

NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide

Abstract

NonStop™ Servlets for JavaServer Pages (NSJSP) is a container that runs Java servlets and JavaServer Pages (JSPs) that are platform-independent server-side programs, which programmatically extend the functionality of web-based applications by providing dynamic content from a web server to a client browser over the HTTP protocol.

Product Version

NonStop Servlets for JavaServer Pages 6.1 (T1222H60^AAN)

Supported Release Version Updates (RVUs)

This publication supports J06.04 and all subsequent J-series RVUs and H06.15 and all subsequent H-series RVUs, until otherwise indicated by its replacement publications.

Part Number	Published
596210-006	June 2013

Document History

Part Number	Product Version	Published
596210-001	NonStop Servlets for JavaServer Pages 6.1 (T1222H60^AAN)	June 2010
596210-002	NonStop Servlets for JavaServer Pages 6.1 (T1222H60^AAN)	November 2011
596210-005	NonStop Servlets for JavaServer Pages 6.1 (T1222H60^AAN)	March 2012
596210-006	NonStop Servlets for JavaServer Pages 6.1 (T1222H60^AAN)	June 2013

Legal Notices

© Copyright 2012 Hewlett-Packard Development Company L.P.

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

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 U.S. trademark of Sun Microsystems, Inc.

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.

© 1990, 1991, 1992, 1993 Open Software Foundation, Inc. This documentation and the software to which it relates are derived in part from materials supplied by the following:

© 1987, 1988, 1989 Carnegie-Mellon University. © 1989, 1990, 1991 Digital Equipment Corporation. © 1985, 1988, 1989, 1990 Encore Computer Corporation. © 1988 Free Software Foundation, Inc. © 1987, 1988, 1989, 1990, 1991 Hewlett-Packard Company. © 1985, 1987, 1988, 1989, 1990, 1991, 1992 International Business Machines Corporation. © 1988, 1989 Massachusetts Institute of Technology. © 1988, 1989, 1990 Mentat Inc. © 1988 Microsoft Corporation. © 1987, 1988, 1989, 1990, 1991, 1992 SecureWare, Inc. © 1990, 1991 Siemens Nixdorf Informationssysteme AG. © 1986, 1989, 1996, 1997 Sun Microsystems, Inc. © 1989, 1990, 1991 Transarc Corporation.

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 Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide

[Glossary](#)[Index](#)[Examples](#)[Figures](#)[Tables](#)[Legal Notices](#)[What's New in This Manual](#) xv[Manual Information](#) xv[New and Changed Information](#) xv[About This Manual](#) xvii[Who Should Read This Guide](#) xvii[Organization of This Guide](#) xviii[Related Manuals](#) xix[Online Resources](#) xxi[Notation Conventions](#) xxii[Abbreviations](#) xxvii[HP Encourages Your Comments](#) xxx

1. Introduction to NSJSP

[Overview](#) 1-2[NSJSP Product](#) 1-2[Apache Tomcat - A Container for Java Servlets and JSP](#) 1-2[The HP NonStop Servlet and JSP Container](#) 1-2[Installing NSJSP](#) 1-3[Configuring NSJSP](#) 1-8[Management in NSJSP](#) 1-9[Securing Web Applications](#) 1-10[NSJSP Features](#) 1-11[Architecture](#) 1-13[Apache Tomcat Components](#) 1-13[NSJSP Architecture](#) 1-16

2. Installing NSJSP

[Prerequisites](#) 2-1[Installing NSJSP from the CD](#) 2-2[Running the IPSetup Program](#) 2-2

Running the setup Script	2-14
Creating an NSJSP Installation	2-16
Verifying the NSJSP Installation	2-18
NSJSP Installation Directory Structure	2-20
Creating an NSJSP Manager Installation	2-21
Verifying the NSJSP Manager Application Installation	2-23
Updating an NSJSP Installation	2-24
Removing an NSJSP Configuration	2-25
Support for Multiple NSJSP Installations in a Single iTP Secure WebServer Environment	2-26

3. Configuring NSJSP

Overview	3-2
Configuration Files for the Server Classes	3-5
The Generic <code>servlet.config</code> File	3-6
The Installation-Specific <code>servlet.config</code> File	3-7
The <code>nsjspadmin.config</code> File	3-29
The <code>filemaps.config</code> File	3-32
The <code>jdbc.config</code> File	3-34
Configuration Files for the Servlet Container	3-36
The <code>server.xml</code> File	3-36
The <code>context.xml</code> File	3-58
The <code>web.xml</code> File	3-65
Virtual Hosts	3-72
Configuring Virtual Hosts	3-73
Session Management	3-74
Sessions in NSJSP	3-74
In-Memory Sessions (<code>SessionBasedLoadBalancing = true</code>)	3-74
Persistent Manager Sessions (<code>SessionBasedLoadBalancing = false</code>)	3-79
Mixed-Mode Sessions	3-87
Determining the Storage Capacity of the Persistent Store	3-91
Configuring the Manager Element	3-95

4. Managing NSJSP

NSJSP Manager Application	4-1
Overview	4-1
NSJSP Manager Features	4-2
NSJSP Security	4-3
NSJSP Manager Operations	4-3
Admin Web Application	4-56

Overview and Architecture	4-57
Admin Web Application Features	4-59
Login and Security Considerations	4-59
Managing Admin Web Application Operations	4-62
Administering the Server	4-62
Administering a Service	4-64
Administering a Connector	4-66
Administering a Host	4-67
Administering a Context	4-69
Administering a Realm	4-71
Administering a Valve	4-72
Administering Resources	4-74
Administering User Definitions	4-79
Access Security Considerations	4-83
Persisting Changes to the <code>server.xml</code> File and Context Files	4-83
Manager Web Application	4-85
Operations Using the Command-line Interface	4-85
iTP Secure WebServer Operations	4-85
Server Class Operations	4-86
Manual Deployment and Undeployment of Web Applications	4-90
Deploying Applications at Startup	4-90
Deploying Applications on a Running NSJSP Server	4-91
Comparison of the Management Applications	4-92
Comparison of Architectures	4-92
Comparison of Features	4-98
Comparison of Management Application Access Roles	4-100
Single Point of Management Using the NSJSP Manager	4-100
The Architecture of the NSJSP Manager	4-104

5. Logging in NSJSP

Logging Architecture	5-1
Loggers	5-1
Handlers	5-2
Formatters	5-2
Log Manager	5-3
Apache Tomcat Enhancements to the Logging Architecture	5-4
NSJSP Enhancements to the Logging Architecture	5-5
NSJSP Formatter	5-5
NSJSP Log Handler	5-5

Log Rollover	5-6
Logging Configuration	5-9
Configuring Handlers	5-10
Configuring Loggers	5-13
Configuring the NSJSP Formatter Class	5-14
Configuring for Log Rollover	5-15
Configuring the <code>logging.properties</code> File	5-20
Configuring Logging for the NSJSP Container and Web Applications	5-24
Log Files Related to NSJSP	5-27
The <code>out</code> and <code>err</code> Log Files	5-27
Log File Created by JULI	5-28
Programming Considerations for Logging	5-28
LogFactory	5-28
Log	5-29
Commons Logging	5-30

6. Debugging NSJSP

Debugging using Java Debugger tool	6-1
Debugging using Eclipse platform	6-2

7. Migrating to NSJSP 6.1

Comparison of NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1	7-1
Comparing Installation Properties in NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1	7-1
Comparing Configuration Properties in NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1	7-4
Comparing Management Properties in NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1	7-14
Comparing Logging Infrastructure in NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1	7-15
Logging Configuration of Servlet Container Components	7-15
Comparing Miscellaneous Properties	7-18
Considerations for Migrating Web Applications from NSJSP 5.0 to NSJSP 6.1	7-19
Considerations for Migrating Web Applications from NSJSP 6.0 to NSJSP 6.1	7-26
Migrating the Session Store	7-27
Migrating to NSJSP Manager Application in NSJSP 6.1	7-27
Support for Multiple NSJSP Installations in a Single ITP Secure WebServer Environment	7-27

8. Security Considerations

Securing Web Applications	8-1
Establishing a Secure Link	8-2

Authenticating a User	8-3
HTTP Basic Authentication	8-3
HTTP Digest Authentication	8-4
Form-Based Authentication	8-5
HTTPS Client Authentication	8-7
Realms	8-7
Digested Passwords	8-28
Single Sign-On	8-28
Authorizing a User	8-29
Web Resource Collection	8-30
Authorization Constraint	8-31
User Data Constraint	8-31
Validating the Sender	8-33
Remote Host Filter	8-33
Remote Address Filter	8-34
Java Security Manager	8-35
Configuring the Java Security Manager	8-35
Starting NSJSP with the Java Security Manager	8-37
Securing NSJSP Resources Using the <code>permissions</code> Directive	8-38
Package Protection in NSJSP	8-40
Troubleshooting the Java Security Manager	8-41
Manager Web Application and NSJSP Manager Security	8-41
Using Realms to Implement Security	8-42
Monitoring Server Classes and Hosts	8-42

[A. MBeans in the NSJSP Container](#)

Prerequisites	A-1
Overview	A-1
Object Names and Attributes of MBeans	A-2
MBeans Representation in NSJSP Manager	A-4
Commonly Used MBeans in NSJSP	A-10

[Glossary](#)

[Index](#)

Examples

Example 3-1.	A Generic <code>servlet.config</code> File	3-6
Example 3-2.	The <code>httpd.config</code> File Referencing <code>servlet.config</code> File	3-7
Example 3-3.	An Installation-Specific <code>servlet.config</code> File	3-8

Example 3-4.	A Sample Configuration of the <code>server_objectcode</code> Variable	3-12
Example 3-5.	The Default Value of the <code>NSJSP_HOME</code> Variable	3-12
Example 3-6.	The Default Value of the <code>JVM_POLICY_FILE</code> Variable	3-13
Example 3-7.	The Default Value of the <code>NSJSP_SECMGR_POLICY</code> Variable	3-13
Example 3-8.	The Default Value of the <code>NSJSP_SECMGR</code> Variable	3-13
Example 3-9.	The Default Value of the <code>JAAS_CONFIG_FILE</code> Variable	3-14
Example 3-10.	The Default Value of the <code>NSJSP_JAAS_CONFIG</code> Variable	3-14
Example 3-11.	The Default Value of the <code>JAVA_HOME</code> Variable	3-15
Example 3-12.	The Default Value of the <code>JVCP</code> Variable	3-15
Example 3-13.	The Default Value of the <code>USRCP</code> Variable	3-16
Example 3-14.	Default Value of the <code>SERVLET_BANK</code> Variable	3-17
Example 3-15.	Default Value of the <code>NSJSP_DLL_PATH</code> Variable	3-17
Example 3-16.	The Default Value of the <code>CLASSPATH</code> Variable	3-17
Example 3-17.	The Default Value of the <code>JAVA_HOME</code> Variable	3-18
Example 3-18.	The Default Value of the <code>JREHOME</code> Variable	3-18
Example 3-19.	The Default Value of the <code>_RLD_LIB_PATH</code> Variable	3-18
Example 3-20.	The Default Value of the <code>TANDEM_FILEMAPS_CONFIG</code> Variable	3-18
Example 3-21.	The Default Value of the <code>BANK_CATALOG</code> Variable	3-19
Example 3-22.	The Default Value of the <code>NSJSP_CONFIG_FILE</code> Variable	3-19
Example 3-23.	The Default Value of the <code>TANDEM_RECEIVE_DEPTH</code> Variable	3-19
Example 3-24.	The Default <code>Xmx</code>, <code>Xms</code>, <code>Xss</code>, and <code>Xnoclassgc</code> Variables	3-20
Example 3-25.	The Default Value of the <code>java.util.logging.manager</code> Variable	3-20
Example 3-26.	The Default Value of the <code>java.util.logging.config.file</code> Variable	3-21
Example 3-27.	The Default Value of the <code>javax.management.builder.initial</code> Variable	3-21
Example 3-28.	The Default Value of the <code>java.io.tmpdir</code> Variable	3-21
Example 3-29.	The Default Values for <code>catalina.home</code> and <code>catalina.base</code> Variables	3-22
Example 3-30.	A Sample Configuration of the <code>SessionBasedCookieExpiry</code> Variable	3-23
Example 3-31.	New Filemap Information	3-24
Example 3-32.	Filemap History Available as a Comment	3-24
Example 3-33.	Configuration for <code>JMXProxyServlet</code> in the <code>web.xml</code> file	3-25
Example 3-34.	The Default Value of the <code>SaveSessionOnCreation</code> Variable	3-25
Example 3-35.	The Default Configuration of the <code>Region</code> Directive	3-28
Example 3-36.	The Default Configuration of the <code>Filemap</code> Directive	3-29
Example 3-37.	The <code>nsjspadmin.config</code> File	3-30

Example 3-38.	Checking for the presence of the <code>nsjspadmin.config</code> file and sourcing the <code>nsjspadmin.config</code> file in the <code>servlet.config</code> file	3-31
Example 3-39.	The <code>filemaps.config</code> File	3-32
Example 3-40.	Checking for the presence of the <code>filemaps.config</code> file and sourcing the <code>filemaps.config</code> file in the <code>servlet.config</code> file	3-32
Example 3-41.	Specific User Application Filemap Definitions	3-33
Example 3-42.	The <code>jdbc.config</code> File	3-35
Example 3-43.	The <code>server.xml</code> File	3-38
Example 3-44.	The Default Value for the <code>GlobalNamingResources</code> Element	3-42
Example 3-45.	The Default Values for the <code>Service</code> Element	3-43
Example 3-46.	The Default Values for the <code>Connector</code> Element	3-45
Example 3-47.	The Default Values of JMX Connection Listener	3-49
Example 3-48.	The Default Values for the <code>Engine</code> Element	3-50
Example 3-49.	The Default Values for <code>Realm</code> Element	3-52
Example 3-50.	The Default Values for the <code>Host</code> Element	3-53
Example 3-51.	The Default Values for the <code>RequestTrackerValve</code>	3-58
Example 3-52.	The default <code>context.xml</code> File	3-59
Example 3-53.	Configuration of the Default Servlet	3-66
Example 3-54.	Configuration of the Invoker Servlet	3-67
Example 3-55.	Configuration of the JSP Page Compiler and Execution Servlet	3-68
Example 3-56.	Configuration of the SSI Servlet	3-69
Example 3-57.	Configuration of the CGI processing Servlet	3-70
Example 3-58.	Configuration of the Static Content Filter	3-71
Example 3-59.	Configuration of the Session Timeout Parameter	3-71
Example 3-60.	Configuring Virtual Hosts	3-73
Example 3-61.	Arglist from a <code>servlet.config</code> file with <code>SessionBasedLoadBalancing</code> Enabled	3-76
Example 3-62.	The Default Configuration of the NSJSP Specific Standard Manager	3-76
Example 3-63.	The Default Configuration of <code>session-timeout</code> in the <code><NSJSP_HOME>/conf/web.xml</code> File	3-79
Example 3-64.	An Arglist from a <code>servlet.config</code> file with <code>SessionBasedLoadBalancing</code> set to <code>false</code>	3-81
Example 3-65.	The Sample Configuration of the <code>Manager</code> Element	3-81
Example 3-66.	Default Configuration of <code>session-timeout</code> in the <code><NSJSP_HOME>/conf/web.xml</code> File	3-83
Example 3-67.	SQL/MP script to create a Persistent Store	3-84
Example 3-68.	SQL/MX script to create a Persistent Store	3-85
Example 3-69.	Sample Configuration of the <code>Store</code> Element	3-85

Example 3-70.	An Arglst from a <code>servlet.config</code> file with <code>SessionBasedLoadBalancing</code> Set to <code>true</code>	3-88
Example 3-71.	The Sample Configuration of the <code>Manager</code> Element	3-89
Example 3-72.	Default Configuration of <code>session-timeout</code> in the <code><NSJSP_HOME>/conf/web.xml</code> File	3-90
Example 3-73.	The Sample Output of a <code>showddl</code> Command	3-92
Example 3-74.	The Sample Output of a <code>fup info <filename>,detail</code> Command	3-93
Example 3-75.	SQL Session Data Cleanup Script	3-94
Example 3-76.	Using the <code>nsjsp_cleanSessionData</code> Script	3-94
Example 3-77.	Creating A Session Data Table with Four Partitions Across Four Different Disks	3-95
Example 3-78.	Sample Context Configured with a Persistent Manager	3-96
Example 3-79.	Default Configuration of the <code>Manager</code> Element	3-97
Example 8-1.	Creating SQL/MX Tables for use in a <code>JDBCRealm</code>	8-18
Example 8-2.	Sample Realm Configuration	8-19
Example 8-3.	Sample <code>UserDataBase</code> Definition	8-20
Example 8-4.	Defining a Global JNDI <code>Datasource</code>	8-25
Example 8-5.	Sample <code>DataSourceRealm</code> Configuration	8-25
Example 8-6.	Configuring a <code>UserDatabaseRealm</code> and <code>DataSourceRealm</code> Within a <code>CombinedRealm</code>	8-26
Example 8-7.	Configuring an <code>NSJSPLockOutRealm</code>	8-27
Example 8-8.	Java Policy File Entry	8-35
Example 8-9.	Policy File Entry for the <code>NSJSP</code> Container	8-36
Example 8-10.	Starting <code>NSJSP</code> with the Java Security Manager	8-38
Example 8-11.	Setting the Java Security Debug Information	8-41
Example 8-12.	Sample <code>host-access.properties</code> File	8-42
Example 8-13.	Role Definitions	8-43

Figures

Figure 1-1.	Fresh Installation of <code>NSJSP</code> in an <code>iTP Secure WebServer</code> Environment	1-4
Figure 1-2.	<code>NSJSP 5.0/NSJSP 6.0</code> Installation in an <code>iTP Secure WebServer</code> Environment	1-5
Figure 1-3.	Multiple <code>NSJSP</code> Installations in an <code>iTP Secure WebServer</code> Environment	1-6
Figure 1-4.	<code>NSJSP 6.1</code> in an <code>iTP Secure WebServer</code> Environment Configured for Online-Upgrade	1-7
Figure 1-5.	Multiple <code>NSJSP</code> Installations in an <code>iTP Secure WebServer</code> Environment Configured for Online-Upgrade	1-8

Figure 1-6.	NSJSP in an iTP Secure WebServer Environment Configured for Online-Upgrade	1-12
Figure 1-7.	Standalone Apache Tomcat Using the HTTP Connector	1-14
Figure 1-8.	Apache Tomcat with the Apache Web Server	1-15
Figure 1-9.	NSJSP Architecture	1-17
Figure 2-1.	The NSJSP 6.1 Home Page	2-19
Figure 2-2.	The NSJSP Admin Web Application Login Page	2-20
Figure 2-3.	The NSJSP Manager Application Login Page	2-24
Figure 3-1.	Files used to Configure NSJSP 6.1	3-4
Figure 3-2.	Maximum Wait Time for a Message with Size = 1MB	3-27
Figure 3-3.	The Element Hierarchy and Relationships in the <code>server.xml</code> file	3-37
Figure 3-4.	Virtual Hosting in a Sample iTP Secure WebServer Environment	3-72
Figure 3-5.	Request Routing within NSJSP Sessions when <code>SessionBasedLoadBalancing</code> is <code>true</code>	3-75
Figure 3-6.	Session Object Handling by NSJSP with the Persistent Store Configuration	3-80
Figure 4-1.	NSJSP Manager Application Login Page	4-4
Figure 4-2.	NSJSP Manager Application Functions	4-8
Figure 4-3.	Applications Page	4-9
Figure 4-4.	Application Summary Page	4-11
Figure 4-5.	Application Summary Page After Clicking Process View	4-15
Figure 4-6.	In-Memory Sessions Page	4-17
Figure 4-7.	URI Statistics Page	4-19
Figure 4-8.	HTTP Method Statistics Page	4-20
Figure 4-9.	Context Descriptor Page	4-21
Figure 4-10.	Deployment Descriptor Page	4-22
Figure 4-11.	Servlet Mappings Page	4-23
Figure 4-12.	Filters Page	4-24
Figure 4-13.	Initialization Parameters Page	4-25
Figure 4-14.	Application Summary Page Showing the Down Status	4-27
Figure 4-15.	Application Summary Page Showing the Running Status	4-28
Figure 4-16.	Application Summary Page Showing the Reloaded Status	4-29
Figure 4-17.	NSJSP Information Page	4-30
Figure 4-18.	Server Class Processes Page	4-32
Figure 4-19.	NSJSP Connector Stats Page	4-34
Figure 4-20.	Configuration Parameters Page	4-35
Figure 4-21.	Server Class Statistics Page	4-37
Figure 4-22.	Server Class with the <code>\$YSB6</code> PATHMON in the <code>FROZEN</code> State	4-40
Figure 4-23.	Server Class with the <code>\$YSB6</code> PATHMON in the <code>RUNNING</code> State	4-41

Figure 4-24.	Server Class with the \$ZSB6 PATHMON in the FROZEN State	4-43
Figure 4-25.	Server Class with the \$ZSB6 PATHMON in the RUNNING State	4-44
Figure 4-26.	NSJSP MBeans Page	4-45
Figure 4-27.	List of MBeans	4-47
Figure 4-28.	Compare - NSJSP MBeans Page	4-48
Figure 4-29.	Comparison Result for the modelerType Attribute	4-49
Figure 4-30.	Modify MBean Page	4-52
Figure 4-31.	Web Application Deployment from Server	4-54
Figure 4-32.	Web Application Deployment from Desktop	4-56
Figure 4-33.	Admin Operations followed by SAVE	4-58
Figure 4-34.	Operator Commit Changes Command	4-58
Figure 4-35.	Admin Login page	4-60
Figure 4-36.	Admin Web Application Home Page	4-61
Figure 4-37.	The Tomcat Server Element	4-63
Figure 4-38.	The Service Element	4-65
Figure 4-39.	The Connector Element	4-67
Figure 4-40.	The Host Element	4-68
Figure 4-41.	The Context Element	4-70
Figure 4-42.	The Properties of MemoryRealm	4-72
Figure 4-43.	The Properties of AccessLogValve	4-73
Figure 4-44.	The Data Sources Element	4-75
Figure 4-45.	The Properties Displayed After Selecting the Create New Mail Session Action	4-76
Figure 4-46.	The Properties Displayed After Selecting the Create New Env Entry Action	4-77
Figure 4-47.	The Properties of the Default User Database	4-78
Figure 4-48.	The Properties Displayed After Selecting the Create New Resource Link Action	4-79
Figure 4-49.	User Properties for the Admin User	4-80
Figure 4-50.	New Group Properties	4-81
Figure 4-51.	Roles List	4-82
Figure 4-52.	Architecture of the Old Manager Application	4-93
Figure 4-53.	Architecture of the Admin Web Application	4-95
Figure 4-54.	Architecture of the NSJSP Manager Application	4-96
Figure 4-55.	NSJSP Manager Supporting Multiple Hosts	4-101
Figure 4-56.	Multiple NSJSP Installations	4-102
Figure 4-57.	NSJSP in a Pathway Domain	4-103
Figure 4-58.	NSJSP Manager	4-104
Figure 5-1.	Logging Work Flow	5-4
Figure 7-1.	Sample conf Directory Created After NSJSP 6.0 Installation	7-29

Figure 7-2.	Sample <code>conf</code> Directory Contents After NSJSP 6.1 Installation	7-29
Figure 7-3.	NSJSP 6.0 and NSJSP 6.1 Installation Locations	7-31
Figure 8-1.	Flow of User Request	8-2
Figure 8-2.	Logon Page for HTTP Basic Authentication	8-4
Figure 8-3.	Logon Page for HTTP Digest Authentication	8-5
Figure 8-4.	Logon Page for a Form-Based Authentication	8-7
Figure A-1.	Tree View of an MBean	A-4
Figure A-2.	Tree View of an MBean Under the <code>Servlet</code> Node	A-6
Figure A-3.	Tree View of an MBean Under the <code>Valve</code> Node	A-7
Figure A-4.	Node with Value <code>none</code>	A-8
Figure A-5.	Values of the MBean Attributes	A-9
Figure A-6.	Description of the MBean Attributes	A-10

Tables

Table 2-1.	NSJSP 6.1 Installation Directories and Files	2-20
Table 3-1.	Terms and Definition	3-1
Table 3-2.	Configuration Files for Server Classes	3-3
Table 3-3.	Configuration Files for the NSJSP Servlet Container and All Web Applications hosted by an NSJSP Installation	3-3
Table 3-4.	The Generic <code>servlet.config</code> File	3-6
Table 3-5.	The Installation-Specific <code>servlet.config</code> File	3-7
Table 3-6.	Overview of the <code>nsjspadmin.config</code> File	3-29
Table 3-7.	Overview of the <code>filemaps.config</code> File	3-32
Table 3-8.	The <code>jdbc.config</code> File	3-34
Table 3-9.	Overview of the <code>server.xml</code> File	3-36
Table 3-10.	Attribute List for the <code>Server</code> element	3-40
Table 3-11.	Descriptions of Listeners configured as child elements of the <code>Server</code> element	3-41
Table 3-12.	Attribute List of the <code>Service</code> Element	3-44
Table 3-13.	Attribute List for the <code>Connector</code> Element in NSJSP	3-46
Table 3-14.	Attribute List for the <code>Engine</code> Element	3-51
Table 3-15.	Attribute List for the <code>Host</code> Element	3-54
Table 3-16.	The default <code>context.xml</code> File	3-59
Table 3-17.	Attribute List for the <code>Context</code> Element	3-60
Table 3-18.	Attribute List for the <code>NSJSPStandardManager</code>	3-77
Table 3-19.	Attribute List for the <code>Manager</code> element	3-82
Table 3-20.	Attribute List for the <code>Store</code> Element	3-86
Table 3-21.	Attribute List of the <code>Manager</code> Element.	3-90
Table 4-1.	Attributes in the Applications Page	4-10
Table 4-2.	Operations in the Application Summary Page	4-12

Table 4-3.	Attributes in the Application Summary Page	4-13
Table 4-4.	Submenu Items Under the Applications Tab	4-13
Table 4-5.	Attributes in the Process View Page	4-16
Table 4-6.	Operations in the In-Memory Sessions Page	4-18
Table 4-7.	Attributes in the In-Memory Sessions Page	4-18
Table 4-8.	Attributes in the URI Statistics Page	4-19
Table 4-9.	Attributes in the HTTP Method Statistics Page	4-21
Table 4-10.	Attributes in the Servlet Mappings Page	4-24
Table 4-11.	Attributes in the Filters Page	4-25
Table 4-12.	Context Initialization Parameters	4-25
Table 4-13.	Attributes in the NSJSP Information Page	4-31
Table 4-14.	Submenu Items Under the Server Class	4-31
Table 4-15.	Attributes in the Server Class Processes Page	4-33
Table 4-16.	Attributes in the NSJSP Connector Statistics page	4-34
Table 4-17.	Server Class Configuration Parameters	4-36
Table 4-18.	Server Class Statistics	4-38
Table 4-19.	Parameters for Viewing MBeans and their Attributes	4-45
Table 4-20.	Parameters in the Comparing MBeans Across NSJSP Processes Page	4-50
Table 4-21.	Parameters in the Modify MBean Page	4-52
Table 4-22.	Attributes on the Web Application Deployment from Server Page	4-54
Table 4-23.	Attributes on the Web Application Deployment from Desktop Page	4-56
Table 4-24.	Comparison of Features of NSJSP Manager, Old Manager, and Admin Web Applications	4-98
Table 4-25.	Comparing Roles	4-100
Table 5-1.	Handlers in the Java Logging Package	5-2
Table 5-2.	Configuration Properties of <code>FileHandler</code>	5-10
Table 5-3.	Configuration Properties of <code>NSJSPLogHandler</code>	5-11
Table 5-4.	Literals in the Format Attribute	5-15
Table 5-5.	Behavior of Log Files in <code>maxfilesize</code>-based Rollover	5-16
Table 5-6.	Timestamp Attributes	5-17
Table 5-7.	Time Intervals for Rollover	5-18
Table 5-8.	Behavior of Log Files in <code>datepattern</code>-based Rollover	5-19
Table 5-9.	Mapping of Logging Methods	5-29
Table 7-1.	Comparison of Installation Properties of NSJSP Versions	7-2
Table 7-2.	Differences in the NSJSP 5.0 and NSJSP 6.1 <code>server.xml</code> Files	7-5
Table 7-3.	Differences in the NSJSP 6.0 and NSJSP 6.1 <code>server.xml</code> Files	7-9
Table 7-4.	Differences in the NSJSP 5.0 and NSJSP 6.1 <code>servlet.config</code> Files	7-11

Table 7-5.	Differences in the NSJSP 6.0 and NSJSP 6.1 <code>servlet.config</code> Files	7-13
Table 7-6.	Differences and Similarities in Management Properties of NSJSP Versions	7-14
Table 7-7.	Differences and Similarities in NSJSP Versions	7-15
Table 7-8.	Miscellaneous Properties of NSJSP Versions	7-18
Table 7-9.	Prerequisites of NSJSP 5.0 and NSJSP 6.1	7-19
Table 7-10.	Context Definition Attributes in NSJSP 5.0 and NSJSP 6.1	7-21
Table 7-11.	Session Manager Configuration	7-22
Table 7-12.	Prerequisites of NSJSP 6.0 and NSJSP 6.1	7-26
Table 8-1.	Attributes in the JNDIRealm	8-12
Table 8-2.	MemoryRealm Attributes	8-15
Table 8-3.	JDBCRealm Attributes	8-16
Table 8-4.	UserDatabaseRealm Attributes	8-20
Table 8-5.	JAASRealm Attributes	8-22
Table 8-6.	DataSourceRealm Attributes	8-24
Table 8-7.	NSJSPLockOutRealm Attributes	8-27
Table 8-8.	Types of Transport Guarantee	8-32
Table 8-9.	Remote Host Filter Attributes	8-34
Table 8-10.	Remote Address Filter Attributes	8-34
Table A-1.	Attributes Associated with Thread Pool	A-10
Table A-2.	Attributes Associated with Host	A-11
Table A-3.	Attributes Associated with Request Dumper	A-11
Table A-4.	Attributes Associated with Application Context	A-12
Table A-5.	Attributes Associated with JSP Statistics	A-12

What's New in This Manual

Manual Information

Abstract

NonStop™ Servlets for JavaServer Pages (NSJSP) is a container that runs Java servlets and JavaServer Pages (JSPs) that are platform-independent server-side programs, which programmatically extend the functionality of web-based applications by providing dynamic content from a web server to a client browser over the HTTP protocol.

Product Version

NonStop Servlets for JavaServer Pages 6.1 (T1222H60^AAN)

Supported Release Version Updates (RVUs)

This publication supports J06.04 and all subsequent J-series RVUs and H06.15 and all subsequent H-series RVUs, until otherwise indicated by its replacement publications.

Part Number	Published
596210-006	June 2013

Document History

Part Number	Product Version	Published
596210-001	NonStop Servlets for JavaServer Pages 6.1 (T1222H60^AAN)	June 2010
596210-002	NonStop Servlets for JavaServer Pages 6.1 (T1222H60^AAN)	November 2011
596210-005	NonStop Servlets for JavaServer Pages 6.1 (T1222H60^AAN)	March 2012
596210-006	NonStop Servlets for JavaServer Pages 6.1 (T1222H60^AAN)	June 2013

New and Changed Information

Changes to 596210-006 Manual:

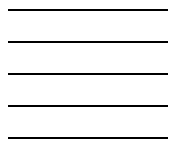
Updated the manual with information about `xnoclassgc` on page 3-20.

Changes to 596219-005 Manual:

Added `maxParameterCount` parameter to the list of Connector element attributes in chapter 3.

Changes to 596210-002 Manual:

Added [Debugging NSJSP](#) chapter.



About This Manual

This guide describes the architecture of NSJSP 6.1, procedures for installing, updating, and removing NSJSP 6.1 and NSJSP Manager, the management tasks you can perform using the NSJSP Manager and Admin Web applications, how to configure NSJSP, logging functionality in NSJSP, a comparison of NSJSP 5.0, 6.0, and 6.1, considerations before migrating user applications from NSJSP 5.0 or NSJSP 6.0 to NSJSP 6.1, and application security and considerations for securing data transfer from a web browser to the web server.

Who Should Read This Guide

The *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide* is intended for experienced HP NonStop system administrators and operators who need to install, configure, and manage NSJSP.

It is assumed that you are an experienced user of HP products and are familiar with:

- The Open System Services (OSS) environment
- The PATHCOM interface of NonStop TS/MP and iTP Secure WebServer
- The Common Gateway Interface (CGI/1.1) standard and the HyperText Transfer Protocol (HTTP/1.1)
- The Java language and tools
- Network security and authentication techniques

This guide also assumes that you have experience operating a secure computing system. For an introduction to the basic network security concepts, see the *iTP Secure WebServer System Administrator's Guide*.

Organization of This Guide

Chapter	Description
1, Introduction to NSJSP	Provides an overview of the NSJSP product. This chapter also describes the NSJSP architecture and its features.
2, Installing NSJSP	Provides the procedures to install, verify, and remove NSJSP and the NSJSP Manager. This chapter also lists the modifications made to support multiple NSJSP installations.
3, Configuring NSJSP	Describes the files for configuring NSJSP in detail. This chapter also discusses specific instances of configuration, such as Virtual Hosts and Session Management.
4, Managing NSJSP	Describes the new NSJSP Manager application that enables you to manage user web applications, server classes, and MBeans. This chapter also describes the Admin Web application and how it enables you to manage the servlet container components.
5, Logging in NSJSP	Describes the logging architecture, logging configuration, and log files in NSJSP. This chapter also describes how to configure logging for user web applications.
6, Debugging NSJSP	Describes the various ways of debugging applications deployed in NSJSP.
7, Migrating to NSJSP 6.1	Describes the installation, configuration, management, and logging characteristics relevant to migrating applications from NSJSP 5.0 and NSJSP 6.0 to NSJSP 6.1. This chapter also describes the procedures involved in migrating web applications from NSJSP 5.0 and NSJSP 6.0 to NSJSP 6.1.
8, Security Considerations	Describes how to secure a web application by establishing a secure link, authenticating a user, and validating the sender. This chapter also discusses how to secure a web application using the Java Security Manager.
Appendix A, MBeans in the NSJSP Container	Describes NSJSP MBeans and how the MBeans are represented in the NSJSP Manager application. This chapter also lists the MBeans that are commonly used in NSJSP.

Related Manuals

NonStop Servlets for JavaServer Pages (NSJSP) Manuals

For information about the features of the NonStop Servlets for JavaServer Pages (NSJSP) 6.0, see *NonStop Servlets for JavaServer Pages (NSJSP) 6.0 System Administrator's Guide*.

iTP manuals

For more information specific to the iTP Secure WebServer environment, see:

<i>iTP Secure WebServer System Administrator's Guide</i>	describes how to install, configure, and manage the iTP Secure WebServer. It also discusses how to develop and integrate Common Gateway Interface (CGI) applications and Java Servlets and JSPs into an iTP Secure WebServer environment. This guide is intended for experienced HP NonStop system administrators and operators who need to install, configure, and manage the iTP Secure WebServer on an HP NonStop system.
--	--

<i>WEBSERV Messages</i> chapter of the <i>Operator Messages Manual</i>	describes the operator messages reported by components of the iTP Secure WebServer and related products.
--	--

The following manuals contain additional information about installing, configuring, and managing HP NonStop systems or other products that may be used with NSJSP.

TCP/IP Manuals

For information specific to managing the TCP/IP subsystem, see:

<i>TCP/IP Configuration and Management Manual</i>	describes the installation, configuration, and management of the NonStop TCP/IP subsystem, which is also called conventional TCP/IP. It is for system managers, operators, and others who require a basic understanding of the HP NonStop TCP/IP implementation.
---	--

<i>HP NonStop TCP/IPv6 Configuration and Management Manual</i>	describes the installation, configuration, and management of the NonStop TCP/IPv6 subsystem. It is for system managers, operators, and others who require a basic understanding of the HP NonStop TCP/IPv6 implementation.
--	--

<i>HP NonStop TCP/IP Programming Manual</i>	describes how to program to the Guardian sockets library for NonStop TCP/IP, Parallel Library TCP/IP, and NonStop TCP/IPv6.
---	---

Open System Services (OSS) Manuals

For information specific to the OSS environment, see:

<i>Open System Services User's Guide</i>	describes the Open System Services (OSS) environment: the shell, file-system, and user commands.
<i>Open System Services Installation Guide</i>	describes how to install and configure the HP NonStop Kernel OSS environment.
<i>Open System Services Management and Operations Guide</i>	describes how to manage and operate the NonStop Kernel OSS environment.

NonStop Transaction Services/MP (NonStop TS/MP) Manuals

For information specific to managing PATHMON environments and managing Pathway domain, see:

<i>TS/MP System Management Manual</i>	describes the PATHCOM and TACL commands used to configure and manage PATHMON environments. This manual also includes manageability guidelines, information about monitoring and tuning a PATHMON environment to optimize performance, and methods for diagnosing and correcting problems.
<i>TS/MP Management Programming Manual</i>	describes how to start, configure, and manage PATHMON environments programmatically and describes the event messages that report errors and other occurrences of interest to operators.
<i>TS/MP Release Supplement</i>	describes how to start and use the new command line interface called Pathway Domain Management Command interpreter (PDMCOM). PDMCOM provides an easy configuration and management interface for multiple Pathway environments grouped together in a domain.

NonStop Java Manuals

For information about the features of the NonStop Server for Java, or if you plan to use JDBC, see *NonStop Server for Java (NSJ) Programmer's Guide*.

NonStop SQL Manuals

For information specific to the NonStop SQL environment, see:

<i>SQL/MP Reference Manual</i>	describes NonStop SQL/MP, the HP relational database management system that uses SQL to describe and manipulate data in a NonStop SQL/MP database. The manual includes information about SQLCI, the conversational interface to NonStop SQL/MP.
<i>SQL/MX Reference Manual</i>	describes SQL language elements—data types, literals, expressions, functions, and predicates—and SQL statements of NonStop SQL/MX, the HP relational database management system based on ANSI SQL-92. It also includes MXCI commands.

These manuals contain additional information about NonStop systems:

<i>H06.nn Release Version Update Compendium</i>	provides a summary for the products that have major changes in the H-series, including the products' new features, migration issues, and fallback considerations. The compendium is written for system managers or anyone who needs to understand how migrating to H-series RVU affects installation, configuration, operations, system management, maintenance, applications, networks, and databases.
<i>NonStop NS-Series Planning Guide</i>	describes the HP Integrity NonStop NS-series system hardware and provides examples of system configurations to assist in planning for installation of a new system. It also provides a guide to other Integrity NonStop NS-series manuals.

Online Resources

These URL references are available on the Internet:

- General references:
<http://www.w3.org>
- HyperText Transfer Protocol (HTTP) references:
<http://www.w3.org/Protocols/rfc2616/rfc2616.txt>
- Common Gateway Interface (CGI) references:
<http://hoohoo.ncsa.uiuc.edu/cgi>
- Tomcat 6.0 Documentation:
<http://tomcat.apache.org/tomcat-6.0-doc/index.html>
- Java Servlet Specification Version 2.5:
<http://jcp.org/aboutJava/communityprocess/mrel/jsr154/index2.html>

- JavaServer Pages API Specification Version 2.1:
<http://jcp.org/aboutJava/communityprocess/final/jsr245/index.html>

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 in [Backup DAM Volumes and Physical Disk Drives](#) on page 3-2.

