the product CD by double clicking on the CD drive.
2. Click the **View Readme** file button. Setup opens the `README` file in Notepad. Be sure to review the entire `README` file before proceeding.
3. Click the **IPSetup** button to launch IPSetup. The program displays a Welcome Screen and a License Agreement screen. To continue the installation, click **Next** on each of these screens.
4. On the Placement Options screen select the NonStop Kernel RISC option.
   - If you plan on using DSM/SCM, check the box for "Use DSM/SCM to complete installation on host." Click **Next.**
   - If you do not plan on using DSM/SCM, uncheck the box for "Use DSM/SCM to complete installation on host." Click **Next.**
5. On the Product Selection screen, highlight NonStop Server for Java Message Service as the product you want to install. Click the **ADD** button. Click **Next**.
6. Log on using a user ID that can write to the `/usr` OSS directory (for example, super.super) by following the instructions on the Host Information screen. You can use either the system name or the system IP address to log on. Click **Next**.
7. On the Host Target screen you can either accept the default locations for Work and Backup subvolumes or browse to locations of your choice. Click **Next** when you are satisfied with the locations.

8. On the Host File Placement screen you can either accept the default disk locations or browse to locations of your choice. Click **Next** when you are satisfied with the locations.
9. On the Placement Manifest screen review the file locations. You can click Back to go back and change the file locations. When you are satisfied with the locations, click **Next**. This step can take a few minutes to complete.
10. On the Placement Complete screen you can choose to view the release documentation or to launch DSM/SCM. You should review the release documentation.
11. After you review the release documentation, click **Finish** to complete running IPSetup.

Unpax the NSJMS Product Files

Select one of these processes to unpax the NSJMS product files.

If you will be using DSM/SCM:

- Use DSM/SCM to Place the Product Files in /usr/tandem on page 2-4
- Use DSM/SCM and PINSTALL to Place the Product Files in a User-Specified Installation Directory on page 2-5

If you will not be using DSM/SCM

- Use COPYOSS to Place the Product Files in /usr/tandem on page 2-6

Use DSM/SCM to Place the Product Files in `/usr/tandem`

If you use DSM/SCM, this process will unpax the NSJMS PAX file, `T1251PAX`, in the standard OSS location, that is, in `/usr/tandem`. For additional information about using DSM/SCM, see the *DSM/SCM User’s Guide*.

1. RECEIVE the product files from disk (DSV locations) or tape.
2. **Check** the `Manage OSS Files` option for the target configuration within the DSM/SCM Planner Interface.

   Note. You must be installing NSJMS on a NonStop S-series system running G06.18 or later RVU.

3. COPY the received product files to a new software revision of the configuration you want to update.
4. BUILD and APPLY the configuration revision.

5. Run ZPHIRNM to perform the RENAME step.

Note. If the option `Manage OSS Files` was not selected in the DSM/SCM planner interface in Step 2 above, the NSJMS PAX file is in the Guardian subvolume `$`*`ISV`*`.ZOSSUTL` (where *`ISV`* is the name of your installation subvolume). Use `COPYOSS` to extract and place the contents of the NSJMS PAX file into the OSS file system. (Go to Use COPYOSS to Place the Product Files in /usr/tandem on page 2-6).

6. Go to Install and Configure NSJMS on page 2-6 for detailed instructions about running the install script and configuring NSJMS.

Use DSM/SCM and PINSTALL to Place the Product Files in a User-Specified Installation Directory

If you use DSM/SCM and PINSTALL, this process will unpax the NSJMS PAX file, `T1251PAX`, in a user-specified installation directory. For additional information about using DSM/SCM, see the *DSM/SCM User’s Guide*. For additional information about using PINSTALL, see the *Open System Services Management and Operations Guide.*

1. RECEIVE the product files from disk (DSV locations) or tape.
2. **Uncheck** the `Manage OSS Files` option for the target configuration within the DSM/SCM Planner Interface.

   Note. You must be installing NSJMS on a NonStop S-series system running G06.18 or later RVU.

3. COPY the received product files to a new software revision of the configuration you want to update.
4. BUILD and APPLY the configuration revision.
5. Run ZPHIRNM to perform the RENAME step.
6. Use PINSTALL to unpax the NSJMS PAX file into the OSS file system.

   a. On your NonStop S-series server, log on as Super.Super:

   ```
   TACL> LOGON SUPER.SUPER
   ```

   b. Navigate to the Guardian subvolume `$`*`ISV`*`.ZOSSUTL`:

   ```
   TACL> VOLUME $ISV.ZOSSUTL
   ```

   where *`ISV`* is the name of your installation subvolume.

   c. Unpax `T1251PAX` using the PINSTALL utility:

   ```
   TACL> PINSTALL -s:/usr/tandem:install-dir: -rvf
         T1251PAX
   ```

   PINSTALL extracts the files contained in `T1251PAX` and places them in the version-specific OSS directory *`install-dir`*`/nsjms/`*`version`*, where *install-dir* is the user-specified installation directory and *version* is the vproc of this release (for example, T1251V30_30SEP2003_V30).

Note. Running the PINSTALL command does not affect any environment currently running on your NonStop S-series server.

7. Go to Install and Configure NSJMS on page 2-6 for detailed instructions about running the install script and configuring NSJMS.

Use COPYOSS to Place the Product Files in /usr/tandem

If you use COPYOSS, this process will unpax the NSJMS PAX file, T1251PAX, in the standard OSS location, that is, in /usr/tandem. For additional information about using COPYOSS, see the *Open System Services Management and Operations Guide*.

1. Ensure that the NSJMS PAX file, T1251PAX, has been transferred into the DSV location.
2. Use COPYOSS to unpax the NSJMS PAX file into the OSS file system.
    a. On your NonStop S-series server, log on as Super.Super:

    ```
    TACL> LOGON SUPER.SUPER
    ```

    b. Navigate to the Guardian subvolume $*ISV*.ZOSSUTL:

    ```
    TACL> VOLUME $ISV.ZOSSUTL
    ```

    where *ISV* is the name of your installation subvolume.

    c. Unpax T1251PAX using the TACL macro COPYOSS:

    ```
    TACL> COPYOSS T1251PAX
    ```

    COPYOSS extracts the files contained in T1251PAX and places them in the version-specific OSS directory /usr/tandem/nsjms/*version*, where *version* is the vproc of this release (for example, T1251V30_30SEP2003_V30).

    Note. Running the copyoss command does not affect any environment currently running on your NonStop S-series server.

3. Go to Install and Configure NSJMS on this page for detailed instructions about running the install script and configuring NSJMS.

Install and Configure NSJMS

1. Run the /usr/tandem/nsjms/*version* installation script located in the OSS file system directory.

    Note. If you are running NonStop Server for Java (T0083V31), the NSJ Makefile INCLUDE_JDBCMX variable must be set to yes, and the user ID must match the user ID that was used to install the JVM NSJ software. This Makefile is located in /usr/tandem/java/install.

For example:

```
TACL> LOGON SUPER.SUPER
TACL> OSH
OSH: cd /usr/tandem/nsjms/version
OSH: chmod u+x install
OSH: ./install
```

Note. Run the installation script only once after you unpax the NSJMS product files into the version-specific OSS directory. Run the installation script a second time only when you reinstall the T1251 NSJMS libraries or class files.

After running the installation script:

- The nsjms.jar file is in the /usr/tandem/nsjms/*version*/lib directory.
- For NonStop Server for Java (T0083V31), the NSJMS JNI code (libnsjms.so and libnsjms_u64.so for 64-bit support) is in the /usr/tandem/java/jre/lib/oss/posix_threads directory.

  For NonStop Server for Java (T2766H70), the NSJMS JNI code (libnsjms.so and libnsjms_u64.so for 64-bit support) is in the /usr/tandem/java_public_lib directory.
- The Java binaries have been automatically rebuilt to include the NSJMS JNI code (libnsjms.so and libnsjms_u64.so for 64-bit support).
- The JVM (/usr/tandem/java) has been updated and relinked to include this version of the NSJMS software libraries and class files.

2. Verify that the installation script did not report any errors.

△ **Caution.** After verifying installation, do not delete or modify the version-specific directory (/usr/tandem/nsjms/*version*) or its subdirectories because OSS symbolic links point back to the version-specific directory tree. If any part of the version-specific directory tree is deleted or modified, you must reinstall NSJMS by unpaxing the NSJMS PAX file. (For more information, see Unpax the NSJMS Product Files on page 2-4.)

3. Copy the JNDI files from your workstation to the NSJMS directory:

   Using FTP in binary mode, transfer the providerutil.jar file and the fscontext.jar file from your workstation to the /usr/tandem/nsjms/*version*/lib directory where *version* is the vproc of this release (for example, T1251V30_30SEP2003_V30).

   Note. The jndi.jar file is included in the NSJ 2.0 release and resides in the /usr/tandem/java/jre/lib/ext directory.

4. Verify the vproc of the Java Binary.

   A T1251 vproc should match with the *version* portion of the installation directory.

   Note. If there is no match, an installation error occurred. Correct the error before proceeding.

For example:

- If NonStop Server for Java (T0083V31) is installed use these commands:

```
OSH: cd /usr/tandem/nssjava/version/bin/oss/posix_threads
OSH: vproc java
```

  where *`version`* is the vproc of this JVM release (for example, `jdk131_v10`)

- If NonStop Server for Java (T2766V10) is installed use these commands:

```
OSH: cd /usr/tandem/nssjava/version/bin
OSH: vproc java
```

  where *`version`* is the vproc of this JVM release (for example, `jdk141_v10`)

The vproc should contain a version procedure similar to:

```
Version procedure:  T1251V30_30SEP2003_BASE_LIBNSJMS
```

Note. The remaining installation steps do not require super.super privileges. However, to maintain a secure environment, you should choose or create a user ID that is configured for the OSS environment and from under which you run your JMS applications. Use this user ID to perform the post-installation steps.

5. Use the `export` command to update the NSJMS_HOME variable. For example:

```
OSH: cd /usr/tandem/nsjms/version
OSH: export NSJMS_HOME=$PWD
```

6. Use the `export` command to update the CLASSPATH variable. For example:

```
OSH: cd /usr/tandem/nsjms/version
OSH: export JAVA_HOME=/usr/tandem/java
OSH: export JREHOME=$JAVA_HOME/jre
OSH: export CLASSPATH=$NSJMS_HOME:$NSJMS_HOME/examples:$CLASSPATH
OSH: export CLASSPATH=`echo $NSJMS_HOME/lib/*.jar | tr ' ' ':'`:$CLASSPATH
```

7. Verify that you have access to an existing SQL catalog, or create a new catalog.

   NSJMS uses SQL tables to store messages and destination information.

   To create a new SQL catalog:

```
OSH: gtacl -p sqlci 'create catalog $volume.subvol;exit;'
```

   For additional information about the requirements for creating SQL catalogs, see the *SQL/MP Installation and Management Guide*.

8. Update SQL Catalog and Table Location:

   - If the NSJMS SQL tables reside on the same subvolume as the SQL catalog, using an editor of your choice, update the SQL catalog location in the `/usr/tandem/nsjms/`*`version`*`/creatsql` file to reflect the *volume.subvol* of the existing or newly created SQL catalog specified in Step 7 on page 2-8. For example:

Change:

```
VOLUME;
```

To:

```
VOLUME $volume.subvol;
```

- If the NSJMS SQL tables reside on a different subvolume than the SQL catalog, using an editor of your choice, add the SQL catalog location to the `/usr/tandem/nsjms/`*`version`*`/creatsql` file by adding a CATALOG command referring to the catalog from Step 7 on page 2-8, and update the VOLUME command to refer to the *volume.subvol* where the database tables will reside. For example:

  Change:

  ```
  VOLUME;
  ```

  To:

  ```
  CATALOG $volume1.subvol1
  VOLUME $volume2.subvol2
  ```

9. Use this command to create the NSJMS SQL tables. For example:

```
OSH: gtacl -p sqlci < creatsql
```

Note. Database tables, by default, support a maximum of 2 gigabytes (GB) on each partition. If larger partitions are required, use the FORMAT2ENABLED attribute when creating the tables. For information about using the FORMAT2ENABLED attribute within the CREATE TABLE statement, see the *SQL/MP Reference Manual*.

10. Update the Property Files:

    a. Using an editor of your choice, update these properties in the `nsjms.properties` file:

       1. Set the `Deadmsg.deletecount` property value to 100. For example:

          ```
          Deadmsg.deletecount=100
          ```

       2. Set the `Database.volsubvol` property to the location of the NSJMS SQL tables from Step 8 on page 2-8. For example:

          ```
          Database.volsubvol=$volume.subvol
          ```

       The `nsjms.properties` file is described on page 2-12 and contains properties that are used to configure NSJMS. The property values are read by the NSJMS implementation once during startup.

    b. If the JNDI objects will be stored in a directory other than the default directory (file:`///usr/tandem/nsjms/`*`version`*`/jndi_object_store`), using an editor of your choice, update the `java.naming.provider.url` property in the `jndi.properties` file with the path for the new directory.

The `jndi.properties` file is described on page 2-15 and contains values that are necessary to use a simple JNDI file service provider on the operating system.

The `nsjms.properties` file and the `jndi.properties` file contain default key and value combinations and require few or no changes.

Note. `nsjms.properties` and `jndi.properties` are loaded from directories in the CLASSPATH. You can copy the property files to other directories and edit them for different users or applications. However, you then must update CLASSPATH to include the directories containing the edited files.

11. Add the JNDI and create a sample queue and topic by using these commands:

```
OSH: cd /usr/tandem/nsjms/version
OSH: java com.tandem.nsjms.admin.JmsAdmin
NSJMS Admin started.
nsjms-> add jndi
Jndi Initialized
nsjms-> add queue Q1
Destination added.
nsjms-> add topic T1
Destination added.
nsjms-> exit
```

12. Compile the sample programs by using these commands. The sample programs use the JNDI object store and destinations created in Step 11 on page 2-10:

```
OSH: cd examples
OSH: javac *.java
```

13. Test the sample programs:

You will need two OSS shells to test each sample program.

Note. Before testing each sample program, set the NSJMS_HOME variable and the CLASSPATH variable in each shell; see Step 5 on page 2-8 and Step 6 on page 2-8.

In the sample programs, you can enable the non-blocking features of SQL/MX by issuing the `sqlmx_nowait` system property from the command line using the -D option:

```
java -Dsqlmx_nowait=on client-name client-attribute
```

where *`client-name`* is the name of the sample program and *`client-attribute`* specifies the sample programs attributes. For example:

```
OSH: java -Dsqlmx_nowait=on SimpleReceiver -queue Q1 -ackmode 1
```

For additional information on using the `sqlmx_nowait` system property, see Threading on page 4-15.

a. Test the Point-to-Point sample by using these commands:

In the first shell:

```
OSH: cd examples
OSH: java SimpleReceiver -queue Q1 -ackmode 1
```

In the second shell:

```
OSH: cd examples
OSH: java SimpleSender -queue Q1 -count 5
```

Your Point-to-Point sample output will be similar to:

```
OSH: cd examples
OSH: java SimpleReceiver -queue Q1 -ackmode 1
OSH: RCV Message: Sending message 0
OSH: RCV Message: Sending message 1
OSH: RCV Message: Sending message 2
OSH: RCV Message: Sending message 3
OSH: RCV Message: Sending message 4
```

```
OSH: cd examples
OSH: java SimpleSender -queue Q1 -count 5
OSH: Publishing: Sending message 0
OSH: Publishing: Sending message 1
OSH: Publishing: Sending message 2
OSH: Publishing: Sending message 3
OSH: Publishing: Sending message 4
OSH: Finished Publishing 5 messages
```

b. Test the Publish and Subscribe sample by using these commands:

In the first shell:

```
OSH: cd examples
OSH: java SimpleConsumer -topic T1 -ackmode 1
```

In the second shell:

```
OSH: cd examples
OSH: java SimplePublisher -topic T1 -count 5
```

Your Publish and Subscribe sample output will be similar to:

```
OSH: cd examples
OSH: java SimpleConsumer -topic T1 -ackmode 1
Message listener has been set
OSH: Received message #1: Publishing this message 0
OSH: Received message #2: Publishing this message 1
OSH: Received message #3: Publishing this message 2
OSH: Received message #4: Publishing this message 3
OSH: Received message #5: Publishing this message 4
```

```
OSH: cd examples
OSH: java SimplePublisher -topic T1 -count 5
OSH: Finished publishing 5 messages
```

NSJMS Directory Structure

After completing the product installation, the `/usr/tandem/nsjms/`*`version`* directory contains the following files and subdirectories. Included are detailed descriptions of the properties contained in the `nsjms.properties` file and the `jndi.properties` file:

- `nsjms.properties` – Contains these properties that are used to configure NSJMS:

  Note. All properties must appear in the properties file in a *`PropertyKey`*`=`*`Value`* format. For example, `Logger.type=FileLogger`.

  - Database.delay

    Default: 1000

    Values: *`milliseconds`*

    Value Description: *`milliseconds`*—The time (in milliseconds) that the thread delays after a stream timeout occurs.

    Example: `Database.delay=1000`

  - Database.tabletimeout

    Default: 3000

    Values: *`milliseconds`*

    Value Descriptions: *`milliseconds`*—The time (in milliseconds) for a table lock timeout. If *`milliseconds`* elapses before a lock is acquired on a table, the statement fails, and SQL returns a file system error 73 (disk file or record is locked).

    Example: `Database.tabletimeout=3000`

  - Database.timeout

    Default: 100

    Values: *`milliseconds`*

    Value Descriptions: *`milliseconds`*—The time (in milliseconds) in which the stream selects a block for new messages before a timeout occurs.

    Example: `Database.timeout=100`

- Database.volsubvol

  Default: *`vol.subvol`*

  Values: *`vol.subvol`*

  Value Description: *`vol.subvol`*—The volume and subvolume of the database tables.

  Example: `Database.volsubvol=$data01.nsjms`

- Deadmsg.deletecount

  Default: 0

  Values: 0 through 5000

  Value Description: Dead messages are messages in topics which have been read by all existing subscribers of the topic (including durable subscribers), and are therefore unusable to any subscriber.

  The messages are deleted periodically as part of the regular receive process. The frequency of deletion is determined by the delete-count value. A delete count of 100 means that after every 100th message received from a topic, the client will automatically delete all dead messages. When 0 is specified, no dead messages are deleted.

  Example: `Deadmsg.deletecount=100`

- Logger.filename

  Default: nsjms.log

  Value: *`name`*

  Value Description: *`name`* is the file location that contains log information.

  Example: `Logger.filename=nsjms.log`

- Logger.loglevel

  Default: warning

  Values: fatal | error | exception | warning | info | debug

  Value Descriptions: fatal—Logs a fatal message that leads to abnormal termination.

  error—Logs messages that do not necessarily lead to abnormal termination but still should be noted, such as a message about an incorrect configuration parameter. A log level of `error` displays messages that have log levels of fatal and error

  exception—Logs a specified exception.

  warning—Logs messages that do not effect the integrity of the JMS client but should be brought to the JMS client's attention. A log level of `warning` displays messages that have log levels of fatal, error, exception, and warning,

  info—Logs messages that do not imply an error but could indicate an important event, such as a message that the JMS client is connected. A log level of `info` displays messages that have log levels of fatal, error, exception, warning, and info.

  debug—Logs messages useful during a debugging session. Setting Logger.loglevel to `debug` can substantially degrade performance and should be used only when requested by support personnel. A log level of `debug` displays all levels of messages: fatal, error, exception, warning, info, and debug.

  Example: `Logger.loglevel=warning`

- Logger.type

  Default: FileLogger

  Values: FileLogger | ConsoleLogger

  Value Descriptions: FileLogger—Logs to the file specified in Logger.filename.

  ConsoleLogger—Logs to the client home term.

  Example: `Logger.type=FileLogger`

  - Servlet.xmlvalidate

    Default: false

    Values: true | false

    Value Description: true—Specifies that validation is performed on the administrative servlet XML request and reply using a DTD.

    false—Specifies that no validation is performed on the administrative servlet XML administrative request and reply using a DTD.

    Example: `Servlet.xmlvalidation=false`

  - Stats.active

    Default: false

    Values: true | false

    Value Descriptions: true—Updates the statistics database at each Stats.interval.

    false—Does not update the statistics database at each Stats.interval.

    Example: `Stats.active=false`

  - Stats.interval

    Default: 60

    Values: 1 through 60

    Value Description: 1 through 60—The interval (in minutes) in which Stats are recorded.

    Example: `Stats.interval=60`

- `jndi.properties` – Contains these properties that specify the JNDI initial context and the location of JNDI objects:

  Note. All properties must appear in the properties file in a `PropertyKey=Value` format. For example, `java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory`

  - Java.naming.factory.initial

    Default: `com.sun.jndi.fscontext.RefFSContextFactory`

    Value: `com.sun.jndi.fscontext.RefFSContextFactory`

    Value Description: `com.sun.jndi.fscontext.RefFSContextFactory` specifies the JNDI initial context to be the file system.

    Example: `java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory`

  - Java.naming.provider.url

    Default: file:`////usr/tandem/nsjms/`*`version`*`/jndi_object_store`

    Value: file:`////usr/tandem/nsjms/`*`version`*`/jndi_object_store`

    Value Description: file:`////usr/tandem/nsjms/`*`version`*`/jndi_object_store` specifies the actual directory where the JNDI objects are stored.

    Example: `java.naming.provider.url=file:////usr/tandem/nsjms/`*`version`*`/jndi_object_store`

- `creatsql` – Contains the sqlci database creation commands.
- `install` – Contains the installation script that moves the `nsjms.jar` and `libnsjms.so` and `libnsjms_u64.so` for 64-bit support files to the appropriate JVM locations, runs the NSJ Makefile to bind the NSJMS libraries with the JVM, and creates the SQL/MP tables.
- `AdminHelp.txt` – Contains the NSJMS administrative utility help file.
- `License.txt` – Contains the license agreement for the NSJMS software.
- `Apache.txt` – Contains the license agreement for the apache Software foundation.
- `Exolab.txt` – Contains the license agreement for ExOffice Technologies, Inc.
- `/jndi_object_store` – Contains the JNDI bindings for Administered objects.
- `/lib` – Contains
  - NSJMS library (`libnsjms.so` and `libnsjms_u64.so` for 64-bit support). This library contains the native library.
  - NSJMS Client Classes (`nsjms.jar`). This file contains the core classes for NSJMS that implement the JAVA JMS API.

- `/examples` – Contains
    - `SimpleSender.java`. This sample program sends simple messages to a queue.
    - `SimpleReceiver.java`. This sample program receives simple messages from a queue.
    - `SimplePublisher.java`. This sample program publishes simple messages to a topic.
    - `SimpleConsumer.java`. This sample program subscribes and consumes messages from a topic.
    - `JmsAdminServlet.html`. This sample program uses the administrative servlet.
    - `web.xml`. This file is used with the administrative servlet sample program.
- `/bridge` – Contains
    - `btool.sh`. The btool.sh script is used to launch the reliable messaging bridge configuration tool which is used to map between queues and topics and to define the persistent environment parameters.
    - `start_template.txt`. This file is used by the configuration tool to generate an executable Pathway start script.
    - `stop_template.txt`. This file is used by the configuration tool to generate an executable Pathway stop script.

SQL/MP Table Partitioning

You can optionally partition SQL/MP tables to achieve greater scalability by adding range partitioning to the message table. Partitioning spreads the space required by queues or topics across several disks and also spreads the DP2 execution load across several processors. For information about planning and executing table partitioning, see the *SQL/MP Installation and Management Guide*.

In these examples, the SQLCI ALTER TABLE command partitions SQL/MP tables. An existing message table residing on $VOL1 is split into two partitions resulting in the new partition residing on $VOL2:

```
SQLCI>> ALTER TABLE $VOL1.NSJMS.MESSAGE
+> PARTONLY MOVE FROM KEY (10000,0) TO $VOL2.NSJMS.MESSAGE
+> CATALOG $VOL1.NSJMS
+> EXTENT (1000,200);
--- SQL OPERATION COMPLETE
```

After completion of the previous SQLCI ALTER TABLE command, the table has two partitions that have these starting destination ID values:

Partition Location	**Starting Destination ID**
$VOL1	0
$VOL2	10000

An existing partition within a message table that has three partitions is split. Before the split operation, the message table has three partitions that have these starting destination ID values:

Partition Location	**Starting Destination ID**
$VOL1	0
$VOL2	10000
$VOL3	20000

By using this SQLCI ALTER TABLE command, $VOL1 splits into two partitions, resulting in the new partition residing on $VOL4. Destinations that have ID values greater than or equal to 5000 but less than the starting destination value for $VOL2 (10000) are relocated to $VOL4.

```
SQLCI>> ALTER TABLE $VOL1.NSJMS.MESSAGE
+> PARTONLY MOVE FROM KEY (5000,0) TO $VOL4.NSJMS.MESSAGE
+> CATALOG $VOL1.NSJMS
+> EXTENT (1000,200);
--- SQL OPERATION COMPLETE
```

After completion of the above SQLCI ALTER TABLE command, the message table has four partitions that have these starting destination ID values:

Partition Location	**Starting Destination ID**
$VOL1	0
$VOL4	5000
$VOL2	10000
$VOL3	20000

Adding a Message-Table Dummy Record

You can add a dummy record to a message table when you create a new partition for a message table. A dummy record is a record that is never deleted from a message table. Dummy records improve performance when tables often transition from the empty to non-empty condition by avoiding the overhead of collapsing and expanding a message table. A dummy record is assigned a destination ID value that is never assigned as a destination.

The `creatsql` file that is provided with NSJMS includes this command to insert a dummy record that has a destination ID value of `0`.

```
SQLCI>> INSERT INTO MESSAGE VALUES (0, 0, 1000000000000000000,0,'');
```

The destination ID value of `0` is never assigned as a destination.

When you create a new partition that has a destination ID range of 5000 through 10000, this command adds a dummy record that has a destination ID value of 5000:

```
SQLCI>> INSERT INTO MESSAGE VALUES (5000, 0, 1000000000000000000,0,'');
```

Then use the administrative utility ADD Command on page 7-2 to assign destination IDs for the new partition starting at 5001.

Overflow Table Partitioning

You can also partition an overflow table by using commands similar to those used to partition SQL/MP tables. The overflow table holds messages larger than approximately 3900 bytes. If large messages are used, the overflow table should be partitioned to match the message table.

Uninstalling NSJMS

To remove an installed version of NSJMS, follow these steps from within the OSS environment:

1. Remove the NSJMS library archive.

    - If NonStop Server for Java (T0083V31) is installed use these commands:

```
OSH: cd  /usr/tandem/java/jre/lib/oss/posix_threads
OSH: rm libnsjms.a
```

    - If NonStop Server for Java (T2766V10) is installed use these commands:

```
OSH: cd  /usr/tandem/java_public_lib
OSH: rm libnsjms.a
```

2. Rerun the make utility by using these commands:

```
OSH: cd  /usr/tandem/java/install
OSH: make
```

3. Remove the NSJMS Administrative servlet.

    a. Remove the modifications made to the iTP Secure WebServer servlet.config file in Step 3 on page 6-2.

b. Delete the iTP Secure WebServer NSJMS directory by using these commands:

```
OSH: cd /usr/tandem/webserver/samples/Servlets
OSH: rm nsjms
```

c. Restart the iTPSecure WebServer by using these commands:

```
OSH: cd /usr/tandem/webserver/conf
OSH: ./restart
```

Uninstall Cleanup Steps

After uninstalling NSJMS, you can optionally perform these cleanup steps to remove the version-specific NSJMS SQL tables and the NSJMS install directory:

1. Remove the NSJMS SQL tables and the catalog specified by the `VOLUME` parameter in the `creatsql` file on `/usr/tandem/nsjms/`*`version`*; where version represents a specific version of NSJMS.

2. Remove the NSJMS install directory by using these commands:

```
OSH: cd  /usr/tandem/nsjms
OSH: rm -R version
```

where *`version`* represents a specific version of NSJMS.

Note. Removal of this directory will require any future installation of NSJMS to start with Run the IPSetup Program on page 2-3.

3

NSJMS Installation and Configuration for TNS/E

NSJMS System Requirements

Hardware

HP NonStop NS-series server

Software

When Running NonStop Server for Java (T2766H10)

HP NonStop operating system (T9050H01)

NonStop OSS (T8620H01)

NonStop Server for Java (T2766H70)

NonStop SQL/MX (T1050H32)

NonStop SQL/MX EXE (T1051H32)

NonStop SQL/MP (TXXXXXXX)

JDBC/MX (T1275H32)

Java Naming and Directory Interface (JNDI) 1.2.1 FS Context Service Provider

▲ **WARNING.** The JNDI 1.2.1 FS Context Service Provider from Sun Microsystems Java Naming and Directory Interface (JNDI) Web site (http://java.sun.com/products/jndi/index.html) must be downloaded and extracted on your workstation. These files are required for NSJMS.

Note. NonStop SQL/MP driver is not required from H06.25 release onwards.

Minimum Disk Space

1 Megabyte

NSJMS Installation Procedure

This subsection describes what you need to do before installing NSJMS and explains how to install and configure NSJMS on a NonStop NS-series system. It also includes instructions for creating and testing sample programs.

The following installation instructions are correct as of the time this manual was published; however, the README.TXT file or Softdoc supersedes the information here.

Before You Begin the Installation

- Check that you have downloaded and extracted `FS Context Service Provider` on to your workstation from the Sun Microsystems' JNDI web site (http://java.sun.com/products/jndi/index.html).
- Review the `README` file on the product CD to ensure you have the correct version for all products installed on your NonStop NS-series system.
- Check that your site meets the minimum hardware and software requirements for the installation utility and also any product-specific installation requirements. (For more information, see the `README` on the product CD.)
- Review the `USRGUIDE.PDF` file in the `NSK_SW` subdirectory on the product CD. This file contains the *IPSetup User's Guide* that provides instructions for using IPSetup, a utility that installs Independent Products.
- Determine whether you will use DSM/SCM to move files to Installation Subvolumes (ISVs) after files are placed on your workstation. For additional information about using DSM/SCM, see the *DSM/SCM User's Guide*.

Note. Using DSM/SCM is optional for G06.18 or later RVUs, but is recommended.

Run the IPSetup Program

Use the IPSetup program to place the NSJMS components on your host NonStop NS-series server. If TCP/IP and FTP are unavailable, or if you have problems with automatic file placement, see the *IPSetup User's Guide* (the `USRGUIDE.PDF` file in the `NSK_SW` subdirectory on the product CD) for information about how to manually place files on the NonStop system.

△ **Caution.** Every time you use the Install product to install a release of the operating system software, you must reinstall your Independent Product. Installing a release using the Install product can result in overwriting this Independent Product with older versions of software. This problem does not occur if you are using DSM/SCM to install releases.

To Run IPSetup

Exit all other Windows applications before placing NSJMS software on the host system.

1. Open the product CD by double clicking on the CD drive.
2. Click the **View Readme** file button. Setup opens the `README` file in Notepad. Be sure to review the entire `README` file before proceeding.

3. Click the **IPSetup** button to launch IPSetup. The program displays a Welcome Screen and a License Agreement screen. To continue the installation, click **Next** on each of these screens.
4. On the Placement Options screen select the NonStop Kernel RISC option.
   - If you plan on using DSM/SCM, check the box for “Use DSM/SCM to complete installation on host.” Click **Next**.
   - If you do not plan on using DSM/SCM, uncheck the box for “Use DSM/SCM to complete installation on host.” Click **Next**.
5. On the Product Selection screen, highlight NonStop Server for JMS as the product you want to install. Click the **ADD** button. Click **Next**.
6. Log on using a user ID that can write to the `/usr` OSS directory (for example, super.super) by following the instructions on the Host Information screen. You can use either the system name or the system IP address to log on. Click **Next**.
7. On the Host Target screen you can either accept the default locations for Work and Backup subvolumes or browse to locations of your choice. Click **Next** when you are satisfied with the locations.
8. On the Host File Placement screen you can either accept the default disk locations or browse to locations of your choice. Click **Next** when you are satisfied with the locations.
9. On the Placement Manifest screen review the file locations. You can click Back to go back and change the file locations. When you are satisfied with the locations, click **Next**. This step can take a few minutes to complete.
10. On the Placement Complete screen you can choose to view the release documentation or to launch DSM/SCM. You should review the release documentation.
11. After you review the release documentation, click **Finish** to complete running IPSetup.

Unpax the NSJMS Product Files

Select one of these processes to unpax the NSJMS product files.

If you will be using DSM/SCM:

- Use DSM/SCM to Place the Product Files in /usr/tandem on page 3-4
- Use DSM/SCM and PINSTALL to Place the Product Files in a User-Specified Installation Directory on page 3-4

If you will not be using DSM/SCM:

- Use COPYOSS to Place the Product Files in /usr/tandem on page 3-5

Use DSM/SCM to Place the Product Files in `/usr/tandem`

If you use DSM/SCM, this process will unpax the NSJMS PAX file, `T1251PAX`, in the standard OSS location, that is, in `/usr/tandem`. For additional information about using DSM/SCM, see the *DSM/SCM User’s Guide*.

1. RECEIVE the product files from disk (DSV locations) or tape.
2. **Check** the `Manage OSS Files` option for the target configuration within the DSM/SCM Planner Interface.

   Note. You must be installing NSJMS on a NonStop NS-series system running H06.03 or later RVU. For U64 support you must be installing NSJMS on a NonStop NS-series system running H06.25 or later RVU.

3. COPY the received product files to a new revision of the software configuration you want to update.
4. BUILD and APPLY the configuration revision.
5. Run ZPHIRNM to perform the RENAME step.

Note. If the option `Manage OSS Files` was not selected in the DSM/SCM planner interface in Step 2 above, the NSJMS PAX file is in the Guardian subvolume `$`*`ISV`*`.ZOSSUTL` (where *`ISV`* is the name of your installation subvolume). Use `COPYOSS` to extract and place the contents of the NSJMS PAX file into the OSS file system. (Go to Use COPYOSS to Place the Product Files in /usr/tandem on page 3-5).

6. Go to Install and Configure NSJMS on page 3-6 for detailed instructions about running the install script and configuring NSJMS.

Use DSM/SCM and PINSTALL to Place the Product Files in a User-Specified Installation Directory

If you use DSM/SCM and PINSTALL, this process will unpax the NSJMS PAX file, `T1251PAX`, in a user-specified installation directory. For additional information about using DSM/SCM, see the *DSM/SCM User’s Guide*. For additional information about using PINSTALL, see the *Open System Services Management and Operations Guide.*

1. RECEIVE the product files from disk (DSV locations) or tape.
2. **Uncheck** the `Manage OSS Files` option for the target configuration within the DSM/SCM Planner Interface.

   Note. You must be installing NSJMS on a NonStop NS-series system running H06.25 or later RVU.

3. COPY the received product files to a new revision of the software configuration you want to update.
4. BUILD and APPLY the configuration revision.

5. Run ZPHIRNM to perform the RENAME step.
6. Use PINSTALL to unpax the NSJMS PAX file into the OSS file system.

   a. On your NonStop NS-series server, log on as Super.Super:

      ```
      TACL> LOGON SUPER.SUPER
      ```

   b. Navigate to the Guardian subvolume `$`*`ISV`*`.ZOSSUTL`:

      ```
      TACL> VOLUME $ISV.ZOSSUTL
      ```

      where *`ISV`* is the name of your installation subvolume.

   c. Unpax `T1251PAX` using the PINSTALL utility:

      ```
      TACL> PINSTALL -s:/usr/tandem:install-dir: -rvf
            T1251PAX
      ```

   PINSTALL extracts the files contained in `T1251PAX` and places them in the version-specific OSS directory *`install-dir`*`/nsjms/`*`version`*, where *`install-dir`* is the user-specified installation directory and *`version`* for this RVU is: (for example, `T1251H10_30MAY2005_H10`).

   Note. Running the `PINSTALL` command does not affect any environment currently running on your NonStop NS-series server.

7. Go to Install and Configure NSJMS on page 3-6 for detailed instructions about running the install script and configuring NSJMS.

Use COPYOSS to Place the Product Files in `/usr/tandem`

If you use COPYOSS, this process will unpax the NSJMS PAX file, `T1251PAX`, in the standard OSS location, that is, in `/usr/tandem`. For additional information about using COPYOSS, see the *Open System Services Management and Operations Guide*.

1. Ensure that the NSJMS PAX file, `T1251PAX`, has been transferred into the DSV location.
2. Use COPYOSS to unpax the NSJMS PAX file into the OSS file system.

   a. On your NonStop NS-series server, log on as Super.Super:

      ```
      TACL> LOGON SUPER.SUPER
      ```

   b. Navigate to the Guardian subvolume `$`*`ISV`*`.ZOSSUTL`:

      ```
      TACL> VOLUME $ISV.ZOSSUTL
      ```

      where *`ISV`* is the name of your installation subvolume.

   c. Unpax `T1251PAX` using the TACL macro COPYOSS:

      ```
      TACL> COPYOSS T1251PAX
      ```

COPYOSS extracts the files contained in T1251PAX and places them in the version-specific OSS directory `/usr/tandem/nsjms/`*`version`*, where *`version`* for this RVU is: (for example, `T1251H10_30MAY2005_H10`).

Note. Running the `copyoss` command does not affect any environment currently running on your NonStop NS-series server.

---

3. Go to Install and Configure NSJMS on page 3-6 for detailed instructions about running the install script and configuring NSJMS.

Install and Configure NSJMS

1. Change the directory:

   ```
   OSH: cd /<jms-install-dir>/nsjms/T1251H10_30MAY2005_H10/lib
   ```

   where `<jms-install-dir>` is the standard or non-standard NSJMS install location.

2. Copy the JNDI files from your workstation to the NSJMS directory:

   Using FTP in binary mode, transfer the `providerutil.jar` file and the `fscontext.jar` file from your workstation to the `/<jms-install-dir>/nsjms/`*`version`*`/lib` directory, where *`version`* for this RVU is: (for example, `T1251H10_30MAY2005_H10`).

Note. The installation steps do not require super.super privileges. However, to maintain a secure environment, you should choose or create a user ID that is configured for the OSS environment and from under which you run your JMS applications. Use this user ID to perform the post-installation steps.

---

3. TACL>OSH

   ```
   OSH: cd /<jms-install-dir>/nsjms/T1251H10_30MAY2005_H10/lib
   OSH: chmod u+x libnsjms.so
   OSH: chmod u+x libnsjms_64.so
   ```

4. Use the `export` command to update the NSJMS_HOME variable. For example:

```
OSH: cd /<jms-install-dir>/nsjms/T1251H10_30MAY2005_H10
OSH: export NSJMS_HOME=$PWD
```

5. Use the `export` command to update the CLASSPATH variable. For example:

```
If T2766H10 is installed in the standard location: (For U64 support, Java
SPR T2766H70 or above is supported):
OSH: export JAVA_HOME=/usr/tandem/java
OSH: export JREHOME=$JAVA_HOME/jre
OSH: export CLASSPATH=$NSJMS_HOME:$NSJMS_HOME/examples:$CLASSPATH
OSH: export CLASSPATH=`echo $NSJMS_HOME/lib/*.jar | tr ' ' ':'`:$CLASSPATH
OSH: export _RLD_LIB_PATH=$NSJMS_HOME/lib
```

```
If T2766H10 is installed in a non-standard location:
OSH: export JAVA_HOME=/<java-install-dir>/java
OSH: export JREHOME=$JAVA_HOME/jre
OSH: export CLASSPATH=$NSJMS_HOME:$NSJMS_HOME/examples:$CLASSPATH
OSH: export CLASSPATH=`echo $NSJMS_HOME/lib/*.jar | tr ' ' ':'`:$CLASSPATH
OSH: export _RLD_LIB_PATH=$NSJMS_HOME/lib
```

6. Verify that you have access to an existing SQL catalog, or create a new catalog.

   NSJMS uses SQL tables to store messages and destination information.

   To create a new SQL catalog:

```
OSH: gtacl -p sqlci 'create catalog $volume.subvol;exit;'
```

   For additional information about the requirements for creating SQL catalogs, see the *SQL/MP Installation and Management Guide*.

7. Update SQL Catalog and Table Location:

   - If the NSJMS SQL tables reside on the same subvolume as the SQL catalog, using an editor of your choice, update the SQL catalog location in the /usr/tandem/nsjms/*version*/creatsql file to reflect the *volume*.*subvol* of the existing or newly created SQL catalog specified in Step 6 on page 3-7. For example:

     Change:

     ```
     VOLUME;
     ```

     To:

     ```
     VOLUME $volume.subvol;
     ```

   - If the NSJMS SQL tables will reside on a different subvolume than the SQL catalog, using an editor of your choice, add the SQL catalog location to the /usr/tandem/nsjms/*version*/creatsql file by adding a CATALOG command referring to the catalog from Step 6 on page 3-7, and update the VOLUME command to refer to the *volume*.*subvol* where the database tables will reside. For example:

     Change:

     ```
     VOLUME;
     ```

     To:

     ```
     CATALOG $volume1.subvol1
     VOLUME $volume2.subvol2
     ```

8. Use this command to create the NSJMS SQL tables. For example:

```
OSH: gtacl -p sqlci < creatsql
```

Note. Database tables, by default, support a maximum of 2 gigabytes (GB) on each partition. If larger partitions are required, use the FORMAT2ENABLED attribute when creating the tables. For information about using the FORMAT2ENABLED attribute within the CREATE TABLE statement, see the *SQL/MP Reference Manual*.

9. Update the Property Files:

   a. Using an editor of your choice, update these properties in the `nsjms.properties` file:

      1. Set the `Deadmsg.deletecount` property value to 100. For example:

         `Deadmsg.deletecount=100`

      2. Set the `Database.volsubvol` property to the location of the NSJMS SQL tables from Step 7 on page 3-7. For example:

         `Database.volsubvol=$`*`volume.subvol`*

      The `nsjms.properties` file is described on page 3-10 and contains properties that are used to configure NSJMS. The property values are read by the NSJMS implementation once during startup.

   b. If the JNDI objects will be stored in a directory other than the default directory (file:`///usr/tandem/nsjms/`*`version`*`/jndi_object_store`), using an editor of your choice, update the `java.naming.provider.url` property in the `jndi.properties` file with the path for the new directory.

      The `jndi.properties` file is described on page 3-14 and contains values that are necessary to use a simple JNDI file service provider on the operating system.

   The `nsjms.properties` file and the `jndi.properties` file contain default key and value combinations and require few or no changes.

Note. `nsjms.properties` and `jndi.properties` are loaded from directories in the CLASSPATH. You can copy the property files to other directories and edit them for different users or applications. However, you then must update CLASSPATH to include the directories containing the edited files.

10. Add the JNDI and create a sample queue and topic by using these commands:

```
OSH: cd /usr/tandem/nsjms/version
OSH: java com.tandem.nsjms.admin.JmsAdmin
NSJMS Admin started.
nsjms-> add jndi
Jndi Initialized
nsjms-> add queue Q1
Destination added.
nsjms-> add topic T1
Destination added.
nsjms-> exit
```

11. Compile the sample programs by using these commands. The sample programs use the JNDI object store and destinations created in Step 10 on page 3-8:

```
OSH: cd examples
OSH: javac *.java
```

12. Test the sample programs:

    You will need two OSS shells to test each sample program.

    Note. Before testing each sample program, set the NSJMS_HOME variable and the CLASSPATH variable in each shell; see Step 4 on page 3-6 and Step 5 on page 3-6.

    In the sample programs, you can enable the non-blocking features of SQL/MX by issuing the `sqlmx_nowait` system property from the command line using the -D option:

    `java -Dsqlmx_nowait=on` ***`client-name client-attribute`***

    where *`client-name`* is the name of the sample program and *`client-attribute`* specifies the sample programs attributes. For example:

```
OSH: java -Dsqlmx_nowait=on SimpleReceiver -queue Q1 -ackmode 1
```

    For additional information on using the `sqlmx_nowait` system property, see Threading on page 4-15.

    a. Test the Point-to-Point sample by using these commands:

       In the first shell:

```
OSH: cd examples
OSH: java SimpleReceiver -queue Q1 -ackmode 1
```

       In the second shell:

```
OSH: cd examples
OSH: java SimpleSender -queue Q1 -count 5
```

       Your Point-to-Point sample output will be similar to:

```
OSH: cd examples
OSH: java SimpleReceiver -queue Q1 -ackmode 1
OSH: RCV Message: Sending message 0
OSH: RCV Message: Sending message 1
OSH: RCV Message: Sending message 2
OSH: RCV Message: Sending message 3
OSH: RCV Message: Sending message 4
```

```
OSH: cd examples
OSH: java SimpleSender -queue Q1 -count 5
OSH: Publishing: Sending message 0
OSH: Publishing: Sending message 1
OSH: Publishing: Sending message 2
OSH: Publishing: Sending message 3
OSH: Publishing: Sending message 4
OSH: Finished Publishing 5 messages
```

b. Test the Publish and Subscribe sample by using these commands:

In the first shell:

```
OSH: cd examples
OSH: java SimpleConsumer -topic T1 -ackmode 1
```

In the second shell:

```
OSH: cd examples
OSH: java SimplePublisher -topic T1 -count 5
```

Your Publish and Subscribe sample output will be similar to:

```
OSH: cd examples
OSH: java SimpleConsumer -topic T1 -ackmode 1
Message listener has been set
OSH: Received message #1: Publishing this message 0
OSH: Received message #2: Publishing this message 1
OSH: Received message #3: Publishing this message 2
OSH: Received message #4: Publishing this message 3
OSH: Received message #5: Publishing this message 4
```

```
OSH: cd examples
OSH: java SimplePublisher -topic T1 -count 5
OSH: Finished publishing 5 messages
```

NSJMS Directory Structure

After completing the product installation, the `/usr/tandem/nsjms/`*`version`* directory contains these files and subdirectories. Included are detailed descriptions of the properties contained in the `nsjms.properties` file and the `jndi.properties` file:

- `nsjms.properties` – Contains these properties that are used to configure NSJMS:

  Note. All properties must appear in the properties file in a *`PropertyKey`*`=`*`Value`* format. For example, `Logger.type=FileLogger`.

- Database.delay

  Default: 1000

  Values: *milliseconds*

  Value Description: *milliseconds*—The time (in milliseconds) that the thread delays after a stream timeout occurs.

  Example: `Database.delay=1000`

- Database.tabletimeout

  Default: 3000

  Values: *milliseconds*

  Value Descriptions: *milliseconds*—The time (in milliseconds) for a table lock timeout. If *milliseconds* elapses before a lock is acquired on a table, the statement fails, and SQL returns a file system error 73 (disk file or record is locked).

  Example: `Database.tabletimeout=3000`

- Database.timeout

  Default: 100

  Values: *milliseconds*

  Value Descriptions: *milliseconds*—The time (in milliseconds) in which the stream selects a block for new messages before a timeout occurs.

  Example: `Database.timeout=100`

- Database.volsubvol

  Default: *vol.subvol*

  Values: *vol.subvol*

  Value Description: *vol.subvol*—The volume and subvolume of the database tables.

  Example: `Database.volsubvol=$data01.nsjms`

- Deadmsg.deletecount

  Default: 0

  Values: 0 through 5000

  Value Description: Dead messages are messages in topics which have been read by all existing subscribers of the topic (including durable subscribers), and are therefore unusable to any subscriber.

  The messages are deleted periodically as part of the regular receive process. The frequency of deletion is determined by the delete-count value. A delete count of 100 means that after every 100th message received from a topic, the client will automatically delete all dead messages. When 0 is specified, no dead messages are deleted.

  Example: `Deadmsg.deletecount=100`

- Logger.filename

  Default: nsjms.log

  Value: *`name`*

  Value Description: *`name`* is the file location that contains log information.

  Example: `Logger.filename=nsjms.log`

- Logger.loglevel

Default:	warning
Values:	fatal | error | exception | warning | info | debug
Value Descriptions:	fatal—Logs a fatal message that leads to abnormal termination. error—Logs messages that do not necessarily lead to abnormal termination but still should be noted, such as a message about an incorrect configuration parameter. A log level of `error` displays messages that have log levels of fatal and error exception—Logs a specified exception. warning—Logs messages that do not effect the integrity of the JMS client but should be brought to the JMS client's attention. A log level of `warning` displays messages that have log levels of fatal, error, exception, and warning, info—Logs messages that do not imply an error but could indicate an important event, such as a message that the JMS client is connected. A log level of `info` displays messages that have log levels of fatal, error, exception, warning, and info. debug—Logs messages useful during a debugging session. Setting Logger.loglevel to `debug` can substantially degrade performance and should be used only when requested by support personnel. A log level of `debug` displays all levels of messages: fatal, error, exception, warning, info, and debug.
Example:	`Logger.loglevel=warning`

- Logger.type

Default:	FileLogger
Values:	FileLogger | ConsoleLogger
Value Descriptions:	FileLogger—Logs to the file specified in Logger.filename. ConsoleLogger—Logs to the client home term.
Example:	`Logger.type=FileLogger`

- Servlet.xmlvalidate

  Default: false

  Values: true | false

  Value Description: true—Specifies that validation is performed on the administrative servlet XML request and reply using a DTD.

  false—Specifies that no validation is performed on the administrative servlet XML administrative request and reply using a DTD.

  Example: `Servlet.xmlvalidation=false`

- Stats.active

  Default: false

  Values: true | false

  Value Descriptions: true—Updates the statistics database at each Stats.interval.

  false—Does not update the statistics database at each Stats.interval.

  Example: `Stats.active=false`

- Stats.interval

  Default: 60

  Values: 1 through 60

  Value Description: 1 through 60—The interval (in minutes) in which Stats are recorded.

  Example: `Stats.interval=60`

- `jndi.properties` – Contains these properties that specify the JNDI initial context and the location of JNDI objects:

Note. All properties must appear in the properties file in a `PropertyKey=Value` format. For example,

```
java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFact
ory
```

- Java.naming.factory.initial

  Default: `com.sun.jndi.fscontext.RefFSContextFactory`

  Value: `com.sun.jndi.fscontext.RefFSContextFactory`

  Value Description: `com.sun.jndi.fscontext.RefFSContextFactory` specifies the JNDI initial context to be the file system.

  Example: `java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory`

- Java.naming.provider.url

  Default: file:`////usr/tandem/nsjms/`*`version`*`/jndi_object_store`

  Value: file:`////usr/tandem/nsjms/`*`version`*`/jndi_object_store`

  Value Description: file:`////usr/tandem/nsjms/`*`version`*`/jndi_object_store` specifies the actual directory where the JNDI objects are stored.

  Example: `java.naming.provider.url=file:////usr/tandem/nsjms/`*`version`*`/jndi_object_store`

- `creatsql` – Contains the sqlci database creation commands.
- `AdminHelp.txt` – Contains the NSJMS administrative utility help file.
- `License.txt` – Contains the license agreement for the NSJMS software.
- `Apache.txt` – Contains the license agreement for the apache Software foundation.
- `Exolab.txt` – Contains the license agreement for ExOffice Technologies, Inc.
- `/jndi_object_store` – Contains the JNDI bindings for Administered objects.
- `/lib` – Contains
  - NSJMS library (`libnsjms.so and libnsjms_64.so`).
  - NSJMS Client Classes (`nsjms.jar`). This file contains the core classes for NSJMS that implement the JAVA JMS API.

- `/examples` – Contains
  - `SimpleSender.java`. This sample program sends simple messages to a queue.
  - `SimpleReceiver.java`. This sample program receives simple messages from a queue.
  - `SimplePublisher.java`. This sample program publishes simple messages to a topic.
  - `SimpleConsumer.java`. This sample program subscribes and consumes messages from a topic.
  - `JmsAdminServlet.html`. This sample program uses the administrative servlet.
  - `web.xml`. This file is used with the administrative servlet sample program.
- `/bridge` – Contains
  - `btool.sh`. The btool.sh script is used to launch the reliable messaging bridge configuration tool which is used to map between queues and topics and to define the persistent environment parameters.
  - `start_template.txt`. This file is used by the configuration tool to generate an executable Pathway start script.
  - `stop_template.txt`. This file is used by the configuration tool to generate an executable Pathway stop script.

SQL/MP Table Partitioning

You can optionally partition SQL/MP tables to achieve greater scalability by adding range partitioning to the message table. Partitioning spreads the space required by queues or topics across several disks and also spreads the DP2 execution load across several processors. For information about planning and executing table partitioning, see the *SQL/MP Installation and Management Guide* .