General Syntax Notation

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

UPPERCASE LETTERS. Uppercase letters indicate keywords and reserved words. Type 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. Type 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 typed 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 type 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 this example, no spaces are 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

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

Bold Text. Bold text in an example indicates user input typed 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 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:

```
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 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:

```
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
```

Notation for Management Programming Interfaces

This list summarizes the notation conventions used in the boxed descriptions of programmatic commands, event messages, and error lists in this manual.

UPPERCASE LETTERS. Uppercase letters indicate names from definition files. Type these names exactly as shown. For example:

ZCOM-TKN-SUBJ-SERV

lowercase letters. Words in lowercase letters are words that are part of the notation, including Data Definition Language (DDL) keywords. For example:

token-type

!r. The !r notation following a token or field name indicates that the token or field is required. For example:

ZCOM-TKN-OBJNAME token-type ZSPI-TYP-STRING. !r

!o. The !o notation following a token or field name indicates that the token or field is optional. For example:

ZSPI-TKN-MANAGER token-type ZSPI-TYP-FNAME32. !o

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.

Abbreviations

This list defines abbreviations and acronyms used in this guide. Both industry-standard terms and HP terms are included.

AWT. Abstract Windowing Toolkit

ARPA. Advanced Research Project Agency

ATP. Active Transaction Pages

BSD. Berkeley Software Distribution

C. Country

CA. Certificate Authority

CBC. Cipher Block Chaining

CCITT. Consultative Committee for International Telegraph and Telephone

CGI. Common Gateway Interface

CN. Common Name

CWD. Current Working Directory

DES. Data Encryption Standard

DN. Distinguished Name

DNS. Domain Name Server

DSM/SCM. Distributed Systems Management/Software Configuration Manager

DSV. distribution subvolume

DTD. Document Type Definition

EAS. Enterprise Application Server

EMS. Event Management Service

FBA. Forms Based Administration

FTP. File Transfer Protocol

GIF. Graphics Interchange Format

GUI. Graphical User Interface

HTML. HyperText Markup Language

HTTP. HyperText Transfer Protocol

HTTPD. HyperText Transfer Protocol Daemon

IEEE. Institute of Electrical and Electronics Engineers

IEN. Internet Engineering Note

IP. Internet Protocol

ISV. installation subvolume

J2EE. Java 2 Enterprise Edition

JAR. Java Archive Tool

JDBC. Java DataBase Connectivity

JDK. Java Development Kit

JIT. Just-In-Time (Java compiler)

JNDI. Java Naming and Directory Interface

JNI. Java Native Interface

JSP. JavaServer Pages

JVCP. Java Visual Class Package

JVM. Java Virtual Machine

KEK. Key Exchange Key

L. Locality

LAN. Local Area Network

LDAP. Lightweight Directory Access Protocol

LDIF. LDAP Data Interchange Format

MAC. Message Authentication Code

MD5. Message Digest

MFK. Master File Key

MIME. Multipurpose Internet Mail Extensions

NCSA. National Center for Supercomputing Applications

NSJ. NonStop Java

NSJSP. NonStop Servlets for JavaServer Pages

O. Organization

OLTP. Online Transaction Processing

OSS. Open System Services

OU. Organizational Unit

PAID. Process Accessor ID

PCT. Private Communication Technology

PDF. Portable Document Format

PEM. Privacy Enhanced Message

PKS. Public Key Certificate Standard

PPP. Point to Point Protocol

QIO. Queued Input Output

RFC. Request for Comments

RLS. Resource Locator Service

RSA. Rivest, Shamir, and Adelman

SCF. Subsystem Control Facility

SCT. Secure Configuration Terminal

SGC. Server Gated Cryptography (Microsoft)

SGML. Standard Generalized Markup Language

SHA1. Secure Hash Algorithm

SI. Session Identifier

SLIP. Serial Line IP

SMTP. Simple Mail Transfer Protocol

SSC. Servlet ServerClass (for Java)

SSI. Server Side Include

SSL. Secure Sockets Layer

ST. State

TACL. Tandem Advanced Command Language

TAL. Transaction Application Language

Tcl. Tool Command Language

Tcl/CGI. Tool Command Language/Common Gateway Interface

TCP/IP. Transmission Control Protocol/Internet Protocol

TS/MP. Transaction Services/Massively Parallel

URI. Uniform Resource Identifier

URL. Uniform Resource Locator

WAR. Web Application Archive

WID. WebSafe2 Interface Driver

WISP. WebSafe2 Internet Security Processor

X.509. CCITT Recommendation for Security Service

XML. Extended Markup Language

HP Encourages Your Comments

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

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

1 Introduction to NSJSP

This chapter describes NonStop Servlets for JavaServer Pages (NSJSP), its components, and its architecture.

To understand NSJSP, prior knowledge of the following topics is required:

- Java Servlets and JavaServer Pages (JSP)

A Java Servlet is a programming object that runs in a server application. It receives client requests, processes them, and generates responses. For more information on the Java Servlet 2.5 specification, see

<http://jcp.org/aboutJava/communityprocess/mrel/jsr154/index2.html>.

JSP is a server side technology that enables you to develop and maintain dynamic web pages. It extends the functionality of web-based applications by providing dynamic content from a web server to a client browser over the Hypertext Transfer Protocol (HTTP). For more information on the JavaServer Pages 2.1 specification, see <http://jcp.org/aboutJava/communityprocess/final/jsr245/index.html>.

- iTP Secure WebServer concepts, such as, how the iTP Secure WebServer components are deployed as TS/MP server classes and how the HTTP daemon (HTTPD) interacts with other components, such as the Common Gateway Interface (CGI)-Server using pathsend calls.

For more information on the iTP Secure WebServer concepts, see the *iTP Secure WebServer System Administrator's Guide*.

This chapter discusses the following topics:

- [Overview](#) on page 1-2
- [NSJSP Product](#) on page 1-2
 - [Apache Tomcat - A Container for Java Servlets and JSP](#) on page 1-2
 - [The HP NonStop Servlet and JSP Container](#) on page 1-2
 - [Installing NSJSP](#) on page 1-3
 - [Configuring NSJSP](#) on page 1-8
 - [Management in NSJSP](#) on page 1-9
 - [Securing Web Applications](#) on page 1-10
- [NSJSP Features](#) on page 1-11
- [Architecture](#) on page 1-13

Overview

NSJSP is a Java Servlets and JavaServer Pages (JSP) container available on HP NonStop operating systems. A servlet container provides an environment in which you can deploy, execute, and manage web applications based on servlets or JSPs. NSJSP is written in Java, and it offers a standards-based environment to host Java Servlet and JSP applications.

NSJSP Product

The NSJSP architecture is derived from the Apache Tomcat servlet container.

Apache Tomcat - A Container for Java Servlets and JSP

Apache Tomcat is a servlet container that implements the Java Servlet and the JSP specifications. It was developed, and is currently maintained by the Apache Software Foundation (ASF) project team.

The default installation of Apache Tomcat includes a servlet and JSP container, and a web server. However, the most commonly used configuration is to not use the built-in web server, but to use the Apache Web Server as the front-end. The Apache Web Server communicates with Apache Tomcat using the Apache JServ Protocol (AJP). The Apache Tomcat component that implements the protocol and provides connectivity between the web server and the container is called a connector.

Apache Tomcat is implemented in Java; it can run on any platform that supports a current Java Virtual Machine (JVM).

The HP NonStop Servlet and JSP Container

NSJSP 6.1 is an enhanced implementation of the Apache Tomcat 6.0.20 release. NSJSP is designed to be installed in an iTP Secure WebServer environment and to use the web server to handle the HTTP protocol. A NonStop connector was developed for NSJSP to enable communication between the iTP Secure WebServer and NSJSP, using Interprocess Communication (IPC). NSJSP processes are also designed to run in server classes, which are defined as part of an iTP Secure WebServer TS/MP configuration.

For more information on TS/MP and server classes, see the *TS/MP System Management Manual*.

NSJSP processes are multi-threaded. Because each process in an NSJSP Server Class has the same configuration, servlet/JSP requests can be load balanced across all the processes in the server class. You can also increase the number of processes in the server class statically or dynamically, which allows for significant scaling of web application capacity.

NSJSP provides the following:

- An implementation of the Java Servlets 2.5 and JSP 2.1 specifications.
- A platform to host applications developed using the Java Open Source frameworks, including:
 - MyFaces - An Apache implementation of JavaServer Faces (JSF)
 - Apache Axis2 - An open source web services engine and framework.
 - Spring - A Plain Old Java object (POJO) oriented application framework for development and deployment of Java applications. For more information, see www.springsource.org
 - Hibernate - A popular (Object Relational Mapping) ORM solution for database access.
- Database access through JDBC drivers for NonStop SQL/MP and SQL/MX.
- Additional APIs, such as, the Pathsend and Pathway APIs implemented in the JToolkit, which can be used in web applications that are deployed in NSJSP.

Note. JToolkit is not bundled with NSJSP.

Installing NSJSP

Each NSJSP Server Class runs in an iTP Secure WebServer environment. The iTP WebServer installation consists of a set of directories with executable files, libraries, configuration files, log files, and a TS/MP configuration for one or two PATHMONs.

NSJSP provides a `setup` script to perform basic installation-related tasks. The `setup` script is located in the NSJSP release version directory and is run from that location. The `setup` script enables you to perform the following installation-related tasks:

- Install NSJSP in an iTP Secure WebServer environment. Starting with the NSJSP 6.1 release, you can install NSJSP in a directory location of your choice. You can also create multiple NSJSP installations in an iTP Secure WebServer environment. NSJSP 6.1 can coexist with other NSJSP 6.1 installations. NSJSP 6.1 can also coexist with either NSJSP 5.0 or NSJSP 6.0 in the same iTP Secure WebServer environment.

You can also install NSJSP in an iTP Secure WebServer environment configured for online-upgrade. In this case, the iTP Secure WebServer installation spans two PATHMONs. For more information on installing, see Chapter [2, Installing NSJSP](#).

- Install the NSJSP Manager, which is a new application introduced in NSJSP 6.1. The NSJSP Manager runs in its own server class called MANAGER. You can create only one installation of the NSJSP Manager application in an iTP Secure WebServer environment.
- Update an existing NSJSP 6.1 installation with a newer NSJSP version.

- Remove an NSJSP installation and optionally delete the installation directories and files.

The following section describes the various installation scenarios.

Installing NSJSP in Different Environments Using the `setup` Script

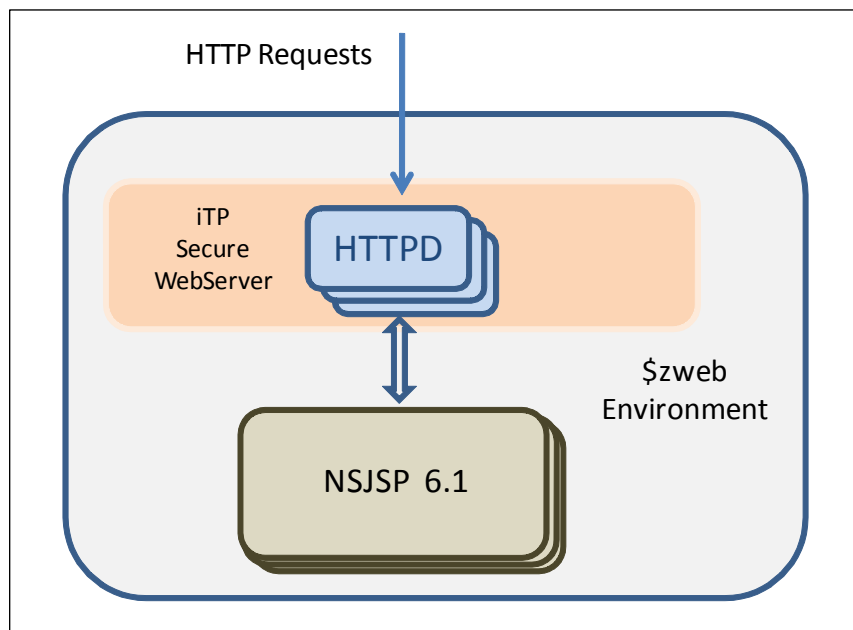
You can use the `setup` script to install NSJSP in the following environments:

- [An iTP Secure WebServer Environment](#)
- [An iTP Secure WebServer Environment Containing Older NSJSP Installations](#)
- [An iTP Secure WebServer Environment Configured for Online-Upgrade](#)

An iTP Secure WebServer Environment

You can create a fresh installation of NSJSP 6.1 in an iTP Secure WebServer environment. In [Figure 1-1](#), the `$zweb` PATHMON environment denotes an iTP Secure WebServer environment, which includes one NSJSP 6.1 installation.

Figure 1-1. Fresh Installation of NSJSP in an iTP Secure WebServer Environment



An iTP Secure WebServer Environment Containing Older NSJSP Installations

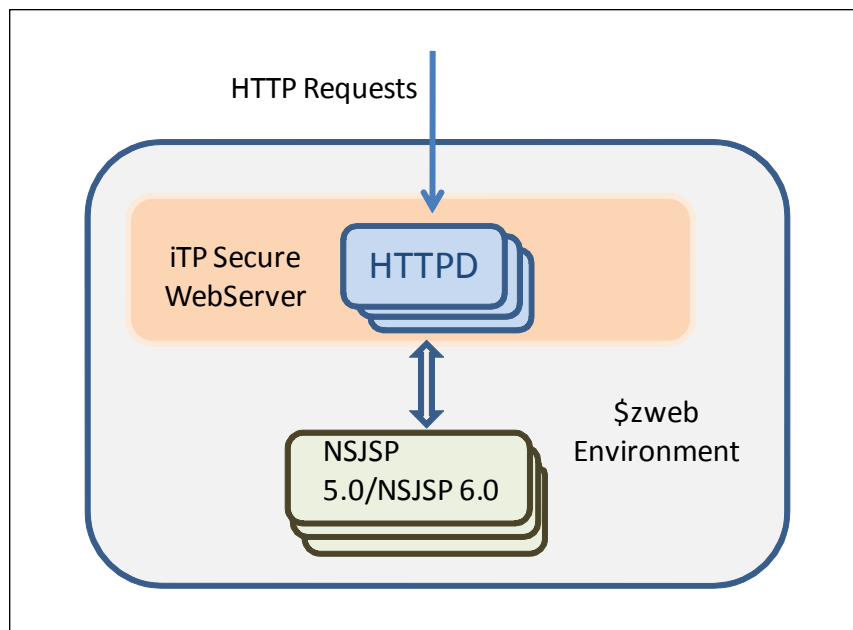
NSJSP 6.1 can coexist with other NSJSP 6.1 installations, and either an NSJSP 5.0 or an NSJSP 6.0 installation. As a result, you can install NSJSP 6.1 in an iTP Secure WebServer environment that includes one or more NSJSP 6.1 installations, and either an NSJSP 5.0 or an NSJSP 6.0 installation. However, you must install each NSJSP installation in a unique directory location.

Note. In the same iTP Secure WebServer environment, NSJSP 6.1 can coexist with either NSJSP 5.0 or NSJSP 6.0, but not with both NSJSP 5.0 and NSJSP 6.0.

NSJSP 6.1 installations will only be included in the iTP Secure WebServer configuration if NSJSP 6.1 is installed after NSJSP 5.0 or NSJSP 6.0 has been installed. If you attempt to install NSJSP 5.0 or NSJSP 6.0 in an iTP Secure WebServer environment that already includes NSJSP 6.1, the older version of NSJSP overwrites part of the NSJSP 6.1 configuration.

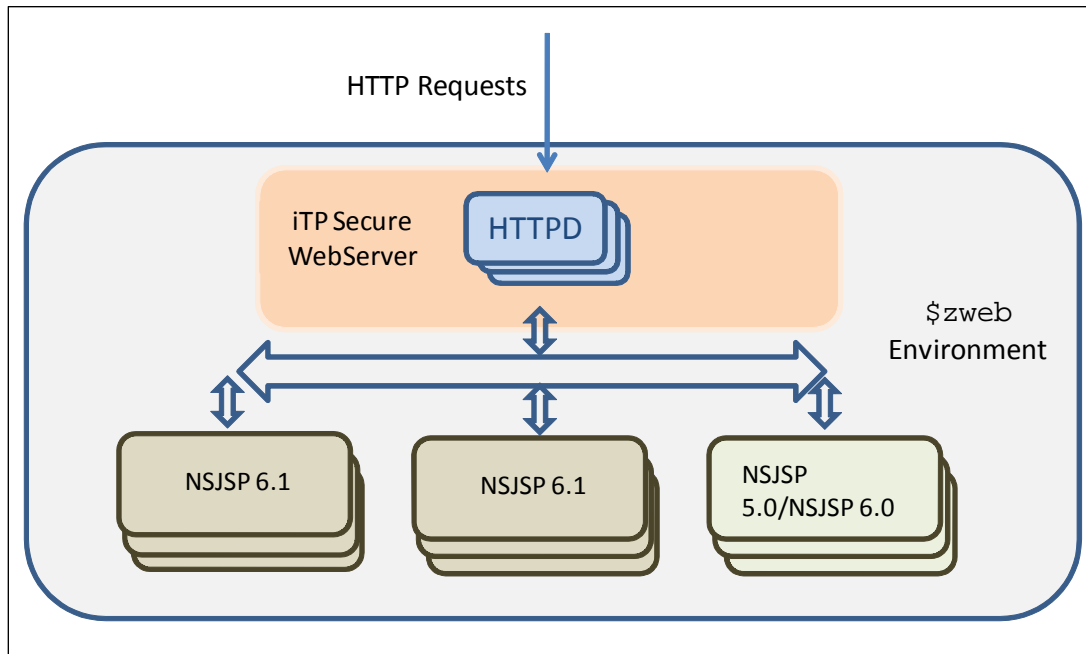
In [Figure 1-2](#), the `$zweb` PATHMON environment denotes an iTP Secure WebServer environment that includes an NSJSP 5.0 or an NSJSP 6.0 installation.

Figure 1-2. NSJSP 5.0/NSJSP 6.0 Installation in an iTP Secure WebServer Environment



In this environment, you can create one or more NSJSP 6.1 installations. [Figure 1-3](#) illustrates an iTP Secure WebServer environment that includes one installation of either NSJSP 5.0 or NSJSP 6.0, and two NSJSP 6.1 installations.

Figure 1-3. Multiple NSJSP Installations in an iTP Secure WebServer Environment



An iTP Secure WebServer Environment Configured for Online-Upgrade

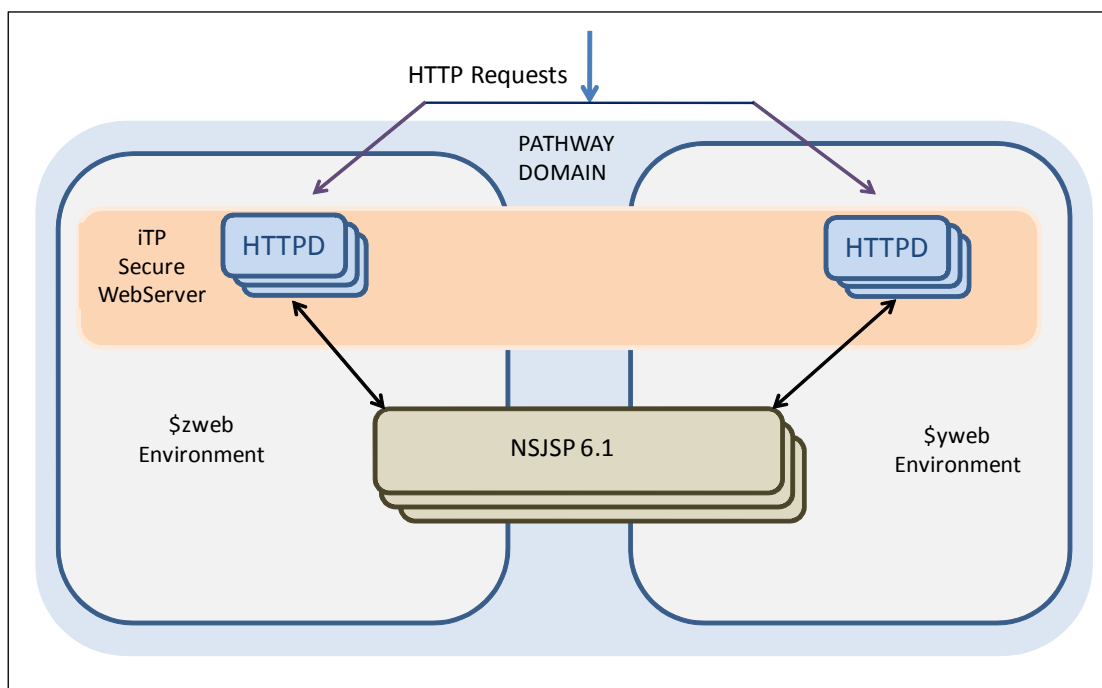
You can install NSJSP 6.1 in an iTP Secure WebServer environment that is configured for online-upgrade.

Note. iTP Secure WebServer 7.0 and later versions provide the capability of online-upgrades. This online-upgrade capability allows an iTP Secure WebServer to be upgraded to a newer version with zero downtime. It requires configuring an iTP Secure WebServer in a Pathway domain that spans two PATHMONs.

For more information on Pathway domains, see the *TS/MP Release Supplement*.

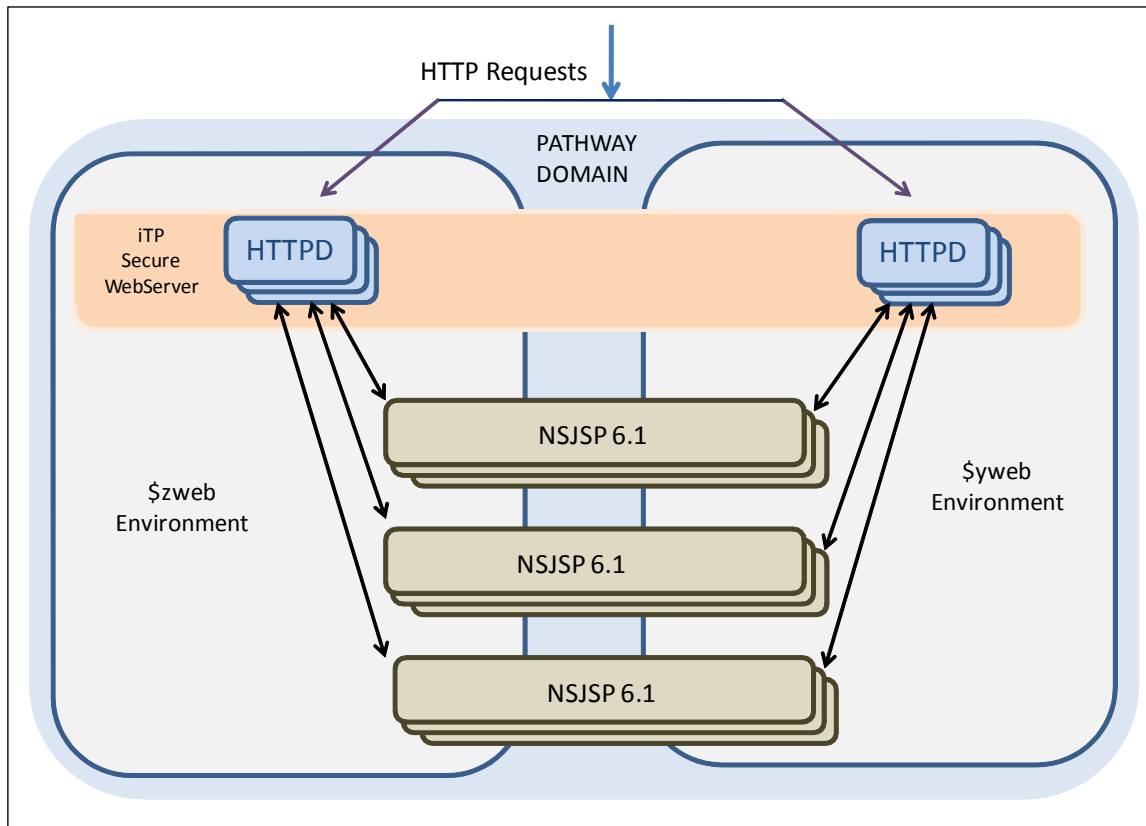
In [Figure 1-4](#), the iTP Secure WebServer environment is configured for online-upgrade. It spans two PATHMONs: \$zweb and \$yweb. You install NSJSP 6.1 in an iTP Secure WebServer environment the same way that it is installed in a single PATHMON environment. In an online-upgrade configuration, however, the iTP Secure WebServer will divide the NSJSP Server Class configuration between the two PATHMONs.

Figure 1-4. NSJSP 6.1 in an iTP Secure WebServer Environment Configured for Online-Upgrade



You can also create multiple NSJSP 6.1 installations in this iTP Secure WebServer environment. However, each NSJSP installation requires a unique directory location. Each NSJSP installation under this iTP Secure WebServer will have its server class processes defined in and run under both the `$zweb` and `$yweb` PATHMON environments. [Figure 1-5](#) illustrates an iTP Secure WebServer environment spanning two PATHMONs, and including multiple NSJSP 6.1 installations.

Figure 1-5. Multiple NSJSP Installations in an iTP Secure WebServer Environment Configured for Online-Upgrade



For more information on installing NSJSP, see Chapter [2, Installing NSJSP](#).

Configuring NSJSP

An NSJSP installation includes a `conf` directory, which contains a `servlet.config` file. If there are multiple NSJSP installations in the iTP Secure WebServer environment, each installation will include its own `servlet.config` file. The `servlet.config` file contains the details of the NSJSP installation, including the TS/MP server class configuration. The TS/MP server class configuration includes the server class name, the URI name that identifies the server class, the JVM configuration, the number of processes, and other PATHMON configuration information.

Additionally, a new `servlet.config` file is created in the `conf` directory within the iTP Secure WebServer installation. This new `servlet.config` file is a generic configuration file. It includes pointers to each of the NSJSP installation-specific `servlet.config` files and a pointer to the NSJSP Manager `servlet.config` file, if the NSJSP Manager is also installed in the iTP Secure Webserver installation directory.

An NSJSP installation also includes a server class for the NSJSP Admin Web application and the old Manager application (Manager Web Application). The configuration file for this server class is located in the `conf` directory within the NSJSP installation, and is called `nsjspadmin.config`.

iTP Secure WebServer Online-Upgrade Configuration Considerations for NSJSP

In a single PATHMON environment, the iTP Secure WebServer creates the NSJSP Server Class as defined in the `conf/servlet.config` file within the NSJSP installation directory. However, if the iTP Secure WebServer is configured for online-upgrade, that is with two PATHMONs in a Pathway domain, the PATHMON configuration is handled differently. The iTP Secure Webserver defines the server class for both Pathmons, but also divides the NSJSP Server Class configuration between the two PATHMONs. One half of the NSJSP Server Class will be configured in one PATHMON and the other half will be configured in the second PATHMON.

Management in NSJSP

Starting with the NSJSP 6.1 release, the new NSJSP Manager application is included. The NSJSP Manager enables single point management of all NSJSP installations within the iTP Secure WebServer environment. The following are some of the important tasks that you can perform:

- Dynamically deploy user web applications on an active NSJSP installation. Also, you can dynamically deploy an application on all the server class processes in an iTP Secure Webserver Pathway domain. Using the old Manager (NSJSP Manager Web Application), you can only deploy an application on the NSJSP Server Class processes running in one PATHMON within a Pathway domain.

Note. You can also deploy a web application to the NSJSP Server Class processes by copying the `.war` file of the application to the `webapps` directory of the NSJSP installation or by copying the application directory, subdirectories, and files to the `webapps` directory of the NSJSP installation.

- View application statistics. You can obtain detailed information about applications, such as, the distribution of application requests across NSJSP Server Class processes and the time taken to process requests.
- View server class statistics, which you can use to tune the server class configuration. You can also perform tasks, such as start and stop the NSJSP Server Class. The new NSJSP Manager provides detailed status and statistics of the server class processes. The old Manager provides limited status and statistics.

Note. Because NSJSP processes run in TS/MP server classes, you can also use PATHCOM and PDMCOM to view the status of server class processes or to display server class statistics.

Starting, Stopping, and Restarting the iTP Secure WebServer

Because NSJSP installations are created in an iTP Secure WebServer environment, when the iTP Secure WebServer environment is stopped, started, or restarted, all the NSJSP installations in the iTP Secure WebServer environment are also stopped, started, or restarted respectively. You can perform these operations on the iTP Secure WebServer by using the scripts located in the iTP Secure Webserver `conf` directory or by using the iTP Secure WebServer Admin application.

You can perform operations on the NSJSP installation server classes by using the NSJSP Manager, PATHCOM, or PDMCOM.