In these examples, the SQLCI ALTER TABLE command partitions SQL/MP tables. An existing message table residing on $VOL1 is split into two partitions resulting in the new partition residing on $VOL2:

```
SQLCI>> ALTER TABLE $VOL1.NSJMS.MESSAGE
+> PARTONLY MOVE FROM KEY (10000,0) TO $VOL2.NSJMS.MESSAGE
+> CATALOG $VOL1.NSJMS
+> EXTENT (1000,200);
--- SQL OPERATION COMPLETE
```

After completion of the previous SQLCI ALTER TABLE command, the table has two partitions that have these starting destination ID values:

Partition Location	**Starting Destination ID**
$VOL1	0
$VOL2	10000

An existing partition within a message table that has three partitions is split. Before the split operation, the message table has three partitions that have these starting destination ID values:

Partition Location	**Starting Destination ID**
$VOL1	0
$VOL2	10000
$VOL3	20000

By using this SQLCI ALTER TABLE command, $VOL1 splits into two partitions, resulting in the new partition residing on $VOL4. Destinations that have ID values greater than or equal to 5000 but less than the starting destination value for $VOL2 (10000) are relocated to $VOL4.

```
SQLCI>> ALTER TABLE $VOL1.NSJMS.MESSAGE
+> PARTONLY MOVE FROM KEY (5000,0) TO $VOL4.NSJMS.MESSAGE
+> CATALOG $VOL1.NSJMS
+> EXTENT (1000,200);
--- SQL OPERATION COMPLETE
```

After completion of the above SQLCI ALTER TABLE command, the message table has four partitions that have these starting destination ID values:

Partition Location	**Starting Destination ID**
$VOL1	0
$VOL4	5000
$VOL2	10000
$VOL3	20000

Adding a Message-Table Dummy Record

You can add a dummy record to a message table when you create a new partition for a message table. A dummy record is a record that is never deleted from a message table. Dummy records improve performance when tables often transition from the empty to non-empty condition by avoiding the overhead of collapsing and expanding a message table. A dummy record is assigned a destination ID value that is never assigned as a destination.

The `creatsql` file that is provided with NSJMS includes this command to insert a dummy record that has a destination ID value of `0`.

```
SQLCI>> INSERT INTO MESSAGE VALUES (0, 0, 1000000000000000000,0,'');
```

The destination ID value of 0 is never assigned as a destination.

When you create a new partition that has a destination ID range of 5000 to 10000, this command adds a dummy record that has a destination ID value of 5000:

```
SQLCI>> INSERT INTO MESSAGE VALUES (5000, 0, 1000000000000000000,0,'');
```

Then use the administrative utility ADD Command on page 7-2 to assign destination IDs for the new partition starting at 5001.

Overflow Table Partitioning

You can also partition an overflow table by using commands similar to those used to partition SQL/MP tables. The overflow table holds messages larger than approximately 3900 bytes. If large messages are used, the overflow table should be partitioned to match the message table.

4

NSJMS and JMS Client Applications

This section provides an overview of the JMS model, describes NSJMS administered objects, and provides information about the common tasks performed by JMS client applications including information about how NSJMS implements the JMS interfaces when performing the tasks.

The JMS Model

This defines an abstract view of a messaging service but does not define an implementation. The JMS model is based on a set of generic interfaces that are defined in Sun’s javax.jms package. The generic JMS interfaces described below are only a few of the generic interfaces used in application programs to perform common tasks.

Generic JMS Interfaces	Description
Connection	Provides access to the underlying NSJMS implementation, and is used to create a Session.
Session	Provides a context for sending and receiving messages, including the methods used to create MessageProducers and MessageConsumers.
MessageProducer	Used to send messages.
MessageConsumer	Used to receive messages.

Messaging Models

The JMS model supports Point-to-Point and Publish/Subscribe messaging models. Within each messaging model, the generic JMS interfaces are further defined to address behavior specific to each messaging model.

Point-to-Point (PTP)

A PTP messaging application is a one-to-one messaging model which uses queues for messaging destinations. A message is sent by a sending client to a specific queue where the message can be received by the receiving client. A queue retains a message until it is received by the receiving client or until the message expires.

The generic JMS interfaces are further defined within the PTP messaging application as:

Queue Interfaces	Description
QueueConnection	Provides an active connection to NSJMS that is used to create one or more QueueSessions.
QueueSession	Provides methods for creating QueueSender's and QueueReceiver's.
QueueSender	Used to send messages to a queue.
QueueReceiver	Used to receive messages that have been delivered to a queue.

Publish/Subscribe (Pub/Sub)

A Pub/Sub messaging application is a one-to-many messaging model which uses topics for messaging destinations. A message is published by a sending client to a topic where any receiving client who is subscribed to the topic will receive the message. A topic retains a message until the message expires.

The generic JMS interfaces are further defined within the Pub/Sub messaging application as:

Topic Interfaces	Description
TopicConnection	Provides an active connection to NSJMS that is used to create one or more TopicSessions.
TopicSession	Provides methods for creating TopicPublisher's and TopicSubscriber's.
TopicPublisher	Used to publish messages to a topic.
TopicSubscriber	Used to receive messages that have been published to a topic.

Administered Objects

Administered objects are built using the administrative utility and are stored in a JNDI namespace. For information about using the administrative utility to build NSJMS administered objects, see Section 7, Managing the NSJMS Environment. A JMS client can retrieve the administered objects from the JNDI namespace and use them without needing to know how they have been implemented.

JMS clients use administered objects to create connections to NSJMS and to specify the destinations for messages. All NSJMS-specific information is contained in implementations of these NSJMS administered objects:

NSJMS Administered Objects	**Description**
QueueConnectionFactory	Used within a PTP messaging application by JMS clients for creating connections to NSJMS.
TopicConnectionFactory	Used within a Pub/Sub messaging application by JMS clients for creating connections to NSJMS.
Queue	Used within a PTP messaging application by JMS clients to specify the destinations to and from which the JMS clients send and receive messages.
Topic	Used within a Pub/Sub messaging application by JMS clients to specify the destinations to and from which the JMS clients send and receive messages.

Writing JMS Client Applications

JMS clients hosted on the operating system use NSJMS to send and receive messages. Descriptions of common tasks that enable JMS clients to send and receive messages and information about how NSJMS implements the JMS interfaces when performing the tasks are described in:

- Creating a Connection on page 4-4
- Transactions on page 4-6
- Sending Messages on page 4-7
- Specifying a Destination on page 4-7
- Destination Names on page 4-8
- Temporary Destinations on page 4-8
- Destination IDs on page 4-8
- Persistence on page 4-8
- Priority on page 4-9
- Receiving Messages on page 4-9
- No-Local on page 4-11
- Using Message Listeners on page 4-11
- Best Practices for Developing a JMS Client on page 4-12
- Threading on page 4-15
- Message Expiration Values on page 4-16

Creating a Connection

A JMS client uses a ConnectionFactory object to create a connection with NSJMS. To create a connection between a JMS client and NSJMS:

- Create an Initial Context on page 4-4
- Retrieve the Connection Factory on page 4-4
- Use the Factory Object to Create a Connection on page 4-5
- Start the Connection on page 4-5
- Create a Session on page 4-5.

Note. At this time, no interoperability exists among the JMS providers. A user achieves platform interoperability by using a JMS provider that is present on all the required platforms. Two or more JMS implementations can coexist. A user must establish JMS connections to each provider from the same Java program. The user can then receive messages from one provider and send the message using another provider. NSJMS includes code that transforms foreign messages to native format when the messages are sent using a NSJMS session.

Create an Initial Context

You must create an initial context to retrieve the ConnectionFactory object from a JNDI namespace. This is shown in the code fragment taken from the `SimpleSender.java` sample program file:

```
import javax.jms.*;
import javax.naming.*;
.
.
.
 Context jndiContext = new InitialContext();
```

Retrieve the Connection Factory

After you have created an initial context, perform a JNDI lookup to retrieve the ConnectionFactory object from a JNDI namespace.

Note. To protect the JMS application from NSJMS-specific information, factory objects are stored in a JNDI namespace.

Retrieving a QueueConnectionFactory is shown in this code fragment taken from the `SimpleSender.java` sample program file:

```
QueueConnectionFactory factory = (QueueConnectionFactory)
jndiContext.lookup(“QueueConnectionFactory”);
```

Retrieving a TopicConnectionFactory is shown in this code fragment taken from the `SimpleSender.java` sample program file:

```
TopicConnectionFactory factory = (TopicConnectionFactory)
jndiContext.lookup(“TopicConnectionFactory”);
```

Create Connection Factories at Runtime

You can, optionally, create ConnectionFactory objects at runtime. However, this requires references to NSJMS specific classes that will reduce the portability of the JMS application.

This code fragment creates a QueueConnectionFactory with the default settings:

```
QueueConnectionFactory factory=
com.tandem.nsjms.client.JmsQueueConnectionFactory();
```

This code fragment creates a TopicConnectionFactory with the default settings:

```
TopicConnectionFactory factory=
com.tandem.nsjms.client.JmsTopicConnectionFactory();
```

Use the Factory Object to Create a Connection

You use a queue or topic connection method on the factory object to create a connection.

Creating a queue connection is shown in this code fragment taken from the `SimpleSender.java` sample program file:

```
QueueConnection connection = factory.createQueueConnection();
```

Creating a topic connection is shown in this code fragment taken from the `SimpleSender.java` sample program file:

```
TopicConnection connection = factory.createTopicConnection();
```

Start the Connection

The JMS specification defines that connections should be created in a stopped state. You must start the connection to enable MessageConsumers associated with the connection to receive messages. To start the connection, run this command:

```
connection.start();
```

Create a Session

After you have started the connection, use a queue or topic session method to create a session.

Creating a queue session is shown in this code fragment taken from the `SimpleSender.java` sample program file:

```
QueueSession session = connection.createQueueSession(false,
Session.AUTO_ACKNOWLEDGE);
```

Creating a topic session is shown in this code fragment taken from the `SimpleSender.java` sample program file:

```
TopicSession session = connection.createTopicSession(false,
Session.CLIENT_ACKNOWLEDGE);
```

Both the `createQueueSession()` method and the `createTopicSession()` method accept a boolean parameter that specifies whether the session is transacted.

Transactions

NSJMS supports transacted sessions and external transactions. Transacted sessions allow you to send and receive multiple messages in a single transaction that is local to the session. External transactions are inherited transactions that start and end external to NSJMS and allow you to combine other database or server work with the same transaction that reads or sends messages. External transactions are defined using either the current class methods or the JTA both of which are described in the *NonStop Server for Java Programmer's Reference*.

The JMS client process' current TMF transaction is inherited by NSJMS send-and-receive operations. If no current transaction exists, a transaction is started within NSJMS and committed within NSJMS.

Messages sent within a transaction are received by consumers only after the transaction is committed. Messages received from a queue within a transaction are not removed from the queue until the transaction is committed.

Transacted Sessions

JMS sessions are created as transacted sessions. Transacted sessions use JMS `commit()` and `rollback()` methods to commit or abort the transaction started for the session. NSJMS supports transacted sessions, but the use of external transactions overrides the transacted session. Within a transacted session, an external transaction can send and receive messages. The JMS session `commit()` and `rollback()` methods affects only those operations that were performed without an external transaction.

△ **Caution.** When a transacted session is used to receive messages and an external transaction is started without first issuing a `commit()` or `rollback()` method for the transacted session, an error may occur if a `receive()` method is issued under the external transaction. This is because two active transactions are associated with the cursor used to read messages. The transacted session should be committed or rolled back before using the external transaction.

When writing JMS client code, check that no more than one current transaction exists for each subscriber, receiver, or publisher.

Non-Transacted Sessions

Non-Transacted sessions use client acknowledgment to signal when receives are committed. Client acknowledgment is determined by a mode established for the session and can be implicit (`AUTO_ACKNOWLEDGE` and `DUPS_OK_ACKNOWLEDGE`) or explicit (`CLIENT_ACKNOWLEDGE`). An external transaction overrides the client acknowledgment mode. A client's explicit message acknowledgment acknowledges only those messages received without an external transaction.

Sending Messages

A JMS client uses a MessageProducer to send messages to a specified destination. A message producer is an object created by a session that is used by JMS clients to send messages to queues or topics. The PTP messaging application uses the `QueueSender` interface as its message producer, and the Pub/Sub messaging application uses the `TopicPublisher` interface as its message producer.

Sending a message to a Queue

To send a message to a queue, you create a QueueSender using the `createSender()` method. This code fragment sends a message to a queue:

```
QueueSender.send(aMessage);
```

Sending a message to a Topic

To send a message to a topic, you create a TopicPublisher using the `createPublisher()` method. This code fragment publishes a message to a topic:

```
TopicPublisher.publish(aMessage);
```

Specifying a Destination

A QueueSender or TopicPublisher is typically created for a specific queue or topic. This ensures that all messages sent using the QueueSender or TopicPublisher are sent to the specific queue or topic. The destination of the queue or topic is specified using a Queue or Topic object. Queue and Topic objects can be built and stored in a JNDI namespace or created at runtime.

Retrieve Queue or Topic Objects from a JNDI Namespace

Retrieving a Queue object from a JNDI namespace is shown in this code fragment:

```
Queue queue = (Queue)jndiContext.lookup(queueName);
```

Retrieving a Topic object from a JNDI namespace is shown in this code fragment:

```
Topic topic = (Topic)jndiContext.lookup(topicName);
```

Create Queue or Topic Objects at Runtime

This code fragment creates a Queue object dynamically:

```
Queue queue = (Queue)session.createQueue(queueName);
```

This code fragment creates a Topic object dynamically:

```
Topic topic = (Topic)session.createTopic(topicName);
```

Destination Names

Queue names and topic names are Java letters and Java digits, limited to 100 bytes, where the first character is a Java letter. This syntax also describes subscription names and client IDs.

Temporary Destinations

JMS allows for the creation of temporary queues and topics that exist only for the duration of a JMS connection. A temporary queue or topic is a unique destination that survives only as long as the JMS connection that created it. NSJMS creates these as specially named destinations with these prefixes: __temp_queue__ or __temp_topic__.

When a JMS connection is closed, any temporary destinations along with associated messages are removed. If an application ends abnormally and a `close` operation is not performed, the temporary destinations and associated messages remain on the database until the next DELETE MESSAGE EXPIRED operation is performed.

This code fragment creates a temporary queue destination:

```
Queue queue = session.createTemporaryQueue();
```

This code fragment creates a temporary topic destination:

```
Topic topic = session.createTemporaryTopic();
```

Destination IDs

Destinations (queues and topics) within NSJMS are assigned IDs that represent the destinations in the database. The IDs allow efficient storage for messages and flexible partitioning of the message table.

Assignment of the destination ID is done when the destination is first created. This assignment can be automatic (implicit) or it can be explicitly provided within the administrative commands.

Explicit destination-ID assignment is useful when you need message-table partitioning. The IDs are chosen so that ranges are defined over the IDs to identify the partitions. For example, you can group destinations by 100s with separate partitions assigned to each range of 100.

Destination IDs must be positive numbers, `1 to 32767 (SHORT_MAX)`. If no explicit ID is specified, the ID is generated by adding one to the current highest ID. Destination IDs cannot be changed once assigned. You must delete and re-create a destination to change an ID.

Persistence

NSJMS does not optimize for `NON_PERSISTENT` messages (deliver at most once). All messages are treated as having `PERSISTENT` delivery mode (deliver once and only once).

Priority

JMS allows message priority to be set by the `setPriority()` method on the MessageProducer object and retrieved using the `getPriority()` method on the MessageConsumer object. Although NSJMS allows message priority to be set and retrieved, this priority does not affect message-delivery order.

Receiving Messages

A JMS client uses a MessageConsumer to receive messages from a specified destination. A message consumer is an object created by a session that is used by JMS clients to receive messages from queues or topics. The PTP messaging application uses `QueueReceiver` as its message consumer, and the Pub/Sub messaging application uses `TopicSubscriber` as its message consumer.

Receiving Messages from a Queue

To receive a message from a queue, you create a QueueReceiver using the `createReceiver()` method. This method uses a Queue parameter to define where the messages are received from. This code fragment receives a message from a queue and then reads back the message:

```
QueueReceiver receiver = session.createReceiver(queue);
Message message = receiver.receive();
```

The `receive()` method without parameters blocks indefinitely unless a timeout parameter is specified. When specified, this parameter defines how long the `receive()` method should wait when no messages are available. No delay occurs if the `receiveNoWait()` method is used.

The `receive()` method returns a message of the same type as was sent to the queue. For example, if a StreamMessage is sent to a queue, the `receive()` method returns an object of type StreamMessage.

To pull the content from the body of the message, you must identify the subclass of the message contents, such as TextMessage. If you do not know the message-contents subclass, use `instanceof` to determine if the contents of a message are of a certain type. It is good practice always to test the message class before casting, so that unexpected errors are handled gracefully.

This code fragment shows the use of `instanceof` when pulling the contents from a TextMessage:

```
if (inMessage instanceof TextMessage){
  String replyString =((TextMessage)inMessage).getText();
  .
  .
  .
} else {
  System.out.println("Reply message was not a TextMessage");
}
```

Receiving Messages from a Topic

To receive a message from a topic, you create a TopicSubscriber using the `createSubscriber()` method. This method creates TopicSubscribers as either non-durable or durable subscribers.

Non-durable Subscribers

These subscribers only receive messages that are published on a chosen topic while the subscriber is active.

To create a non-durable subscriber use the `createSubscriber()` method for the session object. For example:

```
TopicSubscriber subscriber = session.createSubscriber(topic);
```

Non-durable subscribers automatically terminate themselves when their `close()` method is called.

Durable Subscribers

These subscribers receive all messages published on a topic, including messages published while the subscriber is inactive.

A durable subscriber registers a durable subscription with a unique identity that is retained by the NSJMS. Subsequent subscriber objects with the same identity resume the subscription in the same state as left by the previous subscriber. If a durable subscription has no active subscriber, the NSJMS retains the subscription's messages until they are received by the subscription or until they expire.

To establish a unique identity for a durable subscriber specify:

- A client ID for the connection.
- A topic and a subscription name for the subscriber.

The client ID is set by calling the `setClientID()` method for the connection object. For example, `connection.setClientID ("client_ID_01");`

The client ID must be set before creating the durable subscription. To create a durable subscriber use the `createDurableSubscriber()` method for the session object.

This method is similar to creating a non-durable subscriber, except that you must pass a subscription name to identify the unique subscriber. For example:

```
TopicSubscriber subscriber=
session.createDurableSubscriber(topic,MY_SUB_01);
```

The subscriber becomes active after you start the `TopicConnection`. Later on, the `TopicSubscriber` can be closed by calling the `close()` method of `TopicSubscriber` object. For example, `subscriber.close();`

If the program or another application calls `createDurableSubscriber` method with the same client ID, topic, and subscription name, then the subscription is reactivated. Subsequently, the NSJMS delivers the messages that were published while the subscriber was inactive.

To terminate a durable subscriber, use the `unsubscribe()` method and submit the unique name that created the subscriber. For example:

```
session.unsubscribe(MY_SUB_01);
```

No-Local

You can ignore messages that are sent or published on the subscriber's own connection by setting the third parameter of the `createSubscriber()` method to `true`.

This code fragment causes messages published on the subscriber's connection to be ignored:

```
TopicSubscriber sub =
session.createSubscriber(topic,null,true);
```

Using Message Listeners

A message listener uses the `onMessage()` method to enable asynchronous notification to the listener when a message is sent to a queue or published to a topic. The message listener is registered by the JMS client when subscribing to a queue or topic and is used instead of making calls to the `receive()` method.

Note. Use of asynchronous delivery with a QueueReceiver or TopicSubscriber marks the entire Session as asynchronous. It is an error to make an explicit call to the receive methods of a QueueReceiver or TopicSubscriber that is associated with a Session that is using asynchronous delivery.

This code fragment creates a message listener for a queue:

```
import javax.jms.*;
public class MyClass implements MessageListener
{
  public void onMessage(Message message)
  {
    System.out.println("The message is "+message);
    .
    .
    .
   }
 }
.
.
.
 MyClass listener =new MyClass();
 queueReceiver.setMessageListener(listener);
```

This code fragment creates a message listener for a topic:

```
import javax.jms.*;
public class MyClass implements MessageListener
{
  public void onMessage(Message message)
  {
    System.out.println("The message is "+message);
    .
    .
    .
   }
 }
.
.
.
 MyClass listener =new MyClass();
 topicSubscriber.setMessageListener(listener);
```

Best Practices for Developing a JMS Client

You should use caution when creating active connections to NSJMS. Multiple connections are not recommended. Creating multiple Session objects from the same connection is more efficient because sessions share access to the same connection.

Note. Although multiple sessions are allowed, they should not be repetitively created for Consumers or Producers.

A typical JMS client performs these JMS setup process tasks:

- Uses JNDI to find a ConnectionFactory object.
- Uses JNDI to find one or more Destination objects.
- Uses the ConnectionFactory to create a JMS Connection with message delivery inhibited.
- Uses the Connection to create one or more JMS Sessions.
- Uses a Session along with the Destinations to create MessageProducers and MessageConsumers.
- Instructs the Connection to start delivery of messages.

At this point a client has the basic JMS setup needed to produce messages and consume messages using a MessageListener or a `receive()` loop. An example of the recommended setup process is shown in Topic Publisher/Subscriber Sample Setup (Recommended) on page 4-13. An example of a setup process that is not recommended for use is shown in Topic Publisher/Subscriber Sample Setup (Not Recommended) on page 4-14.

Topic Publisher/Subscriber Sample Setup (Recommended)

This is a sample of the recommended setup process in which the setup steps are performed only once:

```
TopicConnectionFactory factory = TopicConnectionFactory)
   jndiContext.lookup("TopicConnectionFactory");
TopicConnection connection = factory.createTopicConnection();

TopicSession sessionSub = connection.createTopicSession(true,
   Session.AUTO_ACKNOWLEDGE);
TopicSession sessionPub = connection.createTopicSession(true,
   Session.AUTO_ACKNOWLEDGE);

Topic topicSub = (Topic)jndiContext.lookup(topicSubName);
Topic topicPub = (Topic)jndiContext.lookup(topicPubName);

TopicSubscriber subscriber =
   sessionSub.createDurableSubscriber(topicSub,
   "MySubscription");

TopicPublisher publisher =
   sessionPub.createPublisher(topicPub);

TextListener topicListener = new TextListener();
subscriber.setMessageListener(topicListener);
connection.start();

class TextListener implements MessageListener
{
   public void onMessage(Message message)
   {
      TextMessage messageSub = (TextMessage) message;

      // Only the message processing, i.e. publish() of the
      // message is done after each receive
      // Most efficient
      TextMessage messagePub =sessionPub.createTextMessage();
      messagePub.setText(messageSub.getText());
      publisher.publish(messagePub);
      // session commits can be done after each message or
      // batched
   }
```

Topic Publisher/Subscriber Sample Setup (Not Recommended)

This is a sample of a setup process that is not recommended for use because the setup steps are repeated with each message:

```
TopicConnectionFactory factory =
   (TopicConnectionFactory)
jndiContext.lookup("TopicConnectionFactory");
TopicConnection connection = factory.createTopicConnection();

TopicSession sessionSub = connection.createTopicSession(true,
   Session.AUTO_ACKNOWLEDGE);
Topic topicSub = (Topic)jndiContext.lookup(topicSubName);
TopicSubscriber subscriber =
   sessionSub.createDurableSubscriber(topicSub,
   "MySubscription");

TextListener topicListener = new TextListener();
subscriber.setMessageListener(topicListener);
connection.start();

class TextListener implements MessageListener
{
   public void onMessage(Message message)
   {
      TextMessage messageSub = (TextMessage) message;

      // Connection, session & publisher recreated after each
      //receive, very expensive
      TopicConnection connectionNew =
         factory.createTopicConnection();
      TopicSession sessionPub =
         connectionNew.createTopicSession(true,
         Session.AUTO_ACKNOWLEDGE);
      Topic topicPub =
         (Topic)jndiContext.lookup(topicPubName);
      TopicPublisher publisher =
         sessionPub.createPublisher(topicPub);

      TextMessage messagePub =
         sessionPub.createTextMessage();
      messagePub.setText(messageSub.getText());
      publisher.publish(messagePub);
      //session commits can be done after each message or
      //batched
   }
}
```

Threading

You can use JMS with multithreaded Java applications. Although threading is supported, the send calls block the process during send operations and the receive calls support the non-blocking features of SQL/MX during receive operations when `sqlmx_nowait` is set to `on`.

Two NSJMS property values specify the threading behavior. The `Database.Timeout` property value is the time in milliseconds during which stream selects block for new messages before a timeout. The `Database.Delay` property value is the time in milliseconds that the thread is delayed after a timeout.

Setting the `Database.Timeout` and `Database.Delay` values below 1000 gives more responsive threads because the read is stopped and restarted frequently, which allows other threads to run. This responsiveness costs in system throughput, however, because the read is tried several times for each message. In single-threaded applications and applications where inter-thread responsiveness is not critical, use `Database.Timeout` and `Database.Delay` values greater than 1000 so that the JVM is not busy waiting. When messages are not available, the process blocks, allowing other processes to run.

Send Operations

Queue-send operations and Topic publish operations block the process for a short duration during a SQL INSERT operation.

Receive Operations

Queue-receive operations and Topic-receive operations support the non-blocking features of SQL/MX during receive operations only when the system property `sqlmx_nowait` is set to `on`. The non-blocking features of SQL/MX are useful in multi-threaded programs that use separate threads to receive messages simultaneously from different sources or to perform processing while awaiting messages.

The system property `sqlmx_nowait` toggles the non-blocking function. The default value is off. `sqlmx_nowait` can be set from the command line by using the -D option (`-Dsqlmx_nowait=on|off`) or programmatically, before obtaining the first NSJMS connection, by using the `System.setProperty()` method.

The `sqlmx_nowait` value is obtained from the environment only at the time of the first JDBC/MX connection. The JDBC/MX driver is then configured with the value of the `sqlmx_nowait` system property for any subsequent connections within the same JVM.

After initiating the receive operation, NSJMS yields the thread control to the JVM. When a message arrives, the JVM wakes up the thread and returns the message to the application.

If `sqlmx_nowait` is set to `on`, users may want to set the `Database.timeout` value to 1000+ milliseconds so the thread waits on messages for a longer period. The larger timeout value can improve performance because the SQL operation is stopped and restarted less frequently while awaiting a message; the longer timeout does not block the process since other threads run during the time spent waiting for the message.

Message Expiration Values

You can change the default expiration value for a message sent to a destination by specifying a value for the `expiry` attribute in the ADD command. The expiry attribute sets the default expiration value for a message to a specified number of milliseconds (see ADD Command on page 7-2).

Queue Message Removal

The application determines the rate of removal for messages in queues, which are deleted as they are received. Messages in queues might also have expiration attributes that specify that the messages can be removed before they are received.

Topic Message Removal

Messages on topics are not removed by subscribers. The messages exist until they are removed because they have expired or are dead.

A topic message is expired when the message-expiration value is exceeded. An expiration value is specified when the message is sent. If no expiration value is specified, the message never expires.

A topic message is dead after it has been read by all subscribers.

To remove expired and dead messages, do either of these:

- Use the NSJMS administrative utility DELETE command.
- Set the `Deadmsg.deletecount` property in the `nsjms.property` file to a value greater than `0`. This value represents the number of messages received between the deletion of dead messages. For example, a delete count of 100 means that after every 100th message received from a topic, all existing dead messages are deleted. Setting the `Deadmsg.deletecount` property to `0` disables dead-message removal.

5 Reliable Messaging Bridge

Introduction

The reliable messaging bridge is hosted on the NonStop platform and sits between a locally hosted JMS client library from a foreign JMS provider and the NSJMS JMS API. Using the standard Sun Microsystems JMS API, the reliable messaging bridge reads from a locally hosted foreign JMS provider and writes to NSJMS, or reads from NSJMS and writes to a locally hosted foreign JMS provider. Figure 5-1 on page 5-2 illustrates the components used in a reliable messaging bridge along with the flow of information.

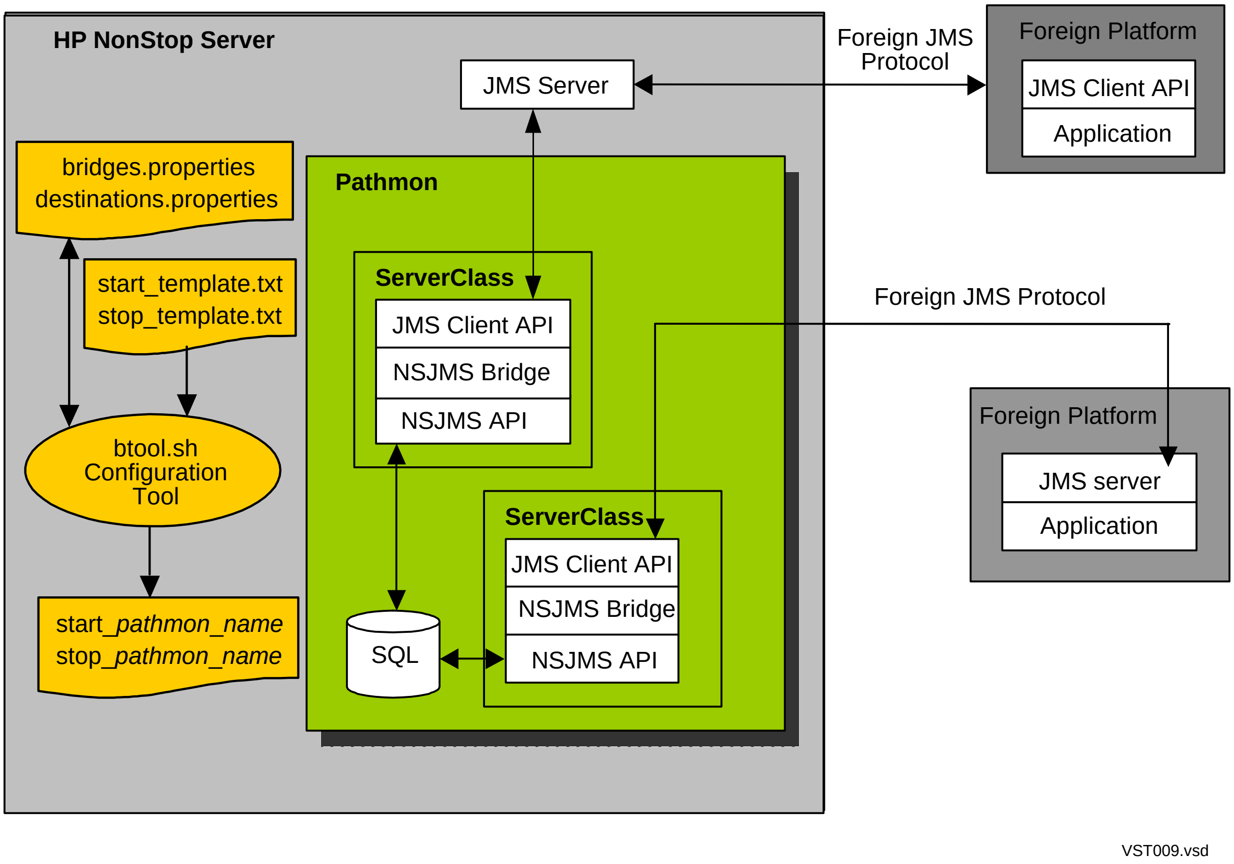
The bridge moves messages reliably using persistent processes and transactions. Within each process, reliability is achieved by enclosing the movement of messages between a locally hosted foreign JMS provider and NSJMS in JMS transacted sessions. In the event of an unrecoverable error, the bridge will roll back any messages transferred during the transaction.

Persistence is achieved by using Pathway. Messages can be batched together and moved in a single transaction which improves throughput performance. The batch size is specified during bridge pathway configuration. Durable subscriptions are supported for source topics, and both source queue and topics may specify selectors in their configurations. Each bridge is composed of one or more processes that are managed to assure they are restarted after failures. For queues, multiple processes may be defined, whereas topics may only have a single process per topic.

The reliable messaging bridge components included in the installation of NSJMS in the `/usr/tandem/nsjms/`*`version`*`/bridge` directory, where *`version`* is the vproc of this RVU (for example, `T1251V30_30SEP2003_V30`) are:

- `btool.sh` – This script invokes the configuration tool. The configuration tool maps between queues and topics and defines the persistent environment parameters. (For more information, see Configure a Reliable Messaging Bridge on page 5-2.)
- `start_template.txt` – This file is used by the configuration tool to generate an executable Pathway start script.
- `stop_template.txt` – This file is used by the configuration tool to generate an executable Pathway start script.

Figure 5-1. Reliable Messaging Bridge



Configure a Reliable Messaging Bridge

To configure a reliable messaging bridge, you must

- Set the environment variables
- Run the btool.sh script
- Use the configuration tool to add, alter, delete, or inquire about a destination or a bridge configuration

Setting the Environment Variables

Check that the following environment variables are set before running the btool.sh script. The script will abort if the environment variables are not accurately set.

JAVA_HOME Environment Variable

Use the `export` command to set the JAVA_HOME variable. For example:

```
OSH: cd /usr/tandem/nsjms/version/bridge
OSH: export JAVA_HOME=/usr/tandem/java
```

where *`version`* is the vproc of this RVU (for example, `T1251V30_30SEP2003_V30`).

NSJMS_HOME Environment Variable

Use the `export` command to set the NSJMS_HOME variable. For example:

```
OSH: cd /usr/tandem/nsjms/version
OSH: export NSJMS_HOME=/usr/tandem/nsjms/version
```

where *`version`* is the vproc of this RVU (for example, `T1251V30_30SEP2003_V30`).

Add Foreign-Provider Client Jar Files

Modify the btool.sh script to include all foreign-provider client jar files that will be used in the bridge Pathway environment:

1. Use the `cd` command to go to the directory where the btool.sh script is located. For example:

```
OSH: cd /usr/tandem/nsjms/version/bridge
```

   where *`version`* is the vproc of this RVU (for example, `T1251V30_30SEP2003_V30`)

2. Using an editor of your choice, modify the btool.sh script as:

   a. Find the line `export BRIDGE_SUPPORT_JARS=`

   b. Append the directory location for all foreign-provider client jar files to the end of `export BRIDGE_SUPPORT_JARS=`. Use colons to separate the directory locations.

Example 5-1 on page 5-4 shows where in the btool.sh script the modifications should be made.

Example 5-1. Modifying the btool.sh Script to Include Foreign-Provider Client Jar Files

```
#!/bin/sh
# Script for starting the NSJMS bridge configuration tool
#
#
# Copyright 2003
# Hewlett Packard Development Company, L.P.
# Protected as an unpublished work.
# All rights reserved.
#
# The computer program listings, specifications and
# documentation herein are the property of Compaq Computer
# Corporation and successor entities such as Hewlett Packard
# Development Company, L.P., or a third party supplier and
# shall not be reproduced, copied, disclosed, or used in whole
# or in part for any reason without the prior express written
# permission of Hewlett Packard Development Company, L.P.
#
if [[ -z $JAVA_HOME ]] then
   print "Environment variable JAVA_HOME must be set."
   return 1
fi

if [[ -z $NSJMS_HOME ]] then
   print "Environment variable NSJMS_HOME must be set prior to running
      $0."
   print "Set it to the path name of the NSJMS product directory."
   return 3
fi

#
# Add supporting jar files here.
#
export BRIDGE_SUPPORT_JARS=
   FOREIGN_PROVIDER_CLIENT_JAR_FILE_DIRECTORY_LOCATION:
   FOREIGN_PROVIDER_CLIENT_JAR_FILE_DIRECTORY_LOCATION:
   FOREIGN_PROVIDER_CLIENT_JAR_FILE_DIRECTORY_LOCATION:
   FOREIGN_PROVIDER_CLIENT_JAR_FILE_DIRECTORY_LOCATION
#
#
#
export
CLASSPATH=$PWD:$CLASSPATH:$NSJMS_HOME/lib/nsjms.jar:$NSJMS_HOME/lib/jndi.jar:
$NSJMS_HOME/lib/providerutil.jar:$NSJMS_HOME/lib/fscontext.jar:$BRIDGE_SUPPOR
T_JARS
echo java -classpath $CLASSPATH com.tandem.nsjms.bridge.Admin $*
java -classpath $CLASSPATH com.tandem.nsjms.bridge.Admin $*
```

Run the btool.sh Script

From within the OSS Shell (`OSH`), run the btool.sh script located in the `/usr/tandem/nsjms/`*`version`*`/bridge` directory. Successful completion of the btool.sh script will start the configuration tool. For example, run the btool.sh script as:

```
OSH: /usr/tandem/nsjms/version/bridge: btool.sh

java -classpath /usr/tandem/nsjms/bridge:/usr/tandem/nsjms
/T1251V30_30SEP2003_V30/lib/nsjms.jar:/usr/tandem/nsjms/T1251V30_30SEP
2003_V30/lib/jndi.jar:/usr/tandem/nsjms/T1251V30_30SEP2003_V30/lib/pro
viderutil.jar:/usr/tandem/nsjms/T1251V30_30SEP2003_V30/lib/fscontext.j
ar: com.tandem.nsjms.bridge.Admin

NSJMS Messaging Bridge Configuration Tool
Copyright 2003 Hewlett Packard Development Company, L.P.

command:
```

Using the Configuration Tool

Note. Always use the btool.sh script to invoke the configuration tool.

The configuration tool is used to create destination configurations and bridge configurations that are stored in the `destinations.properties` file and the `bridges.properties` file respectively, and to generate executable Pathway scripts (`start_`*`pathmon_name`* and `stop_`*`pathmon_name`*).

When the configuration tool has started and the `command:` prompt appears, you can configure or inquire about a bridge Pathway configuration by performing any of these tasks. These tasks are described in detail in the following pages.

△ **Caution.** All tasks can only be performed from within the configuration tool.

- Add a Destination on page 5-6
- Alter a Destination on page 5-7
- Delete a Destination on page 5-8
- Request Information about a Destination on page 5-9
- Add a Bridge Pathway Configuration with Server Class on page 5-10
- Alter a Bridge on page 5-12
- Delete a Bridge on page 5-15
- Request Information about a Bridge on page 5-16

Add a Destination

The `add dest` command defines a destination and generates the `destinations.properties` file.

Add Destination Command Prompts

The `add dest` command steps through these command prompts. At each prompt, enter the appropriate value or press Enter to proceed to the next prompt. If a value appears in brackets ([ ]) at a prompt, for example, `Store queue0 [N]:` pressing Enter without entering a new value will select the value in the brackets.

Prompt	Usage
`Destination Name:`	Required. Enter a unique destination name. The first character must be a letter and the remaining characters are restricted to a letter, a digit, or a hyphen.
`Destination Type:`	Required. Enter “Queue” or “Topic”.
`JNDI Initial Context Factory:`	Required. Enter the JNDI provider class name.
`JNDI Provider URL:`	Required. Enter the JNDI store location.
`JNDI Connection Factory Name:`	Required. Enter the JNDI name as it exists in the JNDI store.
`JNDI Destination Name:`	Required. Enter the JNDI name as it exists in the JNDI store.
`Store` *`Destination_Name`*`:`	Enter y to save the destination.

Example

An example of using the configuration tool to add a destination named `queue0` is shown below. Bold text indicates user input.

```
NSJMS Messaging Bridge Configuration Tool
Copyright 2003 Hewlett Packard Development Company, L.P.

command: add dest

Destination Name: queue0

Destination Type: queue

JNDI Initial Context Factory:
com.sun.jndi.fscontext.RefFSContextFactory

JNDI Provider Url:
file:////usr/tandem/nsjms/T1251V30_30SEP2003_V30/jndi_object_store

JNDI Connection Factory Name: QueueConnectionFactory

JNDI Destination Name: queue0

Store queue0 [N]: y

Destination added
```

Alter a Destination

The `alter dest` command alters the configured properties for an existing destination and updates the `destinations.properties` file with the newly configured properties.

△ **Caution.** If you alter the configured properties of a destination that is in use by a bridge, you must update the bridge pathway configuration using the `alter bridge` command.

Alter Destination Command Prompts

The `alter dest` command steps through these command prompts. At each prompt, enter the appropriate value or press Enter to proceed to the next prompt. Pressing Enter without entering a new value will select the value, if any, which appears in brackets ([ ]) at each prompt. For example, `Destination Name [queue0]:`

Prompt	Usage
`Destination Name:`	Required. Enter the name of the destination to be altered.
`Destination Type:`	Optional. To change, enter “Queue” or “Topic”.
`JNDI Initial Context Factory:`	Optional. To change, enter a new JNDI provider class name.
`JNDI Provider URL:`	Optional. To change, enter a new JNDI store location.
`JNDI Connection Factory Name:`	Optional. To change, enter a new JNDI name as it exists in the JNDI store.
`JNDI Destination Name:`	Optional. To change, enter a new JNDI name as it exists in the JNDI store.
`Store` *`Destination_Name`*`:`	Enter y to save the altered values. Enter n to leave the task without any changes taking effect.

Example

An example of using the configuration tool to alter the JNDI Destination Name from `queue0` to `queue99` is shown below. Bold text indicates user input.

```
NSJMS Messaging Bridge Configuration Tool
Copyright 2003 Hewlett Packard Development Company, L.P.

command: alter dest

Destination Name: queue0

Destination Type [queue]:

JNDI Initial Context Factory
[com.sun.jndi.fscontext.RefFSContextFactory]:

JNDI Provider Url
[file:////usr/tandem/nsjms/T1251V30_30SEP2003_V30/jndi_object_store]:

JNDI Connection Factory Name [QueueConnectionFactory]:

JNDI Destination Name [queue0]:queue99

Store queue0 [N]: y

Destination altered
```

Delete a Destination

The `delete dest` command deletes an existing destination and updates the `destinations.properties` file.

△ **Caution.** If you delete a destination that is in use by a bridge, you must update the bridge to specify an existing destination using the `alter bridge` command.

Delete Destination Command Prompts

The `delete dest` command steps through these command prompts. At each prompt, enter the appropriate value or press Enter to proceed to the next prompt. Pressing Enter without entering a new value will select the value, if any, which appears in brackets ([ ]) at each prompt. For example, `Destination Name [queue0]:`.

Prompt	Usage
`Destination Name:`	Required. Enter the name of the destination you want to delete.
`Delete` *`Destination_Name`*`:`	Enter `y` to delete the destination from the `destination.properties` file. Enter `n` to leave the task without deleting the destination.

Example

An example of using the configuration tool to delete the destination named `queue0` from the `destinations.properties` file is shown below. Bold text indicates user input.

```
NSJMS Messaging Bridge Configuration Tool
Copyright 2003 Hewlett Packard Development Company, L.P.

command: delete dest

Destination Name: queue0

Delete queue0 [N]: y

Destination deleted
```

Request Information about a Destination

The `info dest` command returns configuration information about destinations.

Info Destination Command Prompt

At the `Destination Name:` prompt, enter the appropriate value and press Enter to obtain configuration information. Pressing Enter without entering a new value will select the value, if any, which appears in brackets ([ ]) at the prompt. For example, `Destination Name [queue0]:`

Prompt	Usage
`Destination Name:`	Required. To obtain configuration information for a specific destination, enter the destinations name. To obtain configuration information for all configured destinations, enter the wildcard character (*).

Example

An example of using the configuration tool to request information for all configured destinations, `Destination Name: *`, is shown below. Bold text indicates user input.

```
NSJMS Messaging Bridge Configuration Tool
Copyright 2003 Hewlett Packard Development Company, L.P.

command: info dest

Destination Name: *


Info for - queue0

DestinationType=queue

JndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory

JndiProviderUrl=file:////usr/tandem/nsjms/T1251V30_30SEP2003_V30/jndi_
object_store

JndiConnectionFactoryName=QueueConnectionFactory

JndiDestinationName=queue0
```

Add a Bridge Pathway Configuration with Server Class

The `add bridge` command defines a bridge Pathway configuration, and generates the `bridges.properties` file and the executable Pathway scripts (`start_`*`pathmon_name`* and `stop_`*`pathmon_name`*).

Note. You must first add destinations using the `add dest` command before adding a bridge using the `add bridge` command.

Add Bridge Command Prompts

The `add bridge` command steps through these command prompts. At each prompt, enter the appropriate value or press Enter to proceed to the next prompt. If a value appears in brackets ([ ]) at a prompt, for example, `Number of Servers [1]:`, pressing Enter without entering a new value will select the value in the brackets.

Prompt	**Usage** (page 1 of 3)
`Pathmon Process Name:`	Required. Specifies the name of the PATHMON process. The name can have a maximum of five alphanumeric characters. The first alphanumeric character must be a letter (for example, `sampl`). If the name is to be used across a network, the name can have no more than four characters (for example, `sam1`).
`Pathway Server Class Name:`	Required. Specifies the name of the server class. The name can have from 1 to 15 alphanumeric characters including the hyphen, and must start with a letter, be unique within the Pathway environment, and not be a Pathway reserved word (for example, `qtoq`).

Prompt	**Usage** (page 2 of 3)
`Database vol/subvol:`	Required. Must follow the same form as the nsjms.properties Database.volsubvol name/value pair. See Database.volsubvol on page 2-13. NOTE: This property is global to the `Pathmon Process Name` and any changes to the name/value pair will be implemented across every server class running under the `Pathmon Process Name`. For information on changing the Database.volsubvol for a specific server class, see pathmon_process_name.server_class_name.properties File on page 5-28.
`Source:`	Required. Enter the name of the destination created using the `add dest` command and from which information will originate (for example, `queue0`). The first character must be a letter and the remaining characters are restricted to a letter, a digit, or a hyphen. This property is local to the specific Pathway server class.
`Target:`	Required. Enter the name of the destination created using the `add dest` command and to which information will be sent. For example, `queue1`. The first character must be a letter and the remaining characters are restricted to a letter, a digit, or a hyphen. This property is local to the specific Pathway server class.
`Durable:`	Optional. For topics, the source destination can be made durable. The default is N. This property is local to the specific Pathway server class.
`Selector:`	Optional. A JMS selector. Default is “” (null), This property is local to the specific Pathway server class.
`Batch Size:`	Optional. Enter the number of messages to transfer in each transaction. The default is one. This property is local to the specific Pathway server class.

Prompt	**Usage** (page 3 of 3)
`Number of Servers:`	Required. The default is one. For queues, the number of servers may be greater than one. For topics, the number of servers must be one. This property is local to the specific Pathway server class.
`Store` *`Pathmon_Process_Name._PathwayServer_Class_Name`*`:`	Enter y to save the bridge configuration. Enter n to leave the task without adding the bridge.

Example

An example of using the configuration tool to add a bridge Pathway configuration is shown below. Bold text indicates user input.

```
NSJMS Messaging Bridge Configuration Tool
Copyright 2003 Hewlett Packard Development Company, L.P.

command: add bridge

Pathmon Process Name: sampl

Pathway Server Class Name: qtoq

Database vol/subvol: $myvol.mysubvol

Source: queue0

Target: queue1

Selector []:

Batch Size [1]:

Number of Servers [1]:

Store sampl.qtoq [N]: y

Generating start_sampl script

Generating sampl.properties

Generating stop_sampl script

Bridge added
```

Alter a Bridge

The `alter bridge` command alters an existing bridge Pathway configuration, updates the `bridges.properties` file with the new values, and regenerates the start and stop scripts.

△ **Caution.** The `Database vol/subvol` property is global to the `Pathmon Process Name` and any change to the `Database.volsubvol` name/value combination will be implemented across every server class running under the `Pathmon Process Name`. For information on changing the Database.volsubvol for a specific server class, see pathmon_process_name.server_class_name.properties File on page 5-28.

Alter Bridge Command Prompts

The `alter bridge` command steps through these command prompts. At each prompt, enter the appropriate value or press Enter to proceed to the next prompt. Pressing Enter without entering a new value will select the value, if any, which appears in brackets ([ ]) at each prompt. For example, `Pathway Server Class Name [qtoq]:`.

Prompt	**Usage** (page 1 of 2)
`Pathmon Process Name:`	Required. Enter the name of the Pathmon Process for the bridge you want to alter.
`Pathway Server Class Name:`	Required. Enter the name of the Pathway Server Class for the bridge you want to alter.
`Database vol/subvol:`	**CAUTION**. This property is global to the `Pathmon Process Name` and any change to the name/value pair will be implemented across every server class running under the `Pathmon Process Name`. For information on changing the Database.volsubvol for a specific server class, see pathmon_process_name.server_class_name.properties File on page 5-28. Optional. Must follow the same form as the nsjms.properties Database.volsubvol name/value pair. See Database.volsubvol on page 2-13.
`Source:`	Optional. To change the source destination, enter a new destination from which information will originate. This property is local to the specified Pathway Server Class and any change to the name/value combination will only be implemented within the specified Pathway Server Class.
`Target:`	Optional. To change the target destination, enter a new destination to which information will be sent. This property is local to the specified Pathway Server Class and any change to the name/value combination will only be implemented within the specified Pathway Server Class.
`Durable:`	Optional. For topics, the source can be made durable. The default is N. This property is local to the specified Pathway Server Class and any change to the name/value combination will only be implemented within the specified Pathway Server Class.

Prompt	**Usage** (page 2 of 2)
`Selector:`	Optional. A JMS selector. The default is “” (null), This property is local to the specified Pathway Server Class and any change to the name/value combination will only be implemented within the specified Pathway Server Class.
`Batch Size:`	Optional. The number of messages to transfer in each transaction. The default is one. This property is local to the specified Pathway Server Class and any change to the name/value combination will only be implemented within the specified Pathway Server Class.
`Number of Servers:`	Optional. The default is one. For queues, the number of servers may be greater than one. For topics, the number of servers must be one. This property is local to the specified Pathway Server Class and any change to the name/value combination will only be implemented within the specified Pathway Server Class.
`Store` *Pathmon_Process_Name.Pathway_Server_Class_Name*`:`	Enter `y` to save the altered values. Enter `n` to leave the task without any changes taking effect.

Example

An example of using the configuration tool to alter the batch size in the range `1` through `10` is shown below. Bold text indicates user input.

```
NSJMS Messaging Bridge Configuration Tool
Copyright 2003 Hewlett Packard Development Company, L.P.

command: alter bridge

Pathmon Process Name [sampl]:

Pathway Server Class Name [qtoq]:

Database vol/subvol [$myvol.mysubvol]:

Source [queue0]:

Target [queue1]:

Selector [ ]:

Batch Size [1]: 10

Number of Servers [1]:

Store sampl.qtoq [N]: y

Generating start_sampl script

Generating sampl.properties

Generating stop_sampl script

Bridge altered
```

Delete a Bridge

The `delete bridge` command deletes an existing bridge Pathway configuration, updates the `bridges.properties` file, and regenerates the start and stop scripts.

Delete Destination Command Prompts

The `delete bridge` command steps through these command prompts. At each prompt, enter the appropriate value or press Enter to proceed to the next prompt. Pressing Enter without entering a new value will select the value, if any, which appears in brackets ([ ]) at each prompt. For example, `Pathway Server Class Name [qtoq]:`.

Prompt	Usage
`Pathmon Process Name:`	Required. Enter the Pathmon Process Name for the bridge you want to delete.
`Pathway Server Class Name:`	Required. Enter the Pathway Server Class Name for the bridge you want to delete.
`Delete` *`Pathmon_Process_Name.Pathway_Server_Class_Name`*`:`	Enter y to delete the bridge from the `bridge.properties` file. Enter n to leave the task without deleting the bridge.

Example

An example of using the configuration tool to delete the bridge named `sampl.qtoq` is shown below. Bold text indicates user input.

```
NSJMS Messaging Bridge Configuration Tool
Copyright 2003 Hewlett Packard Development Company, L.P.

command: delete bridge

Pathmon Process Name [sampl]:

Pathway Server Class Name [qtoq]:

Delete sampl.qtoq [N]: y

Generating start_sampl script

Generating sampl.properties

Generating stop_sampl script

Bridge deleted
```

Request Information about a Bridge

The info bridge command returns bridge Pathway configuration information.

Info Bridge Command Prompts

The `info bridge` command steps through these command prompts. At each prompt, enter the appropriate value or press Enter to proceed to the next prompt. Pressing Enter without entering a new value will select the value, if any, which appears in brackets ([ ]) at each prompt. For example, `Pathway Server Class Name [qtoq]:`

Prompt	Usage
`Pathmon Process Name:`	To obtain configuration information about a specific bridge, enter the Pathmon Process Name and the Pathway Server Class Name. To obtain configuration information for all configured bridges running under a specific Pathmon Process, enter the Pathmon Process Name and the wildcard character (*) for the Pathway Server Class Name. To obtain configuration information for all configured bridges, enter the wildcard character (*) for the Pathmon Process Name.
`Pathway Server Class Name:`	See above usage instructions.