Modifying the NSJSP Configuration

You can make changes to the NSJSP Server Class processes by using the NSJSP Manager and the Admin Web application. Also, you can modify the NSJSP MBeans and any user-defined application MBeans by using the NSJSP Manager.

Note. If you make changes to the configuration files in an NSJSP installation, you must restart the iTP Secure WebServer for the changes to take effect.

Note. When the iTP Secure WebServer is configured for online-upgrade, each NSJSP installation is split across the two PATHMONs. Only the new NSJSP Manager can manage the NSJSP Server Class processes and configurations in both PATHMONs. The old Manager and the Admin Web application can manage only one PATHMON.

For more information on Management in NSJSP, see Chapter [4, Managing NSJSP](#).

Securing Web Applications

NSJSP enables you to secure user web applications.

NSJSP can validate and authenticate users when they attempt to access a web application, and authorize users to access resources. The user credentials required for authentication and authorization are stored in an NSJSP repository.

NSJSP also provides the option of using the Java security manager. The Java security manager is used to safeguard application data and services and ensure the security and reliability of NSJSP.

NSJSP also provides advanced security features, such as locking and filtering of requests. The locking feature ensures that users whose credentials are not valid and who have exceeded the maximum allowed attempts are locked out. The filtering feature ensures that only requests received from specified sources are allowed to execute.

For more information on security, see Chapter [7, Migrating to NSJSP 6.1](#).

NSJSP Features

NSJSP is distinguished by the following key features:

- Scalability

Scalability in NSJSP refers to its ability to increase its capacity to process a large number of servlet and JSP requests simultaneously, by adding resources, such as additional processes and logical processors, to a system. The request throughput rate for NSJSP increases in proportion to the increase in system resources, such as logical processors and the number of cores in multi-core processors. NSJSP is highly scalable because it uses the highly scalable iTP Secure WebServer as a front-end and because it runs as a TS/MP server class, it inherits the scalability features from TS/MP.

- Load balancing

With NSJSP, load balancing is achieved through the use of multiple HP NonStop subsystems, including HP NonStop TCP/IPv6, TS/MP, iTP Secure Webserver, and NSJSP. Using NonStop TCP/IPv6 ensures that the incoming HTTP requests are distributed equally across the HTTPD processes running on all the processors. Also, because HTTPD communicates with NSJSP using server class send calls, TS/MP link management can help ensure that HTTPD requests for NSJSP services are balanced across all the available server class processes of NSJSP.

- High-availability

High-availability guarantees continuous availability of services even during a failure of some system resources, such as processors. Because NSJSP is installed as a TS/MP server class, if there is a processor failure, TS/MP ensures that new processes are started on processors that are still available. In addition to the high-availability features provided by TS/MP, NSJSP itself provides features, such as session persistence to ensure that the state of an application is not lost in the event of a resource failure.

If NSJSP is installed in an iTP Secure WebServer environment configured for online-upgrade (that is, the iTP Secure WebServer is configured in two PATHMONs, within a Pathway domain), you can modify the NSJSP configuration while maintaining continuous access to all NSJSP applications and services.

- iTP Secure WebServer Online-Upgrade

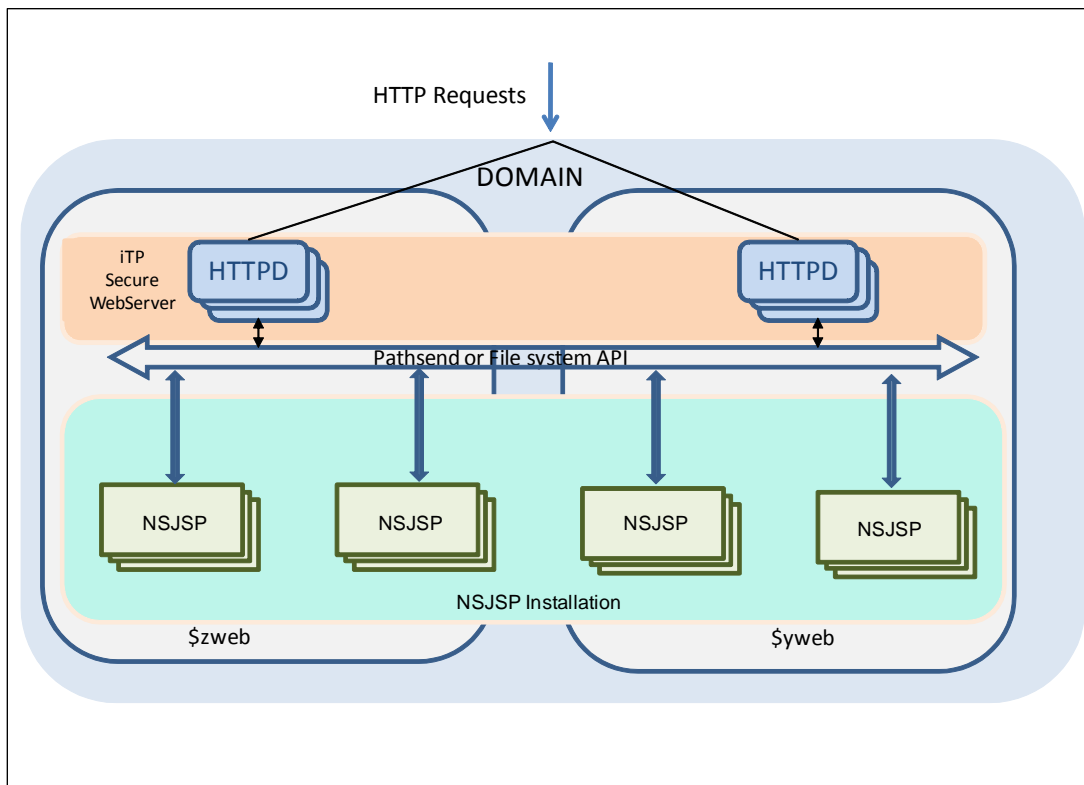
iTP Secure Webserver 7.0 includes the online-upgrade feature, which enables you to perform modifications to the web server or user applications hosted on the iTP Secure WebServer at run time. The online-upgrade feature is enabled by installing the iTP Secure WebServer in a TS/MP Pathway domain environment that includes two PATHMONs. While applications in processes under one PATHMON are shut down for an upgrade, user requests will continue to be processed by the applications running in processes under the other PATHMON.

Starting with the NSJSP 6.1 release, you can install and manage NSJSP in an iTP Secure WebServer 7.0 environment that is configured for online-upgrade.

For more information on the online-upgrade feature, see [An iTP Secure WebServer Environment Configured for Online-Upgrade](#) on page 1-6.

[Figure 1-6](#) illustrates NSJSP in an iTP Secure Webserver environment that is configured for online-upgrade.

Figure 1-6. NSJSP in an iTP Secure WebServer Environment Configured for Online-Upgrade



In [Figure 1-6](#), \$zweb and \$yweb denote two PATHMONs configured in a Pathway domain. The iTP Secure WebServer and the NSJSP installations are distributed across both PATHMONs. The NSJSP processes running in the Servlet Server Class are divided between the two PATHMONs. User requests can be routed to the processes under each PATHMON. If one PATHMON needs to be upgraded or if the processes in one PATHMON are stopped, the processes in the other PATHMON can continue to service the requests, thereby ensuring that NSJSP application services are continuously available.

Architecture

This section discusses the relevant components of Apache Tomcat that will help describe how Apache Tomcat was ported onto the HP NonStop platform as NSJSP, and the NSJSP architecture.

Apache Tomcat Components

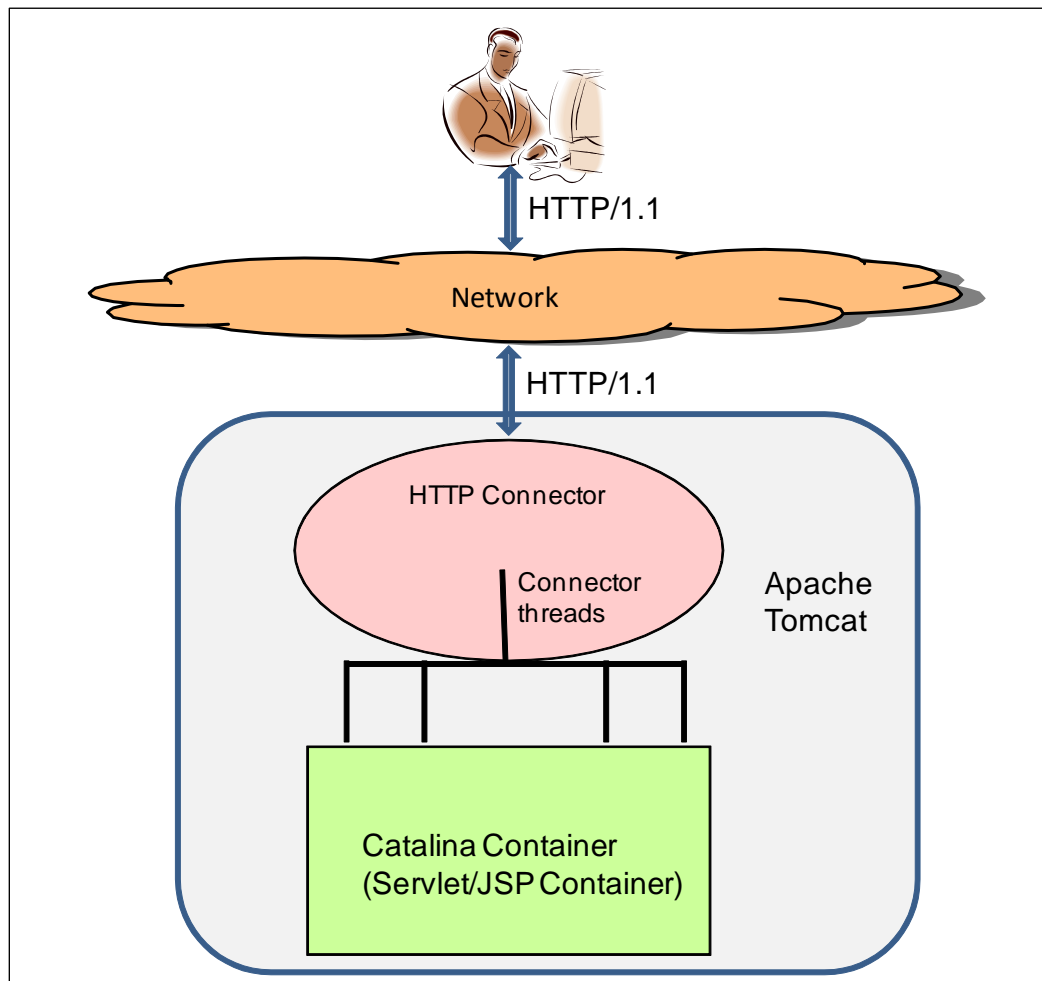
The connector component of Apache Tomcat provides the capabilities of a web server and the container component provides a servlets and JSP container. The connector and the container components are called Coyote and Catalina respectively.

Coyote includes an HTTP connector and an Apache JServ Protocol (AJP) connector. The HTTP connector implements the HTTP/1.1 protocol and the AJP connector implements the AJP 1.3 protocol.

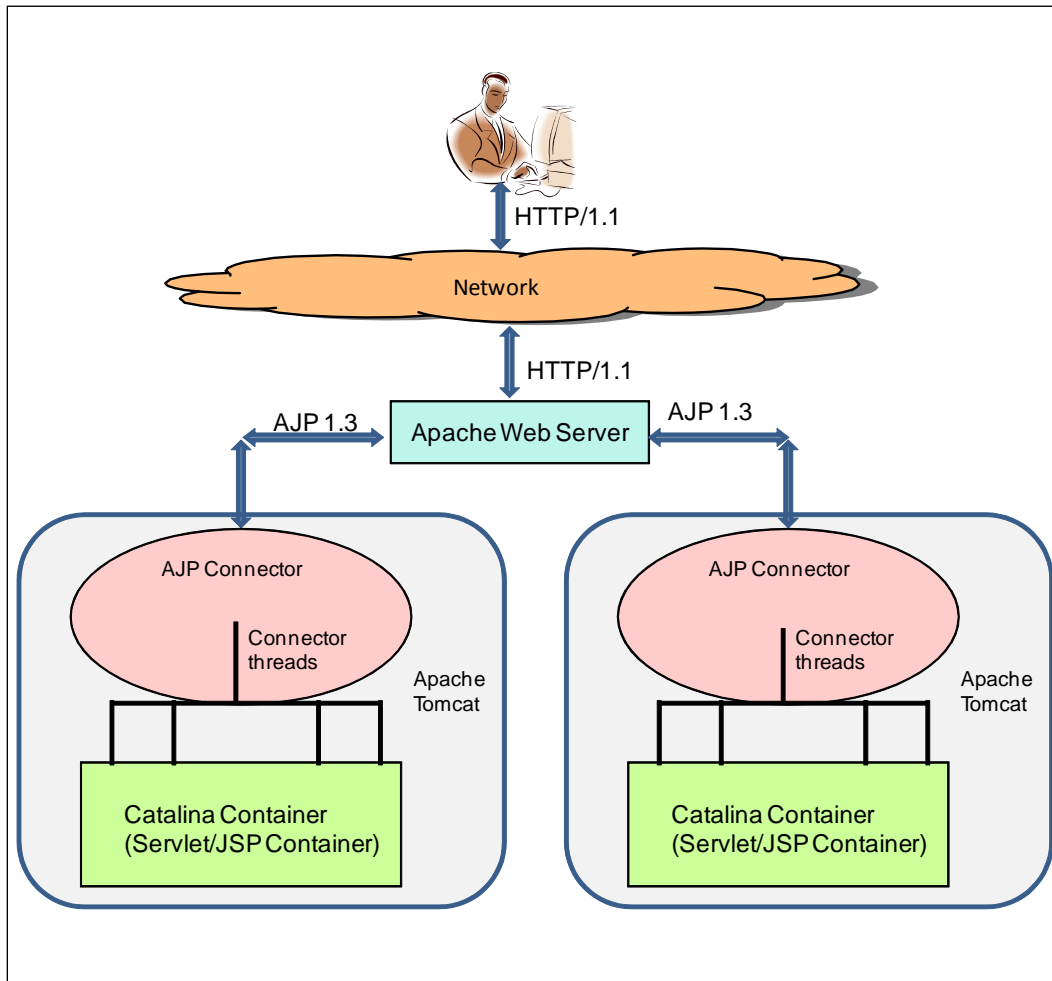
The HTTP Connector allows Apache Tomcat to function as a standalone web server, by listening to and handling HTTP connection requests on a TCP port.

The AJP connector is used to integrate Apache Tomcat with an Apache Web Server. In this case, the Apache Web Server acts as the front end web server for Apache Tomcat. Apache Tomcat then only handles servlet and JSP requests.

[Figure 1-7](#) illustrates a standalone Apache Tomcat servlet container using the HTTP Connector.

Figure 1-7. Standalone Apache Tomcat Using the HTTP Connector

[Figure 1-8](#) illustrates the Apache Tomcat servlet container with the Apache Web Server.

Figure 1-8. Apache Tomcat with the Apache Web Server

The connector validates an incoming message from the web client or the web server for compliance with the respective protocol, such as, HTTP /1.1 or AJP1.3. Subsequently, it creates and allocates a thread for processing the message. The message is then passed to the Catalina component for processing. Each message is processed in a dedicated thread for the duration of message processing.

Catalina processes each request and further invokes a chain of configured Catalina components, such as, the Engine, a Host, and Valves until the request has been completely processed. Catalina loads all deployed web applications upon server startup, maintains a distinct context for each web application that is deployed, maintains all the resources for each web application, prevents unauthorized access of resources, maintains the state of each web application stored in sessions, and provides security to the web applications.

After a message is processed by Catalina, Coyote returns the message response using the configured protocol, such as, HTTP/1.1 or AJP1.3, and sends the data to the

source of the request, such, the web client or the Apache Web Server. After sending the response, the thread is free to process a new message.

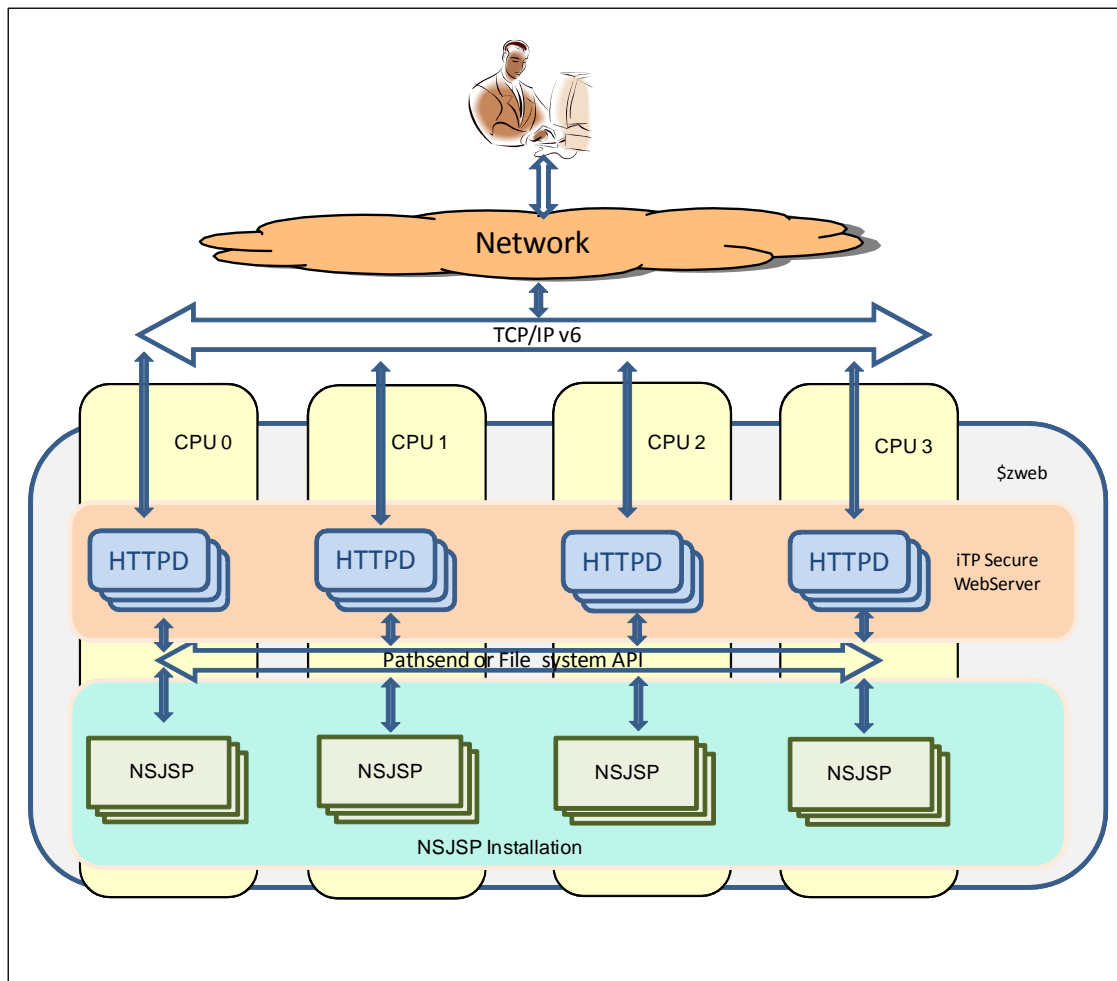
NSJSP Architecture

This section discusses the NSJSP architecture.

HTTPD processes convert the HTTP or Hypertext Transfer Protocol Secure (HTTPS) requests into a set of variables, including environment, HTTP header, and Pathway variables, sets their values, and passes them to the NSJSP processes either through Pathsend or the file system API. The NSJSP connector receives the request data from the HTTPD processes through the NSJSP connector's `$RECEIVE`, which is an Interprocess Communication (IPC) request queue through which NSJSP processes receive messages. The NSJSP connector reads the message contents from `$RECEIVE`, extracts the necessary HTTP request data from the message, and passes the message contents to the Catalina container.

The Catalina container processes the requests it receives from the NSJSP connector, and sends the response back to the NSJSP connector. The NSJSP connector builds a message from the response data returned by Catalina, and returns it to HTTPD. Subsequently, HTTPD sends the response to the originator of the request.

[Figure 1-9](#) illustrates the NSJSP architecture.

Figure 1-9. NSJSP Architecture

In [Figure 1-9](#), NSJSP architecture shows the following:

- An iTP Secure WebServer installation in a TS/MP environment with a PATHMON, called \$zweb. The iTP Secure WebServer Server Class has multiple HTTPD processes distributed across four CPUs.
- A single NSJSP installation in this iTP Secure WebServer environment. The NSJSP installation also comprises multiple NSJSP processes distributed across the four CPUs.
- The HTTPD processes communicate with NSJSP processes using either Pathsend Application Programming Interface (API) or the filesystem API.

The following is the sequence of events that occur in the processing of an HTTP request by NSJSP:

1. NonStop TCP/IPv6 distributes the HTTP(S) user requests across the HTTPD processes.
2. The HTTPD processes map the HTTP requests into a set of variables with values. This information is passed in a message to the NSJSP Server Class through a Pathsend or the File system API.
3. The NSJSP connector receives the request information from an HTTPD process and extracts the required HTTP request data from the message.

For more information on request information variables, see the chapter on Using Common Gateway Interface Programs in the *iTP Secure WebServer System Administrator's Guide*.
4. Subsequently, the connector passes the request information to the Catalina container for further processing.
5. The Catalina container processes the request. The container will invoke any relevant filters and valves, and will pass the request information to the user web application for processing.
6. Catalina sends the application response to the NSJSP connector.
7. The NSJSP connector converts the received response data into an HTTP response message, which is sent to an HTTPD process.
8. The HTTPD process converts the response message to an HTTP protocol response and sends the response to the web client.

2 Installing NSJSP

This chapter discusses the NonStop Servlets for JavaServer Pages (NSJSP) installation script and the prerequisites for NSJSP. This chapter also describes the procedures to install, update, and remove NSJSP and the NSJSP Manager.

Note. NSJSP operates with both iTP Secure WebServer and iTP WebServer. Although this chapter primarily mentions the iTP Secure WebServer, those references apply to the basic iTP WebServer as well. You can install NSJSP in either version of the iTP WebServer.

In the NSJSP context, an iTP Secure WebServer installation creates a TS/MP environment with a PATHMON (or two, if a Pathway domain is configured). Therefore, references to the iTP Secure WebServer environment in this chapter will also include the associated TS/MP environment.

For more information on iTP Secure WebServer and its TS/MP environment, see the *iTP Secure WebServer System Administrator's Guide*.

This chapter discusses the following topics:

- [Prerequisites](#) on page 2-1
- [Installing NSJSP from the CD](#) on page 2-2
- [Creating an NSJSP Installation](#) on page 2-16
- [Creating an NSJSP Manager Installation](#) on page 2-21
- [Updating an NSJSP Installation](#) on page 2-24
- [Removing an NSJSP Configuration](#) on page 2-25
- [Support for Multiple NSJSP Installations in a Single iTP Secure WebServer Environment](#) on page 2-26

Prerequisites

Before installing NSJSP 6.1, ensure that the following products are installed on your NonStop system:

- One of the following NonStop operating systems:
 - J06.04 or later J-series
 - H06.15 or later H-series
- OSS subsystem
- iTP Secure Webserver 7.0 (T8996H02 or T8997H02) or later
- NSJ 5.1 (SPR ABS or later of T2766H51) or NSJ 6.0 (SPR ABP or later of T2766H60)

- JDBC/MX T2 or T4 driver

Note. To run the sample Bank application, which is integrated with the NSJSP PAX, you must have the JDBC T2 or JDBC T4 driver installed on your system.

Installing NSJSP from the CD

This section describes the procedure to run the IPSetup program and the NSJSP 6.1 setup script.

This section includes the following topics:

- [Running the IPSetup Program](#) on page 2-2
- [Running the setup Script](#) on page 2-14

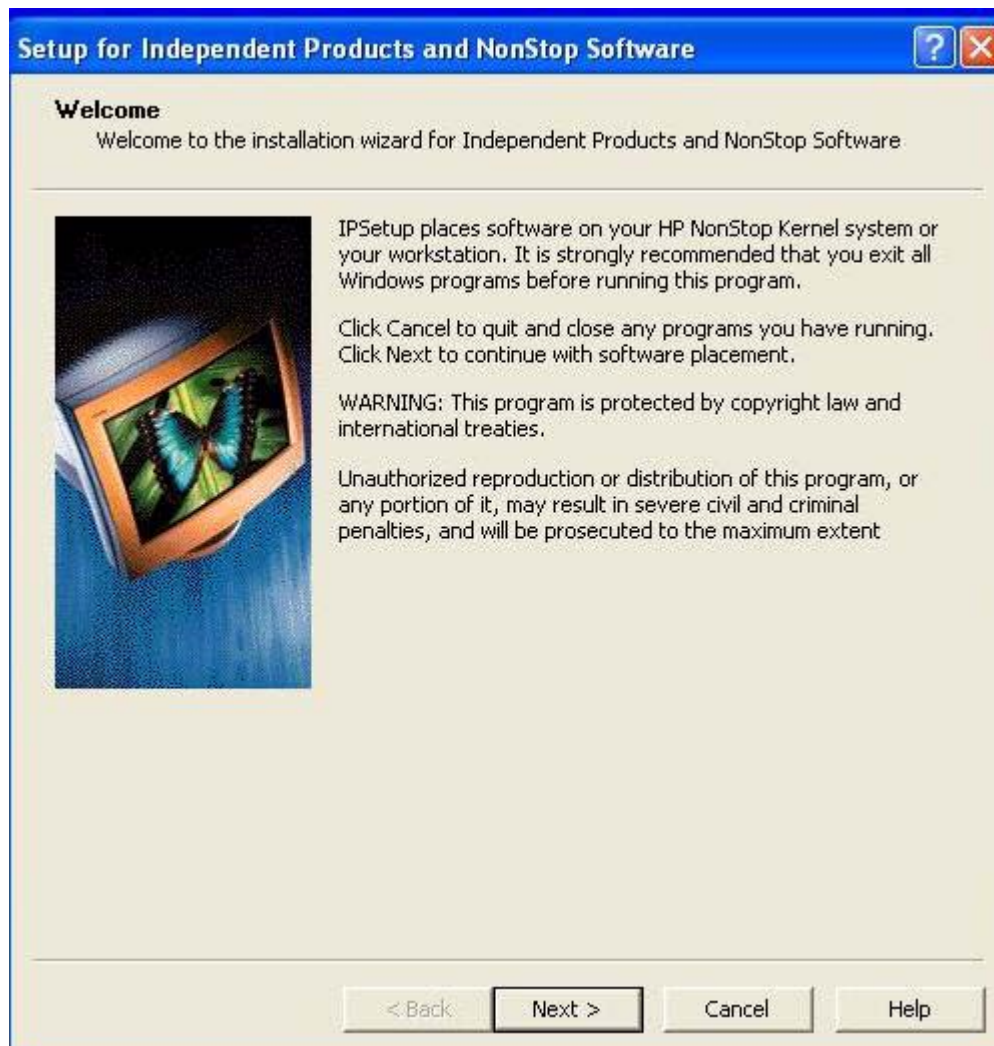
Running the IPSetup Program

The NSJSP software is available on the NSJSP product CD. Use the IPSetup program to move the NSJSP software from the product CD to a NonStop system.

To run the IPSetup program, perform the following steps:

1. Double-click the CD drive to open the product CD and then click the `Setup.exe` file.
The Independent Products Setup screen appears.
2. Click **View the Readme File** option.
The `readme.txt` file opens in Notepad.
3. Review the information provided in the `readme.txt` file and go back to the Independent Products Setup screen.
4. Click **Run IPSetup** option to launch IPSetup.

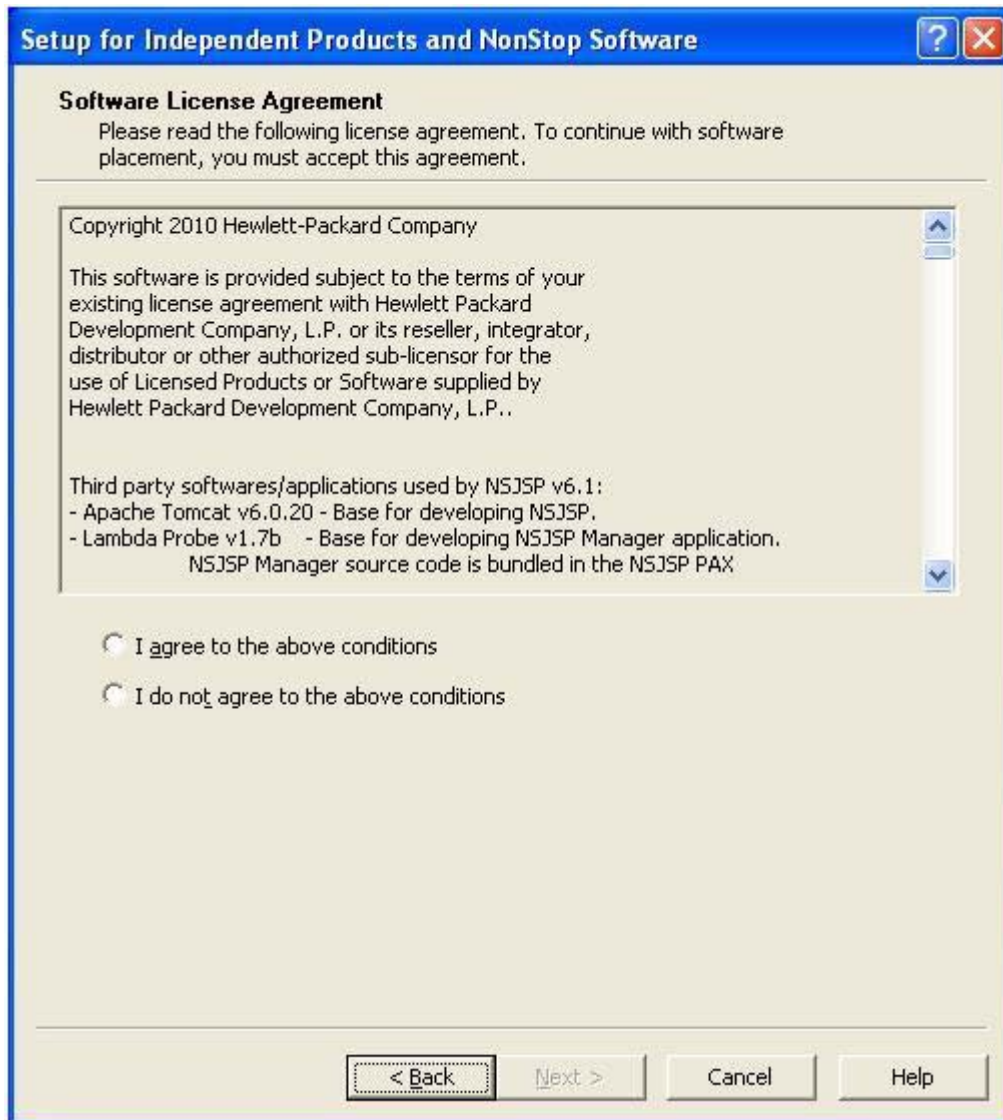
The Welcome screen appears.



Note. HP strongly recommends that you exit all Windows applications before running the IPSetup program.

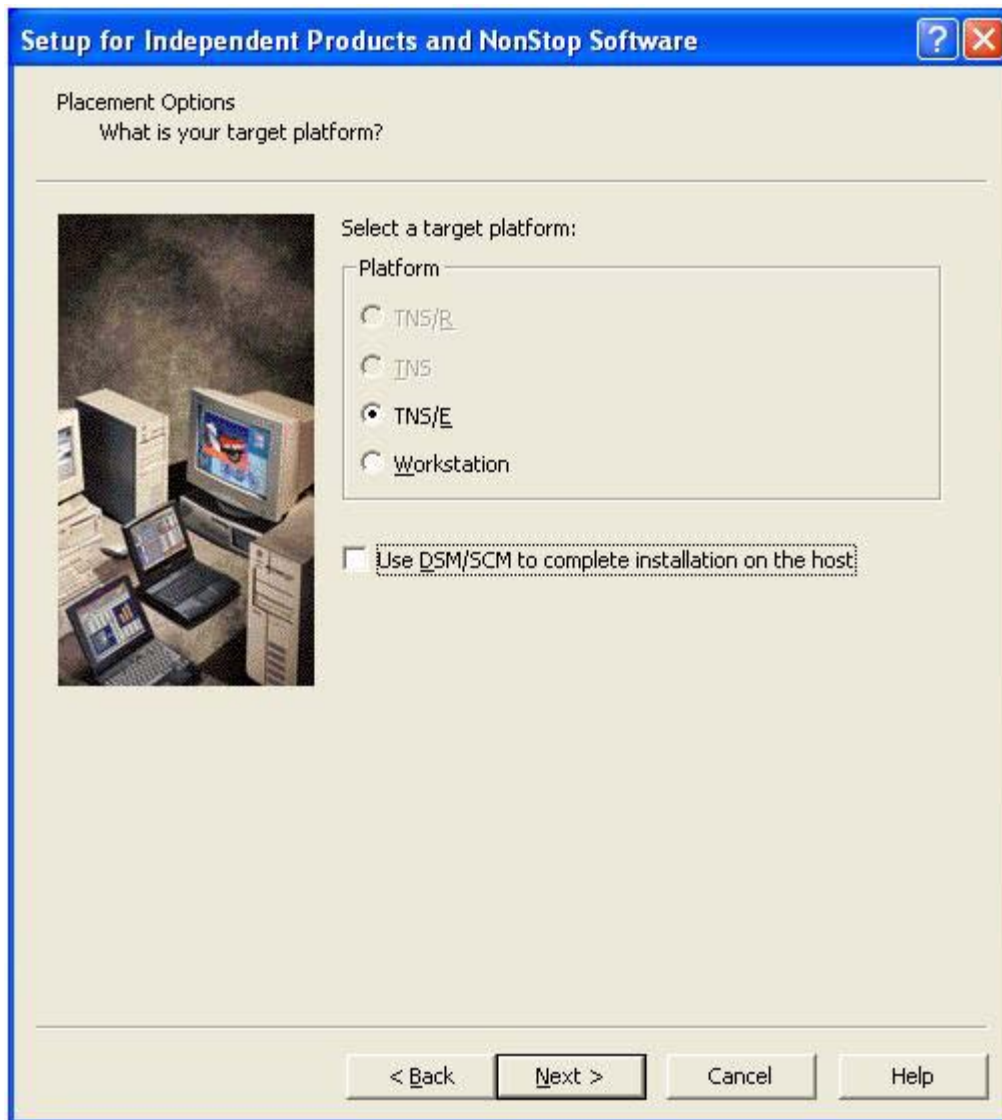
5. Do one of the following:
 - a. Click **Cancel** to exit the IPSetup program and close any other programs that are running.
 - b. Click **Next >** to continue with the IPSetup program.

The Software License Agreement screen appears.



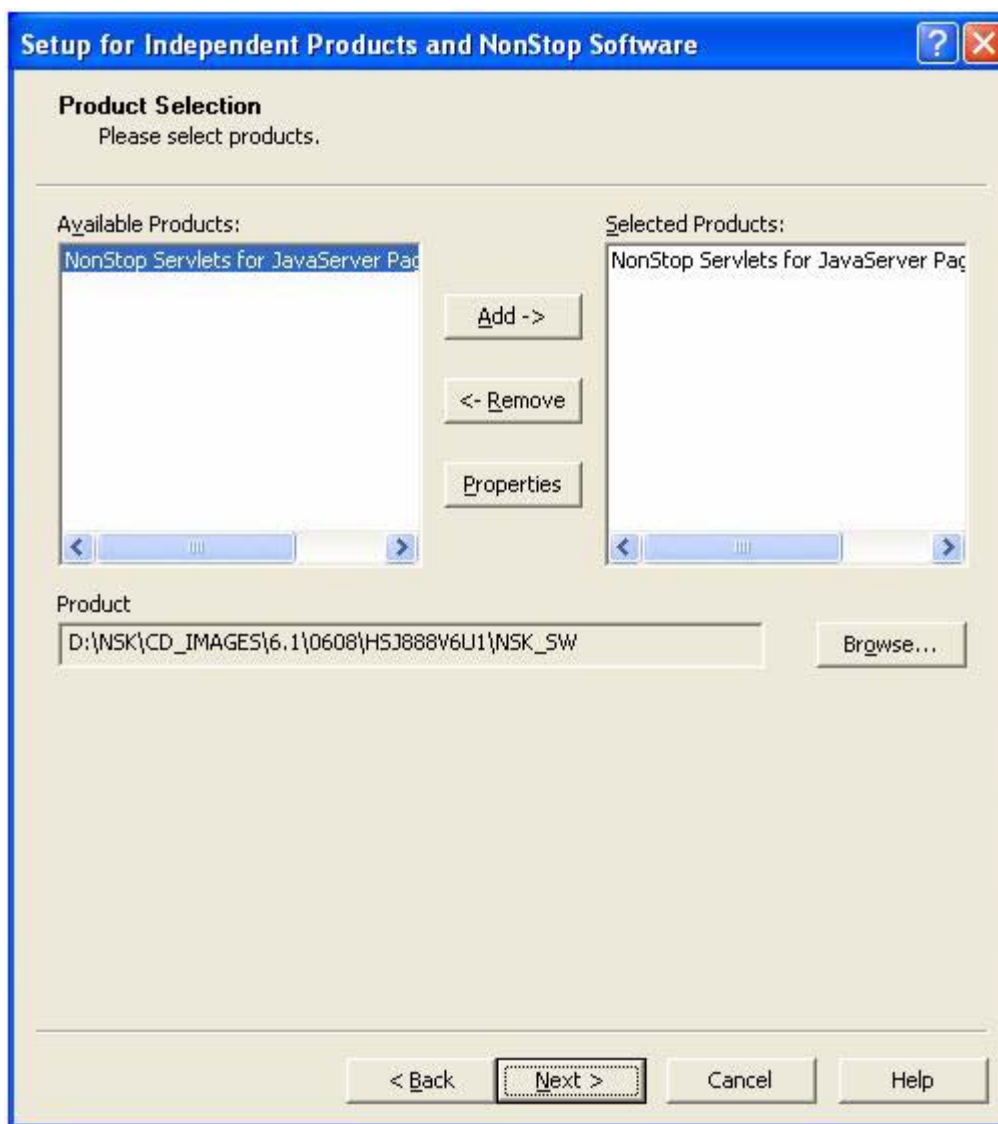
6. Review the License Agreement and do one of the following:
 - a. If you do not want to accept the terms of the agreement, select **I do not agree to the above conditions** and click **Cancel** to exit the IPSetup program.
 - b. To accept all the terms of the agreement, select **I agree to the above conditions** and click **Next >**.

The Placement Options screen appears.



7. Select one of the following as a target platform for your IP software:
 - a. **TNS/E** for H-series and J-series.
 - b. **Workstation** to install the IP on your workstation.
8. Do one of the following and click **Next >**:
 - a. Select the **Use DSM/SCM to complete installation on the host** check box to launch the DSM/SCM planner interface after completing the IPSetup program.
 - b. If you do not want to launch the DSM/SCM after completing the IPSetup program, clear the **Use DSM/SCM to complete installation on the host** check box.

The Product Selection screen appears.



9. From the **Available Products:** list, select **NonStop Servlet for JavaServer Pages** as the product you want to install.
10. Click **Add->**.

The selected product moves to the **Selected Products** list.

11. Click **Next >**.
The Host Information screen appears.

Setup for Independent Products and NonStop Software

Host Information

IPSetup will now gather a list of the available volumes on the host.

Click Next to log on to the host where you would like to place this software.

Host name: 128.88.143.161

User name: super.super

Password: *****

Logonservice: tacl

Select Communication Mode

☒ Telnet ☐ SSH

Telnetport: 23

Ftpport: 21

SSH/Sftpport: 22

< Back Next > Cancel Help

12. Log on to the Host by performing the following steps:
 - a. From the **Host name:** list, select the IP address of a host system where you want to place the selected product.

Note. If the IP address of a Host system is not available in the **Host name:** list, type the IP address.

- b. Enter a user name and password.

- c. Select the communication mode.

Note. Starting with T0316H01^AAK (version 4.1.00.0), IPSetup supports two modes of communication: Telnet and Secure SHell (SSH). To use the SSH mode of communication, ensure that the SSH server is configured correctly and is running on the NonStop server. If the SSH server is not configured or not running on the NonStop server, you will not be allowed to proceed with this mode of communication.

For a secure mode of communication, select **SSH**. Otherwise, accept the default **Telnet**, which sends data in an unsecure mode.

For Telnet mode, enter the logon service that will call the Safeguard prompt. The default service is TACL.

Note.

- HP recommends that you do not change the default service value unless it is required.
- The following are the default port numbers:
 - SSH port — 22
 - Telnet port — 23
 - FTP port — 21

For additional port numbers, consult the system administrator.

- a. Click **Next >**.

The Host Target Settings screen appears.

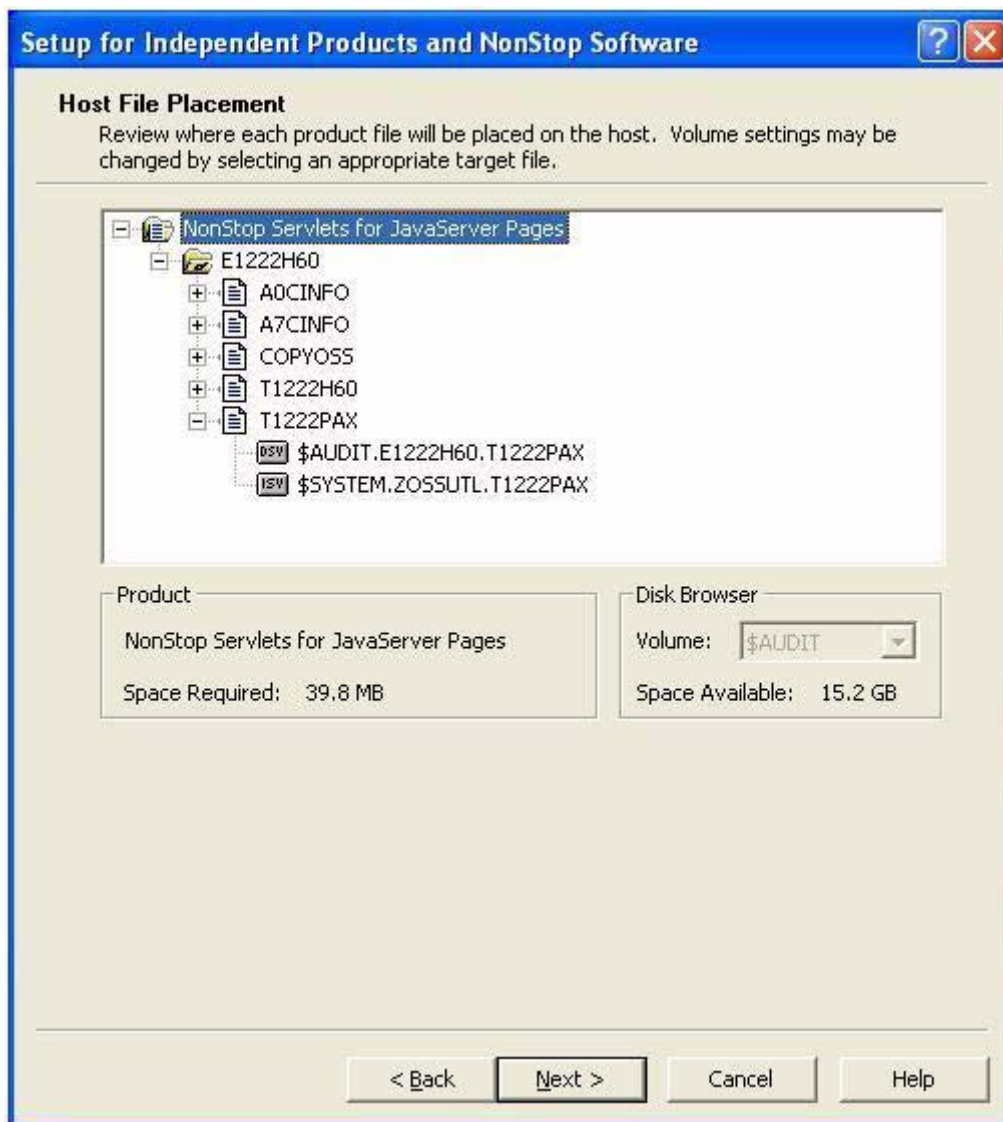


1. Do one of the following:
 - a. Accept the default location for the work subvolume and the subvolume where the existing files will be backed up from the work subvolume.
 - b. Browse the location for a work subvolume and backup of your choice.

Note. If you want to back up the existing files in the work subvolume to another subvolume, select the **Back up existing host files to:** check box.

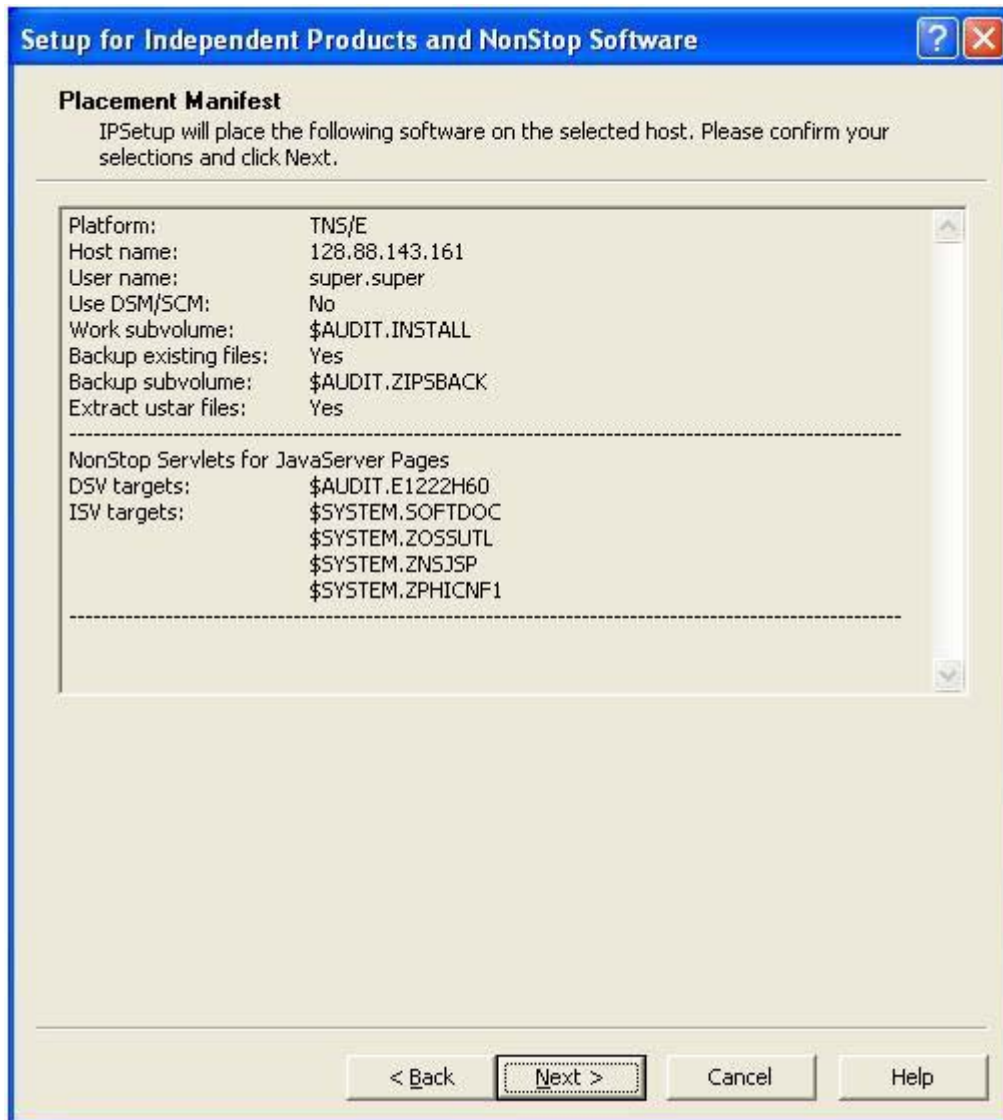
2. Select the **Extract files from ustar archives to OSS file system** check box to extract the ustar files to the OSS file system and click **Next >**.

The Host File Placement screen appears.



3. Verify the location where the product files will be placed on the Host system and click **Next >**.

The Placement Manifest screen appears.



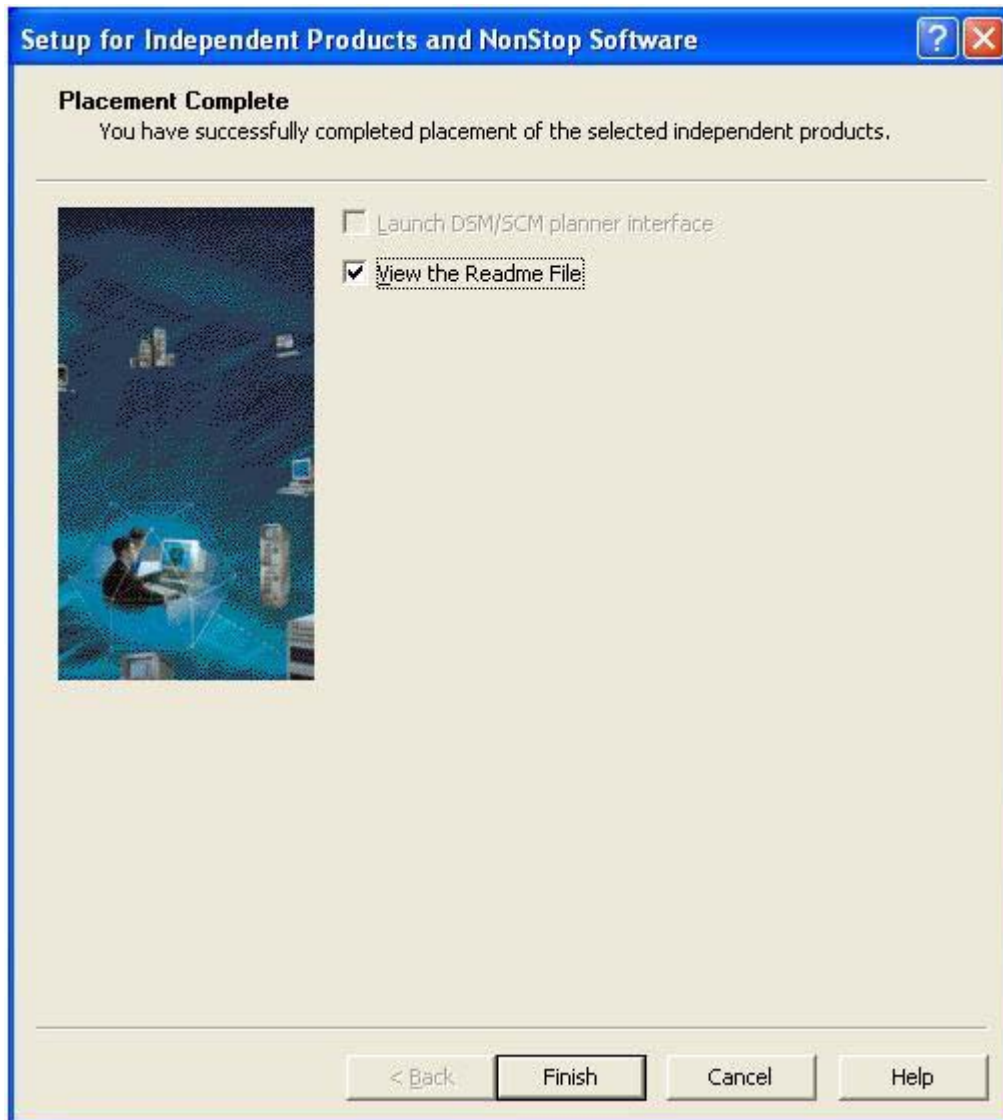
4. Verify the details displayed on the Placement Manifest screen and click **Next >**.

If you select the **Back up existing host files to:** check box in the Host Target Settings screen, IPSetup backs up any existing files to the backup subvolume. If you do not select this check box, IPSetup purges any files in the work subvolume or in the distribution subvolume (DSV) (and installation subvolume (ISV) for IPs) with names identical to files that are about to be placed.

IPSetup then transfers the installation files to the work subvolume and creates DSVs and ISVs. It displays the Placement Status screen, which shows the progress throughout the entire placement process.



After transferring the installation files, the Placement Complete screen appears.



5. Complete your IPSetup program:
 - a. Select the **View the Readme File** check box.
 - b. Click **Finish**.

The IPSetup program completes and opens the `readme.txt` file.

The program saves the contents of the `T1222PAX` file in the version-specific OSS directory located at `<NSJSP base>/<version>`, where:

`<NSJSP base>` is the `/usr/tandem/nsjsp` directory.

`version` is the VPROC string (`T1222H60_27APR2010_AAN_V610_1`).

Note. The `/usr/tandem/nsjsp` directory is referred to as the `<NSJSP base>` directory throughout this chapter.

After the NSJSP files are transferred to a NonStop system using the IPSetup program, complete the [Running the setup Script](#) on page 2-14 procedure to install NSJSP 6.1.

Running the setup Script

Run the `setup` script to complete the NSJSP 6.1 installation. By default, the script is located in the `<version>` directory:

```
<NSJSP base>/<version>
```

Note. A user ID other than `super.super` should be used when running the `setup` script.

To run the `setup` script, complete the following steps:

1. Go to the directory where the `setup` script is located:

```
OSS: cd <NSJSP base>/<version>
```

2. Run the `setup` script at the OSS prompt:

```
OSS: ./setup
```

Note. You can exit the `setup` script at any time by typing `quit` at a prompt. For example:

```
Enter the directory for NonStop™ Server for Java™
[/usr/tandem/java]: quit
```

The following options are displayed:

- a. Create an NSJSP installation

This option enables you to install NSJSP in any of the following environments:

- An iTP Secure WebServer environment.
- An iTP Secure WebServer environment that includes at least one other NSJSP installation—You can install NSJSP 6.1 in an iTP Secure WebServer environment that includes one or both of the following types of installations:
 - one or more NSJSP 6.1 installations
 - an NSJSP 5.0 installation or an NSJSP 6.0 installation
- An iTP Secure WebServer environment configured for online-upgrade—You can install NSJSP 6.1 in a Pathway domain, which is required for an iTP Secure WebServer that is configured for online-upgrade.
- An iTP Secure WebServer environment configured for online-upgrade, and including at least one other NSJSP installation—You can install NSJSP 6.1

in a Pathway domain that includes one or both of the following types of installations:

- one or more NSJSP 6.1 installations
- an NSJSP 5.0 installation or an NSJSP 6.0 installation

For more information on these environments, see [Installing NSJSP in Different Environments Using the setup Script](#) on page 1-4.

For more information on installing NSJSP, see [Creating an NSJSP Installation](#) on page 2-16.

The `Create an NSJSP installation` option also asks if you want to install the NSJSP Manager application if it is not already installed and configured in the selected iTP Secure WebServer environment.

b. `Create an NSJSP Manager installation`

This option enables you to install the NSJSP Manager application. The NSJSP Manager application is used to perform management tasks for the NSJSP server classes and web applications.

For more information on the NSJSP Manager application, see the [NSJSP Manager Application](#) on page 4-1.

For more information on how to install the NSJSP Manager application, see [Creating an NSJSP Manager Installation](#) on page 2-21.

c. `Update an NSJSP installation`

This option enables you to update an existing NSJSP 6.1 installation with a later NSJSP version.

For more information on how to update an NSJSP installation, see [Updating an NSJSP Installation](#) on page 2-24.

d. `Remove an NSJSP configuration`

This option enables you to remove an NSJSP configuration or an NSJSP Manager configuration, and optionally to remove the installation directory and files.

For more information on removing an NSJSP configuration or an NSJSP Manager configuration, see [Removing an NSJSP Configuration](#) on page 2-25.

Creating an NSJSP Installation

After running the `setup` script at the OSS prompt, you must select the `Create an NSJSP installation` option to install NSJSP 6.1. Starting with the NSJSP 6.1 release, the `setup` script can be used to create multiple NSJSP 6.1 installations.

To create an NSJSP installation, complete the following steps:

1. Run the `setup` script, as described in [Running the setup Script](#) on page 2-14.
2. To install NSJSP 6.1, enter **1** at the `setup` script prompt.
3. Enter the directory where the iTP Secure WebServer is installed or press **Enter** to use the default directory, `/usr/tandem/webserver`.
4. Enter a name for the NSJSP installation directory or press **Enter** to use the default directory name, `servlets`.

If you enter a directory that already has an NSJSP 6.1 installation, the `setup` script displays the following message:

```
The directory <NSJSP 6.1 Installation Directory> already
contains an NSJSP installation.
Use the 'Update an NSJSP installation' option to update this
installation.
```

If you enter a directory that already exists, but that does not contain an NSJSP 6.1 installation, the script displays the following message:

```
The directory <user specified directory> already exists.
Files may be overlaid, do you want to continue <y or [n]>:
```

5. Enter the directory that contains the JDBC/MX JAR and library files or press **Enter** to use the default location, `/usr/tandem/jdbcMx/current/lib`.
6. Enter the directory for NonStop Server for Java or press **Enter** to use the default directory, `/usr/tandem/java`.

The script then displays the following message:

```
The initial user name for the Admin and Manager applications
is 'admin'.
```

7. Enter the password for the admin user. After installation, you can log in to the NSJSP Admin Web application or the NSJSP Web Application Manager with the admin user ID and password.

Note. The password must contain at least eight characters, with a combination of upper and lower case characters.

8. Re-enter the password to confirm the value.

9. Enter the NSJSP server class name for this installation or press **Enter** to use the displayed default.

Note. The NSJSP server class name must start with a letter and must contain 1 to 11 alphanumeric or hyphen characters.

10. Enter the uniform resource identifier (URI) name for mapping requests to this installation or press **Enter** to use the displayed default.

For example, if you enter the URI name as `testenviron`, the NSJSP home page can be accessed using the following web address:

`http://15.148.2.1:1088/testenviron`

where,

`15.148.2.1` specifies the *IP address*.

`1088` specifies the *Port number*.

`testenviron` specifies the *NSJSP_6.1_Installation_URI*.

Note. Each NSJSP installation requires a unique URI name.

The script then displays the following information:

- iTP Secure WebServer installation directory
- NSJSP installation directory
- JDBC/MX library directory
- NonStop Server for Java directory
- NSJSP installation server class
- NSJSP URI name

11. Review the information that will be used for the installation and if it is correct, type **y** and press **Enter** to complete the installation.

The `setup` script copies all the required files from the `<NSJSP base>/<version>` directory to the specified NSJSP 6.1 installation directory. It also adds the path of the NSJSP 6.1 installation directory to the `servlet.config` file located in the `<iTP Installation Directory>/conf` directory.

The script then displays 'Installation complete', and shows the URL that can be used to access the NSJSP 6.1 installation and the URL that can be used to access the Admin Web application for this installation.

Note. All the operations performed by the `setup` script are logged to the `install.log` file.

To verify that NSJSP 6.1 is installed successfully, complete the steps described in [Verifying the NSJSP Installation](#) on page 2-18.

After you install NSJSP 6.1, the `setup` script prompts you to install the NSJSP Manager application, if it is not installed in the iTP Secure WebServer. If you have already installed the NSJSP Manager application, the script exits. To create an NSJSP Manager installation, complete the steps described in [Creating an NSJSP Manager Installation](#) on page 2-21.

Verifying the NSJSP Installation

After you have installed NSJSP 6.1, the iTP Secure WebServer must be cold started for the changes to take effect. You can verify the installation by accessing either the NSJSP home page or the Admin Web application login page. The `setup` script displays the uniform resource locators (URLs) for these web pages after completing the installation.

To cold start the iTP Secure WebServer and to verify if NSJSP 6.1 has been successfully installed, complete the following steps:

1. Perform one of the following steps depending on the state of the iTP Secure WebServer:

- Stop and start the iTP Secure WebServer if it is in the running state:

```
OSS: cd <iTP Installation Directory>/conf
OSS: ./stop
OSS: ./start
```

- Run the `start` script from the `<iTP Installation Directory>/conf` directory if the iTP Secure WebServer is not in the running state:

```
OSS: cd <iTP Installation Directory>/conf
OSS: ./start
```

2. Enter the URL for the NSJSP home page into your browser.

The URL must be in the following format:

```
http://<IP address>:<Port number>/<URI for NSJSP 6.1
Installation>
```

For example:

```
http://15.148.2.1:1088/SCP1URI
```

where,

15.148.2.1 specifies the *IP address*.

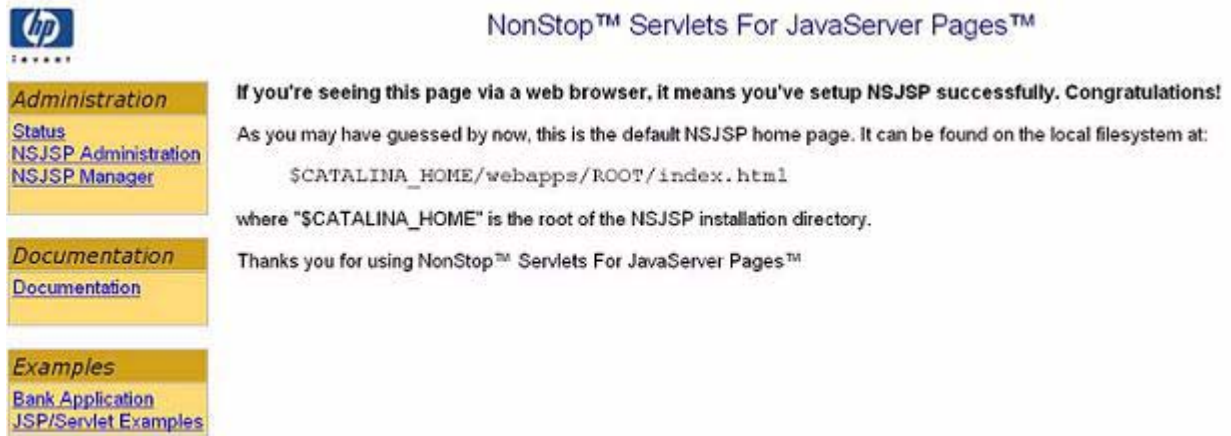
1088 specifies the *Port number*.

SCP1URI specifies the *NSJSP_6.1_Installation_URI*.

Note. The URL for the NSJSP home page can also be retrieved from the `<NSJSP 6.1 Installation Directory>/install.log` file.

[Figure 2-1](#) shows the NSJSP home page for a successful NSJSP 6.1 installation.

Figure 2-1. The NSJSP 6.1 Home Page



You can also verify the NSJSP 6.1 installation by accessing the NSJSP Admin Web application. To access the NSJSP Admin Web application, enter a URL in the following format:

`http://IP address:Port number/<URI for NSJSP 6.1 Installation>/admin`

The NSJSP Admin login page appears.

Note. The URL for the NSJSP Admin Web application can be retrieved from the `<NSJSP 6.1 Installation Directory>/install.log` file.

[Figure 2-2](#) shows the NSJSP Admin Web application login page.

Figure 2-2. The NSJSP Admin Web Application Login Page


NSJSP Installation Directory Structure

[Table 2-1](#) describes the NSJSP 6.1 installation directories located in the NSJSP home directory.

Table 2-1. NSJSP 6.1 Installation Directories and Files (page 1 of 2)

Directory	Description
README	Contains information that will help you to set up an NSJSP environment.
bin/	Contains JAR files that will be included in the classpath.
conf/	Contains NSJSP configuration files and scripts.
deployer/	Contains configuration files used by the Client Deployer.
install.log	Contains log entries for operations performed by the <code>setup</code> script, such as copying the files, making directories, and providing URLs for the NSJSP 6.1 Installation and for the NSJSP Manager application.
lib/	Contains DLLs and the JAR files required for the NSJSP server at runtime.
logs/	Contains the NSJSP log files.

Table 2-1. NSJSP 6.1 Installation Directories and Files (page 2 of 2)

Directory	Description
temp/	Stores temporary working files. This directory is for NSJSP internal use.
webapps/	Contains web applications included within NSJSP and user written web applications.
work/	It is for internal use by the NSJSP servlet container.

Creating an NSJSP Manager Installation

NSJSP 6.1 introduces a new management application, called the NSJSP Manager. The NSJSP Manager application can be used to manage NSJSP server classes running under an iTP Secure WebServer environment. For more information on the NSJSP Manager application, see [NSJSP Manager Application](#) on page 4-1.

Note. You can install only one NSJSP Manager application per iTP Secure WebServer environment.

To install the NSJSP Manager application, complete the following steps:

Note. If you install the NSJSP Manager application as a part of `setup` script option 1, *Creating an NSJSP Installation*, you will only be asked to perform steps 4, 5, 8, 9, and 10 from the following procedure.

1. Run the `setup` script, as described in [Running the setup Script](#) on page 2-14.
2. To install the NSJSP Manager application, enter **2** at the `setup` script prompt.
3. Enter the directory where the iTP Secure WebServer is installed or press **Enter** to use the default directory, `/usr/tandem/webserver`.
4. Enter a name for the NSJSP Manager installation directory or press **Enter** to use the default directory, `manager`.

If you enter a directory that already contains an NSJSP 6.1 installation or an NSJSP Manager application, the script displays the following message:

```
The directory <NSJSP 6.1 Installation Directory> already
contains an NSJSP 6.1.0 installation.
```

```
Use the 'Update an NSJSP installation' option to update this
installation.
```

5. Type **y** at the `setup` script prompt and press **Enter** to create the NSJSP Manager installation directory.
6. Enter the directory that contains the JDBC/MX JAR and library files or press **Enter** to use the default location, `/usr/tandem/jdbcMx/current/lib`.
7. Enter the directory for NonStop Server for Java or press **Enter** to use the default directory, `/usr/tandem/java`.

The script then displays the following message:

```
The initial user name for the Admin and Manager applications
is 'admin'.
```

Note. If you continue to install the NSJSP Manager application after the NSJSP 6.1 installation is complete, the `setup` script uses the same iTP Secure WebServer installation, JDBC/MX library, and NSJ directories that you specified during the NSJSP 6.1 installation.

8. Enter the password for the admin user. After installation, you can log in to the NSJSP Manager application with the admin user ID and password.

Note. The password must contain at least eight characters, with a combination of upper and lower case characters.

9. Re-enter the password to confirm the value.

The `setup` script then displays the following information:

- iTP Secure WebServer installation directory
- NSJSP Manager installation directory
- JDBC/MX library directory
- NonStop Server for Java directory
- NSJSP Manager server class name

The script also displays the following message:

```
This manager will be configured to manage all NSJSP
installations under the PATHMON<s> <user specified PATHMON
names>.
```

Note. By default, the NSJSP Manager Server Class name is `manager`.

10. Review the information that will be used for the NSJSP Manager installation and if it is correct, type **y** at the `setup` script prompt and press **Enter** to install the NSJSP Manager application.

The `setup` script copies the NSJSP Manager files from the `<NSJSP base>/<version>` directory to the specified NSJSP Manager installation directory. It also adds the path of this NSJSP Manager installation directory to the `servlet.config` file located in the `<iTP Installation Directory>/conf` directory.

The script then displays the following message:

```
The NSJSP Manager has been successfully installed.
```

The script will then display the URL that can be used to access the NSJSP Manager.

Verifying the NSJSP Manager Application Installation

After you have installed the NSJSP Manager application, the iTP Secure WebServer must be cold started for the changes to take effect. You can verify the installation by accessing the NSJSP Manager application login page. The `setup` script displays the URL of the login page after completing the installation.

To access the NSJSP Manager application, complete the following steps:

1. Run the `start` script from the `<iTP Installation Directory>/conf` directory of the iTP Secure WebServer that was specified during installation of the NSJSP Manager application:

```
OSS: cd <iTP Installation Directory>/conf
OSS: ./start
```

Note. If the iTP Secure WebServer is in the running state while installing the NSJSP Manager application, you must stop it and then run the `start` script.

2. Enter the URL for the NSJSP Manager application into your browser.

The URL must be in the following format:

`http://IP address:Port number/manager`

For example:

`http://15.148.2.1:1088/manager`

where,

`15.148.2.1` specifies the *IP address*.

`1088` specifies the *Port number*.

`manager` is the *NSJSP Manager server class name*.

The NSJSP Manager login page appears.

Note. The URL of the NSJSP Manager application can also be retrieved from the `<NSJSP Manager Installation Directory>/install.log` file.

[Figure 2-3](#) shows the NSJSP Manager application login page.

Figure 2-3. The NSJSP Manager Application Login Page

NSJSP Manager

User Name

Password

© Copyright 2010 Hewlett-Packard Development Company, L.P.

Updating an NSJSP Installation

The `setup` script includes an option to update an NSJSP installation with a later NSJSP version.

Note. You must copy the new version of NSJSP to the `<NSJSP base>` directory.

To update an existing NSJSP 6.1 installation, complete the following steps:

1. Run the `setup` script, as described in [Running the setup Script](#) on page 2-14.
2. To update an existing installation of NSJSP, enter **3** at the `setup` script prompt.
3. Enter the directory where the iTP Secure WebServer is installed or press **Enter** to use the default directory, `/usr/tandem/webserver`.
4. Enter the name of the NSJSP 6.1 installation directory or the NSJSP Manager installation directory with the version you want to update.

The script displays the list of NSJSP files that will be updated and prompts for the following:

```
Do you want to proceed with the upgrade <y or [n]>:
```

5. Type **y** at the `setup` script prompt and press **Enter** to update the existing installation with the new version of NSJSP.

The `setup` script creates a backup of the old files and replaces them with the new files. It then displays the following message:

Update successful. The iTP Secure Webserver needs to be restarted for the upgrade to take effect.

Removing an NSJSP Configuration

The `setup` script enables you to remove an NSJSP 6.1 configuration or an NSJSP Manager configuration from the iTP Secure WebServer installation using the `Remove an NSJSP Configuration` option. The `setup` script also provides the option to delete the NSJSP installation files and directories.

To remove an NSJSP configuration, complete the following steps:

1. Run the `setup` script, as described in [Running the setup Script](#) on page 2-14.
2. To remove the NSJSP configuration, enter **4** at the `setup` script prompt.
3. Enter the iTP Secure WebServer installation directory that contains the NSJSP installation to be removed or press **Enter** to use the default directory, `/usr/tandem/webserver`.
4. Enter the name of the NSJSP 6.1 installation directory or the NSJSP Manager installation directory to be removed. The script displays the following message:

```
Are you sure you want to remove <NSJSP 6.1 Installation
Directory> from the iTP Secure WebServer configuration
<y or [n]>:
```

5. Type **y** and press **Enter** to remove the NSJSP configuration.

Note. Before removing an NSJSP configuration, ensure that the NSJSP server class or the NSJSP Manager server class is not in the running state. Otherwise, the `setup` script exits and displays the following message at the command prompt:

```
The installation server class
<installation server class name> is still running.
Please shutdown the server class before removing the installation.
```

For an NSJSP 6.1 installation, the script removes the `<NSJSP server class name>.ssc` and `<NSJSP server class name>-adm.ssc` files located in the `<iTP Installation Directory>/bin` directory. The script also removes the path of the NSJSP installation directory from the `servlet.config` file located in the `<iTP Installation Directory>/conf` directory.

For an NSJSP Manager installation, the script removes the `manager.ssc` file located in the `<iTP Installation Directory>/bin` directory. The script also removes the path of the NSJSP Manager installation directory from the `servlet.config` file located in the `<iTP Installation Directory>/conf` directory.

The script then displays the following message:

```
Before deleting <NSJSP 6.1 Installation Directory> ensure  
that this installation was not linked to another webserver.
```

```
Do you want to delete the directory and its contents  
<y or [n]>:
```

6. To remove the NSJSP 6.1 installation directory or the NSJSP Manager installation directory, and its contents, type **y** and press **Enter**.

The NSJSP 6.1 installation directory or the NSJSP Manager installation directory, and its contents are deleted from the iTP Secure WebServer installation directory.

Support for Multiple NSJSP Installations in a Single iTP Secure WebServer Environment

The NSJSP installation script supports multiple NSJSP installations in a single iTP Secure WebServer environment. Following are the modifications made to support multiple NSJSP installations:

- NSJSP installation directory

In releases prior to NSJSP 6.1, NSJSP was installed in the directory (*<iTP Installation Directory>/servlet_jsp*). This default NSJSP location was derived from the location of the iTP Secure WebServer, and it could not be modified. An iTP Secure WebServer directory could include only one *servlet_jsp* directory. As a result, you could have only one NSJSP installation in an iTP Secure WebServer environment. Starting with the NSJSP 6.1 release, you can install NSJSP 6.1 with a directory name of your choice. You can also have multiple NSJSP 6.1 installations in the same iTP Secure WebServer environment. However, each installation must be present in a separate directory, have a unique server class name, and be assigned a unique Uniform Resource Identifier (URI) name.

- NSJSP configuration files required by the iTP Secure WebServer

In releases prior to NSJSP 6.1, during the NSJSP installation, the following iTP Secure WebServer related configuration files were created in *<iTP Installation Directory>/conf*:

- *jdbc.config*
- *nsjspadmin.config*
- *servlet.config*
- *filemaps.config*

Starting with the NSJSP 6.1 release, each NSJSP 6.1 installation includes its own set of the listed configuration files. The configuration files for NSJSP 6.1 installations are no longer present in the iTP Secure WebServer *conf* directory.

Instead, the files are present in the `conf` directory within each NSJSP installation. The location of these NSJSP 6.1 installation-specific configuration files is
<NSJSP 6.1 Installation Directory>/conf.

Note. Within the same iTP Secure WebServer environment, configuration files for NSJSP versions earlier than NSJSP 6.1 will be present in the
<iTP Installation Directory>/conf directory.

For more information about multiple NSJSP installations in a single iTP Secure WebServer environment, see Chapter [6, Migrating to NSJSP 6.1](#).

Note. Multiple NSJSP installations are accessible only if NSJSP 6.1 is installed after installing NSJSP 5.0 or NSJSP 6.0. If you attempt to install or update an NSJSP 5.0 or NSJSP 6.0 installation in an iTP Secure WebServer environment that already includes NSJSP 6.1, the `servlet.config` file of the pre-NSJSP 6.1 version will replace the NSJSP 6.1 generic `servlet.config` file. As a result, the iTP Secure WebServer will not detect the NSJSP 6.1 configurations.

3

Configuring NSJSP

This chapter describes how to configure NSJSP. This chapter assumes that you have installed NSJSP 6.1. For more information on Installing NSJSP, see Chapter [2, Installing NSJSP](#).

This chapter discusses the following topics:

- [Overview](#)
- [Configuration Files for the Server Classes](#)
- [Configuration Files for the Servlet Container](#)
- [Virtual Hosts](#)
- [Session Management](#)

[Table 3-1](#) lists the terms and definitions used in this chapter.

Table 3-1. Terms and Definition (page 1 of 2)

Term	Definition
JSP	A server side technology that enables you to develop and maintain dynamic web pages. It extends the functionality of web-based applications by providing dynamic content from a web server to a client browser over the Hypertext Transfer Protocol (HTTP).
Servlet	A Java Servlet is a programming object that generates dynamic content and runs as part of a server application. It receives client requests, processes them, and generates responses.
NSJSP Servlet Container	A servlet container is a program that executes servlets. A servlet container provides an environment in which you can deploy, execute, and manage web applications implemented with servlets or JSPs.
Servlet Server Class	Refers to one of the server classes configured with an installation of NSJSP. Each installation of NSJSP results in two server classes. One server class hosts web applications and processes requests for user applications and the other contains the Admin application and the old Manager Web Application. The server class that will host web applications and process requests for user applications is referred to as the Servlet Server Class. With NSJSP 6.1, the name of the Servlet Server Class is specified during installation.

Table 3-1. Terms and Definition (page 2 of 2)

Term	Definition
Admin Server Class	Refers to one of the server classes configured with an installation of NSJSP. Each installation of NSJSP results in 2 server classes. One server class will host web applications and process requests for user applications and the other contains the Admin application and the old Manager Web Application. The server class that is used by the Admin application is referred to as the Admin Server Class. With NSJSP 6.1, the name of the Servlet Server Class is specified during installation and the Admin server class name is automatically created by appending <code>-ADM</code> to the name of the servlet server class specified during installation.
Session object	Also called an HTTP Session Object, it is a Java object. It is used to store state between client interactions with web application servlets executing in NSJSP.
Session	A session, also called an HTTP session, provides the means to associate an HTTP Client and an HTTP Server. This association or session, persists over multiple connections and/or requests during a given time period. Sessions are used to maintain the state and identity a user across multiple requests and connections. An example of session state would be the contents of a shopping cart, which is stored in a session object.
<code><iTP Secure WebServer Home></code>	Refers to the directory where iTP Secure WebServer has been installed.
<code><NSJSP_HOME></code>	Refers to the directory where NSJSP has been installed.

Overview

NSJSP is configured to meet specific user requirements by modifying the configuration files installed with NSJSP 6.1. This section provides an overview of the files used to configure NSJSP 6.1.

The files used to configure NSJSP are broadly classified into the following categories:

- Configuration files that describe the NSJSP server classes and how they are used by the iTP Secure WebServer. [Table 3-2](#) provides a brief description of these files.

Table 3-2. Configuration Files for Server Classes

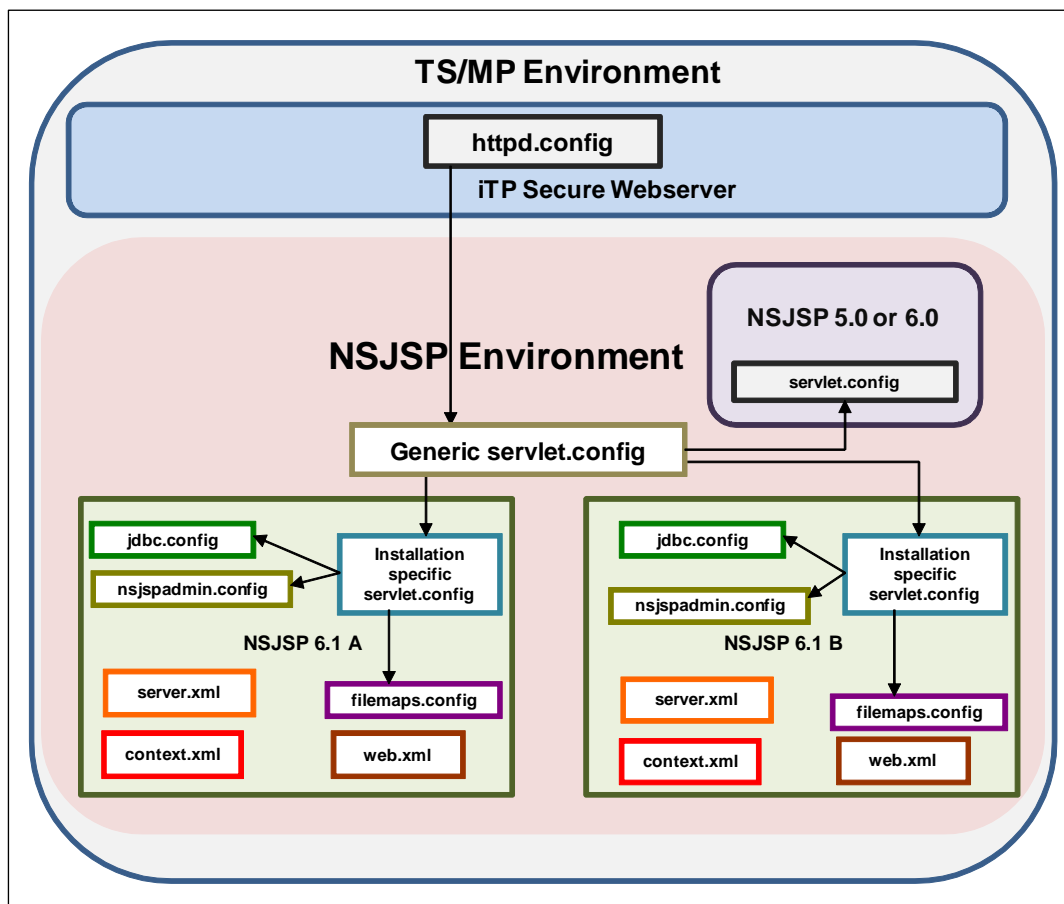
Configuration File	Description
Generic <code>servlet.config</code>	Provides an iTP Secure WebServer with links to the installation-specific <code>servlet.config</code> files. It is common to all NSJSP installations.
Installation-specific <code>servlet.config</code>	Contains the server class specific information used to configure a servlet server class. For more information, see the definition for Servlet Server Class in Table 3-1 .
<code>nsjspadmin.config</code>	Contains the configuration information required by the Admin Server Class.
<code>filemaps.config</code>	Contains the dynamically added Filemaps.
<code>jdbc.config</code>	Contains the JDBC-specific configuration for an NSJSP installation.

- Configuration files for the NSJSP servlet container and configuration files for web applications hosted on NSJSP. [Table 3-3](#) provides a brief description of these files.

Table 3-3. Configuration Files for the NSJSP Servlet Container and All Web Applications hosted by an NSJSP Installation

Configuration File	Description
<code>server.xml</code>	Contains the configuration information for the NSJSP servlet container.
<code>context.xml</code>	The default context loaded by all applications. Each web application can create an application-specific context.
<code>web.xml</code>	A deployment descriptor file containing a set of servlets, such as default, invoker, jsp, ssi, and cgi, along with other definitions that are available to all web applications.

[Figure 3-1](#) shows the relationship between the configuration files across NSJSP installations.

Figure 3-1. Files used to Configure NSJSP 6.1

[Figure 3-1](#) shows two NSJSP 6.1 installations NSJSP 6.1 A and NSJSP 6.1 B along with their configuration files. It also shows an older NSJSP installation that may be either an NSJSP 5.0 or NSJSP 6.0 installation. The `httpd.config` file in the iTP Secure WebServer references the generic `servlet.config` file. The generic `servlet.config` file has links to each installation-specific `servlet.config` file of each NSJSP installation.

Each installation-specific `servlet.config` file in NSJSP 6.1 has links to a `jdbc.config` file, an `nsjspadmin.config` file, and a `filemaps.config` file. The `server.xml` file, the `context.xml` file, and the `web.xml` file are independent of each other.

Note. You can use the default NSJSP configuration settings for development activities without making any changes. However, when deploying NSJSP in a production environment, the configuration might have to be modified to address the load, degree of fault tolerance in the production environment and other considerations. For more information on tuning NSJSP for performance, see the NonStop Servlets for JavaServer Pages (NSJSP) Configuration White Paper at the following location:
<http://www.docs.hp.com/en/588255-001/588255-001.pdf>.

Configuration Files for the Server Classes

An NSJSP installation, by default, creates two server classes, the Servlet server class and the Admin server class. For definitions of the Servlet and the Admin Server Classes, see [Table 3-1](#).

The following files are used to configure these server classes and how they are used by iTP Secure WebServer:

- [The Generic servlet.config File](#)
- [The Installation-Specific servlet.config File](#)
- [The nsjspadmin.config File](#)
- [The filemaps.config File](#)
- [The jdbc.config File](#)

Note. The syntax used in the generic `servlet.config`, installation-specific `servlet.config`, `nsjspadmin.config`, `filemaps.config`, and `jdbc.config` files is from the Tool Command Language (Tcl) and the configuration directives are defined by iTP Secure WebServer. For more information about Tcl and the configuration directives, see the *iTP Secure WebServer System Administrator's Guide*.

The Generic `servlet.config` File

[Table 3-4](#) provides an overview of the generic `servlet.config` file.

Table 3-4. The Generic `servlet.config` File

Location	<iTP Secure WebServer Home>/conf
Description	Contains references to all the active NSJSP installations in an iTP Secure WebServer environment. A <code>source</code> directive is included for each active NSJSP installation in the iTP Secure WebServer environment. Each <code>source</code> directive includes a fully qualified file reference for the installation-specific <code>servlet.config</code> file.
Recommendation	The <code>source</code> directives in the generic <code>servlet.config</code> file should be managed by using the setup script. Note. New <code>source</code> directives are created when the script is used to create a new NSJSP or NSJSP Manager installation and when the script is used to remove an NSJSP installation, the corresponding <code>source</code> directive is removed.

[Example 3-1](#) shows the sample content of a generic `servlet.config` file.

Example 3-1. A Generic `servlet.config` File

```
#
#  VERSION=GENERIC
#

#####
#
# The configuration for SSCAUX will be removed in subsequent
# releases of NSJSP. NSJSP6.1 does not make use of sscaux any more.
#

#####
#
#Server $root/bin/sscaux {
#    eval $DefaultServerAttributes
#    CWD [pwd]
#    Env TANDEM_RECEIVE_DEPTH=1
#    Arglist -server -noautoaccept [HTTPD_CONFIG_FILE]
#    Priority 170
#    Maxservers 5
#    Numstatic 0
#    Maxlinks 16
#}

#####
# List of individual NSJSP configuration files, if any.
#####

source /usr/tandem/webserver/instA/conf/servlet.config
source /usr/tandem/webserver/manager/conf/servlet.config
source /home/usra/nsjsp/priv/conf/servlet.config
```

In [Example 3-1](#), the directories `/usr/tandem/webserver/instA/`, `/usr/tandem/webserver/manager/`, and `/home/usra/nsjsp/priv/` refer to the active NSJSP installations within an iTP Secure WebServer environment. An NSJSP

installation that was removed using the setup script may still be present, including its `servlet.config` file. However, the `source` directive for the installation would have been removed from the generic `servlet.config` file. That would be an example of an inactive NSJSP installation.

The `httpd.config` file uses the `source` directive to access the generic `servlet.config` file as shown in the following example.

Note. `source` is a Tcl command and it executes the contents of the file as a Tcl script. It is also referred as the `source` directive.

Example 3-2. The `httpd.config` File Referencing `servlet.config` File

```
#####
#
# This does an existential check for a servlet.config file. If
# it is there, it will be included in the configuration.
#
if { [file exists $root/conf/servlet.config] } {
    source $root/conf/servlet.config
}
```

The Installation-Specific `servlet.config` File

[Table 3-5](#) provides an overview of the installation-specific `servlet.config` file.

Table 3-5. The Installation-Specific <code>servlet.config</code> File	
Location	<NSJSP_HOME>/conf
Description	Contains configuration parameters required to configure the Servlet Server Class.
Recommendation	Read this complete section to understand the available configuration options. Configuration parameters may need to be changed, based on specific requirements. For more information on performance tuning recommendations, see the NonStop Servlets for JavaServer Pages (NSJSP) Configuration White Paper at the following location: http://www.docs.hp.com/en/588255-001/588255-001.pdf .

[Example 3-3](#) shows the default content of the installation-specific `servlet.config` file.

Example 3-3. An Installation-Specific servlet.config File

```
#
#  VERSION=6.1.0
#

#
#  The server_objectcode represents a SYMBOLIC LINK to your installed
#  copy of the JVM (default /usr/tandem/java/bin/java)
#
set  server_objectcode  $root/bin/instA.ssc

#
#  The NSJSP installation directory.
#
set  env(NSJSP_HOME)  /usr/tandem/java/

#####
#
#  Security Manager options. This allows the Java2 system code to check
#  the policy currently in effect and perform access control checks. This
#  enables us to allow the application and user Servlet/JSP code to run
#  inside its own sandbox.
#
#  NSJSP Java2 System policy file and Java2 VM option.
#
#  Note: the "double" equalto signs "==" is not a typo!! This informs the
#  JVM to use this file exclusively and that all others are to be ignored.
#

set  env(JVM_POLICY_FILE)  $env(NSJSP_HOME)/conf/ITP_catalina.policy
set  NSJSP_SECMGR_POLICY  -Djava.security.policy==$env(JVM_POLICY_FILE)

#
#  By default, the JVM is run without a security manager.
#

set  NSJSP_SECMGR  -Dnsjsp.security.manager=none

#
#  If you wish to run with a security Manager, uncomment the next
#  statement ("set  NSJSP_SECMGR  ...").
#
#  set  NSJSP_SECMGR  -Djava.security.manager

#####
#
#  NSJSP JAAS NonStopLoginModule configuration file.
#
set  env(JAAS_CONFIG_FILE)  $env(NSJSP_HOME)/conf/ITP_jaas.config

#
#  By default, the JVM is run without a JAAS configuration file.
#

set  NSJSP_JAAS_CONFIG  -Dnsjsp.jaas.login.config=none

#
#  If you wish to use the NSJSP JAAS NonStopLoginModule, uncomment the
#  next statement ("set  NSJSP_JAAS_CONFIG  ...").
#
#  Note: the "double" equalto signs "==" is not a typo!! This informs the
#  JVM to use this file exclusively and that all others are to be ignored.
#
#  set  NSJSP_JAAS_CONFIG
#  -Djava.security.auth.login.config==$env(JAAS_CONFIG_FILE)
```

```
#####
#
# Set the JAVA_HOME and GUARDIAN_SUBVOL
#
set env(JAVA_HOME) /nsjsp/software/java6.0/java

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

#####
#
# The following code sets up the Java Specific class PATH variables into
# a Tcl variable called JVCP. JVCP will be prepended to the classpath of
# the JVM.
#

set JVCP ""
append JVCP $env(NSJSP_HOME) "/bin/bootstrap.jar:"
append JVCP $env(NSJSP_HOME) "/bin/commons-daemon.jar:"
append JVCP $env(NSJSP_HOME) "/bin/tomcat-juli.jar:"
append JVCP $env(NSJSP_HOME) "/bin/ns-logger.jar:"
append JVCP $env(NSJSP_HOME) "/bin/nsjspmbeanserver.jar:"

#
# This trims off the rightmost colon char.
#

set JVCP [string trimright $JVCP ":"]

#
# Set the default value of the User ClassPath blank
#

set USRCP ""

#####
#
# Related to the BankDemo example application provided with NSJSP
#
# The value of SERVLET_BANK should be <catalog>.<schema>.<tablename>
# Catalog and schema should have been created. The application will create
# the given table if one is not present

set SERVLET_BANK bankcat.banksch.sraccts

#####
#
# NSJSP SHARED RUNTIME LIBRARY (SRL) location
#

set NSJSP_DLL_PATH $env(NSJSP_HOME)/lib

#####
# Do an existence check on the JDBC specific configuration
# file (jdbc.config). If the file exists, it will be
# included in the configuration.
if { [file exists $env(NSJSP_HOME)/conf/jdbc.config] } {
    source $env(NSJSP_HOME)/conf/jdbc.config
}
}
```