Examples

An example of using the configuration tool to request information about a specific bridge, `sampl.qtoq`, is shown below. Bold text indicates user input.

```
NSJMS Messaging Bridge Configuration Tool
Copyright 2003 Hewlett Packard Development Company, L.P.
command: info bridge
Pathmon Process Name [sampl]:
Pathway Server Class Name []: qtoq

Info for - sampl.qtoq
DbVolSubvol=$myvol.mysubvol
Source=queue0
Target=queue1
Durable=N
Selector=null
BatchSize=10
NumServers=1
```

An example of using the configuration tool to request information about all currently configured bridges, `Pathmon Process Name [sampl]: *`, is shown below. Bold text indicates user input.

```
NSJMS Messaging Bridge Configuration Tool
Copyright 2003 Hewlett Packard Development Company, L.P.
command: info bridge
Pathmon Process Name [sampl]: *

Info for - sampl.qtoq
DbVolSubvol=$myvol.mysubvol
Source=queue0
Target=queue1
Durable=N
Selector=null
BatchSize=1
NumServers=1
```

Run the Bridge from within a Pathway Environment

The bridge program runs within a Pathway environment using the latest version of executable Pathway scripts (start_*pathmon_name* and stop_*pathmon_name*). These scripts assist you in starting and stopping the Pathway environment.

Starting the Bridge

To start a bridge within a Pathway environment, run the start script from within the OSS shell (`OSH`) by passing in a configuration filename (start_*pathmon_name*). For example, to start the bridge added in Add a Bridge Pathway Configuration with Server Class on page 5-10, type this command:

```
OSH>./start_sampl
```

Stopping the Bridge

To stop a bridge within a Pathway environment, run the stop script from within the OSS shell (`OSH`) by passing in a configuration filename (stop_*pathmon_name*). For example, to stop the bridge added in Add a Bridge Pathway Configuration with Server Class on page 5-10, type this command:

```
OSH>./stop_sampl
```

Monitor the Pathway and Bridge Server Process Log Files

All PATHMON process events that occur while your PATHMON environment is running are written to a central log file, *pathmon_process_name*`.pathmon.log`. For example, output generated by Pathway when the Pathway environment is started in Starting the Bridge on page 5-18 are contained in the `sampl.pathmon.log` file.

All bridge processes write their log records to a common log file, *pathmon_process_name*`.log`. For example, any log records generated by the bridge processes in Starting the Bridge on page 5-18 are contained in the `sampl.log` file.

Files Generated by the Configuration Tool

The configuration tool generates the executable Pathway start and stop scripts (start_*pathmon_name* and stop_*pathmon_name*), and the *pathmon_process_name*`.properties` file. These files are described in the following pages.

Start Script

The configuration tool replaces generic configuration placeholders contained in the `start_template.txt` file with bridge Pathway configured values when the executable Pathway start script (`start_`*`pathmon_name`*) is generated using the `add bridge`, `alter bridge`, or `delete bridge` commands. The start script is used to start a bridge within a Pathway environment (see Starting the Bridge on page 5-18).

The generic configuration placeholders replaced in the `start_template.txt` file are shown in bold text in Example 5-2 on page 5-20). A sample executable Pathway start script containing the bridge Pathway configured values in bold text is shown in Example 5-3 on page 5-23).

Example 5-2. start_template.txt File (page 1 of 3)

```
#! /bin/sh
# Script for starting a NSJMS Messaging bridge environment
#
# Usage:
#    start
# The "start" script will start the messaging bridge environment.
# Pathcom status messages are redirected to a file.
#
# Expected environment variables:
#    JAVA_HOME        top level java directory
#
# Optional environment variables:
#    HOMETERM
#
# If the environment variable HOMETERM is non-null it will be
# used as the home terminal for output display.  Otherwise the
# window from which the script was run is used for output display.
#
# (C) Copyright 2003 Hewlett-Packard
#
# Get -v option if supplied
#
while getopts ":v" opt; do
case $opt in
    v ) VERBOSE="-v" ;;
    \? ) print "usage: start [-v]"
    return 1
    esac
done
shift $(($OPTIND - 1))
if [[ -z $JAVA_HOME ]] then
    print "Environment variable JAVA_HOME must be set prior to running $0"
    return 1
fi

#
# Check for NonStop Java - first T0083 then T2766. (For U64 support, Java SPR
# T2766H70 or above is supported).
#
JAVA_BINARY="$JAVA_HOME/bin/oss/posix_threads/java"
if [[ -f $JAVA_BINARY ]]; then
   print "Found NonStop Java in $JAVA_BINARY"
else
   JAVA_BINARY="$JAVA_HOME/bin/java"
   if [[ -f $JAVA_BINARY ]]; then
      print "Found NonStop Java in $JAVA_BINARY"
   else
      print "NonStop Java has not been installed on the system, please  "
      print "install NonStop Server for Java to the $JAVA_HOME directory "
      print "and then run this script again. "
      print ""
      print "Exiting ... "
      print ""
      return 1
   fi
fi
#
# Set G_HOMETERM to the terminal identifier on which standard
# output is displayed.
#
```

Example 5-2. start_template.txt File (page 2 of 3)

```
if [[ -z "$HOMETERM" ]] then
    X=$(who -m)
    CNT=$((1))
    for i in $X; do
      if ((CNT==2))
      then
        O_HOMETERM=$i
        break
    fi
    ((CNT+=1))
    done
    HOMETERM=$(basename $O_HOMETERM)
    TCP_PROCESS=$(basename $(dirname $O_HOMETERM))
    G_HOMETERM=\$$TCP_PROCESS.$HOMETERM
    SLEEP_TIME=10000
else
  O_HOMETERM=/G/ZTNT/$HOMETERM
  G_HOMETERM=$ZTNT.$HOMETERM
  SLEEP_TIME=0
fi
#
# Start a PATHMON envionment for the Messaging bridge processes:
#
export PATHMON=@@PATHMON_NAME@@
PMLOG=$PWD/$PATHMON.pathmon.log
#
gtacl -c "pathmon/name \$$PATHMON,nowait,out $PMLOG/"
sleep 1
gtacl -p pathcom \$$PATHMON > $PMLOG <<eof
set pathway maxassigns          8
set pathway maxparams           8
set pathway maxserverclasses   50
set pathway maxserverprocesses 50
set pathway maxstartups        40
set pathway maxtcps             0
set pathway maxterms            0
set pathway maxdefines         27
set pathway maxpathcoms         8
set pathway maxspi              8
set pathway maxlinkmons        16
set pathway maxexternaltcps     0
set pathway maxprograms         5
start pathway cold !
@@BEGIN_SERVER@@
reset server
set server processtype oss
set server pri         150
set server cwd         $PWD
set server program     $JAVA_BINARY
set server env         JREHOME=$JAVA_HOME/jre
set server hometerm $G_HOMETERM
set server stdin /dev/null
set server createdelay  0 secs
set server deletedelay  0 secs
set server TIMEOUT      0 secs
set server MAXLINKS    16
set server LINKDEPTH    1
set server AUTORESTART 10
set server arglist &
"-Dsqlmx_nowait=on",&
"-classpath",&
@@CLASSPATH@@
"com.tandem.nsjms.bridge.Bridge",&
```

Example 5-2. start_template.txt File (page 3 of 3)

```
@@SERVER_PARAMS@@
set server stdout $PWD/@@STDOUT@@
set server stderr $PWD/@@STDERR@@
set server maxservers @@MAXSERVERS@@
set server numstatic  @@NUMSTATIC@@
add server @@SERVER_NAME@@
start server @@SERVER_NAME@@
@@END_SERVER@@
eof
grep -q -i error $PWD/sampl.pathmon.log
if [[ $? -eq 0 ]] then
    print "Error in starting pathcom, refer to pathmon.log for information"
  else
    print "Messaging Bridge environment was started."
    print "Individual servers can take some time to activate."
fi
```

Example 5-3. Start Script Sample (page 1 of 3)

```
#! /bin/sh
# Script for starting a NSJMS Messaging bridge environment
#
# Usage:
#    start
# The "start" script will start the messaging bridge environment.
# Pathcom status messages are redirected to a file.
#
# Expected environment variables:
#    JAVA_HOME        top level java directory
#
# Optional environment variables:
#    HOMETERM
#
# If the environment variable HOMETERM is non-null it will be
# used as the home terminal for output display.  Otherwise the
# window from which the script was run is used for output display.
#
# (C) Copyright 2003 Hewlett-Packard
#
# Get -v option if supplied
#
while getopts ":v" opt; do
case $opt in
    v ) VERBOSE="-v" ;;
    \? ) print "usage: start [-v]"
    return 1
    esac
done
shift $(($OPTIND - 1))
if [[ -z $JAVA_HOME ]] then
    print "Environment variable JAVA_HOME must be set prior to running $0"
    return 1
fi

#
# Check for NonStop Java - first T0083 then T2766. (For U64 support, Java SPR
# T2766H70 or above is supported).
#
JAVA_BINARY="$JAVA_HOME/bin/oss/posix_threads/java"
if [[ -f $JAVA_BINARY ]]; then
   print "Found NonStop Java in $JAVA_BINARY"
else
   JAVA_BINARY="$JAVA_HOME/bin/java"
   if [[ -f $JAVA_BINARY ]]; then
      print "Found NonStop Java in $JAVA_BINARY"
   else
      print "NonStop Java has not been installed on the system, please  "
      print "install NonStop Server for Java to the $JAVA_HOME directory "
      print "and then run this script again. "
      print ""
      print "Exiting ... "
      print ""
      return 1
   fi
fi
#
# Set G_HOMETERM to the terminal identifier on which standard
# output is displayed.
#
```

Example 5-3. Start Script Sample (page 2 of 3)

```
if [[ -z "$HOMETERM" ]] then
    X=$(who -m)
    CNT=$((1))
    for i in $X; do
      if ((CNT==2))
      then
        O_HOMETERM=$i
        break
    fi
    ((CNT+=1))
    done
    HOMETERM=$(basename $O_HOMETERM)
    TCP_PROCESS=$(basename $(dirname $O_HOMETERM))
    G_HOMETERM=\$$TCP_PROCESS.$HOMETERM
    SLEEP_TIME=10000
else
  O_HOMETERM=/G/ZTNT/$HOMETERM
  G_HOMETERM=$ZTNT.$HOMETERM
  SLEEP_TIME=0
fi
#
# Start a PATHMON envionment for the Messaging bridge processes:
#
export PATHMON=sampl
PMLOG=$PWD/$PATHMON.pathmon.log
#
gtacl -c "pathmon/name \$$PATHMON,nowait,out $PMLOG/"
sleep 1
gtacl -p pathcom \$$PATHMON > $PMLOG <<eof
set pathway maxassigns          8
set pathway maxparams           8
set pathway maxserverclasses   50
set pathway maxserverprocesses 50
set pathway maxstartups        40
set pathway maxtcps             0
set pathway maxterms            0
set pathway maxdefines         27
set pathway maxpathcoms         8
set pathway maxspi              8
set pathway maxlinkmons        16
set pathway maxexternaltcps     0
set pathway maxprograms         5
start pathway cold !
reset server
set server processtype oss
set server pri         150
set server cwd         $PWD
set server program     $JAVA_HOME/bin/oss/posix_threads/java
set server env         JREHOME=$JAVA_HOME/jre
set server hometerm $G_HOMETERM
set server stdin /dev/null
set server createdelay  0 secs
set server deletedelay  0 secs
set server TIMEOUT      0 secs
set server MAXLINKS    16
set server LINKDEPTH    1
set server AUTORESTART 10
set server arglist &
"-Dsqlmx_nowait=on",&
"-classpath",&
```

Example 5-3. Start Script Sample (page 3 of 3)

```
"/usr/tandem/nsjms/T1251V30_30SEP2003_V30:&
/usr/tandem/nsjms/T1251V30_30SEP2003_V30/lib/nsjms.jar:&
/usr/tandem/nsjms/T1251V30_30SEP2003_V30/lib/jndi.jar:&
/usr/tandem/nsjms/T1251V30_30SEP2003_V30/lib/providerutil.jar:&
/usr/tandem/nsjms/T1251V30_30SEP2003_V30/lib/fscontext.jar",&
"com.tandem.nsjms.bridge.Bridge",&
"-pp","sampl",&
"-sc","qtoq",&
"-sd","queue0",&
"-sjd","queue0",&
"-sjicf","com.sun.jndi.fscontext.RefFSContextFactory",&
"-sjurl","file:////usr/tandem/nsjms/T1251V30_30SEP2003_V30
   /jndi_object_store",&
"-sjcf","QueueConnectionFactory",&
"-td","queue1",&
"-tjd","queue1",&
"-tjicf","com.sun.jndi.fscontext.RefFSContextFactory",&
"-tjurl","file:////usr/tandem/nsjms/T1251V30_30SEP2003_V30
   /jndi_object_store",&
"-tjcf","QueueConnectionFactory",&
"-d","N",&
"-s","null",&
"-bs","1"
set server stdout $PWD/sampl.log
set server stderr $PWD/sampl.log
set server maxservers 1
set server numstatic  1
add server qtoq
start server qtoq

eof
grep -q -i error $PWD/sampl.pathmon.log
if [[ $? -eq 0 ]] then
    print "Error in starting pathcom, refer to pathmon.log for information"
  else
    print "Messaging Bridge environment was started."
    print "Individual servers can take some time to activate."
fi
```

Stop Script

The configuration tool replaces a generic configuration placeholder contained in the `stop_template.txt` file with a bridge Pathway configured value when the executable Pathway stop script (`stop_`*`pathmon_name`*) is generated using the `add bridge`, `alter bridge`, or `delete bridge` commands. The stop script is used to stop a bridge within a Pathway environment (see Stopping the Bridge on page 5-18).

The generic configuration placeholder replaced in the `stop_template.txt` file is shown in bold text in Example 5-4 on page 5-26. A sample executable Pathway stop script containing the bridge Pathway configured values in bold text is shown in Example 5-5 on page 5-26.

Example 5-4. stop_template.txt File

```
#! /bin/sh
# Script for shuting down the Messaging Bridge pathway environment
#
# Execute comand to gracefully shut down servers
#
# Tell pathmon to shut down.
#
export PATHMON=@@PATHMON_NAME@@
#
gtacl -p pathcom \$$PATHMON <<eof
shutdown2, mode im, status ag
eof
# Just in case pathway wasn't configured, kill the pathmon process.
# If the shutdown above worked, this will have no effect since the
# pathmon process is gone. Otherwise this gets rid of the pathmon process.
kill -s GUARDIAN /G/$PATHMON > /dev/null 2>&1
```

Example 5-5. Stop Script Sample

```
#! /bin/sh
# Script for shuting down the Messaging Bridge pathway environment
#
# Execute comand to gracefully shut down servers
#
# Tell pathmon to shut down.
#
export PATHMON=sampl
#
gtacl -p pathcom \$$PATHMON <<eof
shutdown2, mode im, status ag
eof
# Just in case pathway wasn't configured, kill the pathmon process.
# If the shutdown above worked, this will have no effect since the
# pathmon process is gone. Otherwise this gets rid of the pathmon process.
kill -s GUARDIAN /G/$PATHMON > /dev/null 2>&1
```

pathmon_process_name.properties File

The *pathmon_process_name*.properties file is generated using the add bridge command and is regenerated by the alter bridge and delete bridge commands. The file is read by the bridge at startup, and contains the two properties listed below in addition to the properties contained in the nsjms.properties file. For information on the contents of the nsjms.properties file, see page 2-12.

- Bridge.receive.timeout

Default:	60000
Values:	*milliseconds*
Value Descriptions:	*milliseconds*—The time (in milliseconds) the receive() method waits for a message on the source queue or topic.
Example:	Bridge.receive.timeout=60000

- Bridge.reconnect.delay

Default:	60000
Values:	*`milliseconds`*
Value Descriptions:	*`milliseconds`*—The time (in milliseconds) between when a connection is lost and an attempt is made to reconnect.
Example:	`Bridge.reconnect.delay=60000`

An example of the `sampl.properties` file generated during the Add a Bridge Pathway Configuration with Server Class on page 5-10 is

```
#NSJMS Messaging Bridge Properties for sampl
#Fri Aug 08 10:24:21 PDT 2003
Bridge.receive.timeout=60000
Logger.filename=sampl.log
Bridge.reconnect.delay=60000
Database.delay=1000
Logger.loglevel=info
Database.timeout=100
Database.volsubvol=$myvol.mysubvol
```

Optional Files Created by the User

You can override or add to the properties for a server class by creating files which contain properties that the bridge process will apply only to the specified server class. The *`server_class`*`.jndi.properties` file and the *`pathmon_process_name`*`.`*`server_class_name`*`.properties` file are described below.

`server_class``.jndi.properties` File

The *`server_class`*`.jndi.properties` file is an optional file you create if you have additional JNDI properties to specify for a specific server class in addition to the general JNDI properties specified in the `add dest` command.

Note. The properties in the *`server_class`*`.jndi.properties` file are applied only to the specified server class.

You create this file using an editor of your choice and populate the file with the additional JNDI properties that you want to apply to the specified server class. The file must be named

`server_class``.jndi.properties`

where *`server_class`* is the name entered at the `Pathway Server Class Name:` prompt in the `add bridge` command (for example, `qtoq.jndi.properties`).

At startup, the JNDI values specified in the `add dest` command are passed to the bridge process on the command line in the start script. The bridge process then searches it's CLASSPATH for the *server_class*.jndi.properties file. If the file is found, the bridge process uses the properties in the *server_class*.jndi.properties file to override or add to the properties previously passed in from the command line. The bridge process then initializes it's JNDI context.

pathmon_process_name.*server_class_name*.properties File

The *pathmon_process_name*.*server_class_name*.properties file is an optional file you create if you want to specify property name/value pairs for a specific server class. The bridge process uses the *pathmon_process_name*.*server_class_name*.properties file to override or add to the property name/value pairs contained in the *pathmon_process_name*.properties file.

Note. The properties in the *pathmon_process_name*.*server_class_name*.properties file are applied only to the specified server class.

You create this file using an editor of your choice and populate the file with the property name/value pairs that you want to apply to the specified server class. The file must be named

pathmon_process_name.*server_class_name*.properties

where

pathmon_process_name is the name entered at the Pathmon Process Name: prompt for the add bridge command

and

server_class_name is the name entered at the Pathway Server Class Name: prompt for the add bridge command.

For example, sampl.qtoq.properties.

At startup, the bridge process reads the properties contained in the *pathmon_process_name*.properties file. The bridge process then searches it's CLASSPATH for the *pathmon_process_name*.*server_class_name*.properties file. If the file is found, the bridge process reads the properties contained in the *pathmon_process_name.server_class_name*.properties file and applies these properties to the specified server class in addition to the existing properties in the *pathmon_process_name*.properties file.

For example, the bridge process reads in the properties in the below sampl.properties file generated during the Add a Bridge Pathway Configuration with Server Class on page 5-10.

```
#NSJMS Messaging Bridge Properties for sampl
#Fri SEP 30 10:24:21 PDT 2003
Bridge.receive.timeout=60000
Logger.filename=sampl.log
Bridge.reconnect.delay=60000
Database.delay=1000
Logger.loglevel=info
Database.timeout=100
Database.volsubvol=$myvol.mysubvol
```

The bridge process then searches its CLASSPATH and finds the below user created sampl.qtoq.properties file.

```
Database.volsubvol=$myvol.anothersubvol
```

The bridge process uses the Database.volsubvol property, $myvol.anothersubvol, in the sampl.qtoq.properties file as the location for the Database table for the server class named qtoq. All other server classes running under the Pathmon process named sampl would still use the Database.volsubvol property, $myvol.mysubvol, in the sampl.properties file.

6

NSJMS Administrative Servlet Installation and Configuration

This section describes the process for installing the optional NSJMS administrative servlet on your NonStop servers. The administrative servlet enables you to access the NSJMS administrative utility through an XML interface by using an XML formatted request/reply paradigm in a user-defined management application (for example, your web browser). For information about the administrative utility accessed through an XML interface, see The XML Interface on page 7-17.

NSJMS Administrative Servlet Installation Process

NSJMS Administrative Servlet System Requirements

Hardware

HP NonStop S-series server

Software

- ITP Secure WebServer (T8997), Release 5

And one of these:

- Java Servlets (T0094V20)
- NonStop Servlets for JavaServer Pages (T1222V1X)
- NonStop Servlets for JavaServer Pages (T1222V20)

Administrative Servlet Installation Procedure for Java Servlets

Use this installation procedure if you are running Java Servlets (T0094V20).

1. Log on as the owner of the `/usr/tandem/webserver/conf` directory.

Note. By default, access to the `/usr/tandem/webserver/conf` directory is restricted to the owner of the directory structure, the user ID under which the iTP WebServer was installed. The directory owner can allow anyone to access the directory, however, the system supervisor can always access the directory.

2. Use the `cd` command to go to the directory where the ITP Secure WebServer configuration file is located. For example:

```
OSH: cd /usr/tandem/webserver/conf
```

3. Using an editor of your choice, modify the iTP Secure WebServer `servlet.config` file as:

   a. Locate the line `set server_objectcode $root/bin/servlet.ssc` and insert this line directly below it:

   ```
   set env(NSJMS_HOME) /usr/tandem/nsjms/version
   ```

   The `env` parameter specifies the environment variable for the NSJMS administrative servlet, and `/usr/tandem/nsjms/`*`version`* represents the home directory for the NSJMS installation.

   b. Locate where the Tool Command language (Tcl) variable JVCP is set and insert this command directly below where JAVA_HOME is set:

   ```
   foreach i [glob -nocomplain $env(NSJMS_HOME)/lib/*.zip
   $env(NSJMS_HOME)/lib/*.jar] { append JVCP $i ":" }
   ```

   This Tcl command sets up the class PATH variable for NSJMS.

   c. Locate the line `set USRCP $root/samples/Servlets:$root/samples/Servlets/SunSamples` and append this information to the end of that line:

   ```
   :$root/samples/Servlets/nsjms:$env(NSJMS_HOME)
   ```

4. Create the NSJMS servlet directory `/usr/tandem/webserver/samples/Servlets` by executing these commands:

```
OSH: cd /usr/tandem/webserver/samples/Servlets
OSH: mkdir nsjms
```

5. Copy the `JmsAdminServlet.html` sample program file located in the NSJMS examples directory to the NSJMS directory created in Step 4:

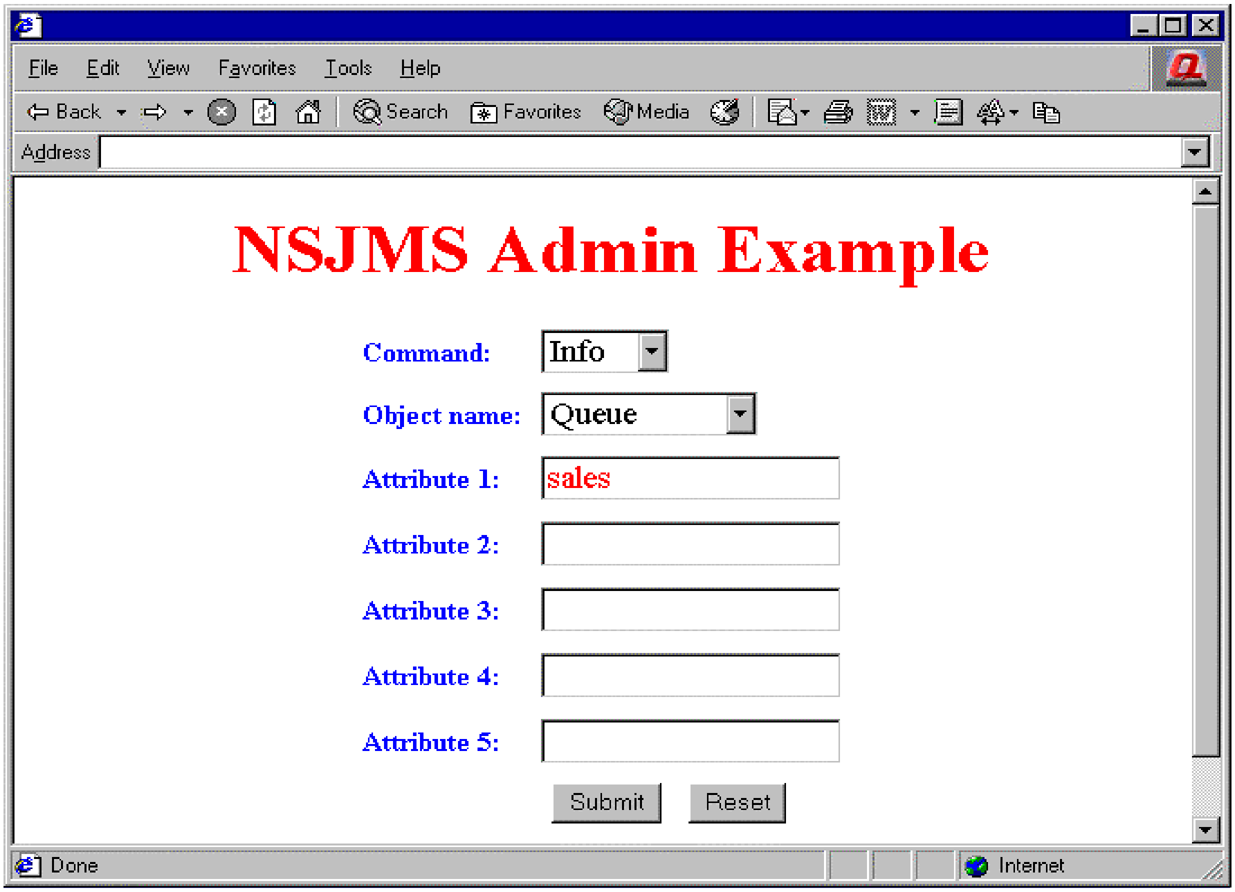
```
OSH: cp /usr/tandem/nsjms/version/examples/JmsAdminServlet.html_
        /usr/tandem/webserver/samples/Servlets/nsjms
```

   where `/usr/tandem/nsjms/`*`version`* represents the home directory of the NSJMS installation.

6. Restart the iTPSecure WebServer by executing these commands:

```
OSH: cd /usr/tandem/webserver/conf
OSH: ./restart
```

7. The servlet can now be activated by accessing the JmsAdminServlet.html file, http://www.*yourserversaddress.com*/samples/Servlets/nsjms/JmsAdminServlet.html, through your web browser. For example:



NSJMS Admin Example

Command: Info

Object name: Queue

Attribute 1: sales

Attribute 2:

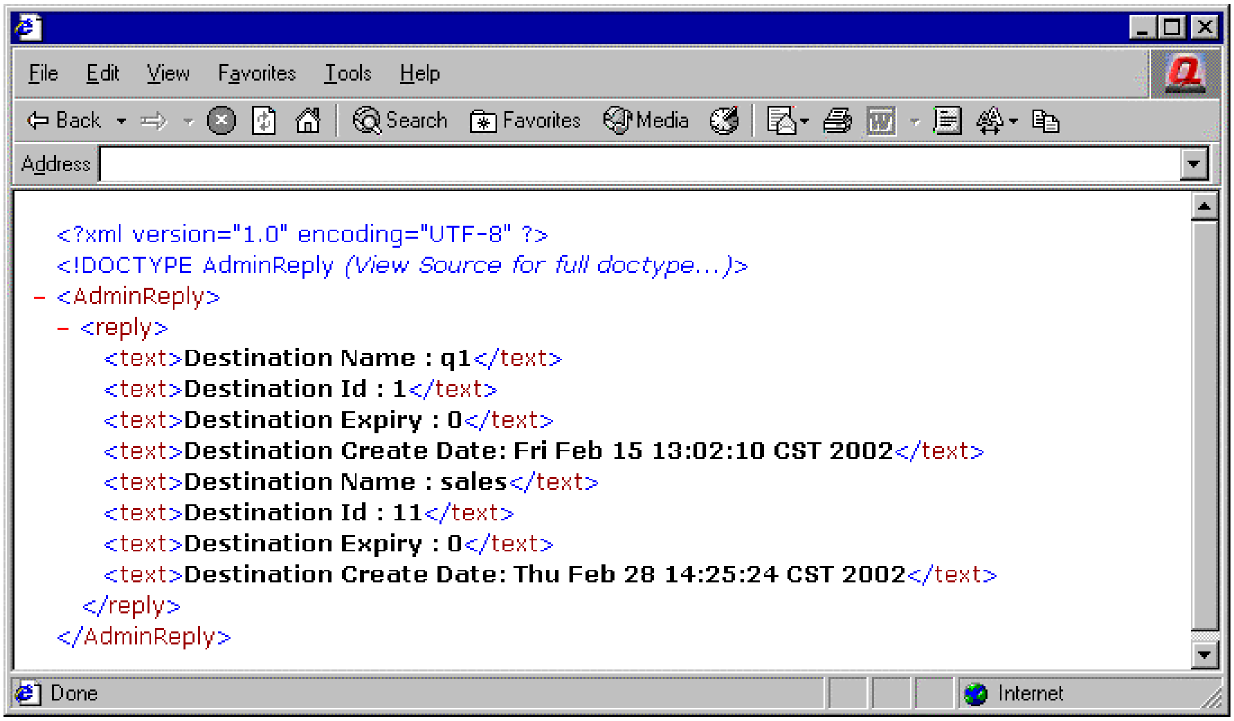
Attribute 3:

Attribute 4:

Attribute 5:

VST003.vsd

Servlet replies are returned as XML formatted data. For example:


File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Done
Internet

```
  <?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE AdminReply (View Source for full doctype...)>
- <AdminReply>
  - <reply>
      <text>Destination Name : q1</text>
      <text>Destination Id : 1</text>
      <text>Destination Expiry : 0</text>
      <text>Destination Create Date: Fri Feb 15 13:02:10 CST 2002</text>
      <text>Destination Name : sales</text>
      <text>Destination Id : 11</text>
      <text>Destination Expiry : 0</text>
      <text>Destination Create Date: Thu Feb 28 14:25:24 CST 2002</text>
    </reply>
  </AdminReply>
```

VST004.vsd

Administrative Servlet Installation Procedure for NonStop Servlets for JavaServer Pages (T1222V1X)

Use this installation procedure if you are running NonStop Servlets for JavaServer Pages (T1222V1X)

1. Log on as the owner of the `/usr/tandem/webserver/conf` directory.

Note. By default, access to the `/usr/tandem/webserver/conf` directory is restricted to the owner of the directory structure, the user ID under which the iTP WebServer was installed. The directory owner can allow anyone to access the directory, however, the system supervisor can always access the directory.

2. Use the `cd` command to go to the directory where the ITP Secure WebServer configuration file is located. For example:

```
OSH: cd /usr/tandem/webserver/conf
```

3. Using an editor of your choice, modify the iTP Secure WebServer `servlet.config` file:

    a. Locate the statement:

    ```
    if {![info exists env(GUARDIAN_SUBVOL)]} {set
    env(GUARDIAN_SUBVOL) /G/system/zweb }
    ```

    Insert this line directly below it:

    ```
    if {![info exists env(NSJMS_HOME)]} {set
    env(NSJMS_HOME)  /usr/tandem/nsjms/version }
    ```

    The `env` parameter specifies the environment variable for the NSJMS administrative servlet, and `/usr/tandem/nsjms/`*`version`* represents the home directory for the NSJMS installation.

    b. Locate the line `set USRCP $root/root/samples/Servlets` and append this information to the end of that line:

    ```
    :$env(NSJMS_HOME):$env(NSJMS_HOME)/lib/nsjms.jar:$env
    (NSJMS_HOME)/lib/fscontext.jar:$env(NSJMS_HOME)/lib/
    providerutil.jar
    ```

4. Use the `cd` command to go to the directory where the ITP Secure WebServer `iTP_server.xml` file is located; for example:

```
OSH: cd /usr/tandem/webserver/servlet_jsp/conf
```

5. Using an editor of your choice, modify the iTP Secure WebServer `iTP_server.xml` file:

    Locate the comment:

    ```
    <!--  Add your context mappings here. -->
    ```

    Insert this statement directly below it:

    ```
    <Context path="/servlet_jsp/nsjms" docBase="webapps/nsjms"
    debug="0" reloadable="false"> </Context>
    ```

6. Create the NSJMS servlet directory `/usr/tandem/webserver/servlet_jsp/webapps/nsjms` by executing these commands:

```
OSH: cd /usr/tandem/webserver/servlet_jsp/webapps
OSH: mkdir nsjms
```

7. Copy the `JmsAdminServlet.html` sample program file located in the NSJMS examples directory to the NSJMS directory created in Step 6:

```
OSH: cp /usr/tandem/nsjms/version/examples/JmsAdminServlet.html
        /usr/tandem/webserver/servlet_jsp/webapps/nsjms
```

   where `/usr/tandem/nsjms/`*`version`*`/examples/JmsAdminServlet.html` represents the examples directory of the NSJMS installation.

8. Using an editor of your choice, modify the `JmsAdminServlet.html` file copied in Step 7:

   Locate the command:

```
<FORM METHOD="POST"  NAME="AdminRequest"
ACTION="/servlet/com.tandem.nsjms.admin.JmsAdminServlet">
```

   Change the command to:

```
<FORM METHOD="POST"  NAME="AdminRequest"
ACTION="servlet/JmsAdminServlet">
```

9. Create the NSJMS servlet directory `/usr/tandem/webserver/servlet_jsp/webapps/nsjms/WEB-INF` by executing these commands:

```
OSH: cd /usr/tandem/webserver/servlet_jsp/webapps/nsjms
OSH: mkdir WEB-INF
```

10. Use the cd command to go to the WEB-INF directory; for example:

```
OSH: cd /usr/tandem/webserver/servlet_jsp/webapps/nsjms/WEB-INF
```

11. Copy the `web.xml` sample program file located in the NSJMS examples directory to the NSJMS directory created in Step 10:

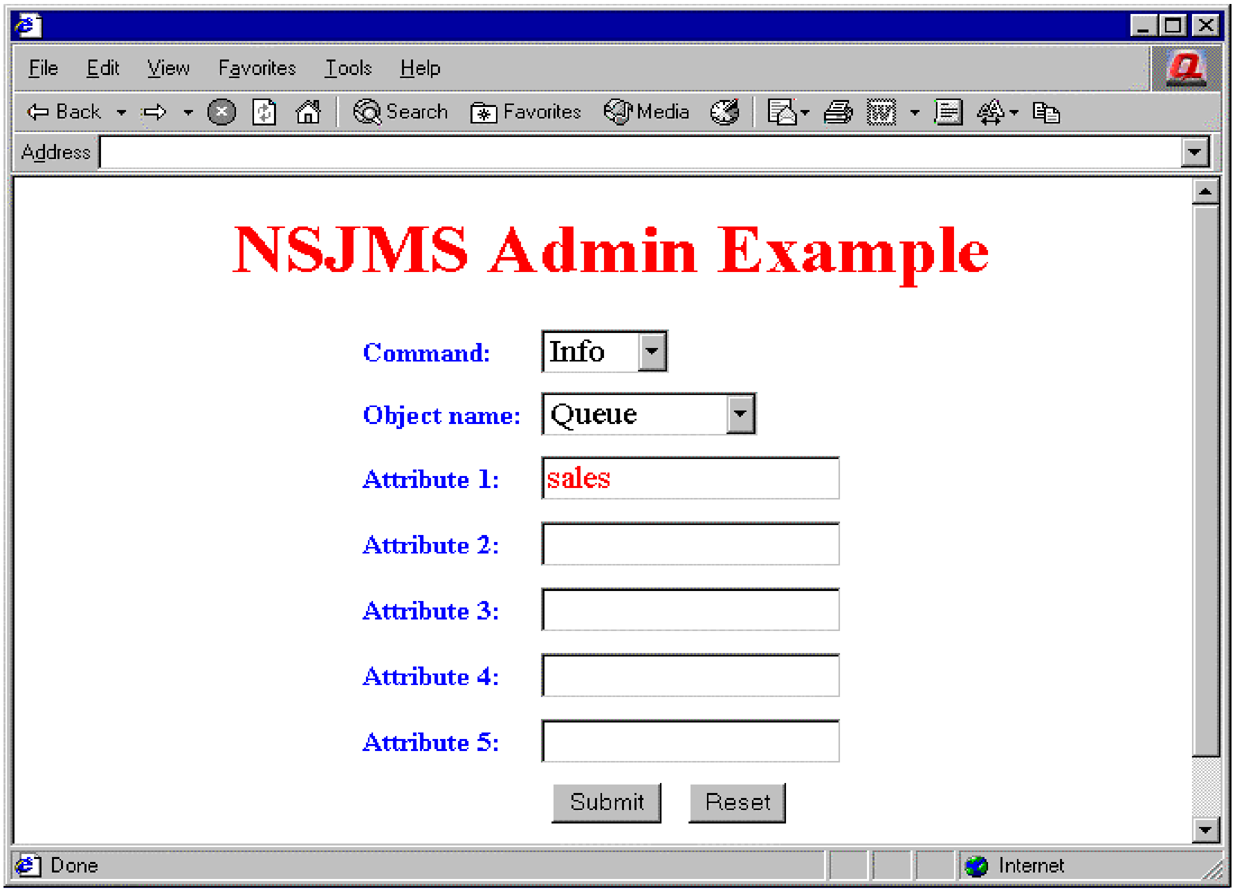
```
OSH: cp /usr/tandem/nsjms/version/examples/web.xml
        /usr/tandem/webserver/servlet_jsp/webapps/nsjms/WEB-INF
```

   where `/usr/tandem/nsjms/`*`version`*`/examples/web.xml` represents the examples directory of the NSJMS installation.

12. Restart the iTPSecure WebServer by executing these commands:

```
OSH: cd /usr/tandem/webserver/conf
OSH: ./restart
```

13. The servlet can now be activated by accessing the JmsAdminServlet.html file, http://www.`yourserversaddress.com`/servlet_jsp/nsjms/JmsAdminServlet.html, through your web browser. For example:



NSJMS Admin Example

Command: Info

Object name: Queue

Attribute 1: sales

Attribute 2:

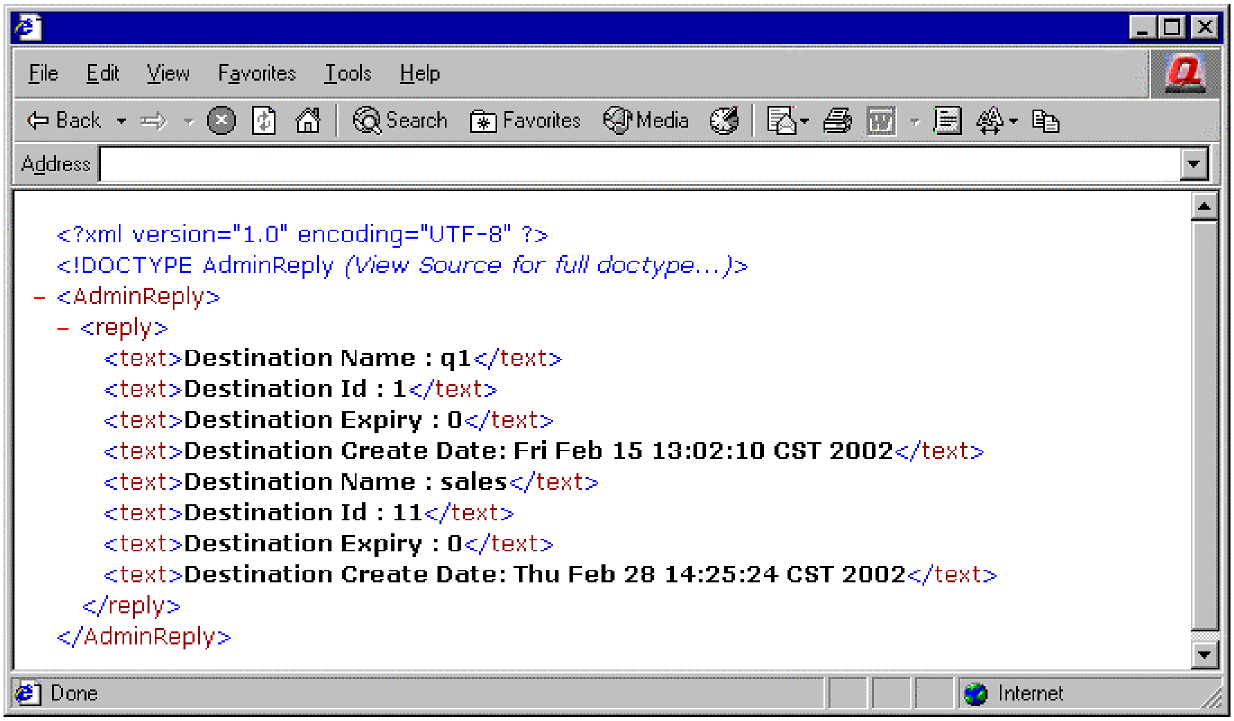
Attribute 3:

Attribute 4:

Attribute 5:

VST003.vsd

Servlet replies are returned as XML formatted data. For example:



```
  <?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE AdminReply (View Source for full doctype...)>
- <AdminReply>
  - <reply>
      <text>Destination Name : q1</text>
      <text>Destination Id : 1</text>
      <text>Destination Expiry : 0</text>
      <text>Destination Create Date: Fri Feb 15 13:02:10 CST 2002</text>
      <text>Destination Name : sales</text>
      <text>Destination Id : 11</text>
      <text>Destination Expiry : 0</text>
      <text>Destination Create Date: Thu Feb 28 14:25:24 CST 2002</text>
    </reply>
  </AdminReply>
```

VST004.vsd

Administrative Servlet Installation Procedure for NonStop Servlets for JavaServer Pages (T1222V20)

Use this installation procedure if you are running NonStop Servlets for JavaServer Pages (T1222V20).

1. Log on as the owner of the `/usr/tandem/webserver/conf` directory.

Note. By default, access to the `/usr/tandem/webserver/conf` directory is restricted to the owner of the directory structure, the user ID under which the iTP WebServer was installed. The directory owner can allow anyone to access the directory, however, the system supervisor can always access the directory.

2. Use the `cd` command to go to the directory where the ITP Secure WebServer configuration file is located; for example:

```
OSH: cd /usr/tandem/webserver/conf
```

3. Using an editor of your choice, modify the iTP Secure WebServer `servlet.config` file:

   a. Locate the statement:

      ```
      if {![info exists env(GUARDIAN_SUBVOL)]} {set
      env(GUARDIAN_SUBVOL) /G/system/zweb }
      ```

      Insert this line directly below it:

      ```
      if {![info exists env(NSJMS_HOME)]} {set
      env(NSJMS_HOME)  /usr/tandem/nsjms/version }
      ```

      The `env` parameter specifies the environment variable for the NSJMS administrative servlet, and `/usr/tandem/nsjms/`*`version`* represents the home directory for the NSJMS installation.

   b. Locate the line `set USRCP $root/root/samples/Servlets` and append this information to the end of that line:

      ```
      :$env(NSJMS_HOME)
      ```

4. Copy the NSJMS jar files located in the NSJMS installation lib directory to the directory where the iTP Secure WebServer common library files are located:

   ```
   OSH: cp /usr/tandem/nsjms/version/lib/*.jar
          /usr/tandem/webserver/servlet_jsp/common/lib
   ```

   where `/usr/tandem/nsjms/`*`version/`*`lib` represents the lib directory of the NSJMS installation.

5. Use the cd command to go to the directory where the ITP Secure WebServer `iTP_server.xml` file is located; for example:

   ```
   OSH: cd /usr/tandem/webserver/servlet_jsp/conf
   ```

6. Using an editor of your choice, modify the iTP Secure WebServer `iTP_server.xml` file:

   Locate the comment:

   ```
   <!-- Servlet BankDemo Context
     commented out, to use the example, remove the comment
     <Context path="/servlet_jsp/bankdemo" docBase="BankDemo"
     reloadable="false" debug="0"/>
    -->
   ```

   Insert this statement directly below it:

   ```
   <Context path="/servlet_jsp/nsjms" docBase="nsjms"
   debug="0" reloadable="false"> </Context>
   ```

7. Create the NSJMS servlet directory `/usr/tandem/webserver/servlet_jsp/webapps/nsjms` by executing this command:

```
OSH: mkdir /usr/tandem/webserver/servlet_jsp/webapps/nsjms
```

8. Copy the `JmsAdminServlet.html` sample program file located in the NSJMS examples directory to the NSJMS directory created in Step 7:

```
OSH: cp /usr/tandem/nsjms/version/examples/JmsAdminServlet.html
        /usr/tandem/webserver/servlet_jsp/webapps/nsjms
```

   where `/usr/tandem/nsjms/`*`version`*`/examples/JmsAdminServlet.html` represents the examples directory of the NSJMS installation.

9. Using an editor of your choice, modify the `JmsAdminServlet.html` file copied in Step 8, as:

   Locate the command:

```
<FORM METHOD="POST"  NAME="AdminRequest"
ACTION="/servlet/com.tandem.nsjms.admin.JmsAdminServlet">
```

   Change the command to:

```
<FORM METHOD="POST"  NAME="AdminRequest"
ACTION="servlet/JmsAdminServlet">
```

10. Create the NSJMS servlet directory `/usr/tandem/webserver/servlet_jsp/webapps/nsjms/WEB-INF` by executing this command:

```
OSH: mkdir /usr/tandem/webserver/servlet_jsp/webapps/nsjms/WEB-INF
```

11. Use the cd command to go to the WEB-INF directory; for example:

```
OSH: cd /usr/tandem/webserver/servlet_jsp/webapps/nsjms/WEB-INF
```

12. Copy the `web.xml` sample program file located in the NSJMS examples directory to the NSJMS directory created in Step 11:

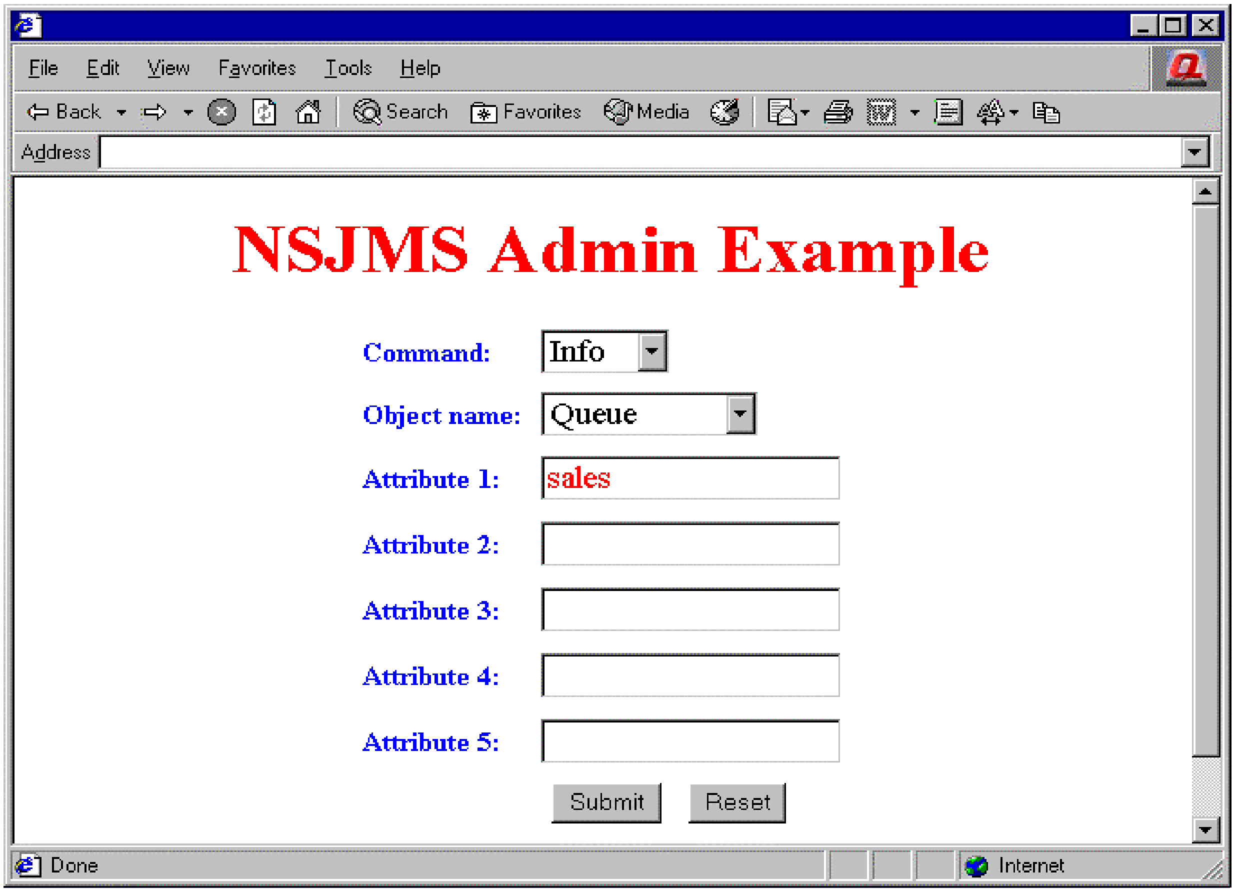
```
OSH: cp /usr/tandem/nsjms/version/examples/web.xml
        /usr/tandem/webserver/servlet_jsp/webapps/nsjms/WEB-INF
```

   where `/usr/tandem/nsjms/`*`version`*`/examples/web.xml` represents the examples directory of the NSJMS installation.

13. Restart the iTPSecure WebServer by executing these commands:

```
OSH: cd /usr/tandem/webserver/conf
OSH: ./restart
```

14. The servlet can now be activated by accessing the JmsAdminServlet.html file, http://www.`yourserversaddress.com`/servlet_jsp/nsjms/JmsAdminServlet.html, through your web browser. For example:



NSJMS Admin Example

Command: Info

Object name: Queue

Attribute 1: sales

Attribute 2:

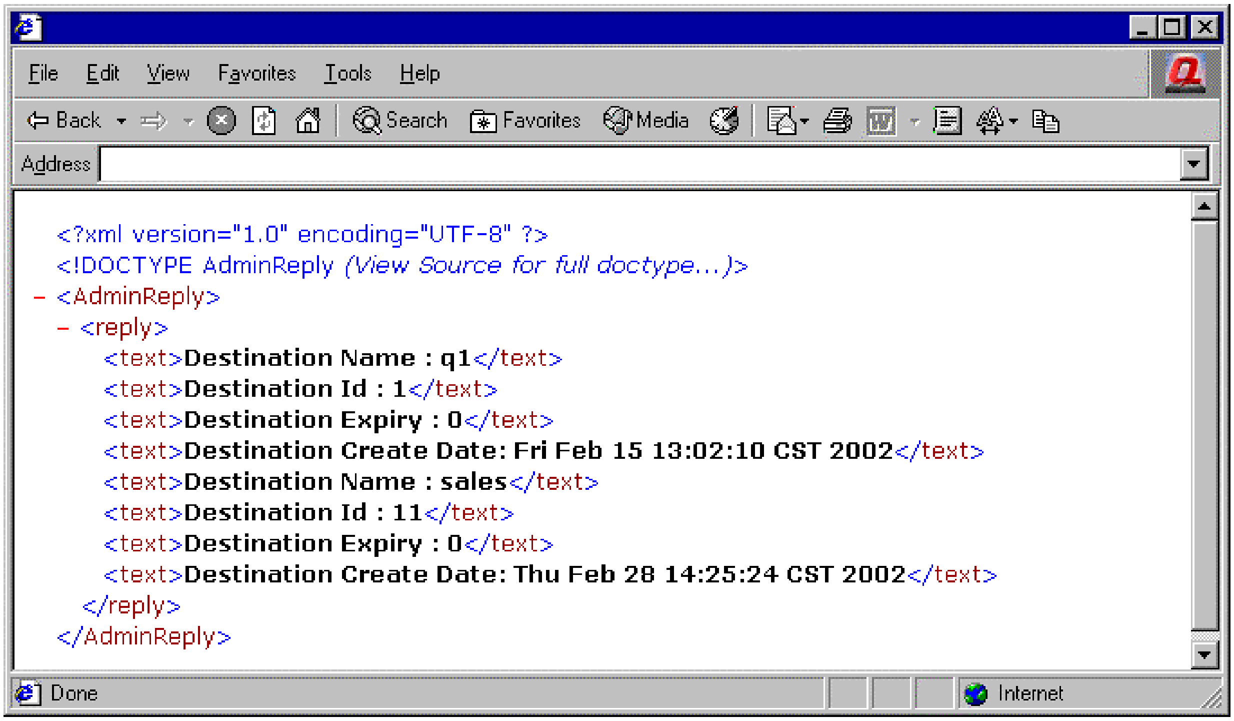
Attribute 3:

Attribute 4:

Attribute 5:

VST003.vsd

Servlet replies are returned as XML formatted data. For example:



```
  <?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE AdminReply (View Source for full doctype...)>
- <AdminReply>
  - <reply>
      <text>Destination Name : q1</text>
      <text>Destination Id : 1</text>
      <text>Destination Expiry : 0</text>
      <text>Destination Create Date: Fri Feb 15 13:02:10 CST 2002</text>
      <text>Destination Name : sales</text>
      <text>Destination Id : 11</text>
      <text>Destination Expiry : 0</text>
      <text>Destination Create Date: Thu Feb 28 14:25:24 CST 2002</text>
    </reply>
  </AdminReply>
```

VST004.vsd

7

Managing the NSJMS Environment

This section provides information about using the administrative utility to manage the NSJMS environment and information about using the SQL conversational interface (SQLCI) to query and manage an NSJMS database.