```

if {[info exists jdbcMx_LIB_PATH]} {
    if {[info exists jdbcMp_LIB_PATH]} {
set NSJSP_DLL_PATH $env(NSJSP_HOME)/lib:$jdbcMx_LIB_PATH:$jdbcMp_LIB_PATH
    } else {
set NSJSP_DLL_PATH $env(NSJSP_HOME)/lib:$jdbcMx_LIB_PATH
    }
}

#
# Any custom classpaths can be added by uncommenting and modifying
# the next statement ("append USRCP ...").
#
# append USRCP " :<Custom-ClassPath>"

#
# Add region commands to deny access to internal Catalina Servlets.
#
Region */servlet/org.apache.catalina.servlets.* {
DenyHost *
}

Region */servlet/jsp/* {
DenyHost *
}

Server $server_objectcode {
    CWD $env(NSJSP_HOME)
    Env CLASSPATH=$JVCP:$USRCP
    Env JAVA_HOME=$env(JAVA_HOME)
    Env JREHOME=$env(JAVA_HOME)/jre
    Env _RLD_LIB_PATH=$NSJSP_DLL_PATH
    Env TANDEM_FILEMAPS_CONFIG=$env(NSJSP_HOME)/conf/filemaps.config
    Env BANK_CATALOG=$SERVLET_BANK
    MapDefine =TCPIP^PROCESS^NAME $transport
    Maxservers 4
    Numstatic 4
    Maxlinks 50

    Linkdepth 50
    Env TANDEM_RECEIVE_DEPTH=50

    #
    # File locations to direct standard input, output and error.
    #
    Stdin /dev/null
    Stdout $env(NSJSP_HOME)/logs/sca3.out
    Stderr $env(NSJSP_HOME)/logs/sca3.err

    #
    # This is the actual Arglist used to start up the NSJSP
    # Container.

    Arglist -Xmx64m -Xms64m -Xss128k -Xnoclassgc \
-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager \
-Djava.util.logging.config.file=$env(NSJSP_HOME)/conf/logging.properties \
-Djavax.management.builder.initial=
                                com.tandem.servlet.jmx.NSJSPMBeanServerBuilder \
$NSJSP_SECMGR \
$NSJSP_SECMGR_POLICY \
$NSJSP_JAAS_CONFIG \
-Dcom.tandem.servlet.CONTEXT_PREFIXES=/sca3url \
-Dcatalina.home=$env(NSJSP_HOME) \
-Dcatalina.base=$env(NSJSP_HOME) \

```

```

-Djava.io.tmpdir=$env(NSJSP_HOME)/temp \
org.apache.catalina.startup.Bootstrap start
}

#####
#
# Do an existence check on the nsjspadmin configuration
# file (nsjspadmin.config). If it exists, source it in.
#
if { [file exists $env(NSJSP_HOME)/conf/nsjspadmin.config] } {
source $env(NSJSP_HOME)/conf/nsjspadmin.config
}

#####
#
# Do an existence check on the filemaps configuration
# file (filemaps.config). If it exists, source it in.
#
if { [file exists $env(NSJSP_HOME)/conf/filemaps.config] } {
source $env(NSJSP_HOME)/conf/filemaps.config
}

#
# The filemap directive should always map to wherever you have installed
# the NonStop(tm) Servlets For JavaServer Pages(tm) server object code.
#

Filemap /sca3url $server_objectcode

```

The contents of the installation-specific `servlet.config` file are broadly classified as shown below:

Environment Variables	Sets all the necessary environment variables that are used in the <code>Server</code> directive to configure the server class.
Server Class Configuration using the Server Directive	The actual configuration of the server class using the <code>Server</code> directive.
Miscellaneous	Configuration that is used by the HTTPD component of the iTP Secure WebServer, such as <code>Region</code> and <code>Filemap</code> directives.

Environment Variables

This section describes the environment variables that are explicitly set in the installation-specific `servlet.config` file. Environment variables are used to configure the Servlet Server Class. The installation-specific `servlet.config` file also sources in the variables defined in other configuration files, such as `jdbc.config`.

`server_objectcode`

The `server_objectcode` variable is used to set the complete path of the object file of the server class. The name of the `server_objectcode` must be `<svc>.ssc`. Where `<svc>` is the name provided for the Servlet Server Class when the installation script was run. The extension `ssc` is recognized by the iTP

Secure WebServer. In NSJSP, the `server_objectcode` is a symbolic link to the Java executable. For example, `/usr/tandem/java/bin/java`.

Note. The value for this variable is set by the `setup` script. Do not modify this value.

[Example 3-4](#) shows a sample configuration of the `server_objectcode` variable.

Example 3-4. A Sample Configuration of the `server_objectcode` Variable

```
#
# The server_objectcode represents a SYMBOLIC LINK to your installed
# copy of the JVM (default /usr/tandem/java/bin/java)
#
set server_objectcode $root/bin/instA.ssc
```

NSJSP_HOME

The `NSJSP_HOME` variable is used to set the full path of the directory where NSJSP is installed.

Note. The value for this variable is set by the `setup` script. Do not modify this value.

[Example 3-5](#) shows the default value of the `NSJSP_HOME` variable, where the iTP Secure WebServer is installed in the default location `/usr/tandem/webserver`, and that the default NSJSP installation directory is used, which is `servlets`.

Example 3-5. The Default Value of the `NSJSP_HOME` Variable

```
#
# The NSJSP installation directory.
#
set env(NSJSP_HOME) /usr/tandem/webserver/servlets
```

JVM_POLICY_FILE

The `JVM_POLICY_FILE` variable specifies the security policy file that needs to be used if NSJSP is run with a security manager. The default security policy file in NSJSP is `<NSJSP_HOME>/conf/iTP_catalina.policy`. The `JVM_POLICY_FILE` variable is used to set the value of the `NSJSP_SECMGR_POLICY` variable to indicate the security policy file. For more information about the security policy file and syntax of the file, see Chapter [7](#), [Migrating to NSJSP 6.1](#).

[Example 3-6](#) shows the default value of the `JVM_POLICY_FILE` variable.

Example 3-6. The Default Value of the `JVM_POLICY_FILE` Variable

```
set  env(JVM_POLICY_FILE)  $env(NSJSP_HOME)/conf/itp_catalina.policy
```

`NSJSP_SECMGR_POLICY`

The value of this variable is the Java command-line argument that is used to indicate a security policy file. The value reads `-Djava.security.policy==$env(JVM_POLICY_FILE)`. The `==` sign is intentional and indicates to the JVM that it must only use the mentioned policy file.

[Example 3-7](#) shows the default value of the `NSJSP_SECMGR_POLICY` variable.

Example 3-7. The Default Value of the `NSJSP_SECMGR_POLICY` Variable

```
set  NSJSP_SECMGR_POLICY
      -Djava.security.policy==$env(JVM_POLICY_FILE)
```

`NSJSP_SECMGR`

The value of `NSJSP_SECMGR` variable is the Java command-line argument that is used to enable the Java Security Manager. The default value of this variable is `-Dnsjsp.security.manager=none` and by default the Java Security Manager is not enabled. If this variable is set to `-Djava.security.manager`, the security manager is enabled.

Note. Based on the Java Security Specification, a security policy file must be used only if Java is configured to run with the security manager enabled. The `NSJSP_SECMGR_POLICY` variable is effective only if the Java Security Manager has been enabled using the `NSJSP_SECMGR` variable.

[Example 3-8](#) shows the default value of the `NSJSP_SECMGR` variable.

Example 3-8. The Default Value of the `NSJSP_SECMGR` Variable

```
#
#  If you wish to run with a security Manager, uncomment the next
#  statement ("set  NSJSP_SECMGR  ...").
#
#  set  NSJSP_SECMGR  -Djava.security.manager
```

`JAAS_CONFIG_FILE`

The `JAAS_CONFIG_FILE` variable specifies the location of Java Authentication and Authorization Service (JAAS) configuration file. The default JAAS configuration file in NSJSP is `<NSJSP_HOME>/conf/itp_jaas.config`. The `JAAS_CONFIG_FILE` variable is used by the `NSJSP_JAAS_CONFIG` variable to

obtain the location of the JAAS configuration file. For more information on JAAS, see <http://java.sun.com/javase/6/docs/guide/security/jaas/JAASRefGuide.html>.

[Example 3-9](#) shows the default value of the `JAAS_CONFIG_FILE` variable.

Example 3-9. The Default Value of the `JAAS_CONFIG_FILE` Variable

```
#
# NSJSP JAAS NonStopLoginModule configuration file.
#
set  env(JAAS_CONFIG_FILE)  $env(NSJSP_HOME)/conf/itp_jaas.config
```

`NSJSP_JAAS_CONFIG`

The value of the `NSJSP_JAAS_CONFIG` variable is the Java command-line argument that is used to turn on the JAAS module. This enables the authentication and authorization procedure to be handled as per the definitions in the JAAS config file that is specified by the variable `JAAS_CONFIG_FILE`. For more information on JAAS configuration, see [Realms](#) on page 7-7.

[Example 3-10](#) shows the default value of the `NSJSP_JAAS_CONFIG` variable.

Example 3-10. The Default Value of the `NSJSP_JAAS_CONFIG` Variable

```
#
# By default, the JVM is run without a JAAS configuration file.
#
set  NSJSP_JAAS_CONFIG  -Dnsjsp.jaas.login.config=none

#
# If you wish to use the NSJSP JAAS NonStopLoginModule, uncomment
# the
# next statement ("set  NSJSP_JAAS_CONFIG  ...").
#
# Note: the "double" equalto signs "==" is not a typo!! This informs
# the # JVM to use this file exclusively and that all others are to be
# ignored.
#
# set  NSJSP_JAAS_CONFIG
# -Djava.security.auth.login.config==$env(JAAS_CONFIG_FILE)
```

`JAVA_HOME`

The `JAVA_HOME` variable is set to the location where NonStop Server for Java (NSJ) is installed. The value of the `JAVA_HOME` variable is set when the `setup` script is run. After installing NSJSP, do not modify the value of this variable unless the NSJ installation is moved to a different directory.

[Example 3-11](#) shows the default value of the `JAVA_HOME` variable.

Example 3-11. The Default Value of the `JAVA_HOME` Variable

```
#####
#
# Set the JAVA_HOME and GUARDIAN_SUBVOL
#
set env(JAVA_HOME) /nsjsp/software/java6.0/java

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

JVCP

The `JVCP` variable includes additions for the Java classpath and is used to set the `CLASSPATH` variable. Do not modify the `jar` files listed in this variable. Use the `USRCP` variable to identify any installation-specific `jar` files that also need to be added to the `CLASSPATH` variable.

[Example 3-12](#) shows the default value of the `JVCP` variable.

Example 3-12. The Default Value of the `JVCP` Variable

```
#####
#
# The following code sets up the Java Specific class PATH variables
# into
# a Tcl variable called JVCP. JVCP will be prepended to the
# classpath of
# the JVM.
#
#
set JVCP ""
append JVCP $env(NSJSP_HOME) "/bin/bootstrap.jar:"
append JVCP $env(NSJSP_HOME) "/bin/commons-daemon.jar:"
append JVCP $env(NSJSP_HOME) "/bin/tomcat-juli.jar:"
append JVCP $env(NSJSP_HOME) "/bin/ns-logger.jar:"
append JVCP $env(NSJSP_HOME) "/bin/nsjspmbeanserver.jar:"

#
# This trims off the rightmost colon char.
#
set JVCP [string trimright $JVCP ":"]
```

USRCP

By default the `USRCP` variable is empty. This variable can be used to include any installation-specific `jar` files that need to be added to the `CLASSPATH`. If the `USRCP` variable is not empty, the format of this variable must be the same as for `JVCP`. For example, `<full path to jar1>:<full path to jar2>:...`

[Example 3-13](#) shows the default value of the `USRCP` variable.

Example 3-13. The Default Value of the `USRCP` Variable

```
#
# Set the default value of the User ClassPath blank
#
set USRCP ""
```

`SERVLET_BANK`

The `SERVLET_BANK` variable is used to specify the name of the required database table in the format `<catalog>.<schema>.<tablename>` used in example bank application distributed with NSJSP. If the table does not exist, the bank application uses this variable to create the table at run time. The `SERVLET_BANK` variable can be deleted if the bank application is removed.

The bank application is installed by default. The application name is `bankapp.war`. This application is located in `<NSJSP_HOME>/webapps`.

Note. The bank application creates (if it does not already exist) the table specified by the `SERVLET_BANK` environment variable. If NSJSP is started by a super group, the application may attempt to create the table on the `$SYSTEM` volume. Normally the `$SYSTEM` volume is not audited by Transaction Management Facility (TMF). In that case, the table creation will fail and the sample bank application will not work.

The bank application does not create a new table if it already exists. To prevent an attempt to create a table on `$SYSTEM` volume, HP recommends that you create the table manually. For example, while creating a table in SQL/MX, you can use the `LOCATION` clause to specify the volume on which the table is created. A sample SQL/MX script to create the required table for bank application is as shown below:

```
create catalog bankcat;
set catalog bankcat;

create schema banksch;
set schema banksch;

create table bankcat.banksch.bankdata (
    acctno      smallint unsigned not null,
    lastname    char(20)           not null,
    firstname   character(20)      not null,
    city        character(20)      not null,
    balance     int                not null,
    primary     key(acctno)) location $DATA01;
```

The script to create the table for the sample bankapp application is `<NSJSP_HOME>/conf/bankapp_mx_ddl.sql`. Before running the script, ensure that you replace suitable values for `catalog`, `schema`, and `tablename`.

[Example 3-14](#) shows the default value of the `SERVLET_BANK` variable.

Example 3-14. Default Value of the `SERVLET_BANK` Variable

```
#####
#
# Related to the BankDemo example application provided with NSJSP
#
# The value of SERVLET_BANK should be <catalog>.<schema>.<tablename>
# Catalog and schema should have been created. The application will
# create the given table if one is not present

set SERVLET_BANK bankcat.banksch.sraccts
```

`NSJSP_DLL_PATH`

The value of the `NSJSP_DLL_PATH` variable contains the directory where the NSJSP native libraries are located. This variable can be used to include other directories where application specific native libraries are located. The value of the `NSJSP_DLL_PATH` variable is used to set the `_RLD_LIB_PATH`. For more information on the `_RLD_LIB_PATH`, see the *RLD Manual*.

[Example 3-15](#) shows the default value of the `NSJSP_DLL_PATH` variable.

Example 3-15. Default Value of the `NSJSP_DLL_PATH` Variable

```
#####
#
# NSJSP SHARED RUNTIME LIBRARY (SRL) location
#

set NSJSP_DLL_PATH $env(NSJSP_HOME)/lib
```

Server Class Configuration using the `server` Directive

The `Server` directive is used to configure the Servlet Server Class. This section describes the Servlet Server Class configuration. For more information about the `Server` directive, see the *ITP Secure WebServer System Administrator's Guide*.

The following environment variables are defined in the `Server` directive:

`CLASSPATH`

The `CLASSPATH` variable contains the Java classpath. The classpath is defined by the `JVCP` and `USRCP` variables. This environment variable is mandatory and is used by the JVM when the server class is started. For more information on the `CLASSPATH`, see the *NonStop Server for Java Reference Manual*.

[Example 3-16](#) shows the default value of the `CLASSPATH` variable.

Example 3-16. The Default Value of the `CLASSPATH` Variable

```
Env CLASSPATH=$JVCP:$USRCP
```

JAVA_HOME

The `JAVA_HOME` variable is mandatory and specifies the directory where NonStop Server for Java (NSJ) is installed.

[Example 3-17](#) shows the default value of the `JAVA_HOME` variable.

Example 3-17. The Default Value of the `JAVA_HOME` Variable

```
Env  JAVA_HOME=$env(JAVA_HOME)
```

JREHOME

The `JREHOME` variable is mandatory and specifies the location of the Java Runtime Environment (JRE).

Note. This variable is not mandatory in NSJ 6.0 and later versions.

[Example 3-18](#) shows the default value of the `JREHOME` variable.

Example 3-18. The Default Value of the `JREHOME` Variable

```
Env  JREHOME=$env(JAVA_HOME)/jre
```

_RLD_LIB_PATH

The `_RLD_LIB_PATH` variable is mandatory and specifies the location of the required native libraries.

[Example 3-19](#) shows the default value of the `_RLD_LIB_PATH` variable.

Example 3-19. The Default Value of the `_RLD_LIB_PATH` Variable

```
Env  _RLD_LIB_PATH=$NSJSP_DLL_PATH
```

TANDEM_FILEMAPS_CONFIG

The `TANDEM_FILEMAPS_CONFIG` variable contains the location of the `filemaps.config` file. This environment variable is mandatory. For more information on the `filemaps.config` file, see [The filemaps.config File](#) on page 3-33.

[Example 3-20](#) shows the default value of the `TANDEM_FILEMAPS_CONFIG` variable.

Example 3-20. The Default Value of the `TANDEM_FILEMAPS_CONFIG` Variable

```
Env  TANDEM_FILEMAPS_CONFIG=$env(NSJSP_HOME)/conf/filemaps.config
```

BANK_CATALOG

The `BANK_CATALOG` variable is set to the value of the `$SERVLET_BANK` variable. For more information, see [SERVLET_BANK](#) on page 3-16.

[Example 3-21](#) shows the default value of the `BANK_CATALOG` variable.

Example 3-21. The Default Value of the `BANK_CATALOG` Variable

```
Env BANK_CATALOG=$SERVLET_BANK
```

NSJSP_CONFIG_FILE

Specifies the NSJSP configuration file that is used by the NSJSP Manager. This variable is only present in the NSJSP Manager server class configuration file. For more information, see Chapter 4, [Managing NSJSP](#).

[Example 3-22](#) shows the default value of the `NSJSP_CONFIG_FILE` variable.

Example 3-22. The Default Value of the `NSJSP_CONFIG_FILE` Variable

```
Env NSJSP_CONFIG_FILE=$env(NSJSP_HOME)/conf/nsjsp_manager.config
```

TANDEM_RECEIVE_DEPTH

The `TANDEM_RECEIVE_DEPTH` variable is mandatory and it must have a numeric value. This variable indicates the maximum number of requests that a single server class process handles simultaneously. For more information on how this affects the behavior of NSJSP, see the *NonStop Servlets for JavaServer Pages (NSJSP) Configuration White Paper*.

[Example 3-23](#) shows the default value of the `TANDEM_RECEIVE_DEPTH` variable.

Example 3-23. The Default Value of the `TANDEM_RECEIVE_DEPTH` Variable

```
Env TANDEM_RECEIVE_DEPTH=50
```

The following command-line arguments are passed to the JVM when the server class is started. The arguments are specified using the `Arglist` command. The command-line arguments include the arguments defined by NSJ and NSJSP. Any command-line argument valid in NSJ can also be used in NSJSP.

Xmx

Specifies the maximum size, in bytes, of the JVM memory allocation pool. The default value is 64m, which is sufficient to run an application like the Java Pet Store. This is enough memory to handle about 50 concurrent requests in a single instance of server class. You can change this number to meet the requirements of applications that are deployed in NSJSP.

Xms

Specifies the initial memory allocation pool (also called the heap) size that is allocated when NSJSP starts up. This value is set to the same value as `Xmx`.

Xss

Specifies the stack size. The default value is `128k`. This value should be sufficient for most applications hosted on NSJSP. If you experience stack overflow exceptions in your application, consider increasing this value.

Xnoclassgc

Disables the class garbage collection. This argument is specified by NSJ.

When you specify `-XnoClassGC` in the `Arglist` of `servlet.config` file in `<NSJSP_HOME>/conf` directory, the class objects in the web applications are left as it is during garbage collection, and are always considered to be active. Turning off the garbage collection class eliminates the overhead of loading and unloading the same class multiple times. If we want to periodically check and reduce the consumption of the heap memory, `-XnoClassGC` can be removed from the `Arglists` of `servlet.config` file. However, this creates a small overhead of running the garbage collection mechanism to free the heap space occupied.

[Example 3-24](#) shows the default `Xmx`, `Xms`, `Xss`, and `Xnoclassgc` variables.

Example 3-24. The Default `Xmx`, `Xms`, `Xss`, and `Xnoclassgc` Variables

```
Arglist -Xmx64m -Xms64m -Xss128k -Xnoclassgc
```

java.compiler

The `java.compiler` argument is not set in NSJSP 6.1 for the following reason:

In releases prior to NSJ 4.2 where the Just In Time (JIT) compiler was used, the `java.compiler` argument was used to disable JIT compilation. Starting with the NSJ 4.2 release, the JIT compiler is replaced with the HotSpot compiler. However, for backward compatibility, the `java.compiler` argument is still supported. If you set the value to `none`, the HotSpot compiler is disabled, which is the equivalent of setting the `-Xint` argument. HP recommends that you must not use the `java.compiler` argument in order to allow the JVM to use the default value for the HotSpot compiler.

java.util.logging.manager

Specifies the `LogManager` that must be used for all logging purposes. Do not modify this value unless you want to implement a new logging infrastructure. For more information about logging, see Chapter [5, Logging in NSJSP](#). For more information about the Java logging infrastructure, see <http://java.sun.com/javase/6/docs/technotes/guides/logging/index.html>

[Example 3-25](#) shows the default value of the `java.util.logging.manager` variable.

Example 3-25. The Default Value of the `java.util.logging.manager` Variable

```
-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager
```

`java.util.logging.config.file`

Specifies the configuration file used by the LogManager. For more information about the logging configuration file, see Chapter [5, Logging in NSJSP](#). For more information about the Java logging infrastructure, see

<http://java.sun.com/javase/6/docs/technotes/guides/logging/index.html>

[Example 3-26](#) shows the default value of the `java.util.logging.config.file` variable.

Example 3-26. The Default Value of the `java.util.logging.config.file` Variable

```
-Djava.util.logging.config.file=
    $env(NSJSP_HOME)/conf/logging.properties
```

`javax.management.builder.initial`

Specifies the NonStop-specific MBean server builder class that NSJSP provides to create an MBean server to administer NSJSP. The default value is `com.tandem.servlet.jmx.NSJSPMBeanServerBuilder`. This is used by the NSJSP admin application. Do not modify the default value.

For more information on MBean and the JMX technology, see <http://java.sun.com/javase/6/docs/technotes/guides/jmx/overview/JMXoverviewTOC.html>

[Example 3-27](#) shows the default value of the `javax.management.builder.initial` variable.

Example 3-27. The Default Value of the `javax.management.builder.initial` Variable

```
-Djavax.management.builder.initial=
    com.tandem.servlet.jmx.NSJSPMBeanServerBuilder
```

`java.io.tmpdir`

Specifies the location of the `temp` directory. The value can be changed to any other directory. The directory must be created before the Servlet Server Class is started. The `temp` directory is used internally by the NSJSP servlet container.

[Example 3-28](#) shows the default value of the `java.io.tmpdir` variable.

Example 3-28. The Default Value of the `java.io.tmpdir` Variable

```
-Djava.io.tmpdir=$env(NSJSP_HOME)/temp
```

`catalina.home` and `catalina.base`

The `catalina.home` and `catalina.base` arguments are mandatory and specify the installation directory for NSJSP. The values for these variables must be the same. The `catalina.home` argument is used to identify the directory containing the common libraries for all the applications in an NSJSP environment. The `catalina.home` argument must always point to a directory containing `lib` and `bin` directories which include all the container libraries that come with a default installation of NSJSP. The `catalina.base` directory is used to identify the directory that contains `conf`, `logs`, `temp`, `webapps`, and `work` directories. These directories are specific to the container configuration and the web applications.

Note. HP recommends that both `catalina.home` and `catalina.base` should always point to the same directory location. The `update` option of the setup script assumes that an entire NSJSP installation is within a single directory and not split across two directories.

[Example 3-29](#) shows the default values for `catalina.home` and `catalina.base` variables.

Example 3-29. The Default Values for `catalina.home` and `catalina.base` Variables

```
-Dcatalina.home=$env(NSJSP_HOME)
-Dcatalina.base=$env(NSJSP_HOME)
```

SessionBasedLoadBalancing

Specifies a NonStop-specific argument. This value indicates whether sessions must be considered as sticky sessions. Sticky sessions have an affinity to the process (a `ServerClass` instance) where the session was created and all requests within that session will always go to the same process. The argument can have two values `true` or `false`. The default value is `true`. If this variable is not set in the installation-specific `servlet.config` file, the value is considered to be `true`.

For more information on how this parameter effects the behavior of NSJSP, see [Session Management](#) on page 3-75.

SessionBasedCookieExpiry

Specifies a NonStop-specific argument. The default value is `false`. If this value is set to `true`, the `JSESSIONID` cookie has the same expiry date as the corresponding session object. If the value is `false`, the cookie persists in the web

browser as long as the browser is open. If sessions are configured to be persistent and to expire with short durations, then setting this attribute to `true` might improve performance.

When setting this value to `true`, it is important that the date and time on the browser workstation match or is very close to that of the NonStop server where NSJSP is running. If the time are greatly out of synchronization, the cookie in the browser may expire before the corresponding session expires.

[Example 3-30](#) shows a sample configuration of the `SessionBasedCookieExpiry` variable.

Example 3-30. A Sample Configuration of the `SessionBasedCookieExpiry` Variable

```
-DSessionBasedCookieExpiry=false
```

`com.tandem.servlet.nsjsp`

Specifies a NonStop-specific argument. This argument indicates whether the running process is an NSJSP process. The default value is `true`. If the value is set to `false`, NSJSP exhibits only Apache Tomcat behavior. Although NSJSP is based on Apache Tomcat, the default behavior of some of the features in NSJSP is different from that of Apache Tomcat as shown below.

Default session manager

NSJSP	<code>com.tandem.servlet.catalina.NSJSPStandardManager</code>
Apache Tomcat	<code>org.apache.catalina.session.StandardManager</code>

Default context configuration class

NSJSP	<code>com.tandem.servlet.catalina.startup.NSJSPContextConfig</code>
Apache Tomcat	<code>org.apache.catalina.startup.ContextConfig</code>

HTTP protocol processor

NSJSP	<code>com.tandem.servlet.coyote.http11.NSJSPHttp11Processor</code>
Apache Tomcat	<code>org.apache.coyote.http11.Http11Processor</code>

Attribute: port and shutdown

NSJSP	Not applicable
Apache Tomcat	Used in Apache Tomcat

It is recommended that the `com.tandem.servlet.nsjsp` variable is set to `true`. Setting the variable to `false` might require additional changes to the configuration files, such as `server.xml`, because the default Tomcat behavior might require different configuration parameters than NSJSP.

Note. To get pure Tomcat behavior, you can install Apache Tomcat on a NonStop server.

DiscardFileMapHistory

Preserves or discards the history of all changes to the Filemaps. The default value is `false`.

Filemap changes are made by the NSJSP Manager application when deploying and undeploying web applications. If the `DiscardFileMapHistory` parameter is set to `true`, and if an existing Filemap definition for a URI needs to be overwritten, the existing Filemap definition is first deleted and then a new definition is inserted. If the `DiscardFileMapHistory` parameter is set to `false`, the old definition is retained as a comment and the new definition is added.

[Example 3-31](#) shows an example of a new Filemap definition being added when `DiscardFileMapHistory` is `true`.

Note. HP recommends that you retain the default value of `false` to preserve the Filemap history.

Example 3-31. New Filemap Information

```
#
# 2010-04-08 14:47:10.765 - Added Filemap '/bank'.
Filemap /bank /usr/tandem/webserver/bin/dev.ssc
```

[Example 3-32](#) shows an example of retaining the previous Filemap definition when `DiscardFileMapHistory` is `false`.

Example 3-32. Filemap History Available as a Comment

```
#
# 2010-04-08 14:47:10.765 - Added Filemap '/bank'.
# Filemap /bank /usr/tandem/webserver/bin/dev.ssc

#
# 2010-04-09 15:23:11.123 - Added Filemap '/bank'.
Filemap /bank /usr/tandem/webserver/bin/prod.ssc
```

EnableJMXProxyServlet

In NSJSP 6.0 and later versions, the JMX Proxy Servlet is either enabled or disabled in the deployment descriptor (`web.xml`) of the old manager application. This argument was used in NSJSP 5.0 to enable or disable the JMX Proxy Servlet. This argument must not be used in NSJSP 6.1. In NSJSP 6.1, the JMX Proxy

Servlet is enabled by default. The `JMXProxy` is defined in the old manager application's deployment descriptor (`web.xml`). [Example 3-33](#) shows the configuration for JMX Proxy Servlet in the `web.xml` file.

Example 3-33. Configuration for `JMXProxyServlet` in the `web.xml` file

```
...
...
    <servlet>
        <servlet-name>JMXProxy</servlet-name>
        <servlet-
            class>org.apache.catalina.manager.JMXProxyServlet</servlet-class>
    </servlet>
...
...
    <servlet-mapping>
        <servlet-name>JMXProxy</servlet-name>
        <url-pattern>/jmxproxy/*</url-pattern>
    </servlet-mapping>
...
...
```

The NSJSP Manager GUI provides features to query, compare, and modify MBean attributes across the NSJSP server class processes. Although the JMX Proxy Servlet is available in NSJSP 6.1, HP recommends that you use the NSJSP Manager GUI for MBean operations.

While the JMX Proxy Servlet provides features to modify MBean attributes across NSJSP server class processes in a single PATHMON, the NSJSP Manager GUI provides similar functionality across NSJSP server class processes running across multiple PATHMONS.

The JMX Proxy Servlet can be used in scripts to manage NSJSP Mbeans. However, the NSJSP Manager GUI cannot be used in scripts. Therefore, if you are using scripts to manage NSJSP MBeans, you can continue to use the JMX Proxy servlet in NSJSP6.1 to execute those scripts.

SaveSessionOnCreation

Enables or disables saving sessions into a persistent store during creation time.

The default value is `false`. This is to prevent saving a session before it is modified by the user application. This also reduces the number of database operations for each client request.

If both the `SaveSessionOnCreation` and the `SessionBasedLoadBalancing` options are set to `false`, and if a persistent manager is configured with a persistent store, then all sessions are written to the store at the end of each request processing cycle. As a result, all changes made to the session by the user application are persisted to the store.

[Example 3-34](#) shows the default value of the `SaveSessionOnCreation` variable.

Example 3-34. The Default Value of the `saveSessionOnCreation` Variable

```
-DSaveSessionOnCreation=false
```

`com.tandem.servlet.CONTEXT_PREFIXES`

This is the URI name that is specified during installation when you run the setup script. The value must begin with `/` followed by the URI name, which is made up of alphanumeric characters. This value does not support multiple `/` characters. For more information about the URI name, see Chapter [2, Installing NSJSP](#).

Although, by default, all applications have a context name which follows the value of the `com.tandem.servlet.CONTEXT_PREFIXES` parameter, it is possible to configure different ways to access an application.

For example, if the value of this parameter is `/dev` and the application `foo.war` is deployed, by default, the context path for this application will be `/dev/foo`.

Therefore, the application can be accessed using the URL

`http://<ipaddress:port>/dev/foo`.

The default behavior of the NSJSP servlet container can be overridden in two ways:

- Deploy applications using the NSJSP Manager and check the Automatically Add Filemap option. For more information on NSJSP Manager, see Chapter [4, Managing NSJSP](#).
- Deploy applications using the Client Deployer and set `appendContext` to `false`. For more information on Client Deployer, see Chapter [4, Managing NSJSP](#).

The NSJSP Manager and the Client deployer each provide an option to deploy an application without requiring the value of this parameter. Using the example quoted above, it is possible to deploy the `foo.war` application such that it can be accessed using the URL `http://<ipaddress:port>/foo`. In this case, the context path of the application will be `/foo` and not `/dev/foo`.

`maxWaitTimeSecs`

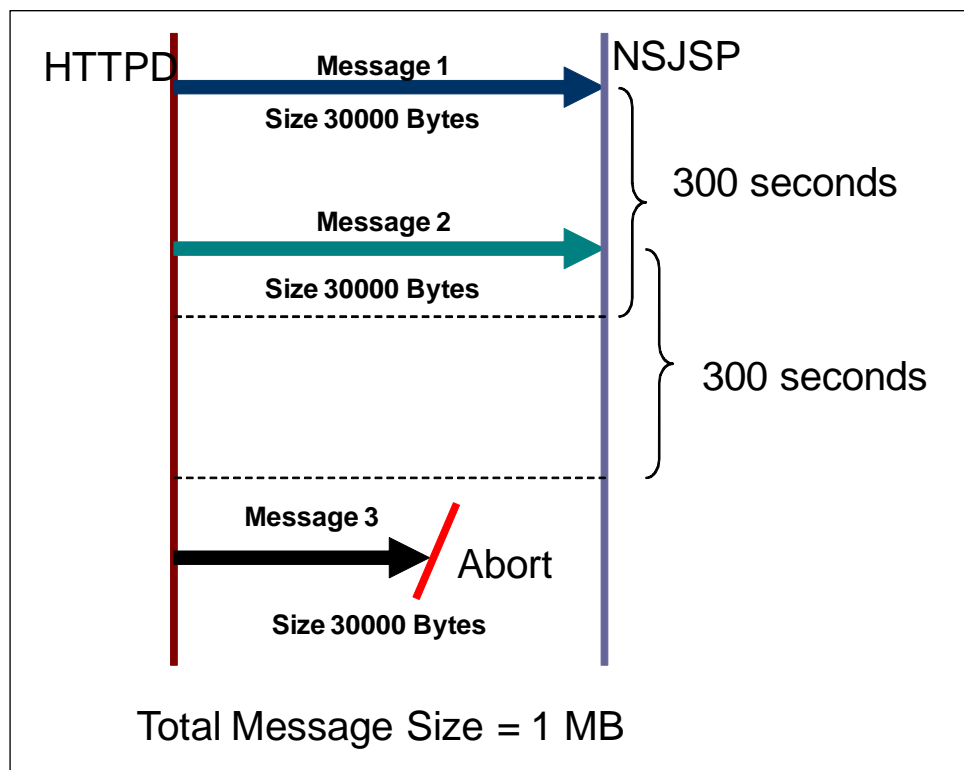
Any message greater than 30000 bytes is referred to as a big message. The value of `maxWaitTimeSecs` is relevant when the size of the message transfer between NSJSP and HTTPD is big and requires exchanging multiple messages between HTTPD and NSJSP. The value of `maxWaitTimeSecs` is a number indicating the number of seconds that the NSJSP Connector should wait before aborting a message exchange between HTTPD and NSJSP. For more information on message exchange between HTTPD and NSJSP, see the *NonStop Servlets for JavaServer Pages (NSJSP) Configuration White Paper*.

Note. HP recommends that you set this value to a value greater than or equal to the `ScriptTimeout` value specified in the *iTP Secure WebServer* configuration.

Note. If a negative value is set, and if there are multiple message exchanges between NSJSP and HTTPD, the NSJSP Connector aborts the message exchange if HTTPD does not respond to the message exchange immediately. Therefore, the value must not be negative since it disables big message transfers by exchanging multiple messages between NSJSP and HTTPD. For more information about `ScriptTimeout` see the *iTP Secure WebServer Administrator's Guide*.

[Figure 3-2](#) illustrates the maximum wait time for a message with a total size of 1 MB.

Figure 3-2. Maximum Wait Time for a Message with Size = 1MB



[Figure 3-2](#) shows three big messages being transferred between HTTPD and NSJSP. These messages are message 1, message 2, and message 3. Message 2 is transferred within the 300 second wait period after sending message 1 and hence transferred successfully. Whereas, message 3 exceeding the 300 second wait period aborts.

The following are the configuration parameters defined by TS/MP for server classes. For more information on these parameters, see the *NonStop TS/MP System Management Manual*. The following descriptions are applicable to NSJSP only.

NUMSTATIC and MAXSERVERS

In the default configuration, both these parameters have the same value. This means there are no dynamic servers are configured for NSJSP. HP recommends that you do not configure dynamic servers. This is because, the long startup time for NSJSP processes might lead to requests getting queued, which might also result in application timeouts. For more information on `NUMSTATIC` & `MAXSERVERS` parameters, see the *NonStop Servlets for JavaServer Pages (NSJSP) Configuration White Paper*.

stdout and stderr

Specifies the standard out and standard error files respectively. The JNI library of NSJSP writes messages to these files. Although most of the messages are written during startup, errors encountered on operations on with `$RECEIVE` are written to the standard error file during run time. Based on the message type, log entries created by NSJ are also written to either the `Stdout` file or the `Stderr` file.

MAXLINKS and LINKDEPTH

The `MAXLINKS` and `LINKDEPTH` parameters are each set to 50 by default. These values may require tuning, depending upon the NSJSP request load and the number of CPUs in the HP NonStop system. For improved load balancing and better response times, smaller values for both parameters may provide better results. For information on `MAXLINKS` and `LINKDEPTH` parameters, see the *NonStop Servlets for JavaServer Pages (NSJSP) Configuration White Paper*.

Miscellaneous

The Region Directive

The `Region` directive is used to prevent access to the Servlets and JSPs through the `Invoker Servlet`. The `Invoker Servlet` allows applications to dynamically register new servlets and to invoke a servlet using the fully qualified servlet class name. Allowing such servlet invocations might lead to security problems. For more information on the `Invoker Servlet`, see <http://tomcat.apache.org/tomcat-6.0-doc/funcspecs/fs-invoker.html>.

[Example 3-35](#) shows the default configuration of the Region directive.

Example 3-35. The Default Configuration of the Region Directive

```
#
# Add region commands to deny access to internal Catalina Servlets.
#
Region */servlet/org.apache.catalina.servlets.* {
DenyHost  *
}

Region */servlet/jsp/* {
DenyHost  *
}
```

The Filemap Directive

The Filemap directive at the end of the `servlet.config` file indicates to the iTP Secure WebServer that all requests whose URI matches the filemap definition must be directed to instances of the specified server class. The value of the `$server_objectcode` variable is the fully qualified object file name, which includes the server class name.

[Example 3-36](#) shows the default configuration of the Filemap directive for an NSJSP installation with a URI name of `sca3url`.

Example 3-36. The Default Configuration of the Filemap Directive

```
#
# The filemap directive should always map to wherever you have installed
# the NonStop(tm) Servlets For JavaServer Pages(tm) server object code.
#
Filemap /sca3url $server_objectcode
```

The nsjspadmin.config File

[Table 3-6](#) provides an overview of the nsjspadmin.config file.

Table 3-6. Overview of the nsjspadmin.config File

Location	<NSJSP_HOME>/conf
Description	Contains the configuration attributes for the Admin Server Class. The name of this server class is created by adding <code>-adm</code> to the servlet server class name, which is specified during the installation process. For example, if you entered the server class name during installation as <code>production</code> , the name of this server class will be <code>production-adm</code> . The <code>nsjspadmin.config</code> file, if present, is included in the installation-specific <code>servlet.config</code> file using the <code>source</code> directive. Only environment variables defined in the installation-specific <code>servlet.config</code> file are also valid in the <code>nsjspadmin.config</code> file.
Recommendation	You need not change the parameters in this file. This is because, the Admin Server Class will only be used to access the Admin application only.

[Example 3-37](#) shows the sample nsjspadmin.config file.

Example 3-37. The nsjspadmin.config File

```

#
#  VERSION=6.1.0
#
#
#  The nsjspadmin_objectcode represents a SYMBOLIC LINK to your installed
#  copy of the JVM (default /usr/tandem/java/bin/java)
#
set nsjspadmin_objectcode $root/bin/instA.ssc

Server $nsjspadmin_objectcode {
    CWD $env(NSJSP_HOME)
    Env CLASSPATH=$JVCP:$USRCP
    Env JAVA_HOME=$env(JAVA_HOME)
    Env JREHOME=$env(JAVA_HOME)/jre
    Env _RLD_LIB_PATH=$NSJSP_DLL_PATH
    Env TANDEM_HTTPD_SC_NAME=HTTPD
    Env TANDEM_SERVLET_SC_NAME=sca
    Env TANDEM_SERVLET_SC_PATH=$server_objectcode
    Env TANDEM_FILEMAPS_CONFIG=
        $env(NSJSP_HOME)/webserver/conf/filemaps.config
    MapDefine =TCPIP^PROCESS^NAME $transport
    Maxservers 1
    Numstatic 1
    Maxlinks 25
#
# Check that the Linkdepth and TANDEM_RECEIVE_DEPTH parameter values
# match.
#
# The value of TANDEM_RECEIVE_DEPTH should be equal to or less than
# the maxThreads attribute in the connector element configured in
# $NSJSPS_HOME/conf/server.xml.
#
    Linkdepth 25
    Env TANDEM_RECEIVE_DEPTH=25
#
# File locations to direct standard input, output and error.
#
    Stdin /dev/null
    Stdout $env(NSJSP_HOME)/logs/nsjspadmin.out
    Stderr $env(NSJSP_HOME)/logs/nsjspadmin.err
#
# This is the actual Arglist used to start up the NSJSPAdmin Container.
#
    Arglist -Xmx64m -Xss128k -Xnoclassgc \
-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager \
-Djava.util.logging.config.file=$env(NSJSP_HOME)/conf/logging.properties \
-Djavax.management.builder.initial=com.tandem.servlet.jmx.NSJSPMBeanServerB
uilder \
    $NSJSP_SECMGR \
    $NSJSP_SECMGR_POLICY \
    $NSJSP_JAAS_CONFIG \
    -Dcom.tandem.servlet.CONTEXT_PREFIXES=/sca \
    -Dcom.tandem.servlet.nsjspadmin=true \
    -Dcatalina.home=$env(NSJSP_HOME) \
    -Dcatalina.base=$env(NSJSP_HOME) \
    -Djava.io.tmpdir=$env(NSJSP_HOME)/temp \
    org.apache.catalina.startup.Bootstrap start
}
Filemap /sca/admin $nsjspadmin_objectcode
Filemap /sca/manager $nsjspadmin_objectcode

```

[Example 3-38](#) shows the installation-specific `servlet.config` file checking for the presence of the `nsjspadmin.config` file and sourcing the `filemaps.config` file, if available.

Example 3-38. Checking for the presence of the `nsjspadmin.config` file and sourcing the `nsjspadmin.config` file in the `servlet.config` file

```
#####
#
# Do an existence check on the nsjspadmin configuration
# file (nsjspadmin.config). If it exists, source it in.
#
if { [file exists $env(NSJSP_HOME)/conf/nsjspadmin.config] } {
source $env(NSJSP_HOME)/conf/nsjspadmin.config
}
```

The following configuration parameters are specific to `nsjspadmin.config` file.

TANDEM_HTTPD_SC_NAME

This is a Server Class environment variable that indicates the server class name of the HTTPD component in the iTP Secure WebServer environment where this Servlet Server Class is configured. The Server Class for the HTTPD component is configured (in the default configuration) in the `httpd.config` file. The default value is HTTPD.

TANDEM_SERVLET_SC_NAME

This is the name of the Servlet Server Class that must be monitored by this Admin Server Class. For more information on monitoring NSJSP using the Admin Server Class, see Chapter [4, Managing NSJSP](#).

TANDEM_SERVLET_SC_PATH

This is the full path to the object file used in the Servlet Server Class. For more information about the object file, see [server_objectcode](#) on page 3-11. This value is set during installation and should not be modified manually.

NUMSTATIC and MAXSERVERS

These are parameters whose value must always equal the number of PATHMONs configured in the iTP Secure WebServer `httpd.config` configuration file so that there is only one instance of this server class in a PATHMON. The value is set during installation and must not be modified manually.

Filemaps

The Filemap directives at the end of the `nsjspadmin.config` file indicate that all requests to both the `admin` and the `old manager` are also directed to the `admin server` class.

Note. The `manager` application mentioned here is different from the new NSJSP Manager. For more information on the manager applications, see Chapter [4, Managing NSJSP](#).

The filemaps.config File

The Filemap directive is defined by ITP Secure WebServer and is described in detail in the *ITP Secure WebServer System Administrator's guide*. This section describes information about the `filemaps.config` file that is specific to NSJSP.

[Table 3-7](#) provides an overview of the `filemaps.config` file.

Table 3-7. Overview of the filemaps.config File

Location	<NSJSP_HOME>/conf
Description	This file is sourced in by the <code>servlet.config</code> file using the <code>source</code> directive.

[Example 3-39](#) shows the default `filemaps.config` file.

Example 3-39. The filemaps.config File

```
#This file should contain all the NSJSP specific filemaps
#The file will be read by the NSJSP SERVLET and NSJSPADMIN instances
#By default filemaps for manager and admin applications are
#mentioned in ITPWS/conf/nsjspadmin.config.
#Any changes to the default filemap definitions should be made
#in this file. If changes are made then the default definitions
#in nsjspadmin.config should be removed.
```

[Example 3-40](#) shows the installation-specific `servlet.config` file checking for the presence of the `filemaps.config` file and sourcing the `filemaps.config` file, if available.

Example 3-40. Checking for the presence of the filemaps.config file and sourcing the filemaps.config file in the servlet.config file

```
#####
#
# Do an existence check on the filemaps configuration
# file (filemaps.config). If it exists, source it in.
#
if { [file exists $env(NSJSP_HOME)/conf/filemaps.config] } {
source $env(NSJSP_HOME)/conf/filemaps.config
}
```

The `filemaps.config` file is only meant to contain `Filemap` definitions for specific user applications. The default `Filemap` definitions for the overall NSJSP installation are in the `servlet.config` and `nsjspadmin.config` files.

[Example 3-41](#) illustrates the usage of the `filemaps.config` file for specific user application `Filemap` definitions.

Example 3-41. Specific User Application Filemap Definitions

Consider an installation with a URI name of `/dev` running an application `foo.war`. By default, the application can be accessed over the URL

`http://<ipaddress:port>/dev/foo` and the full application context path will be `/dev/foo`. There might be situations where the application context and thus the URL used to access the application along with the application cannot include the URI name prefix. In that case, the URL to access the application would be

`http://<ipaddress:port>/foo` and the full application context path is `/foo`. The `filemaps.config` file is used to achieve this objective and it can be achieved in one the following ways:

- By deploying the application through the NSJSP manager and indicating that the context name should not have the default URI name prefix. In this case, the NSJSP manager deploys the application without the default URI name prefix (in this case `/dev`) and then makes an entry in the `filemaps.config` file for the application context name `/foo`. The entry in the `filemaps.config` for the application context name `/foo`, ensures that the configuration is retained across server class restarts.
- By manually making a `Filemap` entry in the `filemaps.config` file. If the application is deployed manually by copying the `war` file to the `webapps` directory, then a `Filemap` entry must be made in the `filemaps.config` file, manually. In this example, the entry in the `filemaps.config` file should be `Filemap /foo $server_objectcode`. The `$server_objectcode` variable can also be replaced by the full path to the `ssc` file for this serverclass. For more information on `server_objectcode`, see [server_objectcode](#) on page 3-11.

The `TANDEM_FILEMAPS_CONFIG` environment variable must specify the fully qualified location of `filemaps.config` file. The NSJSP servlet container reads the file identified by this environment variable during startup and decides the correct context path for the applications. For example, if there is an entry `Filemap /foo $server_objectcode` in the `filemaps.config` and an application `foo.war` is being deployed, the NSJSP servlet container will not prefix the URI name (`/dev`) to the context name (`/foo`) of the application.

The jdbc.config File

[Table 3-8](#) provides an overview of the jdbc.config file.

Table 3-8. The jdbc.config File

Location	<NSJSP_HOME>/conf
Description	The file is source in, if present, by the servlet.config file and is used to configure certain variables to include the JDBC/MX and/or JDBC/MP configuration. One variable that is modified in the jdbc.config file is USRCP. For more information on USRCP, see The Installation-Specific servlet.config File on page 3-7). The jdbc.config file creates and sets the new variables jdbcMx_LIB_PATH and jdbcMp_LIB_PATH. These two variables are used in the installation-specific servlet.config file.

[Example 3-42](#) shows the default jdbc.config file.

Example 3-42. The jdbc.config File

```
#
#  VERSION=6.1.0
#
#####
#
#  This file contains the JDBC/MP and JDBC/MX configuration.
#
#
#
#  If the CLASSPATH contains sqlmp.jar and/or jdbcMx.jar, then first
#  add those to the User ClassPath TCL variable USRCP.
#
if { [info exists env(CLASSPATH)] } {
    foreach i [split $env(CLASSPATH) ":"] {
        if {[string first "sqlmp.jar" $i] > 0} {append USRCP ":" $i}
        if {[string first "jdbcMx.jar" $i] > 0} {append USRCP ":" $i}
    }
}

#
#  Append the sqlmp.jar file (if it exists) from the JDBC/MP location
#  specified when the "setup" script was run.
#
#  NOTE:  If your CLASSPATH contains sqlmp.jar, then that version will be
#         used and will automatically override the add/append done below.
#
#
set jdbcMp_JarFile /usr/tandem/jdbcMp/current/lib/sqlmp.jar

if { [file exists $jdbcMp_JarFile] } {
    append USRCP ":$jdbcMp_JarFile"
}

#
#  Append the jdbcMx.jar file (if it exists) from the JDBC/MX location
#  specified when the "setup" script was run.
#
#  NOTE:  If your CLASSPATH contains jdbcMx.jar, then that version will
#         be used and will automatically override the add/append done below.
#
#
set jdbcMx_JarFile /usr/tandem/jdbcMx/current/lib/jdbcMx.jar

if { [file exists $jdbcMx_JarFile] } {
    append USRCP ":$jdbcMx_JarFile"
}

#####
#
#  Define the jdbc/Mx library location
#
set jdbcMx_LIB_PATH /usr/tandem/jdbcMx/current/lib

#
#  Define the jdbc/Mp library location
#
set jdbcMp_LIB_PATH /usr/tandem/jdbcMp/current/lib
```

Configuration Files for the Servlet Container

This section describes the following configuration files:

- [The server.xml File](#)
- [The context.xml File](#)
- [The web.xml File](#)

The server.xml File

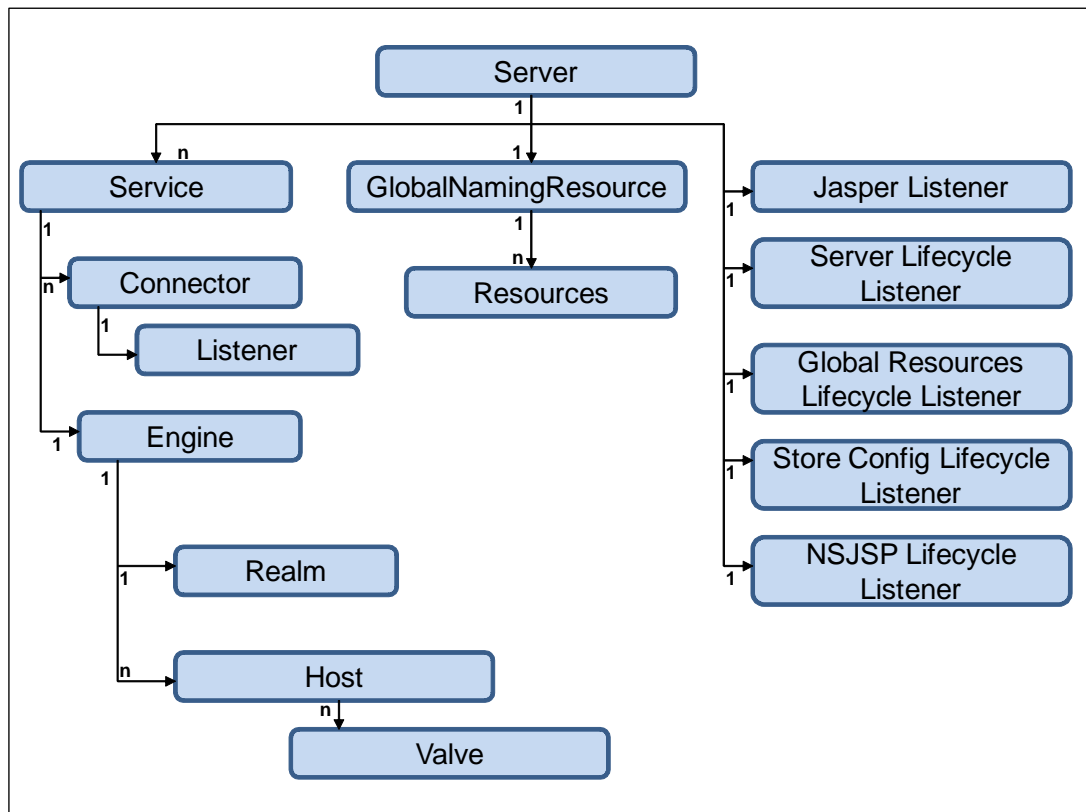
This section discusses only the elements that are used in the default `server.xml` file. Although the behavior of some of these components is the same as in Apache Tomcat, some components behave differently in NSJSP.

[Table 3-9](#) provides an overview of the `server.xml` file.

Table 3-9. Overview of the `server.xml` File

Location	<code><NSJSP_HOME>/conf/server.xml</code>
Description	Used to configure the entire NSJSP servlet container.
Recommendation	Read the complete section and make modifications based on your requirements.

[Figure 3-3](#) shows the element hierarchy and relationships in the `server.xml` file.

Figure 3-3. The Element Hierarchy and Relationships in the `server.xml` file

In [Figure 3-3](#), `n` represents one or more elements and `1` represents one element. This figure illustrates the relationships between only the elements used in the default configuration.

Note. [Figure 3-3](#) is only a depiction of the relationships between the elements and you might not be able to add more elements. For example, although the relationship between the `Server` element and the `Service` element is `1..n`, the default configuration has only one `Service` element nested in the `Server` element.

[Example 3-43](#) shows the default `server.xml` file.

Example 3-43. The server.xml File (page 1 of 2)

```

<!--
Ensure that the port is always set to -1 It only then possible to shutdown
the server using the freeze;stop command of pathway
-->
<Server port="-1">
    <!--
        Initialize Jasper prior to webapps are loaded. Documentation at
        /docs/jasper-howto.html
    -->
    <Listener className="org.apache.catalina.core.JasperListener" />
    <!--
        JMX Support for the Tomcat server. Documentation at
        /docs/non-existent.html
    -->
    <Listener
        className="org.apache.catalina.mbeans.ServerLifecycleListener" />
    <Listener
        className="org.apache.catalina.mbeans.GlobalResourcesLifecycleLi
stener" />
    <!--Listener to instantiate StoreConfig MBeans-->
    <Listener
        className="org.apache.catalina.storeconfig.StoreConfigLifecycleLi
stener" />
    <!--Listener to instantiate NSJSP specific MBeans-->
    <Listener className="com.tandem.servlet.NSJSPLifecycleListener" />

    <GlobalNamingResources>
        <Resource name="UserDatabase" auth="Container"
            type="org.apache.catalina.UserDatabase" description="User
            database that can be updated and saved"
            factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
            pathname="conf/nsjsp-users.xml" />
    </GlobalNamingResources>
    <Service name="NSJSP">
        <!--
            The Listener inside connector component is required to enable
            the JMX generic connector server for NSJSP. If not present
            the JMX Connection server will be disabled.
        -->
        <Connector protocol="HTTP/1.1" connectionTimeout="0"
            acceptCount="25" maxThreads="75">
            <Listener
                className="com.tandem.servlet.JMXConnectionListener"/>
        </Connector>
        <Engine name="NSJSP" defaultHost="localhost" >
            <Realm
                className="org.apache.catalina.realm.UserDatabaseRealm"
                resourceName="UserDatabase" digest="MD5" />

```

Example 3-43. The server.xml File (page 2 of 2)

```

    <!--
        The following is the definition of a request dumper
        valve. If enabled this valve will dump the incoming
        requests and the outgoing response in the logger of its
        container (in this case its the Engine) The
        recordLength attribute specifies the maximum number of
        bytes to write per log record. A record each is created
        for the incoming request and outgoing response If the
        dumperOn attribute is omitted the valve is disabled.
        For the valve to be enabled the dumperOn attribute
        **should** be set to true
    -->
    <!--
        <Valve
            className="com.hp.tandem.nsjsp.valves.NSJSPRequestDum
            perValve" dumperOn="true" recordLength="1000"/>
    -->

    <Host name="localhost" appBase="webapps" unpackWARs="true"
        autoDeploy="true" xmlValidation="false"
        xmlNamespaceAware="false" configClass=
        "com.tandem.servlet.catalina.startup.NSJSPContextConfig">

    <!--
        The RequestTrackerValve should be configured to track the
        requests to applications deployed in this host. The valve
        generates statistics related to the requests for all the
        web applications under the host. This tracker is a must
        to display application statistics in the new Domain
        Manager application
    -->

    <Valve
        className="com.hp.tandem.nsjsp.valves.RequestTrackerValve" />
    </Host>
</Engine>
</Service>
</Server>

```

This section discusses the following topics:

- [Server Element](#)
 - [Listener Elements](#)
 - [GlobalNamingResources Element](#)
 - [Service Element](#)

Server Element

The Server element represents the entire catalina servlet container. Therefore, it must be the single outermost element in the server.xml configuration file.

[Table 3-10](#) shows the attribute list for the Server element.

Table 3-10. Attribute List for the server element

Attribute	Description	Default value
className	Java class name for the implementation to use. This class must implement the <code>org.apache.catalina.Server</code> interface. If no class name is specified, the standard implementation will be used. HP recommends that you do not change the value of this attribute.	<code>org.apache.catalina.core.StandardServer</code>
port	Specifies the port number on which the server receives the shutdown command. NSJSP does not receive the shutdown commands on a TCP port. Shutdown depends on the number of openers for each NSJSP process. An NSJSP process shuts down when all the openers, such as the link manager close their connections to the process. HP recommends that you do not modify the default value.	-1
shutdown	NSJSP does not receive shutdown messages. This attribute is not applicable to NSJSP.	

Child Elements Nested in the server Element

The following child elements are nested in the `Server` element:

- [Listener Elements](#)
- [GlobalNamingResources Element](#)
- [Service Element](#)

Listener Elements

The `Server`, `Service`, `Engine`, and `Host` elements generate lifecycle events, such as `start` and `stop`. Listeners are elements that perform actions based on the lifecycle events of their parent element. For example, the Listeners configured as child elements of the `Server` element perform actions based on the lifecycle events of the `Server` element.

[Table 3-11](#) shows the descriptions of Listeners that are configured as child elements of the `Server` element.

Table 3-11. Descriptions of Listeners configured as child elements of the `server` element

Listener	Description
Jasper Listener	The Jasper Listener initializes the Jasper2 JSP engine before loading any web applications. The Jasper2 JSP engine is an implementation of the Java Server Pages 2.1 specification. The class name of the listener is <code>org.apache.catalina.core.JasperListener</code> .
Server Lifecycle Listener	The Server Lifecycle Listener initializes the <code>MBeanServer</code> . The <code>MBeanServer</code> registers MBeans. Without this Listener, MBeans provided as part of Apache Tomcat will not be available. The class name of the Listener is <code>org.apache.catalina.mbeans.ServerLifecycleListener</code> .
Global Resources Lifecycle Listener	The Global Resources Lifecycle Listener initializes the Global Java Naming and Directory Interface (JNDI) resources defined in the <code>server.xml</code> file and in the <code>GlobalNamingResources</code> element. Without this Listener, none of the Global Resources can be defined. The class name of the Listener is <code>org.apache.catalina.mbeans.GlobalResourcesLifecycleListener</code> .
Store Config Lifecycle Listener	This Listener is used to instantiate MBeans that are used by the <code>admin</code> application to persist the configuration changes made by the <code>admin</code> application to the respective configuration files (for example, the <code>server.xml</code> file and application-specific <code>context.xml</code> files). The class name of the Listener is <code>org.apache.catalina.storeconfig.StoreConfigLifecycleListener</code> .
NSJSP Lifecycle Listener	This Listener is used to instantiate the NSJSP-specific MBeans. These MBeans are mainly used by the <code>admin</code> application. The class name of this Listener is <code>com.tandem.servlet.NSJSPLifecycleListener</code> .

GlobalNamingResources Element

The `GlobalNamingResources` element defines the global JNDI resources for the Server. The resources defined here are listed in the server's global JNDI resource context. The server's global JNDI resource context is different from the JNDI resources defined in each web application. The global JNDI resources are not, by default, available to individual web applications. A global JNDI resource can be made available to a web application by using a `ResourceLink` element in the applications context definition. For more information on the `GlobalNamingResources` element, see <http://tomcat.apache.org/tomcat-6.0-doc/config/globalresources.html>.

The `GlobalNamingResources` element in the default `server.xml` file defines a resource of type `org.apache.catalina.UserDatabase`. This resource is used by the `UserDatabaseRealm` that is nested in the `Engine` element.

[Example 3-44](#) shows the default value for the `GlobalNamingResources` element.

Example 3-44. The Default Value for the `GlobalNamingResources` Element

```
<GlobalNamingResources>
  <Resource name="UserDatabase" auth="Container"
    type="org.apache.catalina.UserDatabase"
    description="User database that can be updated and
    saved"
    factory="org.apache.catalina.users.MemoryUserDatabaseFac
    tory"pathname="conf/nsjsp-users.xml" />
</GlobalNamingResources>
```

Service Element

A `Service` element represents the combination of one or more `Connector` components that share a single `Engine` element for processing incoming requests. One or more `Service` elements may be nested inside a `Server` element.

Note. Although it is possible to add more than one `Service` element, HP recommends only one `Service` element. This is because of the following reasons.

- Each `Service` element requires a `Connector` element and NSJSP provides only one `Connector` implementation that processes messages from the `$RECEIVE` file. Therefore, there cannot be two `Connector` definitions using the same connector implementation thus restricting the number of service definitions to only one.
 - The NSJSP Manager can manage applications in only one service.
-

[Example 3-45](#) shows the default values for the `Service` element.

Example 3-45. The Default Values for the Service Element

```

<Service name="NSJSP">
  <!--
  The Listener inside connector component is required to enable the JMX
  generic connector server for NSJSP. If not present the JMX Connection
  server will be disabled.
  -->

    <Connector protocol="HTTP/1.1"   maxThreads="75">
      <Listener
        className="com.tandem.servlet.JMXConnectionListener" />
    </Connector>

    <Engine name="NSJSP" defaultHost="localhost" >
      <Realm
        className="org.apache.catalina.realm.UserDatabaseRealm"
        resourceName="UserDatabase" digest="MD5" />

      <!--

        <Valve className=
          "com.hp.tandem.nsjsp.valves.NSJSPRequestDumperValve"
          dumperOn="true" recordLength="1000"/>

      -->

      <Host name="localhost" appBase="webapps" unpackWARs="true"
        autoDeploy="true" xmlValidation="false"
        xmlNamespaceAware="false" configClass=
        "com.tandem.servlet.catalina.startup.NSJSPContextConfig">

        <Valve className=
          "com.hp.tandem.nsjsp.valves.RequestTrackerValve"/>
      </Host>
    </Engine>
  </Service>

```

[Table 3-12](#) shows the attribute list for the Service element.

Table 3-12. Attribute List of the service Element

Attribute	Description	Default value
className	The Java class name of the implementation to use. This class must implement the <code>org.apache.catalina.Service</code> interface. If no class name is specified, the standard implementation will be used. HP recommends that this attribute not be modified.	<code>org.apache.catalina.core.StandardService</code>
name	The display name of this Service, which will be included in log messages, if the default implementation with standard catalina components is used. The name of each Service that is defined under a <code>Server</code> element must be unique.	NSJSP

Child Elements Nested in the Service Element

Following are the child elements nested in the `Service` Element:

- [Connector Element](#)
- [Engine Element](#)

Connector Element

The `Connector` element connects the HTTPD component of the iTP Secure WebServer with the NSJSP Manager. The connector receives all NSJSP request messages on the `$RECEIVE` file. Each process can have only one `$RECEIVE` file and since the NSJSP connector receives messages through this file, there can be only one `Connector` element that reads messages from `$RECEIVE`. The connector handles messages based on the type of the message. If the message is from an HTTPD component, an HTTPD message is handed over to a component that handles the message from HTTPD. In NSJSP, the message is handed over to the `NSJSPHttp11Processor` component to process the message. If it is a JMX message, the message is handed over to the component that handles JMX messages, which is `NSJSPMessageConnectionHandler`.

To understand the attributes that affect the connector behavior, it is important to understand the major differences between the NSJSP connector and the default connector supplied with Apache Tomcat.

The Apache Tomcat connector receives inputs through a TCP/IP port in the form of a stream of characters. The connector is responsible for all the HTTP1.1 protocol validation and translation. In this way, the connector functions as a simple web server.

The NSJSP `Connector` element communicates with the HTTPD component of the iTP Secure WebServer and receives all the input request data through the

\$RECEIVE file. The HTTPD component acts as the front-end web server and handles all the protocol validation and translation. After the protocol is validated, HTTPD extracts the necessary information and converts the information to a set of name value pairs. The name value pairs are then sent in messages to the NSJSP connector. The NSJSP connector assembles the request data from the messages with the name value pairs and continues with processing the request. Thus, the NSJSP Connector does not function as a web server. It receives messages from HTTPD, assembles each request and then hands over the request to the catalina container for further processing.

[Example 3-46](#) shows the default values for the Connector element.

Example 3-46. The Default Values for the Connector Element

```
<Service name="NSJSP">
  <!--
  The Listener inside connector component is required to enable the JMX
  generic connector server for NSJSP. If not present the JMX Connection
  server will be disabled.
  -->

    <Connector protocol="HTTP/1.1"    maxThreads="75">

      <Listener
        className="com.tandem.servlet.JMXConnectionListener" />

    </Connector>

    <Engine name="NSJSP" defaultHost="localhost" >

      <Realm
        className="org.apache.catalina.realm.UserDatabaseRealm"
        resourceName="UserDatabase" digest="MD5" />

    <!--

      <Valve className=
        "com.hp.tandem.nsjsp.valves.NSJSPRequestDumperValve"
        dumperOn="true" recordLength="1000"/>

    -->

      <Host name="localhost" appBase="webapps" unpackWARs="true"
        autoDeploy="true" xmlValidation="false"
        xmlNamespaceAware="false" configClass=
        "com.tandem.servlet.catalina.startup.NSJSPContextConfig">

        <Valve className=
          "com.hp.tandem.nsjsp.valves.RequestTrackerValve"/>
      </Host>

    </Engine>

  </Service>
```

[Table 3-13](#) shows the attributes for the Connector element in NSJSP.

Table 3-13. Attribute List for the Connector Element in NSJSP (page 1 of 2)

Attribute	Description	Default Value
<code>allowTrace</code>	A Boolean value that can be used to enable or disable the TRACE HTTP method. If this attribute is not specified, it is set to <code>false</code> .	
<code>emptySessionPath</code>	If set to <code>true</code> , all paths for session cookies will be set to <code>/</code> . If this attribute is not specified, it is set to <code>false</code> . It is important to understand cookie paths before setting this attribute to <code>true</code> .	
<code>enableLookups</code>	Setting this value to <code>true</code> will not enable DNS lookups but rather fetch the remote host information provided by the HTTPD component. If this attribute is not specified, it is set to <code>true</code> .	
<code>maxParameterCount</code>	The maximum number of parameters (GET plus POST) which will be automatically parsed by the container. A value of less than 0 means no limit. If not specified, a default of 10000 is used.	10000
<code>protocol</code>	Specifies the version of the protocol to be used.	HTTP/1.1.
<code>redirectPort</code>	If a request is received for which a matching <code><security-constraint></code> requires SSL transport, catalina will automatically redirect the request to the port number specified here. If this attribute is not specified, it is set to 443.	
<code>URIEncoding</code>	Specifies the character encoding used to decode the URI bytes. If this attribute is not specified, ISO-8859-1 will be used.	
<code>useBodyEncodingForURI</code>	This attribute indicates if the encoding specified in the <code>contentType</code> of the message is to be used for decoding the URI. The default value is <code>false</code> and it is suggested that the default value be retained. Hence, using the <code>URIEncoding</code> attribute for decoding the URI.	
<code>xpoweredBy</code>	Set this attribute to <code>true</code> to cause Tomcat to advertise support for the Servlet specification using the header recommended in the specification. If this attribute is not specified, the default value is <code>false</code> .	
<code>executor</code>	The name of a defined <code>Executor</code> element. If this attribute is provided, and the named executor exists, the connector will use the executor, and all the other thread attributes will be ignored.	

Table 3-13. Attribute List for the Connector Element in NSJSP (page 2 of 2)

Attribute	Description	Default Value
<code>maxHttpHeaderSize</code>	The maximum size of the request and response HTTP header, specified in bytes. If this attribute is not specified, it is set to 8192 (8 KB).	
<code>maxThreads</code>	The maximum number of request processing threads to be created by this Connector. It also determines the maximum number of simultaneous requests that can be handled. If this attribute is not specified, it is set to 200. If an executor is associated with this connector, this attribute is ignored as the connector will execute tasks using the executor rather than an internal thread pool.	75
<code>server</code>	Overrides the Server header for the http response. If set, the value for this attribute overrides the Tomcat default and any Server header set by a web application. If not set, any value specified by the application is used. If the application does not specify a value, Apache-Coyote/1.1 is used.	
<code>threadPriority</code>	The priority of the request processing threads within the JVM. If not set, the thread priority will be NORMAL. For more information about this priority, see the JavaDoc for the <code>java.lang.Thread</code> class.	

The following attributes are used in Apache Tomcat, but not applicable on NSJSP:

- `acceptCount`
- `maxPostSize`
- `maxSavePostSize`
- `proxyName`
- `proxyPort`
- `SSLEnabled`
- `Scheme`
- `secure`
- `useIPVHosts`
- `address`
- `bufferSize`
- `compressableMimeType`

- compression
- connectionLinger
- connectionTimeout
- keepAliveTimeout
- disableUploadTimeout
- maxKeepAliveRequests
- noCompressionUserAgents
- port
- restrictedUserAgents
- socketBuffer
- tcpNoDelay
- algorithm
- clientAuth
- keystoreFile
- keystorePass
- keystoreType
- keystoreProvider
- sslProtocol
- ciphers
- keyAlias
- truststoreFile
- truststorePass
- truststoreType
- truststoreProvider
- sessionCacheSize
- sessionTimeout
- crlFile
- allowUnsafeLegacyRenegotiation

For more information on these attributes, see <http://tomcat.apache.org/tomcat-6.0-doc/config/http.html>.

Child Element Nested in the Connector Element

JMX Connection Listener

The `com.tandem.servlet.JMXConnectionListener` (JMX Connection Listener) has to be configured to accept JMX requests (over NonStop IPC) from the NSJSP manager application. The NSJSP manager communicates with server class instances using JMX and without this listener the NSJSP JMX components that accept JMX requests are not configured.

[Example 3-47](#) shows the default value for the JMX Connection Listener.

Example 3-47. The Default Values of JMX Connection Listener

```
<!--
The Listener inside connector component is required to enable the JMX
generic connector server for NSJSP. If not present the JMX Connection
server will be disabled.
-->

    <Connector protocol="HTTP/1.1"    maxThreads="75">
        <Listener
            className="com.tandem.servlet.JMXConnectionListener" />
    </Connector>
```

Engine Element

The `Engine` element represents the entire request processing machinery associated with a particular `Service` element. It receives and processes all requests from one or more `Connectors`, and returns completed responses to a `Connector` for transmitting back to the client.

Only one `Engine` element must be nested inside a `Service` element. It must be defined after all of the corresponding `Connector` elements are defined for the `Service`.

[Example 3-48](#) shows the default values for the Engine element.

Example 3-48. The Default Values for the Engine Element

```
<Service name="NSJSP">
  <!--
  The Listener inside connector component is required to enable the JMX
  generic connector server for NSJSP. If not present the JMX Connection
  server will be disabled.
  -->

  <Connector protocol="HTTP/1.1"  maxThreads="75">
    <Listener
      className="com.tandem.servlet.JMXConnectionListener" />
  </Connector>

  <Engine name="NSJSP" defaultHost="localhost" >
    <Realm
      className="org.apache.catalina.realm.UserDatabaseRealm"
      resourceName="UserDatabase" digest="MD5" />

  <!--

    <Valve className=
      "com.hp.tandem.nsjsp.valves.NSJSPRequestDumperValve"
      dumperOn="true" recordLength="1000"/>

  -->

    <Host name="localhost" appBase="webapps" unpackWARs="true"
      autoDeploy="true" xmlValidation="false"
      xmlNamespaceAware="false" configClass=
      "com.tandem.servlet.catalina.startup.NSJSPContextConfig">

      <Valve className=
        "com.hp.tandem.nsjsp.valves.RequestTrackerValve"/>
    </Host>

  </Engine>
</Service>
```

[Table 3-14](#) shows the attribute list for the Engine Element.

Table 3-14. Attribute List for the Engine Element

Attribute	Description	Default value
backgroundProcessorDelay	Represents the delay in seconds between the invocation of the <code>backgroundProcess</code> method on this engine and on its child containers, including all the hosts and contexts. Child containers methods will not be invoked if their delay value is not negative. This means that the child containers are using their own processing thread. Setting this to a positive value will cause a thread to spawn. After waiting the specified amount of time, the thread will invoke the <code>backgroundProcess</code> method on this engine and all its child containers. If not specified, the default value for this attribute is 10, which represents a 10 second delay. For more information on the <code>backgroundProcess</code> , see backgroundProcess on page 3-52.	
className	Represents the Java class name of the implementation to use. This class must implement the <code>org.apache.catalina.Engine</code> interface. If this attribute is not specified, the standard value <code>org.apache.catalina.Core.StandardEngine</code> will be used. HP recommends that you use the default value.	
defaultHost	The default host name, which identifies the Host that will process requests directed to host names on this server, but which are not defined in this configuration file. This name must match the name attributes of one of the <code>Host</code> elements nested immediately inside the <code>Engine</code> element.	localhost
jvmRoute	Not applicable to NSJSP.	
name	The logical name of this Engine, used in log and error messages.	NSJSP

Considerations for Configuring the Engine element

backgroundProcess

The `Engine`, `Host`, and `Context` elements implement the `org.apache.catalina.Container` interface. One of the methods of the interface is `backgroundProcess`. Each of the containers implements the `backgroundProcess` method, which may implement some container functionality and some housekeeping functions. For example, when the `Host` element `backgroundProcess` is invoked, it checks if the auto deployer must deploy, undeploy, or redeploy any application.

The Context container `backgroundProcess` is invoked to perform session expiration and class monitoring.

The `backgroundProcess` method for a container is invoked in a separate thread called the background processing thread. It is possible to have a separate thread for each of the containers. A `backgroundProcessor` thread is spawned if the container's `backgroundProcessorDelay` attribute has a positive value. Each container invokes the `backgroundProcess` method of its child container if the child container is not configured to spawn its own `backgroundProcessor` thread.

NSJSP inherits this feature from Apache Tomcat.

Child Elements Nested in the Engine Element

Following are the child elements that are nested in the `Engine` element:

- [Realm](#)
- [Host](#)

`Realm`

The `Realm` element is used for authenticating all the applications hosted on all the hosts under the `Engine` element. For more information about Realms, see [Realms](#) on page 7-7.

[Example 3-49](#) shows the default values for the `Realm` element.

Example 3-49. The Default Values for Realm Element

```
<Engine name="NSJSP" defaultHost="localhost" >

    <Realm
        className="org.apache.catalina.realm.UserDatabaseRealm"
        resourceName="UserDatabase" digest="MD5" />

    <!--

        <Valve className=
            "com.hp.tandem.nsjsp.valves.NSJSPRequestDumperValve"
            dumperOn="true" recordLength="1000"/>

    -->

    <Host name="localhost" appBase="webapps" unpackWARs="true"
        autoDeploy="true" xmlValidation="false"
        xmlNamespaceAware="false" configClass=
            "com.tandem.servlet.catalina.startup.NSJSPContextConfig">

        <Valve className=
            "com.hp.tandem.nsjsp.valves.RequestTrackerValve"/>
    </Host>

</Engine>
```

Host

The `Host` element represents a virtual host. One or more `Host` elements are nested inside an `Engine` element. One of the `Host` elements under an `Engine` must have a name matching the `defaultHost` attribute of that `Engine`. In the default `server.xml` file the value of the `defaultHost` attribute of the `Engine` element is `localhost` and there is only one `Host` configured with the name `localhost`. For more information on virtual hosts, see [Virtual Hosts](#) on page 3-73.

[Example 3-50](#) shows the default values for the `Host` element.

Example 3-50. The Default Values for the `Host` Element

```
<Engine name="NSJSP" defaultHost="localhost" >
    <Realm
        className="org.apache.catalina.realm.UserDatabaseRealm"
        resourceName="UserDatabase" digest="MD5" />
    <!--
        <Valve className=
            "com.hp.tandem.nsjsp.valves.NSJSPRequestDumperValve"
            dumperOn="true" recordLength="1000"/>
    -->

    <Host name="localhost" appBase="webapps" unpackWARs="true"
        autoDeploy="true" xmlValidation="false"
        xmlNamespaceAware="false" configClass=
            "com.tandem.servlet.catalina.startup.NSJSPContextConfig">
        <Valve className=
            "com.hp.tandem.nsjsp.valves.RequestTrackerValve"/>
    </Host>
</Engine>
```

[Table 3-15](#) shows the attribute list for the `Host` element.

Table 3-15. Attribute List for the `Host` Element (page 1 of 2)

Attribute	Description	Default Value
<code>appBase</code>	The Application Base directory for this virtual host. This is the pathname of a directory that might contain web applications to be deployed on this virtual host. You may specify an absolute pathname for this directory, or a pathname that is relative to the <code><NSJSP_HOME></code> directory.	<code>webapps</code>
<code>autoDeploy</code>	This flag value indicates if new web applications, dropped in to the <code>appBase</code> directory while NSJSP is running, should be automatically deployed. For more information, see autoDeploy on page 3-57.	<code>true</code>
<code>backgroundProcessorDelay</code>	This value represents the delay in seconds between the invocation of the <code>backgroundProcess</code> method on this <code>Host</code> and on its child containers, including all contexts. Child containers will not be invoked if their delay value is not negative (which would mean they are using their own processing thread). Setting this to a positive value will cause a thread to be spawned. After waiting the specified amount of time, the thread will invoke the <code>backgroundProcess</code> method on this host and all its child containers. A host will use background processing to perform live web application deployment related tasks. If not specified, the default value for this attribute is -1, which means the host will rely on the background processing thread of its parent Engine.	
<code>className</code>	The Java class name of the implementation to use. This class must implement the <code>org.apache.catalina.Host</code> interface. If this attribute is not specified, the standard host implementation <code>org.apache.catalina.core.StandardHost</code> will be used. HP recommends that you not modify this value.	
<code>deployOnStartup</code>	This attribute indicates if web applications from this host should be automatically deployed when the NSJSP server class is started. Even if this attribute is set to <code>false</code> and <code>autoDeploy</code> is set to <code>true</code> , the applications will still be deployed when the background process for the host runs. If both <code>autoDeploy</code> and <code>deployOnStartup</code> are set to <code>false</code> , the applications must be deployed through the NSJSP Manager.	