Using the Administrative Utility to Manage the NSJMS Environment

The administrative utility is a Java-based tool that provides administrative functions to manage the NSJMS environment. Since the NSJMS C++ API shares the NSJMS environment, the administrative utility is also used to manage the NSJMS C++ API environment. You can invoke the administrative utility through a command-line interface as a command-line tool for local use or through an XML interface as a servlet using an XML formatted request/reply paradigm for use in user-defined management applications. The command-line tool installs automatically with the installation of NSJMS. The administrative servlet is optional; see Section 5, Reliable Messaging Bridge

Note. The administrative utility relies on the security features of the Guardian environment and NonStop SQL/MX to protect the information in the NSJMS database. Therefore, use the same user ID to manage the data as was used to create the database.

Authorized users can use the administrative utility to:

- Clear the database of expired messages (including temporary queues and topics)
- Add or delete queues and topics
- Delete durable connections
- Initialize JNDI connection bindings
- Display information about queues, topics, durable connections, and messages
- Review statistics about queues, topics, and the NSJMS service

The Command-Line Interface

The command-line interface provides a local command-line tool for a Java administrator to access and manage the NSJMS environment. You invoke the command-line interface from the OSS shell (osh) by using this command:

```
osh> java com.tandem.nsjms.admin.JmsAdmin
```

When this command-line interface prompt appears,

```
nsjms->
```

you can access and manage the NSJMS environment by using any of these commands, which are described in detail in the following pages:

ADD HELP

ALTER INFO

DELETE STATS

EXIT

Specifying Lines Per Page

You can specify the number of lines of returned output that is displayed when using the command-line interface by specifying the Lines Per Page (`-llp`) argument from the OSS shell.

The `-llp` argument specifies the number of lines of returned output that is displayed by the processed command before you are prompted to display more lines. The `-llp` argument default is 24 lines per page. This example specifies that 30 lines of returned output will be displayed per page by the processed command:

```
osh>java com.tandem.nsjms.admin.JmsAdmin -llp 30
```

ADD Command

The ADD command adds a queue or topic to the database or NSJMS connection factories to the JNDI space.

```
ADD { QUEUE name [, EXPIRY millisecs ]
                 [, ID dest-id ] |

      TOPIC name [, EXPIRY millisecs ]
                 [, ID dest-id ] |

      JNDI name }
```

name

is the name to be associated with the queue or topic to be added to the database. Queue names and Topic names must be Java letters and digits. The first character must be a Java letter and the length is limited to 100 characters.

EXPIRY *millisecs*

specifies the time (in milliseconds) after a message is sent until the message expires. If the EXPIRY attribute value is set to zero, a message never expires. The expiry attribute value is used when no `timetolive` value is specified during a send message operation.

Default	0
Units	milliseconds
Range	0 through 9223372036854775807

`ID`

specifies the location of the queue or topic in the NSJMS database.

Default	If no ID is specified, the ID is generated by adding one to the current highest ID.
Range	1 through 32767

Rules

- The `jndi` attribute is not applicable for the NSJMS C++ API environment.
- When `queue` or `topic` is specified, *`name`* is required.
- When `jndi` is specified, `name` is invalid.
- The expiry attribute value is used when no `timetolive` value is specified during a send-message operation.

Examples

- To add a new queue (named `sales-queue`) and set messages to expire 10 minutes after they are received:

  ```
  nsjms->add queue sales-queue, expiry 600000
  ```

- To add NSJMS connection factories to the JNDI space:

  ```
  nsjms->add jndi
  ```

ALTER Command

The ALTER command alters the parameter of a queue or topic in the database.

```
ALTER { QUEUE name {, EXPIRY millisecs } |

        TOPIC name {, EXPIRY millisecs } | }
```

`name`

is the name associated with an existing queue or topic whose parameters are to be altered.

`EXPIRY` *`millisecs`*

specifies the time (in milliseconds) after a message is sent until the message expires. If the EXPIRY attribute value is set to zero, a message never expires. The expiry attribute value is used when no `timetolive` value is specified during a send message operation.

Default	0
Units	milliseconds
Range	0 through 9223372036854775807

Rules

- When `queue` or `topic` is specified, *`name`* and `expiry` *`millisecs`* are required.
- The expiry attribute value is used when no `timetolive` value is specified during a send message operation.

Example

To alter a queue (named `sales-queue`) to have messages expire 10 minutes after they are received:

```
nsjms->alter queue sales-queue, expiry 600000
```

DELETE Command

The DELETE command removes a queue, topic, message, or subscription from the database, or NSJMS connection factories from the JNDI space.

```
DELETE { QUEUE { name | * } |

         TOPIC { name | * } |

         MESSAGE { id | queue-name | topic-name |

                   expired | * } |

         SUBSCRIPTION { client-id, subscription-name |

                        * } |

         JNDI }
```

`name`

is the name associated with an existing queue or topic to be deleted from the database. In addition, all messages and subscriptions associated with the queue or topic are deleted.

`*`

is a wild-card character that deletes all queues, topics, messages, or subscriptions from the database. If `QUEUE` or `TOPIC` is specified, all queues or topics and any associated subscriptions or messages are deleted. If `MESSAGE` is specified, all messages are deleted. If `SUBSCRIPTION` is specified, all subscriptions are deleted.

`id`

is the unique ID that identifies a specific message to be deleted from the database.

`queue-name`

is the name of the queue from which all messages will be deleted. The queue itself is not deleted from the database.

`topic-name`

is the name of the topic from which all messages will be deleted. The topic itself is not deleted from the database.

`expired`

specifies that all messages that have an expiration date prior to the current date are deleted from the database.

`client-id`

is the unique ID associated with an existing subscription to be deleted from the database.

`subscription-name`

is the name associated with an existing subscription to be deleted from the database.

Rules

- The `jndi` attribute is not applicable for the NSJMS C++ API environment.
- When an asterisk (`*`) is specified in the delete command, a confirmation message appears before the command is run.
- When deleting a queue or topic, a *`name`* or an asterisk (`*`) must be explicitly specified.
- When `queue` or `topic` is specified, the *`id`* attribute is invalid.
- If *`queue-name`* or *`topic-name`* is specified, all messages and subscriptions associated with the specified queue or topic are deleted.

- When `subscription` is specified, if *`client-id`* and *`subscription-name`* are specified, information about the subscription whose client ID and subscription name match *`client-id`* and *`subscription-name`* is deleted. *`client-id`* and *`subscription-name`* are valid only when used with each other.

Examples

- To delete a topic named `sales-topic`:

  ```
  nsjms->delete topic sales-topic
  ```

- To delete all expired messages:

  ```
  nsjms->delete message expired
  ```

- To delete a message that has the message ID `quotes_994790988290`:

  ```
  nsjms->delete message quotes_994790988290
  ```

- To delete all messages:

  ```
  nsjms->delete message *
  ```

- To delete a subscription with a client ID of `$xyz1` and a subscription name of `subxyz1`:

  ```
  nsjms->delete subscription $xyz1, subxyz1
  ```

- To delete NSJMS connection factories from the JNDI space:

  ```
  nsjms->delete jndi
  ```

EXIT Command

The EXIT command exits from the current administrative session.

```
EXIT
```

Example

To exit from the current administrative session:

```
nsjms->exit
```

HELP Command

The HELP command displays the syntax and meaning of administrative utility commands invoked through a command-line interface.

```
HELP [ command ]
```

`command`

is one of these administrative utility commands for which help is desired:

ADD
ALTER
DELETE
EXIT
HELP
INFO
STATS

Rules

- If the HELP command is run without specifying *`command`*, available administrative commands appear.
- When *`command`* is specified, only information about *`command`* appears.

Examples

- To display the syntax and use of every *`command`*:

  ```
  nsjms->help
  ```

- To display the syntax and use of the ADD command:

  ```
  nsjms->help add
  ```

INFO Command

The INFO command provides information about a specified queue, topic, message, or subscription.

```
INFO { QUEUE { name | * } |

       TOPIC { name | * } |

       MESSAGE { id | queue-name | topic-name |

                 expired |* } |

       SUBSCRIPTION { client-id, subscription-name | * } }
```

`name`

is the name associated with an existing queue or topic about which information is requested.

`*`

is a wild-card character that causes information for all specified objects (`QUEUE`, `TOPIC`, `MESSAGE`, or `SUBSCRIPTION`) to appear.

`id`

is the unique ID that identifies a specific message about which information is requested.

`queue-name`

is the name of the queue containing the messages about which information is requested.

`topic-name`

is the name of the topic containing the messages about which information is requested.

`expired`

specifies that information will be returned from the database about all messages with an expiration date prior to the current date.

`client-id`

is the unique ID associated with an existing subscription about which information is requested.

`subscription-name`

is the name associated with an existing subscription about which information is requested.

Rules

- If a *`name`* is specified, information about the named queue or topic appears.
- If *`queue-name`* or *`topic-name`* is specified, information about all messages in the specified queue or topic appears.
- When requesting information about a queue or topic, a *`name`* or an asterisk (`*`) must be explicitly specified.
- When `queue` or `topic` is specified, the *`id`* attribute is invalid.
- When `subscription` is specified, if *`client-id`* and *`subscription-name`* are specified, information about the subscription whose client ID and subscription name match *`client-id`* and *`subscription-name`* appears. *`client-id`* and *`subscription-name`* are valid only when used with each other.

Examples

The INFO QUEUE command returns a display such as:

```
nsjms-> info queue orders

Destination Name:      orders
Destination ID:        123
Destination Expiry:    1200000
Destination Create Date: Mon Mar 05 18:45:33 PST 1973
```

The fields returned by the INFO QUEUE command are:

`Destination Name`

identifies the name of the queue or topic about which information is requested.

`Destination ID`

identifies the location of the queue or topic in the NSJMS database.

`Destination Expiry`

identifies the time (in milliseconds) after a message is sent until the message expires. The expiry attribute value is used when no `timetolive` value is specified during a send message operation.

`Destination Create Date`

identifies the date the queue or topic was created.

The INFO TOPIC command returns a display such as:

```
nsjms-> info topic sales

Destination Name:      sales
Destination ID:        8942
Destination Expiry:    600000
Destination Create Date: Mon Mar 05 18:45:33 PST 1973
```

The fields returned by the INFO TOPIC command are:

`Destination Name`

identifies the name of the queue or topic about which information is requested.

`Destination ID`

identifies the location of the queue or topic in the NSJMS database.

`Destination Expiry`

identifies the time (in milliseconds) after a message is sent until the message expires. The expiry attribute value is used when no `timetolive` value is specified during a send message operation.

`Destination Create Date`

identifies the date the queue or topic was created.

The INFO MESSAGE command returns a display such as:

```
nsjms-> info message salesqueue_994790988290

#1
JMSMessageID:     salesqueue_994790988290
JMSDestination:   salesqueue
JMSDeliveryMode:  1-(Non-Persistent)
JMSTimestamp:     994790988290 (Tue Jul 10 11:49:48 PDT 2001)
JMSExpiration:    1625942988227 (NO EXPIRATION)
JMSRedelivered:   false
JMSPriority:      1
JMSReplyTo:       null
JMSCorrelationID: null
JMSType:          TextMessage
Properties:       No Properties Found
Message body:     Request an estimate on S74000 processor.

1 message(s) found
```

The JMS header fields returned by the INFO MESSAGE command are:

`JMSMessageID`

identifies a specific message.

`JMSDestination`

identifies the queue or topic name for the specified message.

`JMSDeliveryMode`

identifies the JMS delivery mode. A value of 2 indicates a persistent delivery mode; a value of `1` indicates a nonpersistent delivery mode.

`JMSTimestamp`

identifies when the specified message was sent in milliseconds. It is set automatically by the message producer.

`JMSExpiration`

identifies when the specified message expires.

`JMSRedelivered`

identifies if the specified message was redelivered to the consumer. A value of `true` indicates the specified message was redelivered; a value of `false` indicates the specified message was not redelivered.

`JMSPriority`

identifies the specified message’s priority. A message can fall within two categories of message priorities: Values of `0` through `4` are gradations of normal priority; values of `5` through `9` are gradations of expedited priority.

`JMSReplyTo`

identifies a destination to which a reply to the specified message is sent. The destination is supplied by the JMS client when the specified message is sent. A value of `null` indicates the destination for replying to the specified message was not specified at the time the message was sent.

`JMSCorrelationID`

is not currently supported by NSJMS. A value of `null` is returned in this field.

`JMSType`

identifies the type of message. The message type identifier is supplied by the JMS client when the specified message is sent.

`Properties`

identifies additional headers which provide a mechanism for adding optional header fields to the specified message.

`Message body`

identifies the type of information contained in the body of the specified message. NSJMS supports five forms for the body of a message: StreamMessage, MapMessage, TextMessage, ObjectMessage, and BytesMessage. When the body of a message is in the TextMessage form, the text of the message appears.

The INFO SUBSCRIPTION command returns a display such as:

```
nsjms-> info subscription orders

Client ID:         $xyz1
Subscription Name: subxyz1
Destination ID:    1
Nolocal:           1
Message ID:         994790988290
```

The fields returned by the INFO SUBSCRIPTION command are:

`Client ID`

identifies the JMS client to which the durable subscription belongs.

`Subscription Name`

identifies the name of the specified subscription.

`Destination ID`

identifies the location of the subscription in the NSJMS database.

`Nolocal`

identifies the value of the nolocal attribute. A value of `1` indicates messages published to the topic on the same connection will not be delivered to this subscriber. A value of `0` indicates messages published to the topic on the same connection will be delivered to this subscriber.

`Message ID`

identifies the last acknowledged message.

STATS Command

The STATS command provides statistical information about queues, topics, or the NSJMS service.

```
    STATS { QUEUE { name | * } [ , RESET ] |

            TOPIC { name | * } [ , RESET ] |

            NSJMS  [ , RESET ] }
```

`name`

is the name associated with an existing queue or topic about which statistical information is desired.

`*`

is a wild-card character that causes statistical information about all queues or topics to appear.

`RESET`

if present, specifies that all applicable statistical counters are reset to their initial values after the command has run.

Rules

- The STATS command is not applicable for the NSJMS C++ API environment.
- The nsjms property `Stats.active` in the nsjms.properties file must be set to `true` for statistical information to be returned
- When requesting information about a queue or topic, a *`name`* or an asterisk (*) must be explicitly specified.
- If a *`name`* is specified, statistics for the named queue or topic appear.

- If asterisk (`*`) is specified, statistics for all queues or topics appear.
- If statistical information is returned by multiple queues or topics, a summary block is displayed.
- When `nsjms` is specified, the *name* attribute is invalid.
- When `nsjms` is specified, statistics for all queues and topics appear.

Examples

- The STATS QUEUE command returns a display such as:

```
nsjms-> stats queue orders

Destination: orders

Process ID:                     $xyz1
Stats Interval (min):           10
Msgs Sent:                      901
Msgs Sent Rate (msg/sec):       1.5
Msgs Sent Mean (bytes):         4000
Msgs Received:                  600
Msg Received Rate (msg/sec):    1
Msg Received Mean (bytes):      2000
Largest Msg Sent (bytes):       10000
Largest Msg Received (bytes):   9000
```

- The STATS TOPIC command returns a display such as:

```
nsjms-> stats topic sales

Destination: sales
Process ID:               $xyz2
Stats Interval(min):      10
Msgs Sent:                390
Msgs Sent Rate(msg/sec):  0.65
Msgs Sent Mean(bytes):    4000
Msgs Recv:                360
Msgs Recv Rate(msg/sec):  0.60
Msgs Recv Mean(bytes):    2000
Largest Msg Sent(bytes):  1090
Largest Msg Recv(bytes):  6000

Process ID:               $xyz3
Stats Interval(min):      10
Msgs Sent:                904
Msgs Sent Rate(msg/sec):  1.51
Msgs Sent Mean(bytes):    4000
Msgs Recv:                604
Msgs Recv Rate(msg/sec):  1.01
Msgs Recv Mean(bytes):    2000
Largest Msg Sent(bytes):  13400
Largest Msg Recv(bytes):  4000
```

```
Process ID:               $xyz4
Stats Interval(min):      10
Msgs Sent:                907
Msgs Sent Rate(msg/sec):  1.51
Msgs Sent Mean(bytes):    4000
Msgs Recv:                607
Msgs Recv Rate(msg/sec):  1.01
Msgs Recv Mean(bytes):    2000
Largest Msg Sent(bytes):  1800
Largest Msg Recv(bytes):  2000

Process ID:               $xyz5
Stats Interval(min):      10
Msgs Sent:                909
Msgs Sent Rate(msg/sec):  1.51
Msgs Sent Mean(bytes):    4000
Msgs Recv:                609
Msgs Recv Rate(msg/sec):  1.01
Msgs Recv Mean(bytes):    2000
Largest Msg Sent(bytes):  1900
Largest Msg Recv(bytes):  1000

Process ID:               $xyz6
Stats Interval(min):      10
Msgs Sent:                290
Msgs Sent Rate(msg/sec):  0.48
Msgs Sent Mean(bytes):    4000
Msgs Recv:                260
Msgs Recv Rate(msg/sec):  0.43
Msgs Recv Mean(bytes):    2000
Largest Msg Sent(bytes):  1900
Largest Msg Recv(bytes):  1300

Summary Information:

Total Destinations:               1
Total Process Ids:                5
Total Msgs Sent:                  3400
Total Msgs Recv:                  2440

Summary Msgs Sent Rate(msg/sec):  5.66
Summary Msgs Sent Mean(bytes):    4000
Summary Msgs Recv Rate(msg/sec):  4.06
Summary Msgs Recv Mean(bytes):    2000
Summary Largest Msg Sent(bytes):  13400
Summary Largest Msg Recv(bytes):  6000
```

- The STATS NSJMS command returns a display such as:

```
nsjms-> stats nsjms

Destination: orders

Process ID:               $xyz1
Stats Interval(min):      10
Msgs Sent:                901
Msgs Sent Rate(msg/sec):  1.50
Msgs Sent Mean(bytes):    4000
Msgs Recv:                600
Msgs Recv Rate(msg/sec):  1
Msgs Recv Mean(bytes):    2000
Largest Msg Sent(bytes):  10000
Largest Msg Recv(bytes):  9000
```

```
Destination: sales

Process ID:               $xyz2
Stats Interval(min):      10
Msgs Sent:                390
Msgs Sent Rate(msg/sec):  0.65
Msgs Sent Mean(bytes):    4000
Msgs Recv:                360
Msgs Recv Rate(msg/sec):  0.60
Msgs Recv Mean(bytes):    2000
Largest Msg Sent(bytes):  1090
Largest Msg Recv(bytes):  6000

Process ID:               $xyz3
Stats Interval(min):      10
Msgs Sent:                904
Msgs Sent Rate(msg/sec):  1.51
Msgs Sent Mean(bytes):    4000
Msgs Recv:                604
Msgs Recv Rate(msg/sec):  1.01
Msgs Recv Mean(bytes):    2000
Largest Msg Sent(bytes):  13400
Largest Msg Recv(bytes):  4000

Destination: purchases

Process ID:               $xyza
Stats Interval(min):      10
Msgs Sent:                590
Msgs Sent Rate(msg/sec):  0.98
Msgs Sent Mean(bytes):    4000
Msgs Recv:                560
Msgs Recv Rate(msg/sec):  0.93
Msgs Recv Mean(bytes):    2000
Largest Msg Sent(bytes):  1820
Largest Msg Recv(bytes):  1800

Process ID:               $xyz9
Stats Interval(min):      10
Msgs Sent:                690
Msgs Sent Rate(msg/sec):  1.15
Msgs Sent Mean(bytes):    4000
Msgs Recv:                660
Msgs Recv Rate(msg/sec):  1.10
Msgs Recv Mean(bytes):    2000
Largest Msg Sent(bytes):  10000
Largest Msg Recv(bytes):  14500

Summary Information:

Total Destinations:               3
Total Process Ids:                5
Total Msgs Sent:                  3475
Total Msgs Recv:                  2784

Summary Msgs Sent Rate(msg/sec):  5.79
Summary Msgs Sent Mean(bytes):    4000
Summary Msgs Recv Rate(msg/sec):  4.64
Summary Msgs Recv Mean(bytes):    2000
Summary Largest Msg Sent(bytes):  13400
Summary Largest Msg Recv(bytes):  14500
```

`Destination`

identifies a destination to which the statistics are being reported.

`Process ID`

identifies a process within this destination.

`Stat Interval (min)`

identifies the amount of elapsed time in minutes before this process reports statistics.

`Msgs Sent`

identifies the number of messages sent by this process.

`Msgs Sent Rate (msg/sec)`

identifies the rate at which messages were sent by this process.

`Msgs Sent Mean (bytes)`

identifies the mean message size for messages sent by this process.

`Msgs Recv`

identifies the number of messages received by this process.

`Msgs Recv Rate (msg/sec)`

identifies the rate at which messages were received by this process.

`Msgs Recv Mean (bytes)`

identifies the mean message size for messages received by this process.

`Largest Msg Sent (bytes)`

identifies the largest message size in bytes that was sent by this process.

`Largest Msg Recv (bytes)`

identifies the largest message size in bytes that was received by this process.

`Total Destinations`

identifies the total number of destinations for which statistical information was received.

`Total Process Ids`

identifies the total number of process IDs for which statistical information was received.

`Total Msgs Sent`

identifies the total number of all messages sent.

`Total Msgs Recv`

identifies the total number of all messages received.

`Summary Msgs Sent Rate (msg/sec)`

identifies the summary rate for all messages sent.

`Summary Msgs Sent Mean (bytes)`

identifies the summary mean message size for all messages sent.

`Summary Msgs Recv Rate(msg/sec)`

identifies the summary rate for all messages received.

`Summary Msgs Recv Mean (bytes)`

identifies the summary mean message size for all messages received.

`Summary Largest Msg Sent (bytes)`

identifies the largest message size in bytes that was sent by all processes.

`Summary Largest Msg Recv (bytes)`

identifies the largest message size in bytes that was received by all processes.

The XML Interface

You invoke the administrative utility as a servlet by using an XML formatted request/reply paradigm in a user defined management application. For example, your web browser (see Step 7 on page 6-3 if you are running Java Servlets or Step 13 on page 6-7 if you are running NonStop Servlets for JavaServer Pages).

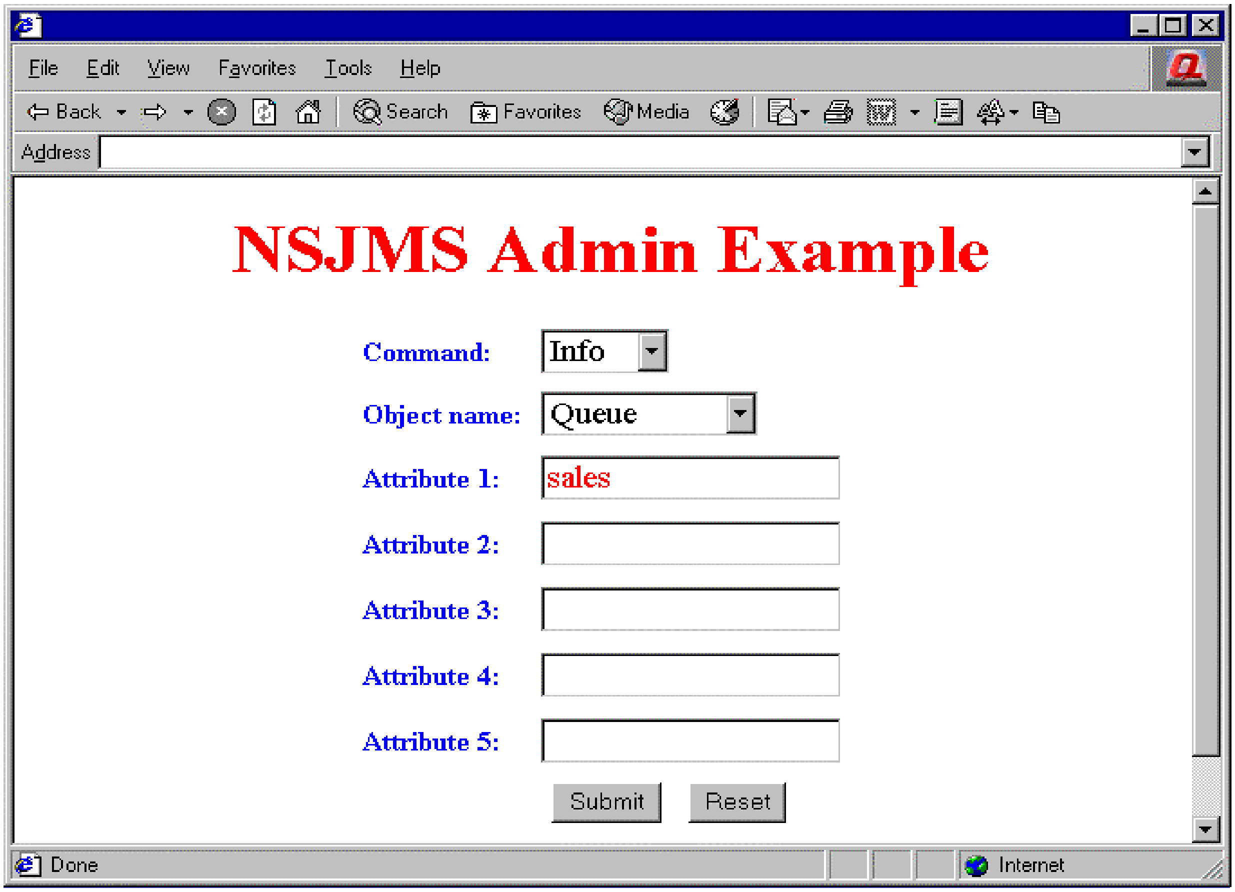
The XML message format syntax resembles that of the NSJMS administrative utility command-line interface. XML request and reply messages must be well-formed XML documents. Optionally, an internal DTD can be used to validate the request and reply messages by setting the `Servlet.xmlvalidate` system property in the `nsjms.properties` file to `true` (see Servlet.xmlvalidate on page 2-15).