Table 3-15. Attribute List for the Host Element (page 2 of 2)

Attribute	Description	Default Value
name	The network name for this virtual host. Regardless of the case used to specify the hostname, NSJSP will convert it to lower case internally. One of the Hosts nested within an Engine must have a name that matches the defaultHost setting for that Engine.	localhost
deployXML	For a description of this attribute, see deployXML on page 3-58.	
errorReportValveClass	The Java class name of the error reporting valve, which will be used by this Host. This valve provides error reports. Setting this property enables you to customize the format of the error pages, which will be generated by NSJSP. If not specified, the default value for this class will be <code>org.apache.catalina.valves.ErrorReportValve</code> . This class must implement the <code>org.apache.catalina.Valve</code> interface.	
unpackWARs	Set this attribute to <code>true</code> if you want web applications that are placed in the <code>appBase</code> directory as web application archive (WAR) files to be unpacked into a corresponding disk directory structure. Set this attribute to <code>false</code> , to run such web applications directly from the WAR file. For more information, see unpackWARs on page 3-57.	true
workDir	The pathname to a scratch directory to be used by applications for this Host. Each application will have its own subdirectory with temporary read-write use. Configuring a context <code>workDir</code> will override the use of the Host <code>workDir</code> configuration. This directory can be made visible to servlets in a web application by configuring a servlet context attribute (of type <code>java.io.File</code>) named <code>javax.servlet.context.tempdir</code> as described in the Servlet Specification. If not specified, a suitable directory below <code>\$CATALINA_BASE/work</code> will be provided.	
configClass	For NSJSP the value of this attribute must be <code>com.tandem.servlet.catalina.startup.NSJSPContextConfig</code>	<code>com.tandem.servlet.catalina.startup.NSJSPContextConfig</code>

Considerations for Configuring the Attributes of the `Host` Element

It is important to understand the effect of the `autoDeploy`, `unpackWARs` and `deployXML` attributes with regard to the security of the NSJSP servlet container and application resources.

`autoDeploy`

Setting `autoDeploy` to `true` could result in the redeployment of an entire application, if any changes are made to watched application resources. For more information on watched application resources, see [WatchedResource](#) on page 3-65. Setting `autoDeploy` to `false` will prevent any changes made to the deployed application from affecting the running application. The basic premise is that, unless a change is propagated through an authorized entry point, such as the NSJSP Manager application, no changes should be made to an already running application. The `autoDeploy` attribute does not have any effect on the Manager application. This means that irrespective of the value of the `autoDeploy` attribute the Manager application can deploy/undeploy applications.

The following scenarios illustrate the effect of the `autoDeploy` attribute:

- **Scenario 1:** An application is deployed using a WAR file and the WAR file is exploded to a directory (because the `unpackWARs` attribute is `true`). In this case, if the `autoDeploy` is `true` and if the WAR file is accidentally removed the entire context gets undeployed. If the `autoDeploy` is set to `false`, the context still exists and the application is still available.

Note. Although the application is still available, if there are any requests needing any of the just deleted resources, those requests will fail.

- **Scenario 2:** An application is deployed as a directory and the `web.xml` (or any `WatchedResource`) is accidentally modified. If `autoDeploy` is set to `true`, the entire application will need to be redeployed with the modified `web.xml`. If the modified `web.xml` has a configuration error (syntax error), the entire application context will be unavailable. If `autoDeploy` is set to `false`, the modified `web.xml` file will not harm the running application.
- **Scenario 3:** An application contains several JSPs and one of the JSPs is accidentally modified. In this case, irrespective of the value of the `autoDeploy` attribute, the modified JSP is effective immediately. In this scenario, the JSP becomes not marked as a `WatchedResource`. If it is marked as a `WatchedResource` then the entire context is reloaded.

`unpackWARs`

Setting `unpackWARs` to `true` will explode WAR files in the Host's `appBase` directory (usually the `webapps` directory). This will expose application contents such as JSPs, and properties files. Although you can prevent modifications to these files from affecting the running application by setting `autoDeploy` to `false`, changes will affect the application upon restart. This represents a potential threat.

There is a potential drawback to setting `unpackWARs` to `false`. A web application's static content will be read from the WAR file directly, instead of from the otherwise exploded directory. It is recommended that all static content be served by the iTP Secure WebServer and not by NSJSP.

`deployXML`

Set this attribute to `false`, if you want to disable parsing the `context.xml` file embedded inside the application (located at `/META-INF/context.xml`). Security-conscious environments should set this to `false` to prevent applications from interacting with the container's configuration. The administrator will then be responsible for providing an external context configuration file in `NSJSP_HOME/conf/[enginename]/[hostname]/`.

Setting this attribute to `true` will allow the NSJSP container to deploy the application using the `context.xml` file in the application's `META-INF` directory. This means that the application can define its own context. There are certain parameters in the context definition that could allow a rogue application to gain access to the NSJSP servlet container's internal resources and also to other applications running alongside the rogue application. The following attributes can be exploited by a rogue application:

`crossContext`

If this value is set to `true`, calls to

`javax.servlet.ServletContext.getContext(<context uri>)` will return the `ServletContext` of the application with the context name `<context uri>`. This means that the caller will have access to contexts for other applications running on the same Host. Although the default value is `false`, the application can still set this attribute to `true` and gain access to other applications' contexts.

`privileged`

If this value is set to `true`, the application is treated as a privileged application and will have access to all the internal classes of NSJSP along with certain container applications, such as the Manager application classes.

Child Element Nested in the `Host` Element

The request tracker Valve is configured as a child element in the `Host` element.

Valve Element

A Valve element represents a component that will be inserted into the request processing pipeline for a container. A Valve element can be configured as a child element of an `Engine`, `Host` or a `Context`. The following valves are configured in the default `server.xml` file:

Request Tracker Valve

The class name of this valve is

`com.hp.tandem.nsjsp.valves.RequestTrackerValve`. This valve

must be configured for every configured `Host` element. The valve tracks all the requests flowing to the applications in that `Host`. This information is used for displaying application statistics in the new NSJSP Manager application. There is no overhead incurred in configuring this valve.

[Example 3-51](#) shows the default values for the `RequestTrackerValve`.

Example 3-51. The Default Values for the `RequestTrackerValve`

```
<Host name="localhost" appBase="webapps" unpackWARs="true"
  autoDeploy="true" xmlValidation="false" xmlNamespaceAware="false"
  configClass="com.tandem.servlet.catalina.startup.NSJSPContextConfig">

  <Valve className="com.hp.tandem.nsjsp.valves.RequestTrackerValve"/>

</Host>
```

Request Dumper Valve

The class name of this valve is `com.hp.tandem.nsjsp.valves.NSJSPRequestDumperValve`. This valve dumps (that is, records) all the incoming requests and the outgoing responses to the parent log file. For example, if this valve is configured under an `Engine` and is turned on by setting the `dumperOn` attribute to `true`, the valve dumps all the requests and responses for all the applications in all the `Hosts` configured for the `Engine`. This valve too, like any other valve, can be configured in an `Engine`, `Host` or a `Context`. The `recordLength` attribute indicates the maximum size of the record printed to the log file. A record is printed for the request and another for the response.

The default `server.xml` file does not have this valve configured. If this valve is configured, the `dumperOn` and the `recordLength` attributes can be changed while the server is running using the MBeans feature of the NSJSP Manager application.

The context.xml File

Before discussing the content of the `context.xml` file, it is important to understand a context. A context in NSJSP means a web application. To configure a context within NSJSP, a Context Descriptor is required. A Context Descriptor is simply an XML file that contains NSJSP related configuration information for a context. It could include, for example, naming resources or the session manager configuration. Application specific context descriptors can be located in

```
<NSJSP_HOME>/conf/[enginename]/[hostname]/<appname>.xml or
<NSJSP_HOME>/webapps/<appname>/META-INF/context.xml.
```

This section will be restricted to the default `context.xml` file located in the `<NSJSP_HOME>/conf` directory. The context definition in this file will be loaded by all web applications in the entire NSJSP Servlet Container. The elements defined by this `context.xml` file can be overridden by an application-specific `context.xml` file. For example, the `NSJSP_HOME/conf/context.xml` file defines a `Manager` element. If

the application-specific context.xml file does not define its own manager element then the one defined in the <NSJSP_HOME>/conf/context.xml file will be used.

[Table 3-16](#) provides an overview of the default context.xml file.

Table 3-16. The default context.xml File

Location	<NSJSP_HOME>/conf/context.xml
Description	The default context.xml file is the context definition loaded by all web applications.

[Example 3-52](#) shows the default context.xml file.

Example 3-52. The default context.xml File

```
<!--
  The contents of this file will be loaded for each web application
-->

<Context>
  <!-- Default set of monitored resources -->
  <WatchedResource>WEB-INF/web.xml</WatchedResource>

  <!--
    NSJSP specific Web application loader.
    All contexts in NSJSP must be loaded using this application loader.
  -->

  <Loader
className="com.tandem.servlet.catalina.loader.NSJSPWebappLoader"/>

  <!--
    NSJSP specific Standard Manager.
    The Manager does no session persistence.
  -->

  <Manager pathname=""
className="com.tandem.servlet.catalina.session.NSJSPStandardManager"/>

  <!--
    <Manager
className="com.tandem.servlet.catalina.session.NSJSPPersistentManager">
      <Store
className="com.tandem.servlet.catalina.session.NonStopSQLJDBCStore"
        driverName="com.tandem.sqlmx.SQLMXDriver"
        connectionURL="jdbc:sqlmx:"
        sessionTable="<catalog>.<schema>.<tablename>"
        sessionIdCol="session_id"
        sessionProcessNameCol="process_name"
        sessionRecNumberCol="rec_number"
        sessionAppCol="app_name"
        sessionDataCol="session_data" sessionValidCol="valid"
        sessionMaxInactiveCol="maxinactiveinterval"
        sessionLastAccessedCol="lastaccessed"/>
      </Manager>
    -->
  </Context>
```

[Table 3-17](#) shows the attribute list for the Context element.

Table 3-17. Attribute List for the Context Element (page 1 of 5)

Attribute	Description
backgroundProcessor Delay	Represents the delay in seconds between the invocation of the <code>backgroundProcess</code> method on this context and its child containers, including all wrappers. Setting this to a positive value causes a thread to spawn. After waiting the specified amount of time, the thread invokes the <code>backgroundProcess</code> method on this context. A context uses background processing to perform session expiration and class monitoring for reloading. If this attribute is not specified, the default value for this attribute is <code>-1</code> , which means the context will rely on the background processing thread of its parent host.
className	Represents the Java class name of the implementation to use. This class must implement the <code>org.apache.catalina.context</code> interface. If this attribute is not specified, the standard value <code>org.apache.catalina.core.StandardContext</code> will be used.
cookies	Enables or disables cookies for session identifier communication. Set to <code>true</code> if you want cookies to be used for session identifier communication if supported by the client (this is the default). Set to <code>false</code> if you want to disable the use of cookies for session identifier communication, and rely only on URL rewriting by the application. If this attribute is not specified, the default value is <code>true</code> .
crossContext	Set to <code>true</code> if you want this application to be able to call <code>ServletContext.getContext()</code> to successfully obtain a request dispatcher for other web applications running on this virtual host. Set to <code>false</code> (the default) in security conscious environments, which makes <code>getContext()</code> always return null. If this attribute is not specified, the default value is <code>false</code> . For more information on the implications of using <code>crossContext</code> , see deployXML on page 3-58.
docBase	The Document Base (also known as the Context Root) directory for this web application, or the pathname to the web application archive file (if this web application is being executed directly from the WAR file). You may specify an absolute pathname for this directory or the WAR file, or a pathname that is relative to the <code>appBase</code> directory of the owning Host. The value of this field must not be set when the Context is configured using a <code>META-INF/context.xml</code> file as it is inferred by the automatic deployment process.

Table 3-17. Attribute List for the Context Element (page 2 of 5)

Attribute	Description
override	<p>Overrides the setting of either the global or Host default contexts.</p> <p>Set to <code>true</code> to have explicit settings in this Context element override any corresponding settings in either the global or Host default contexts. By default, settings from a default context are used.</p> <p>If a symbolic link is used for the <code>docBase</code>, changes to the symbolic link are effective only after an NSJSP restart or by undeploying and redeploying the context. A context reload is not sufficient.</p>
privileged	<p>Changes the context's parent class loader to be the Server Class loader rather than the Shared class loader.</p> <p>Set to <code>true</code> to allow this context to use container servlets, such as the manager servlet. If this attribute is not specified, the default value is <code>false</code>.</p> <p>Note. In a default installation, the Common class loader is used for both the Server Class loader and the Shared class loaders.</p>
path	<p>The value of this field must not be set except when statically defining a Context in <code>server.xml</code>, as it will be inferred from the filenames used for either the <code>.xml</code> context file or the <code>docBase</code>.</p> <p>Note. HP recommends that contexts should not be specified in the <code>server.xml</code> file as this file is meant for container configuration and application specific contexts should be specified in the <code>META-INF/context.xml</code> file of the application.</p>
reloadable	<p>Set to <code>true</code> if you want Catalina to monitor classes in <code>/WEB-INF/classes/</code> and <code>/WEB-INF/lib</code> for changes, and automatically reload the web application if a change is detected. This feature is very useful during application development, but it requires significant runtime overhead and is not recommended for use with applications that have been deployed in production. Hence, the default setting for this attribute is <code>false</code>. As an alternative, however, to trigger reloads of deployed applications on demand, you can use the Manager web application.</p>
wrapperClass	<p>Represents the Java class name of the <code>org.apache.catalina.Wrapper</code> implementation class that will be used for servlets managed by this Context. If this attribute is not specified, a standard default value will be used.</p>

Table 3-17. Attribute List for the Context Element (page 3 of 5)

Attribute	Description
<code>useHttpOnly</code>	<p>Indicates if the <code>HttpOnly</code> flag is included in the HTTP response header.</p> <p>If the <code>HttpOnly</code> flag is included in the HTTP response header, a cookie cannot be accessed through a client side script when the browser supports this flag. As a result, even if a cross-site scripting (XSS) flaw exists, and a user accidentally accesses a link that exploits this flaw, a browser, such as Internet Explorer does not reveal the cookie to a third party. If a browser does not support <code>HttpOnly</code> and a website attempts to set an <code>HttpOnly</code> cookie, the <code>HttpOnly</code> flag is ignored by the browser, thus creating a traditional script accessible cookie. As a result, the session cookie becomes vulnerable to theft or modification by malicious script. If this attribute is not specified, the default value is <code>false</code>.</p>
<code>allowLinking</code>	<p>If the value of this flag is <code>true</code>, symlinks will be allowed inside the web application, pointing to resources outside the web application base path. If this attribute is not specified, the default value is <code>false</code>.</p> <p>It is suggested that the value of this attribute be set to <code>false</code>. Setting this to <code>false</code> instructs the NSJSP servlet container to check if the resource belongs to the application base. If this is set to <code>true</code>, an application can reference resources outside its base directory which could prove to be a security risk in some cases. A good practice is to limit the application references to only those resources that are under its base directory.</p>
<code>antiJARLocking</code>	<p>The default value is <code>false</code> and it is suggested to always keep this value set to <code>false</code>. This will be used in those platforms where access to an application resource like a JAR file ends in file locks. An example would be if <code>URLClassLoader.getResource()</code> accessed a JAR file, that could lead to the jar file getting locked. Such a situation does not occur on <code>NonStop</code> so the value should be set to <code>false</code>.</p>
<code>antiResourceLocking</code>	<p>The default value is set to <code>false</code> and it is suggested that it be kept set to <code>false</code>. If set to <code>true</code>, the NSJSP servlet container copies each application into a separate directory in the <code>temp</code> folder. This is meant for those platforms that lock file resources when accessing them. Setting this to <code>true</code> will result in significant startup times.</p>
<code>cacheMaxSize</code>	<p>Maximum size of the static resource cache in kilobytes. If not specified, the default value is 10240 (10 megabytes).</p> <p>Note. It is suggested that web application static resources be served by the iTP Secure WebServer.</p>

Table 3-17. Attribute List for the Context Element (page 4 of 5)

Attribute	Description
cacheObjectMaxSize	Maximum size of a static resource that will be placed in the cache. If not specified, the default value is 512 (512 kilobytes). If this value is greater than cacheMaxSize/20 it will be reduced to cacheMaxSize/20. Note. It is suggested that web application static resources be serviced by the iTP Secure WebServer.
cacheTTL	Amount of time in milliseconds between revalidation of cache entries. If not specified, the default value is 5000 (5 seconds). Note. It is suggested that web application static resources be serviced by the iTP Secure WebServer.
cachingAllowed	If the value of this flag is true, the cache for static resources will be used. If not specified, the default value of the flag is true. Note. It is suggested that web application static resources be serviced by the iTP Secure WebServer.
caseSensitive	Deprecated
clearReferencesStop Threads	If true, NSJSP attempts to terminate threads that have been started by a web application. Stopping threads is performed through the deprecated <code>Thread.stop()</code> method and is likely to result in instability. Enabling this should be viewed as an option of last resort in a development environment and is not recommended in a production environment. If this attribute is not specified, the default value is false.
processTlds	Checks whether the context should process tag library descriptors (TLDs) on startup. The default is true. The false setting is intended for special cases that know in advance TLDs are not part of the web application.
swallowOutput	If the value of this flag is true, the bytes that are output to <code>System.out</code> and <code>System.err</code> by the web application are redirected to the web application logger. If this attribute is not specified, the default value is false.
tldNamespaceAware	If the value of this flag is true, the TLD files XML validation will be namespace-aware. If you turn this flag on, you should probably also turn <code>tldValidation</code> on. The default value for this flag is false, and setting it to true will have a negative impact on performance.
tldValidation	If the value of this flag is true, the TLD files will be XML validated on context startup. The default value for this flag is false, and setting it to true will incur a performance penalty.
unloadDelay	The time in ms that the container waits for servlets to unload. If this attribute is not specified, the default value is 2000 ms.

Table 3-17. Attribute List for the Context Element (page 5 of 5)

Attribute	Description
unpackWAR	If <code>true</code> , NSJSP unpacks all compressed web applications before running them. If this attribute is not specified, the default value is <code>true</code> .
useNaming	Set to <code>true</code> (the default) to have Catalina enable a JNDI <code>InitialContext</code> for this web application that is compatible with Java2 Enterprise Edition (J2EE) platform conventions. If this attribute is not specified, the default value is <code>true</code> .
workDir	Represents the pathname to a scratch directory. This information is provided by this Context for temporary read-write use by servlets within the associated web application. This directory will be made visible to servlets in the web application by a servlet context attribute (of type <code>java.io.File</code>) named <code>javax.servlet.context.tempdir</code> as described in the Servlet Specification. If this attribute is not specified, a suitable directory underneath <code>\$CATALINA_BASE/work</code> is provided.

The following elements are defined in the `<NSJSP_HOME>/conf/context.xml` file:

WatchedResource

The `WatchedResource` identifies an application resource that has to be watched by the auto deployer. For more information on the auto deployer, see [autoDeploy](#) on page 3-57. There can be multiple `WatchedResource` entries defined in a `context.xml` file. Some resources, such as an application WAR file (if present), the application directory, and the context definition file are by default added as watched resources.

Manager

The `Manager` element configures the session manager. By default, the `NSJSPStandardManager` is configured. The `NSJSPStandardManager` is an in-memory session manager. The pathname attribute is ignored by the manager. For more information about the `NSJSPStandardManager`, see [Session Management](#) on page 3-75.

Note. The `context.xml` file also contains a persistent manager configuration which is commented out.

Loader

The `Loader` element represents the web application class loader that will be used to load Java classes and resources for a web application. NSJSP provides its own loader class

`com.tandem.servlet.catalina.loader.NSJSPWebappLoader`. The loader class name should not be changed.

The web.xml File

The web.xml file in the <NSJSP_HOME>/conf directory defines the default values for all web applications loaded into each instance of the NSJSP Servlet Server Class. As each application is deployed, this web.xml file is processed, followed by the /WEB-INF/web.xml deployment descriptors from individual. This file contains built in servlet definitions and servlet mappings, filters and filter mappings, session parameters, and MIME mappings.

This section discusses the following topics:

- [Built-in Servlet Definitions](#)
- [Static Content Filter](#)
- [Session Timeout](#)
- [MIME Type Mappings](#)

Built-in Servlet Definitions

The following built-in servlet definitions are used in the web.xml file:

- [Default Servlet](#)
- [Invoker Servlet](#)
- [JSP Page Compiler and Execution Servlet](#)
- [SSI Servlet](#)
- [CGI Processing Servlet](#)

Default Servlet

As the name implies, it is generally configured as the default servlet for a web application, by being mapped to the URL pattern /. For more information, see <http://tomcat.apache.org/tomcat-6.0-doc/funcsspecs/fs-default.html>.

[Example 3-53](#) shows the configuration of the default Servlet.

Example 3-53. Configuration of the Default Servlet

```
<servlet>
    <servlet-name>default</servlet-name>
    <servlet-
class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>0</param-value>
    </init-param>
    <init-param>
        <param-name>listings</param-name>
        <param-value>>false</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

.
.
.

<!-- The mapping for the default servlet -->
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

Invoker Servlet

The `invoker` servlet allows a web application to dynamically register new servlet definitions, without having created a new `<servlet>` element in the `/WEB-INF/web.xml` deployment descriptor, and execute requests utilizing the new servlet definitions. From the perspective of the newly registered servlets, all servlet lifecycle requirements of the Servlet Specification (such as calling `init()` and `destroy()` at the correct times) are honoured. For more information, see <http://tomcat.apache.org/tomcat-6.0-doc/funcsspecs/fs-invoker.html>. The `Invoker Servlet` and the `servlet-mapping` for the `Invoker Servlet` are commented out. For more information on the `Invoker Servlet`, see [The Region Directive](#) on page 3-28.

Note. HP recommends that the `Invoker Servlet` not be used, because it could lead to security issues. For example, enabling the `invoker` servlet allows a URL to directly invoke the servlet using its class path, thereby bypassing all the security constraints defined in the web application deployment descriptor.

[Example 3-54](#) shows the configuration of the `invoker` servlet.

Example 3-54. Configuration of the Invoker Servlet

```
<!--
<servlet>
    <servlet-name>invoker</servlet-name>
    <servlet-
class>org.apache.catalina.servlets.InvokerServlet</servlet-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>0</param-value>
    </init-param>

    <load-on-startup>2</load-on-startup>
</servlet>
-->
.
.
.
<!-- The mapping for the invoker servlet -->
<!--
    <servlet-mapping>
        <servlet-name>invoker</servlet-name>
        <url-pattern>/servlet/*</url-pattern>
    </servlet-mapping>
-->
```

JSP Page Compiler and Execution Servlet

The JSP Page Compiler and Execution Servlet is the mechanism used by NSJSP (inherited from Tomcat) to process JSP pages. Traditionally, this servlet is mapped to the URL pattern `*.jsp`. You can use the default configuration for most applications. For more information on the JSP Page Compiler and Execution Servlet, see <http://tomcat.apache.org/tomcat-6.0-doc/jasper-howto.html>.

[Example 3-55](#) shows the configuration of the JSP page compiler and execution Servlet.

Example 3-55. Configuration of the JSP Page Compiler and Execution Servlet

```

<servlet>
    <servlet-name>jsp</servlet-name>
    <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-
class>
    <init-param>
        <param-name>fork</param-name>
        <param-value>>false</param-value>
    </init-param>
    <init-param>
        <param-name>xpoweredBy</param-name>
        <param-value>>false</param-value>
    </init-param>
    <load-on-startup>3</load-on-startup>
</servlet>
.
.
.
<!-- The mapping for the JSP servlet -->
<servlet-mapping>
    <servlet-name>jsp</servlet-name>
    <url-pattern>*.jsp</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>jsp</servlet-name>
    <url-pattern>*.jspx</url-pattern>
</servlet-mapping>

```

SSI Servlet

The Server Side Includes (SSI) processing servlet processes SSI directives in HTML pages consistent with similar capabilities in web servers like Apache. Traditionally, this servlet is mapped to the URL pattern *.shtml. The SSI Servlet and the servlet-mapping for the SSI Servlet are commented out in the <NSJSP_HOME>/conf/web.xml file.

For more information, see <http://tomcat.apache.org/tomcat-6.0-doc/ssi-howto.html>.

Note. If you have configured the iTP Secure WebServer to serve the static content of your web applications, HP recommends that you use the SSI feature offered by the iTP Secure WebServer. For more information, see the *iTP Secure WebServer System Administrator's Guide*.

[Example 3-56](#) shows the configuration of the SSI Servlet.

Example 3-56. Configuration of the SSI Servlet

```
<!--
<servlet>
    <servlet-name>ssi</servlet-name>
    <servlet-class>
        org.apache.catalina.ssi.SSIServlet
    </servlet-class>
    <init-param>
        <param-name>buffered</param-name>
        <param-value>1</param-value>
    </init-param>
    <init-param>
        <param-name>debug</param-name>
        <param-value>0</param-value>
    </init-param>
    <init-param>
        <param-name>expires</param-name>
        <param-value>666</param-value>
    </init-param>
    <init-param>
        <param-name>isVirtualWebappRelative</param-name>
        <param-value>0</param-value>
    </init-param>
    <load-on-startup>4</load-on-startup>
</servlet>
-->
.
.
.
<!--
    <servlet-mapping>
        <servlet-name>ssi</servlet-name>
        <url-pattern>*.shtml</url-pattern>
    </servlet-mapping>
-->
```

CGI Processing Servlet

The Common Gateway Interface (CGI) processing servlet, enables execution of and interaction with external applications that conform to the CGI specification. Typically, this servlet is mapped to the URL pattern `/cgi-bin/*`, which means that any CGI applications that are executed must be present within the webapps directory structure. The CGI Servlet and the servlet-mapping for the CGI Servlet are commented out in the `<NSJSP_HOME>/conf/web.xml` file. For more information, refer to <http://tomcat.apache.org/tomcat-6.0-doc/cgi-howto.html>.

Note. The iTP Secure Webserver can be used to service CGI requests. For more information on servicing CGI requests, see the *Using Common Gateway Interface (CGI) Programs* section in the *iTP Secure WebServer System Administrator's Guide*.

[Example 3-57](#) shows the configuration of the CGI processing Servlet.

Example 3-57. Configuration of the CGI processing Servlet

```

<!--
<servlet>
    <servlet-name>cgi</servlet-name>
    <servlet-class>org.apache.catalina.servlets.CGIServlet
</servlet-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>0</param-value>
    </init-param>
    <init-param>
        <param-name>cgiPathPrefix</param-name>
        <param-value>WEB-INF/cgi</param-value>
    </init-param>
    <load-on-startup>5</load-on-startup>
</servlet>
-->
.
.
.
<!--
    <servlet-mapping>
        <servlet-name>cgi</servlet-name>
        <url-pattern>/cgi-bin/*</url-pattern>
    </servlet-mapping>
-->

```

Static Content Filter

This filter should be enabled when user applications are configured with a Persistent Manager and the `SessionBasedLoadBalancing` parameter in the `servlet.config` file is set to `false`, and the application needs to return static content. The use of the filter in that case, will eliminate some unnecessary session-related database operations.

In the default configuration, the static content filter configuration is provided in the `<NSJSP_HOME>/conf/web.xml` deployment descriptor, but it is commented out. As an alternative, the filter can also be defined in application specific deployment descriptors.

All static content, such as image files, static html pages and so on need to be mapped to this filter using the `filter-mapping` element in the `web.xml` file.

The use of the mapping the `filter-mapping` element by the Persistent Manager is explained below:

When NSJSP is configured for disk-based sessions, the session object is swapped out of memory after each request is processed. However, the session object is persisted to the session store in a database table. When only static resources are returned to the remote client by NSJSP, persisting the entire session object is unnecessary as the session object does not change during static resource processing. The static content filter is used to indicate to the Persistent Manager that the request being serviced is for static content. When the static content filter is enabled, the persistent manager does

not perform the database operations on the session object thus eliminating unnecessary database operation.

HP recommends that all the static content of an application be served directly by the iTP Secure WebServer.

[Example 3-58](#) shows the configuration of the static content filter.

Example 3-58. Configuration of the Static Content Filter

```
<!--
<filter>
    <filter-name>StaticContentFilter</filter-name>
    <filter-
class>com.hp.tandem.nsjsp.filters.StaticContentFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>StaticContentFilter</filter-name>
        <url-pattern>*.gif</url-pattern>
    </filter-mapping>

    <filter-mapping>
        <filter-name>StaticContentFilter</filter-name>
        <url-pattern>*.jpeg</url-pattern>
    </filter-mapping>
-->
```

Session Timeout

You can set the default session timeout (in minutes) for all newly created sessions by modifying the value in the `session-config` element.

[Example 3-59](#) shows the configuration of the session timeout parameter.

Example 3-59. Configuration of the Session Timeout Parameter

```
<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```

MIME Type Mappings

The `<mime-mapping>` element associates a file extension with a MIME type.

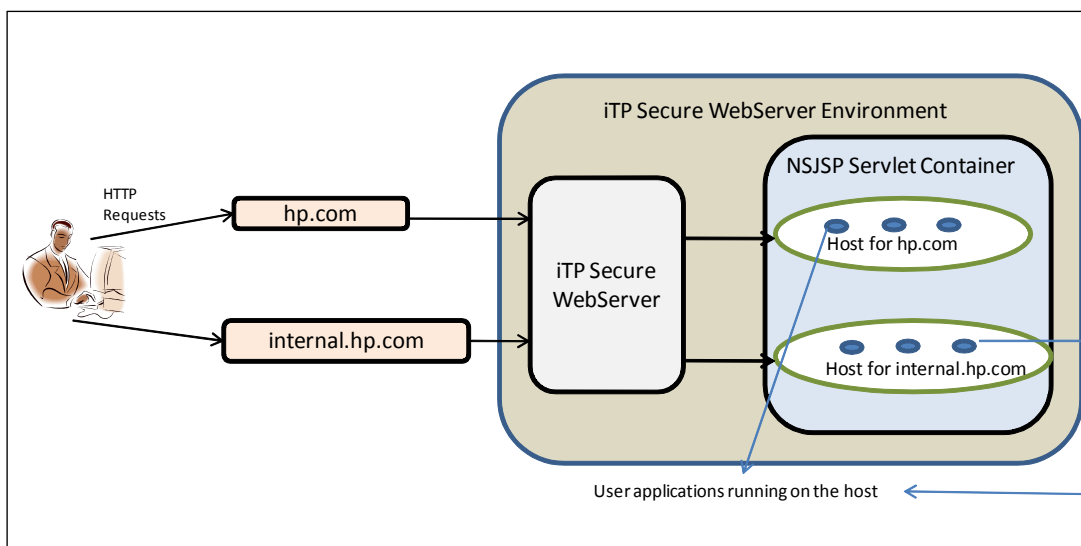
You can add an additional file extension if it is not already listed as a `<mime-mapping>` element.

Virtual Hosts

Assume that a NonStop server has two network names, `hp.com`, and `internal.hp.com` and that NSJSP is required to run applications which use both of these network names. The applications for both these network names could be configured to run with the default configuration of NSJSP, which has only one `Host` element. However, such a configuration is not flexible enough to separate the two sets of applications. The use of Virtual Hosts provides the flexibility to define two different `Host` elements, each one limited to servicing requests for only one of the networks.

[Figure 3-4](#) illustrates a sample iTP Secure WebServer environment in which a user accesses different URLs, for example `hp.com` and `internal.hp.com`.

Figure 3-4. Virtual Hosting in a Sample iTP Secure WebServer Environment



The following scenario shows an example of using Virtual Hosts.

In this example, the two URLs are serviced by the same iTP Secure WebServer environment and the user applications for those URLs are hosted in the same NSJSP servlet container. The `hp.com` URL is a public URL and contains user applications that do not need authentication. The `internal.hp.com` URL is a private URL and contains user applications that require authentication. The different types of user applications need to be segregated so that they can be secured separately. Although user security can be applied at the individual application level, it is preferable to secure applications under one secured entity, which is the `Host`. The virtual hosting feature enables such a security configuration, which permits the implementation of separate security constraints for each `Host` without impacting other `Hosts`.

Configuring Virtual Hosts

To configure virtual hosts, you must configure multiple `Host` elements nested as child elements in the `Engine` element. The value of the `name` attribute of each `Host` is used by the `Engine` element to identify where requests should be routed.

When configuring multiple `Host` elements, it is necessary to have a different `appBase` for each `Host` element and to configure the `RequestTrackerValve` for each `Host`.

Note. With the NSJSP Manager, it is possible to manage all applications running on all the Hosts from a single point of management. For more information, see the [NSJSP Manager Application](#) on page 4-1.

In many situations, system administrators might want to associate more than one network name, such as `www.hp.com` and `www.compaq.hp.com` with the same host. This can be accomplished using host name aliases as shown in [Example 3-60](#).

Example 3-60. Configuring Virtual Hosts

```
<Engine name="NSJSP" defaultHost="localhost">
...
<Host name="localhost" appBase="webapps" unpackWARs="true"
    autoDeploy="true" xmlValidation="false"
    xmlNamespaceAware="false"
    configClass="com.tandem.servlet.catalina.startup.NSJSPContextConfig">
        <Valve className="com.hp.tandem.nsjsp.valves.RequestTrackerValve"/>
    ...
</Host>

<Host name="www.hp.com" appBase="hpapps" unpackWARs="true"
    autoDeploy="false" xmlValidation="false"
    xmlNamespaceAware="false"
    configClass="com.tandem.servlet.catalina.startup.NSJSPContextConfig">
        <Alias>compaq.hp.com</Alias>
        <Valve className="com.hp.tandem.nsjsp.valves.RequestTrackerValve"/>
    ...
</Host>

<Host name="internal.hp.com" appBase="internalapps" unpackWARs="true"
    autoDeploy="true" xmlValidation="false"
    xmlNamespaceAware="false"
    configClass="com.tandem.servlet.catalina.startup.NSJSPContextConfig">
        <Valve className="com.hp.tandem.nsjsp.valves.RequestTrackerValve"/>
    ...
</Host>
```

[Example 3-60](#) shows `www.compaq.hp.com` included as an `Alias` element, which is nested as a child element of the `Host` element where `www.hp.com` is the network name.

Session Management

This section describes the different strategies for session management and the related manager configuration details and also explains how to configure any `manager` element.

This section discusses the following topics:

- [Sessions in NSJSP](#)
- [In-Memory Sessions \(`SessionBasedLoadBalancing = true`\)](#)
- [Persistent Manager Sessions \(`SessionBasedLoadBalancing = false`\)](#)
- [Mixed-Mode Sessions](#)
- [Configuring the Manager Element](#)

Sessions in NSJSP

The NSJSP environment includes multiple NSJSP Server Class processes. In this environment, each NSJSP Server Class process instance is capable of servicing any Servlet or JSP request, and creating and managing session objects required for servicing the Servlet or JSP pages. For user applications that store the application state in a session object, the session object must be made available for processing each request. The shopping cart of the JPetStore application is an example of such a session object.

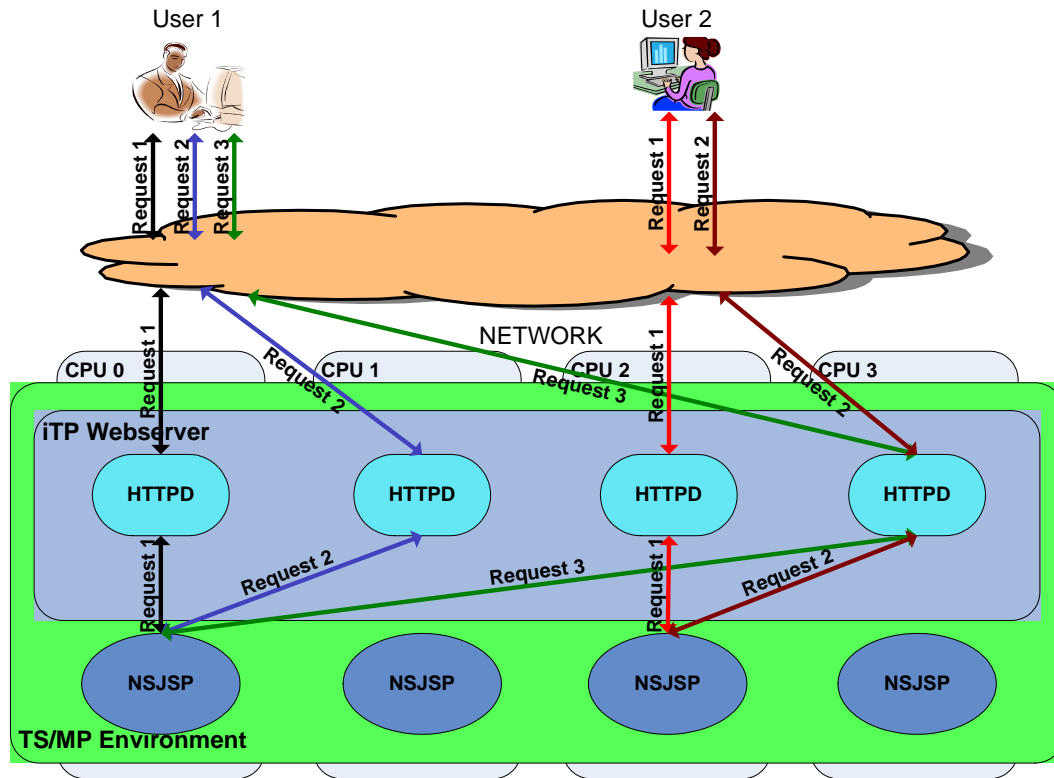
You can configure NSJSP to keep session objects in process memory or to store them in a database called the persistent store. When session objects are stored in process memory, each session object is only available to the process instance that created the session. In addition, if for any reason the process ends or is terminated, the session information can be lost. When the session objects are stored in a persistent store, the session objects are available to any process instance in the NSJSP environment. The persisted session objects are available, if NSJSP processes stop and even after a system reload.

In-Memory Sessions (`SessionBasedLoadBalancing = true`)

If you configure NSJSP to retain session objects in process memory, there must be a mechanism to route all requests for a particular session to the process that has the session object in its memory. Such a routing of HTTP requests based on the session identifier in NSJSP is called session based load balancing. In this case, `SessionBasedLoadBalancing` is enabled (that is, the parameter is set to `true`). This feature is also referred to as Sticky Sessions. Because NSJSP receives all its requests from the iTP Secure WebServer's HTTPD process, the routing mechanism is built into the HTTPD process.

This is the default configuration for NSJSP. [Figure 3-5](#) shows request routing within NSJSP sessions when `SessionBasedLoadBalancing` is `true`.

Figure 3-5. Request Routing within NSJSP Sessions when SessionBasedLoadBalancing is true



In [Figure 3-5](#), each user has their own current session, within which more than one request has been made. Note that all the requests from user 1 are routed to the NSJSP process in CPU 0, which has the session object for user 1. All requests from user 2 are routed to the NSJSP process in CPU 2, which has the session object for user 2.

Configuring In-Memory Sessions

In the installation-specific `servlet.config` file located in `<NSJSP_HOME>/conf`, set the value of the attribute `SessionBasedLoadBalancing` to `true`. [Example 3-61](#) shows the Arglist from an installation-specific `servlet.config` file with `SessionBasedLoadBalancing` enabled.

Example 3-61. Arglist from a servlet.config file with SessionBasedLoadBalancing Enabled

```

Arglist -Xmx64m -Xss128k -Xnoclassgc \
-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager \
-Djava.util.logging.config.file=$env(NSJSP_HOME)/conf/logging.properties \
-
Djavax.management.builder.initial=com.tandem.servlet.jmx.NSJSPMBeanServerB
iilder \
$NSJSP_SECMGR \
$NSJSP_SECMGR_POLICY \
$NSJSP_JAAS_CONFIG \
-Dcom.tandem.servlet.CONTEXT_PREFIXES=/scA \
-Dcatalina.home=$env(NSJSP_HOME) \
-Dcatalina.base=$env(NSJSP_HOME) \
-Djava.io.tmpdir=$env(NSJSP_HOME)/temp \
-DSessionBasedLoadBalancing=true
org.apache.catalina.startup.Bootstrap start

```

After enabling SessionBasedLoadBalancing, configure the NSJSPStandardManager as the session manager. [Example 3-62](#) shows the default configuration of the NSJSP Standard Manager in the <NSJSP_HOME>/conf/context.xml file.