XML Sample Request

This is a sample request for information about a queue named `sales`:

```
<?xml version=”1.0” encoding=”utf-8”?>
<!DOCTYPE AdminRequest>
 <AdminRequest>
      <cmd>info</cmd>
      <objname>queue</objname>
      <attvalue1>sales</attvalue1>
</AdminRequest>
```

This is an example of the previous sample request for information when you invoke the administrative utility as a servlet through your web browser.



File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Submit
Reset
Done
Internet

NSJMS Admin Example

Command: Info

Object name: Queue

Attribute 1: sales

Attribute 2:

Attribute 3:

Attribute 4:

Attribute 5:

VST003.vsd

XML Request DTD

This sample request DTD is used internally to validate XML requests when the system property `Servlet.xmlvalidate` is set to `true`:

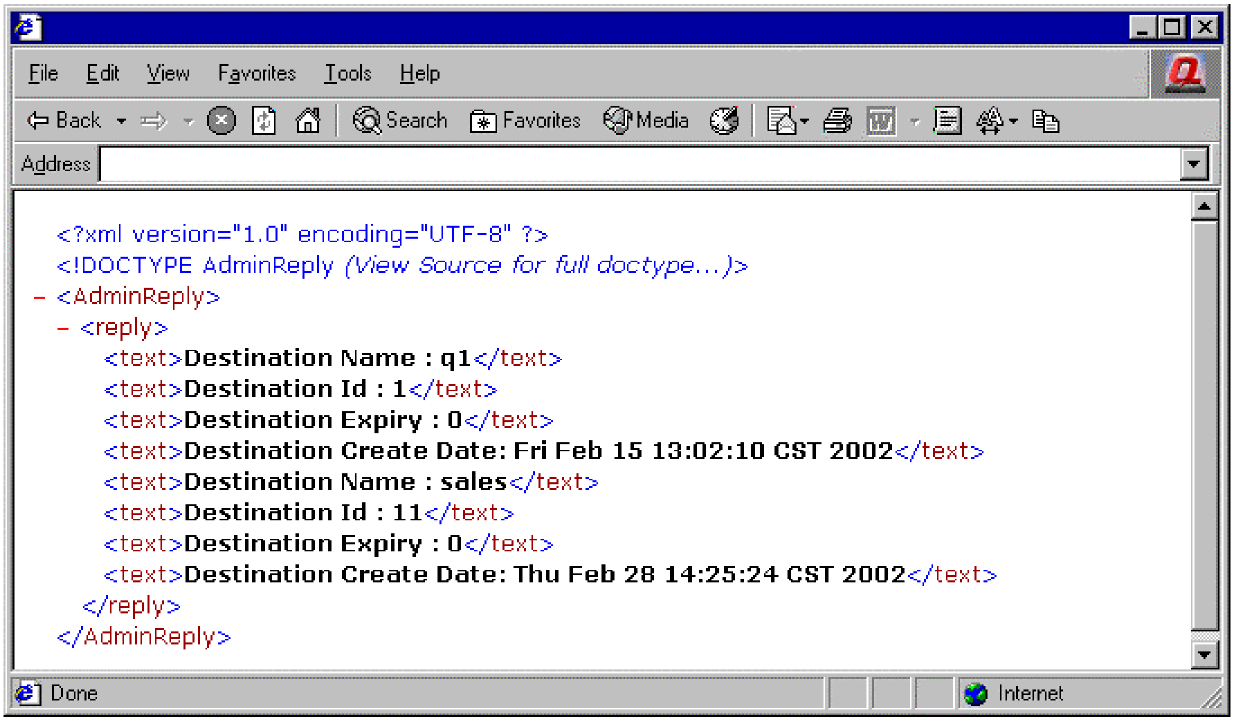
```
<!ELEMENT AdminRequest (cmd, objname?, attvalue1*, attvalue2*, attvalue3*,
                        attvalue4*, attvalue5*,)>"
<!ELEMENT cmd (#PCDATA)>
<!ELEMENT objname (#PCDATA)>
<!ELEMENT attvalue1 (#PCDATA)>
<!ELEMENT attvalue2 (#PCDATA)>
<!ELEMENT attvalue3 (#PCDATA)>
<!ELEMENT attvalue4 (#PCDATA)>
<!ELEMENT attvalue5 (#PCDATA)>
```

XML Sample Reply

This is the sample reply to the previous request for information about a queue named `sales`:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE AdminReply>
  <AdminReply>
     <reply>
       <text>Destination Name : sales</text>
       <text>Destination ID : 1</text>
       <text>Destination Expiry : 0</text>
       <text>Destination Create Date: Mon Mar 05 18:45:33 PST 1997</text>
     </reply>
  </AdminReply>
```

This is an example of the previous sample reply returned through your web browser:



```
  <?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE AdminReply (View Source for full doctype...)>
- <AdminReply>
  - <reply>
      <text>Destination Name : q1</text>
      <text>Destination Id : 1</text>
      <text>Destination Expiry : 0</text>
      <text>Destination Create Date: Fri Feb 15 13:02:10 CST 2002</text>
      <text>Destination Name : sales</text>
      <text>Destination Id : 11</text>
      <text>Destination Expiry : 0</text>
      <text>Destination Create Date: Thu Feb 28 14:25:24 CST 2002</text>
    </reply>
  </AdminReply>
```

VST004.vsd

XML Reply DTD

This sample reply DTD is used internally to validate XML replies when the system property `Servlet.xmlvalidate` is set to `true`:

```
<!ELEMENT AdminReply (reply)>
<!ELEMENT reply (text*)>
<!ELEMENT text   (#PCDATA)>
```

Using SQLCI to Manage an NSJMS Database

NSJMS stores messages and destination information in SQL tables. You can use SQL/MP commands and statements to query and manage an NSJMS database by submitting queries directly through the SQLCI. For additional information about SQL/MP commands and statements, see the *SQL/MP Reference Manual*. For additional information about using SQL/MP to query a database, see the *SQL/MP Installation and Management Guide*.

SQLCI Examples

These examples show how SQLCI can be used to query and manage an NSJMS database. In each example, $*vol.subvol* represents the volume and subvolume where the NSJMS SQL tables were created.

- To delete all messages which have expired, use these commands:

```
SQLCI>> volume $vol.subvol;
SQLCI>> delete from message where msg_expiration < juliantimestamp(current);
SQLCI>> delete from overlow where msg_expiration < juliantimestamp(current);
```

- To retrieve the count of messages in a table names `invoices`, use these commands:

```
SQLCI>> volume $vol.subvol;
SQLCI>> select count(*) from message where dest_id = (select dest_id from dest
SQLCI>>    where dest_name = 'invoices' );
```

- To display a list of destination names and message counts, use these commands:

```
SQLCI>> volume $vol.subvol;
SQLCI>> select dest.dest_name, count(*) from message, dest where
SQLCI>>    message.dest_id = dest.dest_id
SQLCI>>       group by dest.dest_name;
```

- To display a list of destination names and message counts for those destinations that have more than 1000 messages, use these commands:

```
SQLCI>> volume $vol.subvol;
SQLCI>> select dest.dest_name, count(*) from message, dest where
SQLCI>>    message.dest_id = dest.dest_id
SQLCI>>       group by dest.dest_name
SQLCI>>          having count(*) > 1000;
```

A Error Reporting and Messages

This appendix provides information about JMS API error messages, the NSJMS API logging facility, the reliable messaging bridge logging facility, the Pathway logging facility, SQL messages, administrative-utility error messages, and administrative-utility warning messages.

JMS API Error Messages

JMS API error messages and exceptions are described in The *Java Message Service* specification (see Compliance Information on page 1-4).

NSJMS API Logging Facility

All log records generated by the NSJMS API are contained in the `nsjms.log` file. The type of records generated by the NSJMS API is determined by the `Logger.loglevel` setting in the `nsjms.properties` file. For a description of the level of records that can be generated and reported to the `nsjms.log` File, see Logger.loglevel on page 2-14.

Reliable Messaging Bridge Logging Facility

All log records generated by the reliable messaging bridge are contained in the *`pathmon_process_name`*`.log` file.

Pathway Logging Facility

All log records generated by PATHMON processes are contained in the *`pathmon_process_name`*`.pathmon.log`.

SQL/MX Messages

SQL/MX reports exception condition messages during the running of an NSJMS TopicSubscriber or an NSJMS QueueReceiver. Exception conditions are returned to the JMS client as a `javax.jms.JMSException` that is linked with a nested `java.sql.SQLException`. The fields returned in the SQLException area are `SQLCODE`, `SQLSTATE`, `CONDITION_NUMBER` and `MESSAGE_TEXT`.

The SQLCODE field returns a value that describes error information after running an embedded SQL statement:

Value	Description
< 0	An error occurred.
> 0 (<> 100)	A warning occurred.
100	No data was found.
0	The statement completed successfully.

The SQLSTATE field returns a five-character variable string that has two parts. The first part is a two-character class code, and the second part is a three-character subclass code. An SQLSTATE value of 00000 indicates completion. For a listing of the class and subclass values, see the *SQL/MX Programming Manual for C and COBOL* .

The CONDITION_NUMBER field returns the number of an exception condition.

The MESSAGE_TEXT field returns the SQL message number and the message text.

For example, this error message might be displayed if an exception occurs while preparing an embedded SQL statement:

```
javax.jms.JMSException: SQL error -4081 reading messages: java.sql.SQLException:

SQLCODE: -4081
SQLSTATE: 42000
condition number: 1
message text: *** ERROR[4081] SQL/MP error:
*** ERROR from SQL [-4004]: Object \KONA.$DATA30.NSJMS.MESSAGE is not an SQL table
or view.

SQLCODE: -4082
SQLSTATE: 42000
condition number: 2
message text: *** ERROR[4082] Table \KONA.$DATA30.NSJMS.MESSAGE does not exist or
is inaccessible.

SQLCODE: -8822
SQLSTATE: X0X08
condition number: 3
message text: *** ERROR[8822] Unable to prepare the statement.

   at com.tandem.nsjms.client.JmsMessageConsumer.receiveNoWaitIntJNI(JmsMessage
Consumer.java, Compiled Code)
   at com.tandem.nsjms.client.JmsMessageConsumer.receive(JmsMessageConsumer.java,
Compiled Code)
   at com.tandem.nsjms.examples.SimpleReceiver.main(SimpleReceiver.java, Compiled
Code)
```

For information about SQL messages, see the *SQL/MX Messages Manual*.

Administrative Utility Error Messages

Add Failed

```
NSJMS Err: Add queue|topic name failed:queue|topic name
already exists.
```

Cause. An attempt was made to add a queue or topic name that already exists.

Effect. The queue or topic name was not added.

Recovery. Reenter the command, specifying a queue or topic name that does not already exist.

Destination ID Already Exists

```
NSJMS Err: Destination ID already exists
```

Cause. The specified destination ID already exists.

Effect. The command is not run.

Recovery. Reenter the command, specifying a different destination ID.

Destination Name Too Long

```
NSJMS Err: Destination name is too long: destination_name
```

Cause. The length of *Destination_Name* is too long.

Effect. The command is not run.

Recovery. Check the definition of `Destination Name` and reenter the command.

Duplicate Command Attributes

```
NSJMS Err: Duplicate command attributes
```

Cause. The command contains duplicate command attributes.

Effect. The command is not run.

Recovery. Reenter the command, specifying the command attributes once.

Expiry is required

```
NSJMS Err: Expiry millisecs is required
```

Cause. A value for the `expiry` attribute is missing.

Effect. The command is not run.

Recovery. Reenter the command with a value for `expiry`.

Help File Not Found

```
NSJMS Err: Help file not found: help_filename
```

Cause. The specified Help file was not found.

Effect. The command is not run.

Recovery. Verify the name and existence of the Help file and reenter the command.

Help File Text Missing

```
NSJMS Err: Help file has no text: help_filename
```

Cause. The Help file contains no text.

Effect. The command is not run.

Recovery. Reaccess the Help file (see Step 3 on page 6-2) and reenter the command.

Invalid Command Attribute

```
NSJMS Err: Invalid command attribute: attribute
```

Cause. The attribute specified for the command is unknown or invalid.

Effect. The command is not run.

Recovery. Check the command syntax and reenter the command, using valid attributes.

Invalid Command Attributes

```
NSJMS Err: Expected one of the following attribute(s):
attribute_list
```

Cause. The attributes specified for the command are invalid. Valid command attributes are displayed by `attribute_list`.

Effect. The command is not run.

Recovery. Reenter the command, using valid attributes.

Invalid Destination Name

```
NSJMS Err: Invalid Destination name format: destination_name
```

Cause. The Destination Name contains invalid characters.

Effect. The command is not run.

Recovery. Check the definition of `Destination Name` and reenter the command.

Invalid NSJMS Session Argument

```
NSJMS Err: Invalid nsjms session argument
```

Cause. An invalid command-line argument was specified while starting the administrative utility.

Effect. The command is not run.

Recovery. Reenter the command, specifying a valid argument.

Invalid or Out of Range Expiry Value

```
NSJMS Err: Expiry value is invalid or out of range:
expiry_value
```

Cause. The expiry value contains invalid characters or is too large.

Effect. The command is not run.

Recovery. Check the definition of Expiry and reenter the command.

Invalid or Out of Range ID Value

```
NSJMS Err: ID value is invalid or out of range: id_value
```

Cause. The Destination ID value contains invalid characters or is too large.

Effect. The command is not run.

Recovery. Check the definition of Destination ID and reenter the command.

Invalid Syntax

```
NSJMS Err: Invalid Syntax
```

Cause. The command syntax is invalid.

Effect. The command is not run.

Recovery. Check the command syntax and reenter the command.

Invalid Syntax - Comma Expected

```
NSJMS Err: Expected a comma instead of: command_text
```

Cause. The syntax for this command is invalid. A comma was expected where *command text* was found.

Effect. The command is not run.

Recovery. Check the command syntax and reenter the command.

JNDI Add Failed

```
NSJMS Err: JNDI add failed, security violation
```

Cause. A security violation occurred when executing the ADD command.

Effect. The command is not run.

Recovery. Verify the user of the administrative utility has the appropriate permissions to perform this operation.

JNDI Bind Exception

```
NSJMS Err: JNDI bind exception occurred
```

Cause. Review the `nsjms.log` file to determine the exact cause.

Effect. The command is not run.

Recovery. Determine the exact cause and take the appropriate corrective action.

JNDI Context Exception

```
NSJMS Err: JNDI context exception occurred
```

Cause. Review the `nsjms.log` file to determine the exact cause.

Effect. The command is not run.

Recovery. Determine the exact cause and take the appropriate corrective action.

JNDI Delete Failed

```
NSJMS Err: JNDI delete failed, security violation
```

Cause. A security violation occurred when running the DELETE JNDI command.

Effect. The command is not run.

Recovery. Verify the user of the administrative utility has the appropriate permissions to perform this operation.

JNDI Object Directory Not Found

```
NSJMS Err: JNDI object directory not found
```

Cause. The JNDI object store could not be found.

Effect. The command is not run.

Recovery. Update the java.naming.provider.url property in the `jndi.properties` file to reflect the correct location of the JNDI object store.

JNDI Unbind Exception

```
NSJMS Err: JNDI unbind exception occurred
```

Cause. Review the `nsjms.log` file to determine the exact cause.

Effect. The command is not run.

Recovery. Determine the exact cause and take the appropriate corrective action.

Keyword Expected

```
NSJMS Err: Keyword Reset Expected
```

Cause. The keyword `Reset` was expected but not found.

Effect. The command is not run.

Recovery. Reenter the command by using the keyword `Reset`.

Missing Attribute

```
NSJMS Err: Invalid syntax - missing attribute
```

Cause. Required attributes for the specified command are missing.

Effect. The command is not run.

Recovery. Reenter the command, specifying the required attributes.

Missing Attribute Value

```
NSJMS Err: Invalid syntax - missing attribute value
```

Cause. A value was not specified for the command attribute.

Effect. The command is not run.

Recovery. Reenter the command, specifying a value for the command attribute.

No Message Found on Queue

```
NSJMS Err: No message(s) found
```

Cause. An attempt was made to display information about a message from a queue, topic, message, or subscription that does not exist.

Effect. The message was not displayed.

Recovery. Reenter the command, specifying a queue, topic, message, or subscription that contains messages.

NSJMS Connection Problem

```
NSJMS Err: NSJMS Connection problem: JMS_error
```

Cause. A NSJMS database connection problem occurred.

Effect. The command is not run.

Recovery. Check the location of the NSJMS database located in the `nsjms.properties` file and reenter the command.

Queue or Topic Name Missing

```
NSJMS Err: Queue or Topic name missing
```

Cause. A queue or topic name is required for the specified command.

Effect. The command is not run.

Recovery. Reenter the command with a queue or topic name.

Record Already Exists

```
NSJMS Err: Record already exists: Dest Name: destination_name
```

Cause. A record for this Destination Name already exists.

Effect. The command is not run.

Recovery. Reenter the command, specifying a different Destination Name.

SQL Error

```
NSJMS Err: SQL error connecting to database
```

Cause. An error was encountered while attempting to connect to the NSJMS database.

Effect. The command is not run.

Recovery. Check the location of the database as specified in the `nsjms.properties` file.

Too Many Attributes

```
NSJMS Err: Too many attributes for this command: command
```

Cause. Too many attributes were specified for the command.

Effect. The command is not run.

Recovery. Check the command syntax and reenter the command with valid attributes.

Unknown Option

```
NSJMS Err: Unknown option: command
```

Cause. The command displayed is an invalid command.

Effect. The command is not run.

Recovery. Reenter the command, specifying a valid command. To display valid commands, use the help command.

Administrative Utility Warning Messages

Destination Not Found

```
NSJMS Warn: Destination not found
```

Cause. The specified destination (queue or topic) was not found.

Effect. No information is returned for the specified command.

Recovery. This is an informational message. No action is required.

Message Not Found

```
NSJMS Warn: Message not found
```

Cause. The specified message was not found.

Effect. No information is returned for the specified command.

Recovery. This is an informational message. No action is required.

No Destination(s) Found

```
NSJMS Warn: No Destination(s) found
```

Cause. No destinations (queue or topic) were found that meet the specified criteria.

Effect. No information is returned for the specified command.

Recovery. This is an informational message. No action is required.

No Message(s) Found

```
NSJMS Warn: No message(s) found
```

Cause. No NSJMS messages were found that meet the specified criteria.

Effect. No information is returned for the specified command.

Recovery. This is an informational message. No action is required.

No Statistics Found

```
NSJMS Warn: No statistics found
```

Cause. No statistics were found that meet the specified criteria.

Effect. No information is returned for the specified command.

Recovery. This is an informational message. No action is required.

No Subscription(s) Found

```
NSJMS Warn: No subscription(s) found
```

Cause. No subscriptions were found that meet the specified criteria.

Effect. No information is returned for the specified command.

Recovery. This is an informational message. No action is required.

Statistics Not Active/Available

```
NSJMS Warn: Statistice are not active: No statistics
available
```

Cause. The `Stats.active` property in the `nsjms.properties` file is set to false.

Effect. No information is returned for the specified command.

Recovery. This is an informational message. To view and activate statistics, set the `Stats.active` property to true in the `nsjms.properties` file.

Subscription Not Found

```
NSJMS Warn: Subscription not found
```

Cause. The specified subscription was not found.

Effect. No information is returned for the specified command.

Recovery. This is an informational message. No action is required.

Glossary

administered object. See NSJMS administered object.

API. See application program interface (API).

application program. One of the following: A software program written for or by a user for a specific purpose. 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.

attribute. An item of descriptive data associated with a command-line tool command or XML element. An attribute has a name and a value.

class path. The location where the JVM 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.

connection. An open connection between a JMS client and NSJMS.

connection factory. An object used for creating connections to NSJMS.

Disk Process 2 (DP2). The portion of the operating system software that performs read, write, and lock operations on disk volumes. The disk process also implements Enscribe and NonStop SQL/MP file types; creates TMF audit trail records; performs logical REDO operations for Remote Duplicate Database Facility (RDF); and manages disk space, disk controllers, and paths to the disks. This disk process provides enhanced performance, throughput, recoverability, and reliability improvements in high-volume, online transaction processing situations.

Distributed Systems Management/Software Configuration Manager (DSM/SCM). A GUI-based program that installs new software and creates a new operating system. DSM/SCM creates a new software revision and activates the new software on the target system. See Distributed Systems Management/Software Configuration Manager (DSM/SCM).

Document Type Definition (DTD). A specification of valid syntax for a class of XML documents. A DTD specifies required elements and attributes, and permissible and default values of attributes.

DP2. See Disk Process 2 (DP2).

DSM/SCM. See Distributed Systems Management/Software Configuration Manager (DSM/SCM)

DTD. See Document Type Definition (DTD)

element. The basic unit of information in an XML document. An element has a name and can have content and attributes.

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.

Guardian. An environment available for interactive or programmatic use with the HP NonStop operating system. Processes that run in the Guardian environment usually use the Guardian system procedure calls as their application program interface; interactive users of the Guardian environment usually use the HP Tandem Advanced Command Language (TACL) or another HP product's command interpreter. Contrast with Open System Services (OSS).

HP NonStop™ operating system. The operating system for HP NonStop systems.

HP NonStop SQL/MP. HP NonStop Structured Query Language/MP, the HP relational database management system.

HP Tandem Advanced Command Language (TACL). The user interface to the operating system. The TACL product is both a command interpreter and a command language. Users can write TACL programs that perform complex tasks or provide a consistent user interface across independently programmed applications.

HTTP. See Hypertext Transfer Protocol (HTTP).

Hypertext Transfer Protocol (HTTP). The client-server TCP/IP protocol used on the World-Wide Web for the exchange of HTML documents.

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 application program interface (API).

iTP Secure WebServer. The HP web server with which NonStop Server for Java integrates using servlets.

jar. The Java Archive tool, which combines multiple files into a single Java Archive (JAR) file. Also, the command to run the Java Archive Tool.

JAR file. A Java Archive file, produced by the Java Archive Tool, jar. The Java standard for access to relational databases such as SQL/MP.

Java Message Service (JMS). A Java API that enables client applications to create, send, receive, and read messages. See also NonStop Server for Java Message Service (NSJMS).

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 JVM from a C program.

Java Transaction API (JTA). The Sun Microsystems product that specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications. For more information, see the Sun Microsystems JTA document (http://www.javasoft.com/products/jta/index.html).

Java Virtual Machine (JVM). The process that loads, links, verifies, and interprets Java bytecode. The NonStop Server for Java Server for Java JVM has been extended with a JIT Compiler.

JMS. See Java Message Service (JMS).

JMS client. A user-written application used to produce and consume messages.

JMS provider. A messaging system that implements the JMS API and includes administrative and management functionality (See NonStop Server for Java Message Service (NSJMS)).

JNDI. See Java Naming and Directory Interface (JNDI).

JNI. See Java Native Interface (JNI).

jre. The Java run-time environment, which interprets (runs) Java bytecode. Also, the command to run the Java run-time environment.

JTA. See Java Transaction API (JTA).

JVM. See Java Virtual Machine (JVM).

Makefile. In a UNIX or OSS environment, a script used to build an application process.

message. A request, report, or event that is created, sent, and received by JMS clients.

MessageConsumer. An object used for receiving messages sent to a destination.

MessageProducer. An object used for sending messages to a destination.

NonStop Server for Java Message Service (NSJMS). The JMS provider that provides an implementation of the JMS API on a NonStop system. See also JMS provider.

NonStop Server for Java Message Service (NSJMS) C++ API. A JMS provider that implements a subset of the functionality provided by the Sun JMS API, and is used by C++ client applications running on a NonStop system to interoperate with other JMS clients. See also JMS provider.

NSJMS. See NonStop Server for Java Message Service (NSJMS).

NSJMS C++ API. See NonStop Server for Java Message Service (NSJMS) C++ API.

NSJMS administered object. A preconfigured object used by JMS clients to create connections to NSJMS and to specify the destinations for messages.

namespace. A context for resolving names, used to prevent ambiguity (or “collisions”) where a name can have different meanings in different situations. To qualify a name by identifying the namespace expresses the idea that the name is used as defined in that namespace.

Open System Services (OSS). A POSIX-compliant operating environment for NonStop systems.

OSS. See Open System Services (OSS).

PTP. See point-to-point message system (PTP).

Pub/Sub. See publish/subscribe message system (Pub/Sub).

point-to-point message system (PTP). A messaging application based on a one-to-one messaging model which uses queues for messaging destinations. A message is sent by a sending client to a specific queue where the message can be received by the receiving client.

publish/subscribe message system (Pub/Sub). A messaging application based a one-to-many messaging model which uses topics for messaging destinations. A message is published by a sending client to a topic where any receiving client who is subscribed to the topic will receive the message.

queue. A destination for a message in a PTP messaging system.

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.

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 runs in server environments such as World-Wide Web servers. The Servlet API is defined in a draft standard by Sun Microsystems.

session. A context for JMS clients to send and receive messages.

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.

SQLCI. The SQL/MP command interpreter.

SQL/MP. See HP NonStop SQL/MP.

TACL. See HP Tandem Advanced Command Language (TACL).

throw. Java keyword used to raise an exception.

throws. Java keyword used to define the exceptions that a method can raise.

TMF. See Transaction Management Facility (TMF).

TNS. Denotes fault-tolerant HP computers that:

- Support the operating system
- Are based on microcoded complex instruction-set computing (CISC) technology.

TNS systems run the TNS instruction set. Contrast with TNS/R and TNS/E.

TNS/E. Denotes fault-tolerant HP computers that support the operating system and that are based on the Intel Itanium processor-based architecture. TNS/E systems run the Itanium instruction set and can run TNS object files by interpretation or after acceleration. TNS/E systems include all HP systems that use NSAL-x processors. Contrast with TNS and TNS/R.

TNS/R. Denotes fault-tolerant HP computers that:

- Support the operating system
- Are based on 32-bit reduced instruction-set computing (RISC) technology.

TNS/R systems run the MIPS-1 RISC instruction set and can run TNS object files by interpretation or after acceleration. TNS/R systems include all HP systems that use NSR-x processors. Contrast with TNS and TNS/E.

topic. A destination for a message in a Pub/Sub messaging system.

transaction. A series of operations grouped together into a single unit of work.

Transaction Management Facility (TMF). A set of HP software products 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.

virtual machine. See Java Virtual Machine (JVM).

XML. Extensible Markup Language, a standard for tagging data in an HTML document so as to provide semantic information about content elements.

Index