Example 3-62. The Default Configuration of the NSJSP Specific Standard Manager

```

<!--
    NSJSP specific Standard Manager.
    The Manager does no session persistence.
-->

<Manager pathname=""
    className="com.tandem.servlet.catalina.session.NSJSPStandardManager"/>

```

[Table 3-18](#) lists the attributes of the NSJSPStandardManager. For more information on configuring the Manager element, see [Configure the Manager Element](#) on page 3-82.

Table 3-18. Attribute List for the NSJSPStandardManager (page 1 of 2)

Attribute	Description	Default value
className	Java class name of the implementation to use. This class must implement the <code>org.apache.catalina.Manager</code> interface. The value must be <code>com.tandem.servlet.catalina.session.NSJSPStandardManager</code> .	<code>com.tandem.servlet.catalina.session.NSJSPStandardManager</code>
distributable	When set to <code>true</code> , it requires the session manager to enforce the restrictions described in the Servlet Specification on distributable applications. This means that all session attributes must implement <code>java.io.Serializable</code> . When set to <code>false</code> , the session manager does not enforce these restrictions. NOTE: The value for this property is inherited automatically based on the presence or absence of the <code><distributable></code> element in the web application deployment descriptor (<code>/WEB-INF/web.xml</code>).	
algorithm	Represents the name of the Message Digest algorithm used to calculate session identifiers produced by the <code>NSJSPStandardManager</code> . This value must be supported by the <code>java.security.MessageDigest</code> class. If this attribute is not specified, the default value is MD5.	
entropy	A string value that is used when seeding the random number generator used to create session identifiers for this Manager. If the <code>entropy</code> attribute is not specified, an internal value is calculated, but a long string value should be specified in security-conscious environments.	
maxActiveSessions	The maximum number of active sessions that will be created by this Manager, or -1 for no limit. If this attribute is not specified, it is set to -1. For more information, see maxActiveSessions on page 3-79.	

Table 3-18. Attribute List for the NSJSPStandardManager (page 2 of 2)

Attribute	Description	Default value
<code>maxInactiveInterval</code>	<p>The maximum time interval, in seconds, between client requests before a session is invalidated.</p> <p>This attribute provides the initial value whenever a new session is created, but the interval may be dynamically altered by a servlet via the <code>setMaxInactiveInterval</code> method of the <code>HttpSession</code> object.</p> <p>For more information, see maxInactiveInterval on page 3-79.</p>	
<code>pathname</code>	The value of this attribute is ignored by the <code>NSJSPStandardManager</code> .	
<code>processExpiresFrequency</code>	<p>Frequency of the session expiration, and related manager operations. Manager operations will be performed once for the specified amount of <code>backgroundProcess</code> calls (the lower the amount, the more often the checks will occur). The minimum value is 1, and the default value is 6.</p>	
<code>randomClass</code>	<p>This class is used to generate the session id. It is the Java class name of the <code>java.util.Random</code> implementation class to use. If the <code>randomClass</code> attribute is not specified, the default value is <code>java.security.SecureRandom</code>.</p>	
<code>sessionIdLength</code>	The length of session IDs created by this Manager, excluding any JVM route information used for load balancing.	

Considerations for Configuring In-Memory Sessions

`maxActiveSessions`

An active session is one that is still present in the memory of the `NSJSPStandardManager`. When creating new sessions, the `NSJSPPersistentManager` looks at the number of sessions in memory. If the number is equal to the value of `maxActiveSessions` when `maxActiveSessions` is not equal to -1, a new session will not be created and an exception is thrown. There could be situations where some of the sessions in memory are eligible to be expired and removed from the memory. The removal of expired sessions happens only when the `backgroundprocess` runs. Until such time, the expired sessions are also counted as active sessions because they are still in memory.

`maxInactiveInterval`

This variable is used to set the maximum time interval, in seconds, between client requests before a session is invalidated. However, it should be noted that in the current version of NSJSP, during a context start or restart this variable is always overridden by the value specified in the `session-timeout` element in the application's deployment descriptor (`web.xml`). If the application's deployment descriptor does not explicitly specify a `session-timeout` element, the value is taken from the `<NSJSP_HOME>/conf/web.xml` file as shown in [Example 3-63](#). The default value for `session-timeout`, which is expressed in minutes, is 30.

Example 3-63. The Default Configuration of *session-timeout* in the `<NSJSP_HOME>/conf/web.xml` File

```
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
```

It is possible to set the value of this variable at run time using the MBeans feature of the NSJSP Manager. For more information on using the MBeans feature, see [Managing MBeans](#) on page 4-46. The value set at run time will take effect immediately but will not be persisted and will not be available across web application context restarts.

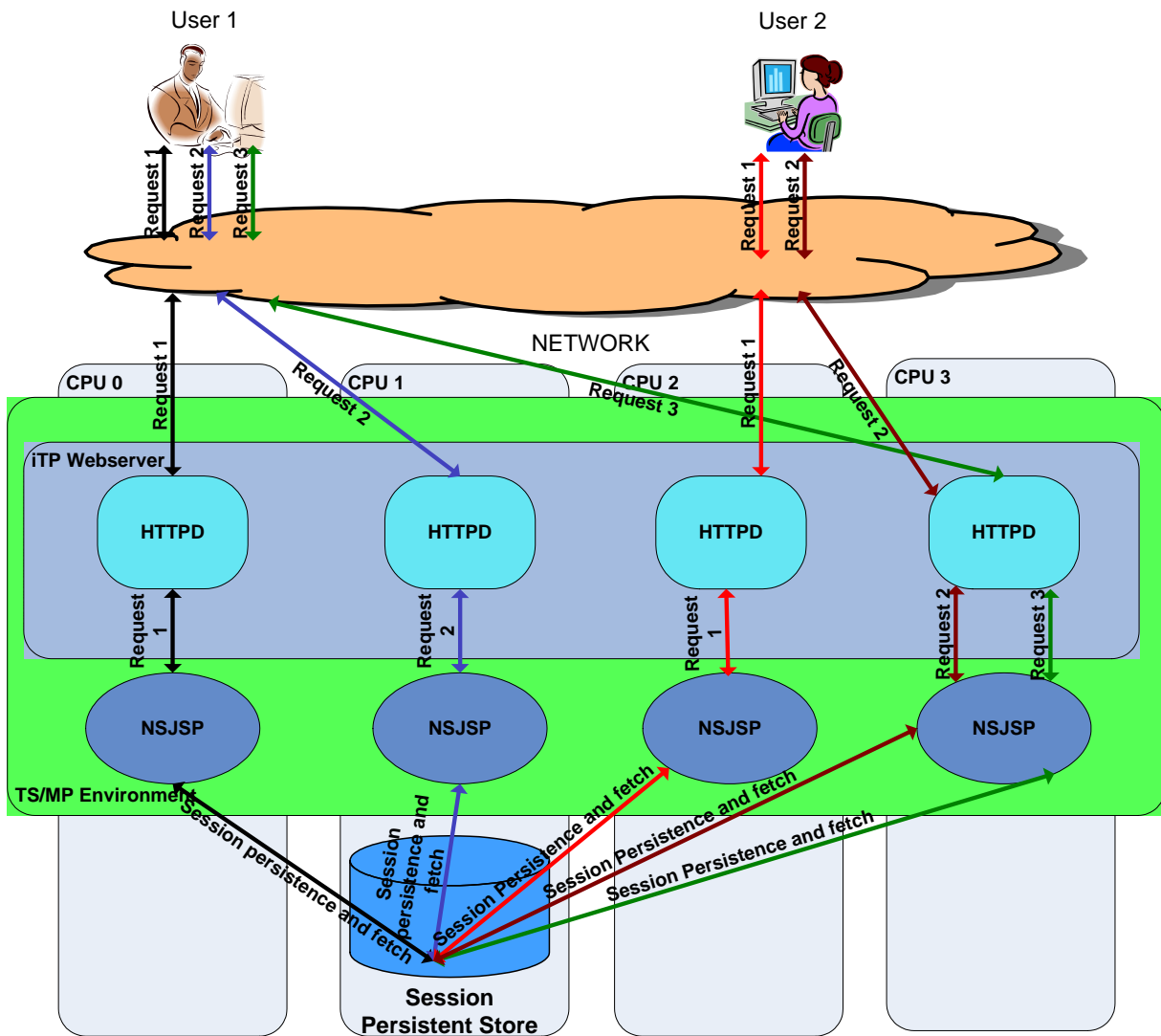
Persistent Manager Sessions (SessionBasedLoadBalancing = false)

You can also configure NSJSP such that, each NSJSP process in the NSJSP Server Class stores its session objects in a persistent store that is accessible to every other NSJSP process. This will enable any request from any HTTPD process to be serviced by any NSJSP process in the server class. In this case, session objects are saved to a persistent store (such as a database) after processing each request within a session. In this case, session based load balancing is said to be disabled. To use persistent sessions, you must set the value of `SessionBasedLoadBalancing` to `false`, and you must configure a persistent manager. For more information, see [Configuring a Persistent Manager](#) on page 3-81.

[Figure 3-6](#) shows request routing, session object fetching and storing session objects upon request completion, by the behavior of NSJSP when the persistent store is

configured.

Figure 3-6. Session Object Handling by NSJSP with the Persistent Store Configuration



In [Figure 3-6](#), user requests are not bound to any particular NSJSP process. An NSJSP process upon receiving a request fetches the session object from the persistent store, forms the response, saves the session object back to the store, and returns the response.

Configuring a Persistent Manager

To configure a persistent manager, complete the following steps:

1. [Set SessionBasedLoadBalancing to false](#)

2. [Configure the Manager Element](#)
3. [Create the Persistent Store](#)
4. [Configure the Persistent Store](#)

Set SessionBasedLoadBalancing to false

Set the SessionBasedLoadBalancing parameter in the installation-specific servlet.config file in <NSJSP_HOME>/conf directory to false.

[Example 3-64](#) shows an Arglst from an installation-specific servlet.config file with SessionBasedLoadBalancing set to false.

Example 3-64. An Arglst from a servlet.config file with SessionBasedLoadBalancing set to false

```
Arglst -Xmx64m -Xss128k -Xnoclassgc \
-Dcom.tandem.servlet.CONTEXT_PREFIXES=/scA \
-Dcatalina.home=$env(NSJSP_HOME) \
-Dcatalina.base=$env(NSJSP_HOME) \
-Djava.io.tmpdir=$env(NSJSP_HOME)/temp \
-DSessionBasedLoadBalancing=false
org.apache.catalina.startup.Bootstrap start
```

Configure the Manager Element

Configure the Manager element in the <NSJSP_HOME>/conf/context.xml file. The default manager configuration in the <NSJSP_HOME>/conf/context.xml file is commented out. Ensure that the className attribute is set to com.tandem.servlet.catalina.session.NSJSPPersistentManager.

[Example 3-65](#) shows the sample configuration of the Manager element in the context.xml file.

Example 3-65. The Sample Configuration of the Manager Element

```
<Manager
  className="com.tandem.servlet.catalina.session.NSJSPPersistentManager">
  <Store
    className="com.tandem.servlet.catalina.session.NonStopSQLJDBCStore"
    driverName="com.tandem.sqlmx.SQLMXDriver"
    connectionURL="jdbc:sqlmx:"
    sessionTable="nsjspcat.nsjspsch.SessData"
    sessionIdCol="session_id"
    sessionProcessNameCol="process_name"
    sessionRecNumberCol="rec_number"
    sessionAppCol="app_name"
    sessionDataCol="session_data" sessionValidCol="valid"
    sessionMaxInactiveCol="maxinactiveinterval"
    sessionLastAccessedCol="lastaccessed"/>
  </Manager>
```

As shown in [Example 3-65](#), the Manager is configured with a nested store element.

[Table 3-19](#) lists the attributes for the Manager element.

Table 3-19. Attribute List for the Manager element (page 1 of 2)

Attribute	Description	Default value
algorithm	Name of the Message Digest algorithm used to calculate session identifiers produced by this Manager. This value must be supported by the <code>java.security.MessageDigest</code> class. If the <code>algorithm</code> attribute is not specified, the default value is MD5.	
className	Java class name of the implementation to use. This class must implement the <code>org.apache.catalina.Manager</code> interface. The value should be <code>com.tandem.servlet.catalina.session.NSJSPPersistentManager</code> .	<code>com.tandem.servlet.catalina.session.NSJSPPersistentManager</code>
entropy	A string value that is used when seeding the random number generator used to create session identifiers for this Manager. If the <code>entropy</code> attribute is not specified, an internal value is calculated, but a long string value should be specified in security conscious environments.	
maxActiveSessions	Sessions are not kept in memory, so this attribute has no significance for the Persistent Manager.	
maxIdleBackup	Sessions are not kept in memory, so this attribute has no significance for the Persistent Manager.	
maxIdleSwap	Sessions are not kept in memory, so this attribute has no significance for the Persistent Manager.	
minIdleSwap	Sessions are not kept in memory, so this attribute has no significance for the Persistent Manager.	
maxInactiveInterval	<p>The maximum time interval, in seconds, between client requests before a session is invalidated.</p> <p>This attribute provides the initial value whenever a new session is created, but the interval may be dynamically altered by a servlet via the <code>setMaxInactiveInterval</code> method of the <code>HttpSession</code> object.</p> <p>For more information, see maxInactiveInterval on page 3-84.</p>	

Table 3-19. Attribute List for the Manager element (page 2 of 2)

Attribute	Description	Default value
randomClass	Java class name of the <code>java.util.Random</code> implementation class to use. If <code>randomClass</code> attribute is not specified, the default value is <code>java.security.SecureRandom</code> .	
saveOnRestart	Sessions are not kept in memory, so this attribute has no significance for the Persistent Manager.	
sessionIdLength	The length of session IDs created by this Manager, excluding any JVM route information used for load balancing. If this attribute is not specified, the default value is 16.	

Considerations for Configuring the Manager Element

`maxInactiveInterval`

This variable is used to set the maximum time interval, in seconds, between client requests before a session is invalidated. However, it should be noted that in the current version of NSJSP, during a context start or restart this variable is always overridden by the value specified in the `session-timeout` element in the application's deployment descriptor (`web.xml`). If the application's deployment descriptor does not explicitly specify a `session-timeout` element, the value is taken from the `<NSJSP_HOME>/conf/web.xml` file as shown in [Example 3-66](#). The default value for `session-timeout`, which is expressed in minutes, is 30.

Example 3-66. Default Configuration of `session-timeout` in the `<NSJSP_HOME>/conf/web.xml` File

```
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
```

It is possible to set the value of this variable at run time using the MBeans feature of the NSJSP Manager. The value set at run time will take effect immediately but will not be persisted and will not be available across web application context restarts.

Create the Persistent Store

To configure NSJSP for persistent sessions, create a NonStop SQL database (catalog and table) to store the persistent session data. The following SQL scripts are used to create a table to store persistent session data:

- For SQL/MP: `nsjsp_createSessionStore_mp.sql`
- For SQL/MX: `nsjsp_createSessionStore_mx.sql`

To create the NonStop SQL database for storing the persistent session data, perform the task for the database product that will be used to create and access the persistent store table:

- For the NonStop SQL/MP database:

Make a copy of the `nsjsp_createSessionStore_mp.sql` file and replace all occurrences of `=TheT1222SessionCatalog` with the Guardian location (of the form `$Volume.SubVol`) where you want the persistent session catalog and table to be created. This subvolume (disk) should be a TMF-audited data volume.

Use the following OSS command to create the table:

```
osh> gtacl -p sqlci <
nsjsp_createSessionStore_mp.sql.your_copy
```

[Example 3-67](#) shows the SQL/MP script to create the persistent store.

Example 3-67. SQL/MP script to create a Persistent Store

```
create catalog $data01.nsjspcat secure "0000";
--
-- Create the NonStop(tm) SQL Table in the above catalog for storing the
-- NonStop(tm) Servlets for JavaServer Pages(tm) persistent session
-- data.
--

create table $data01.nsjspcat.sessdata (
  session_id          CHAR(48)          NO DEFAULT  NOT NULL,
  process_name        CHAR(8)           NO DEFAULT  NOT NULL,
  rec_number          INTEGER UNSIGNED  NO DEFAULT  NOT NULL,
  valid               SMALLINT UNSIGNED NO DEFAULT,
  maxinactiveinterval INTEGER           NO DEFAULT,
  lastaccessed        LARGEINT          NO DEFAULT,
  app_name            VARCHAR(150)      NO DEFAULT  NOT NULL,
  session_data        VARCHAR(3600)     CHARACTER SET ISO88591,
  primary key (session_id, process_name, rec_number) )
catalog $data01.nsjspcat;
```

- For the NonStop SQL/MX database:

Make a copy of the `nsjsp_createSessionStore_mx.sql` file and replace the `=TheT1222SessionCatalog` string with a valid catalog name. Replace the `=TheT1222SessionSchema` string with a valid schema name.

Use the following OSS command to create the mx table:

```
osh> mxci < nsjsp_createSessionStore_mx.sql.your_copy
```

[Example 3-68](#) shows the SQL/MX script to create the persistent store.

Example 3-68. SQL/MX script to create a Persistent Store

```

create catalog nsjspcat;
set catalog nsjspcat;
create schema nsjspsch;
set schema nsjspsch;
--
-- Create the NonStop(tm) SQL Table in the above catalog for storing the
-- NonStop(tm) Servlets for JavaServer Pages(tm) persistent session data.
--

create table SessData (
    session_id          CHAR(48)          NO DEFAULT    NOT NULL,
    process_name        CHAR(8)           NO DEFAULT    NOT NULL,
    rec_number          INTEGER UNSIGNED  NO DEFAULT    NOT NULL,
    valid               SMALLINT UNSIGNED NO DEFAULT,
    maxinactiveinterval INTEGER           NO DEFAULT,
    lastaccessed        LARGEINT          NO DEFAULT,
    app_name            VARCHAR(150)       NO DEFAULT    NOT NULL,
    session_data        VARCHAR(3600)      CHARACTER SET ISO88591,
    primary key (session_id, process_name, rec_number) );

```

Configure the Persistent Store

When configuring a `Manager` element for session persistence, a `Store` element must be configured as a child element of the `Manager` element.

[Example 3-69](#) shows the sample configuration of the `Store` element.

Example 3-69. Sample Configuration of the Store Element

```

<Manager
  className="com.tandem.servlet.catalina.session.NSJSPPersistentManager">
  <Store
    className="com.tandem.servlet.catalina.session.NonStopSQLJDBCStore"
    driverName="com.tandem.sqlmx.SQLMXDriver"
    connectionURL="jdbc:sqlmx:"
    sessionTable="nsjspcat.nsjspsch.SessData"
    sessionIdCol="session_id"
    sessionProcessNameCol="process_name"
    sessionRecNumberCol="rec_number"
    sessionAppCol="app_name"
    sessionDataCol="session_data" sessionValidCol="valid"
    sessionMaxInactiveCol="maxinactiveinterval"
    sessionLastAccessedCol="lastaccessed"/>
  </Store>
</Manager>

```

[Table 3-20](#) lists the attributes for the `Store` element.

Table 3-20. Attribute List for the `store` Element (page 1 of 2)

Attribute	Description	Default value
<code>checkInterval</code>	Not applicable to NSJSP.	
<code>className</code>	Java class name of the implementation to use. This class must implement the <code>org.apache.catalina.Store</code> interface. This must be <code>com.tandem.servlet.catalina.session.NonStopSQLJDBCStore</code> .	<code>com.tandem.servlet.catalina.session.NonStopSQLJDBCStore</code>
<code>driverName</code>	Java class name of the JDBC driver to be used.	<code>com.tandem.sqlmx.SQLMXDriver</code>
<code>connectionURL</code>	The connection URL that will be handed to the configured JDBC driver to establish a connection to the database containing the session table.	<code>jdbc:sqlmx:</code>
<code>sessionTable</code>	Name of the database table to be used for storing swapped out sessions. Note. You must replace this value with the appropriate catalog, schema, and tablename.	
<code>sessionIdCol</code>	Name of the database column, contained in the specified session table, that contains the session identifier of the swapped out session.	<code>session_id</code>
<code>sessionProcessNameCol</code>	Name of the database column that contains the process name that created this session.	<code>process_name</code>
<code>sessionRecordCol</code>	The index number of each record when a session must be split into multiple rows in the table.	<code>rec_number</code>
<code>sessionAppCol</code>	Name of the database column, contained in the specified session table, that contains the Engine, Host, and Web application context name in the format <code>/<Engine_name>/<Host_name>/<Context_name></code> .	<code>app_name</code>
<code>sessionDataCol</code>	Name of the database column, contained in the specified session table, that contains the serialized form of all session attributes for a swapped out session.	<code>session_data</code>

Table 3-20. Attribute List for the store Element (page 2 of 2)

Attribute	Description	Default value
<code>sessionValidCol</code>	Name of the database column, contained in the specified session table, that contains a flag indicating whether this swapped out session is still valid or not.	<code>valid</code>
<code>sessionMaxInactiveCol</code>	Name of the database column, contained in the specified session table, that contains the <code>maxInactiveInterval</code> property of this session.	<code>maxinactiveinterval</code>
<code>sessionLastAccessedCol</code>	Name of the database column, contained in the specified session table, that contains the <code>lastAccessedTime</code> property of this session.	<code>lastaccessed</code>

Mixed-Mode Sessions

In NSJSP 6.1, you can also configure a mixed-mode session. In a mixed-mode session, a session can be configured so that it is kept in memory and also saved to a persistent store at regular intervals.

When configured this way, the sessions exhibit the sticky sessions behavior (similar to the behavior when `SessionBasedLoadBalancing` is enabled for in memory sessions). This means that all requests for a particular session are routed to the same Servlet Server Class process. The sessions are also persisted to the store at regular intervals. The interval is calculated based on the `backgroundProcessorDelay` of the manager's parent element (in this case it is the context). If this attribute is not specified for the context, the Host attribute (the parent element of the context) is used. If the Host does not specify this attribute, then the Engine background processor delay is used.

In the default configuration, the `backgroundProcessorDelay` of the Engine is 60 seconds and the Host and the contexts do not explicitly configure the `backgroundProcessorDelay` attribute. The in memory sessions are checked if they are eligible for persistence every $(60 \times 6) = 360$ seconds. The number 6 is the value of the `processExpiresFrequency` attribute of the persistence manager.

The following are the checks performed to decide if a session must be persisted to the store or not:

1. If the time since the last access is greater than the value indicated by `maxIdleSwap`, the session is persisted to the store and is removed from memory. For more information on `maxIdleSwap`, see [Table 3-21](#).
2. If the number of in-memory sessions is greater than the number indicated by `maxActiveSessions`, then the sessions that are idle for longer than the time indicated by `minIdleSwap` are persisted to the store. In this case, after the

session is persisted, it is removed from process memory. Sessions are persisted till the number of active sessions is less than `maxActiveSessions`. For more information on `maxActiveSessions` and `minIdleSwap`, see [Table 3-21](#).

3. All the remaining in-memory sessions are checked to see if the time since they were last accessed is greater than `maxIdleBackup`. If so, the session is written to the persistent store, but is not removed from process memory. For more information on `maxIdleBackup`, see [Table 3-21](#).

Configuring Mixed-Mode Sessions

To configure a mixed-mode session, complete the following steps:

1. [Set SessionBasedLoadBalancing to true](#)
2. [Configure the Manager Element](#)
3. [Create the Persistent Store](#)
4. [Configure the Persistent Store](#)

Set SessionBasedLoadBalancing to true

Set `SessionBasedLoadBalancing` to `true` in the installation-specific `servlet.config` file in the `<NSJSP_HOME>/conf` directory.

[Example 3-70](#) shows the Arglist of an installation-specific `servlet.config` file with `SessionBasedLoadBalancing` set to `true`.

Example 3-70. An Arglist from a `servlet.config` file with `SessionBasedLoadBalancing` Set to `true`

```
Arglist -Xmx64m -Xss128k -Xnoclassgc \
-Dcom.tandem.servlet.CONTEXT_PREFIXES=/sca \
-Dcatalina.home=$env(NSJSP_HOME) \
-Dcatalina.base=$env(NSJSP_HOME) \
-Djava.io.tmpdir=$env(NSJSP_HOME)/temp \
-DSessionBasedLoadBalancing=true
org.apache.catalina.startup.Bootstrap start
```

Configure the Manager Element

Configure the `Manager` element in the `<NSJSP_HOME>/conf/context.xml` file. Ensure that the `className` attribute is set to `com.tandem.servlet.catalina.session.NSJSPPersistentManager`.

[Example 3-71](#) shows the sample configuration of the `Manager` Element.

Example 3-71. The Sample Configuration of the `Manager` Element

```
<Manager
  className="com.tandem.servlet.catalina.session.NSJSPPersistentManager">

  <Store
    className="com.tandem.servlet.catalina.session.NonStopSQLJDBCStore"
    driverName="com.tandem.sqlmx.SQLMXDriver"
    connectionURL="jdbc:sqlmx:"
    sessionTable="nsjspcat.nsjpsch.SessData"
    sessionIdCol="session_id"
    sessionProcessNameCol="process_name"
    sessionRecNumberCol="rec_number"
    sessionAppCol="app_name"
    sessionDataCol="session_data" sessionValidCol="valid"
    sessionMaxInactiveCol="maxinactiveinterval"
    sessionLastAccessedCol="lastaccessed"/>

  </Manager>
```

As shown in [Example 3-71](#), the `Manager` is configured with a nested store element.

[Table 3-21](#) lists the attributes of the `Manager` element that have a different meaning when compared to a configuration where all sessions are saved to a persistent store.

Table 3-21. Attribute List of the Manager Element.

Attribute	Description
<code>maxActiveSessions</code>	Maximum number sessions that can be kept in-memory. If this attribute is not specified, it is set to -1.
<code>maxIdleBackup</code>	The time interval (in seconds) since the last access to a session before it is eligible for being persisted to the session store, or -1 to disable this feature. By default, this feature is disabled.
<code>maxIdleSwap</code>	The time interval (in seconds) since the last access to a session before it should be persisted to the session store, and removed from the server's memory and written to the database, or -1 to disable this feature. If this feature is enabled, the time interval specified must be equal to or longer than the value specified for <code>maxIdleBackup</code> . By default, this feature is disabled.
<code>minIdleSwap</code>	The time interval (in seconds) since the last access to a session before it will be eligible to be persisted to the session store, and then removed from the server's memory and written to the database, or -1 for this swapping to be available at any time. If specified, this value must be less than that specified by <code>maxIdleSwap</code> .
<code>maxInactiveInterval</code>	<p>The maximum time interval, in seconds, between client requests before a session is invalidated.</p> <p>This attribute provides the initial value whenever a new session is created, but the interval may be dynamically altered by a servlet via the <code>setMaxInactiveInterval</code> method of the <code>HttpSession</code> object.</p> <p>For more information, see maxInactiveInterval on page 3-91.</p>

Considerations for Configuring Mixed-Mode Sessions

`maxInactiveInterval`

This variable is used to set the maximum time interval, in seconds, between client requests before a session is invalidated. However, it should be noted that in the current version of NSJSP, during a context start or restart this variable is always overridden by the value specified in the `session-timeout` element in the application's deployment descriptor (`web.xml`). If the application's deployment descriptor does not explicitly specify a `session-timeout` element, the value is taken from the `<NSJSP_HOME>/conf/web.xml` file as shown in [Example 3-72](#). The default value for `session-timeout`, which is expressed in minutes, is 30.

Example 3-72. Default Configuration of `session-timeout` in the `<NSJSP_HOME>/conf/web.xml` File

```
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
```

It is possible to set the value of this variable at run time using the modify MBeans feature of the NSJSP Manager. The value set at run time will take effect immediately but will not be persisted and will not be available across web application context restarts.

Determining the Storage Capacity of the Persistent Store

It is important to determine how many sessions may need to be stored in the persistent store and then size the persistent store accordingly. Because the size of a session varies from one application to the other, there cannot be a simple formula for sizing the persistent store. The following example explains the factors that need to be considered while sizing a session store.

If at maximum capacity the application produces 20,000 sessions per day and if the session cleanup script is run daily to clean up sessions older than 5 days, the session store should have capacity to hold a minimum of $5 \times 20,000 (=100,000)$ session objects. This example assumes that the session expiry duration (`session-timeout`) is less than 5 days. For more information on cleaning session table, see the [Cleaning the NonStop SQL Session Data](#) on page 3-94.°@

For any given application, perform the following steps to decide the size of the session store.

1. Run your application to create 10% of the maximum number of session objects that are to be stored in the session store. In the above example it is 10% of 100,000 (=10,000).
2. After the sessions are created, locate the physical file of the session table and determine its size. For more information, [Determining the size of the session table](#) on page 3-92. Ensure that the size of the table is at least 10 times greater than this size.

The SQL command that can be used to count the number of sessions in the table is:

```
select count (distinct session_id) from sessdata;
```

Determining the size of the session table

To determine the size of the session table, complete the following steps:

1. Obtain the disk file name for the sessions table. For SQL/MX, use the `showddl` command. Enter `showddl` followed by the `<catalog>.<schema>.<tablename>` of the session table at the SQL/MX command interface (mxci) prompt to determine the disk file name. [Example 3-73](#)

shows the sample output of a `showddl` command.

Example 3-73. The Sample Output of a `showddl` Command

```
>>showddl nsjspcat.nsjspsch.sessdata;
CREATE TABLE NSJSPCAT.NSJSPSCH.SESSDATA
(
  SESSION_ID                CHAR(48) CHARACTER SET ISO88591
  COLLATE
    DEFAULT NO DEFAULT -- NOT NULL NOT DROPPABLE
  , PROCESS_NAME            CHAR(8) CHARACTER SET ISO88591
  COLLATE
    DEFAULT NO DEFAULT -- NOT NULL NOT DROPPABLE
  , REC_NUMBER              INT UNSIGNED NO DEFAULT
    -- NOT NULL NOT DROPPABLE
  , VALID                   SMALLINT UNSIGNED NO DEFAULT
  , MAXINACTIVEINTERVAL    INT NO DEFAULT
  , LASTACCESSED            LARGEINT NO DEFAULT
  , APP_NAME                VARCHAR(150) CHARACTER SET
ISO88591
    COLLATE DEFAULT NO DEFAULT -- NOT NULL NOT DROPPABLE
  , SESSION_DATA            VARCHAR(3600) CHARACTER SET
ISO88591
    COLLATE DEFAULT DEFAULT NULL
  , CONSTRAINT NSJSPCAT.NSJSPSCH.SESSDATA_279712597_5623 PRIMARY KEY
    (SESSION_ID ASC, PROCESS_NAME ASC, REC_NUMBER ASC) NOT
DROPPABLE
  , CONSTRAINT NSJSPCAT.NSJSPSCH.SESSDATA_385269497_5623 CHECK
    (NSJSPCAT.NSJSPSCH.SESSDATA.SESSION_ID IS NOT NULL AND
    NSJSPCAT.NSJSPSCH.SESSDATA.PROCESS_NAME IS NOT NULL AND
    NSJSPCAT.NSJSPSCH.SESSDATA.REC_NUMBER IS NOT NULL AND
    NSJSPCAT.NSJSPSCH.SESSDATA.APP_NAME IS NOT NULL) NOT DROPPABLE
)
LOCATION \POS02.$DATA00.ZSDG8WWM.JM2SKH00
NAME POS02_DATA00_ZSDG8WWM_JM2SKH00
STORE BY (SESSION_ID ASC, PROCESS_NAME ASC, REC_NUMBER ASC)
;
```

2. Use the `fup info <filename>,detail` command at a TACL prompt to display the current size of the file. [Example 3-74](#) shows the sample output of a `fup info <filename>,detail` command.

Example 3-74. The Sample Output of a fup info <filename>,detail Command

```

3> fup info $DATA00.ZSDG8WWM.X8BXVN00 ,detail

$DATA00.ZSDG8WWM.X8BXVN00          26 Apr 2010, 10:44
SQL ANSI TABLE
ANSI NAME NSJSPCAT.NSJSPSCH.SESSDATA
RESOURCE FORK \POS02.$DATA00.ZSDG8WWM.X8BXVN01
SYSTEM METADATA \POS02.$OSS03.ZSD0
VERSION 1200
TYPE K
FORMAT 2
CODE 550
EXT ( 16 PAGES, 64 PAGES, MAXEXTENTS 160 )
PACKED REC 3852
BLOCK 4096
KEY ( COLUMN 0, ASC ,
      COLUMN 1, ASC ,
      COLUMN 2, ASC )
PART ( 0, \POS02.$DATA00.ZSDG8WWM.X8BXVN00 )
PART ( 1, \POS02.$DATA01.ZSDG8WWM.PWHXVN00 )
PART ( 2, \POS02.$FC31.ZSDG8WWM.PWJXVN00 )
PART ( 3, \POS02.$FC34.ZSDG8WWM.QS9XVN00 )
AUDIT
BUFFERED
AUDITCOMPRESS
OWNER 101,2
SECURITY (RWEF): *SQL
DATA MODIF: 5 Apr 2010, 15:12
CREATION DATE: 29 Mar 2010, 12:16
REDEFINITION DATE: 29 Mar 2010, 12:16
LAST OPEN: 5 Apr 2010, 15:11
EOF: 2224128 (10.7% USED)
EXTENTS ALLOCATED: 71
INDEX LEVELS: 2
PARTITION ARRAY FORMAT2ENABLED

```

Cleaning the NonStop SQL Session Data

Sessions saved by the NonStopSQLJDBCStore to a NonStop SQL database may never get deleted and will remain as orphan sessions when:

- The sessions are saved during an NSJSP container restart and are never accessed after the restart, or
- The sessions are backed up, or swapped out to the NonStop SQL database and are never accessed again after an NSJSP container fails and is restarted.

As a result, over a period of time, the session database may become very large. Therefore, It is important to clean the session store periodically to prevent the table from filling up, which can lead to session creation errors.

The <NSJSP_HOME>/conf/nsjsp_cleanSessionData script enables you to delete the sessions that have expired prior to a specified number of days (the nDays parameter) - see [Example 3-75](#) for details. This script should be run periodically to clean the session. You may run this script manually as shown in [Example 3-76](#), or preferably it should be automated through a job scheduler such as cron or NetBatch.

[Example 3-75](#) shows the SQL session data cleanup script.

Example 3-75. SQL Session Data Cleanup Script

Usage: nsjsp_cleanSessionData nDays

where nDays: Number of days for which session data is to be preserved/saved. Sessions that have expired more than 'nDays' ago will be deleted. A value of zero (0) will delete all the expired sessions.

[Example 3-76](#) shows an invocation of the nsjsp_cleanSessionData script.

Example 3-76. Using the nsjsp_cleanSessionData Script

```
osh> /usr/tandem/webserver/servlet_jsp/conf: nsjsp_cleanSessionData 0
```

```
NonStop(tm) Servlets for JavaServer Pages(tm)
```

```
Persistent Sessions Cleanup Script
```

```
T1222 v6.0
```

Cleans up the persistent session data stored in a NonStop(tm) SQL database for NonStop(tm) Servlets for JavaServer Pages(tm).

Persistent Session Store type is SQL/MX [y(default) or n]:

Please enter the Persistent Session Catalog [T1222CAT.T1222SCH] :
nsjspcat.nsjspsch

Persistent Sessions Catalog = nsjspcat.nsjspsch
Persistent Sessions Table = nsjspcat.nsjspsch.SessData

```
Hewlett-Packard NonStop(TM) SQL/MX Conversational Interface 2.2
(c) Copyright 2006 Hewlett-Packard Development Company, LP.
>>DELETE FROM nsjspcat.nsjspsch.SessData
+>WHERE (Juliantimestamp(current) -
+>      Juliantimestamp(timestamp '1970-01-01:00:00:00.00') -
+>      (1000 * lastaccessed) ) >
+>      (1000 * 1000 * (maxinactiveinterval + (0 * 24 * 60 * 60)) );
--- 0 row(s) deleted.
```

Partitioning the Session Table

In applications where there is a lot of session activity and many sessions are created, the time consumed for session related database activity can be considerably reduced by partitioning the session table. Because the value of the session ID is a randomly generated string, a hash partition will be efficient in distributing data across multiple partitions. [Example 3-77](#) shows how to create a session table with four partitions that are spread across four different disks. For better performance, it is recommended that you configure the partitions on disks with their disk processes in different processors.

Example 3-77. Creating A Session Data Table with Four Partitions Across Four Different Disks

```
create table sessdata (
  session_id      CHAR(48)          NO DEFAULT  NOT NULL,
  process_name    CHAR(8)           NO DEFAULT  NOT NULL,
  rec_number      INTEGER UNSIGNED  NO DEFAULT  NOT NULL,
  valid           SMALLINT UNSIGNED NO DEFAULT,
  maxinactiveinterval INTEGER      NO DEFAULT,
  lastaccessed    LARGEINT          NO DEFAULT,
  app_name        VARCHAR(150)      NO DEFAULT  NOT NULL,
  session_data    VARCHAR(3600)     CHARACTER SET ISO88591,
  primary key (session_id, process_name, rec_number))
  location $DATA03
  hash partition by (session_id)(
    add location $DATA00,
    add location $DATA01,
    add location $DATA02
  );
```

Configuring the Manager Element

The `Manager` element represents the session manager that will be used to create and maintain HTTP session objects as requested by a web application.

A `Manager` element is always configured as a child of the `Context` element. Each application can configure its own `Manager` in the context definition in the `META-INF/context.xml` file of the application. [Example 3-78](#) shows a sample context configured with a persistent manager.

Note. Before using a persistent manager, the session table must be created. For information on how to create the session table, see [Create the Persistent Store](#) on page 3-84.

For all applications that do not define a `Manager` explicitly, the `Manager` configured in the `<NSJSP_HOME>/conf/context.xml` file will be used as the default manager. By default, the `NSJSPStandardManager` is used. This is an in-memory session manager. [Example 3-79](#) shows the default `Manager` configuration in the `<NSJSP_HOME>/conf/context.xml` file.

[Example 3-78](#) shows a sample context configured with a persistent manager.

Example 3-78. Sample Context Configured with a Persistent Manager

```
<Context docbase="examples" reloadable="false">
  <Manager
    className="com.tandem.servlet.catalina.session.NSJSPPersistentManager">
    <Store
      className="com.tandem.servlet.session.NonStopSQLJDBCStore"
      driverName="com.tandem.sqlmx.SQLMXDriver"
      connectionURL="jdbc:sqlmx:"
      sessionTable="nsjspcat.nsjpsch.SessData"
      sessionIdCol="session_id"
      sessionProcessNameCol="process_name"
      sessionRecNumberCol="rec_number"
      sessionAppCol="app_name"
      sessionDataCol="session_data" sessionValidCol="valid"
      sessionMaxInactiveCol="maxinactiveinterval"
      sessionLastAccessedCol="lastaccessed" />
    </Store>
  </Manager>
</Context>
```

[Example 3-79](#) shows the default Manager configuration in the `<NSJSP_HOME>/conf/context.xml` file.

Note. A sample Persistent Manager configuration is provided. However, it is commented out.

Example 3-79. Default Configuration of the Manager Element

```

<!--
  The contents of this file will be loaded for each web application
-->

<Context>
  <!-- Default set of monitored resources -->
  <WatchedResource>WEB-INF/web.xml</WatchedResource>

  <!--
    NSJSP specific Web application loader.
    All contexts in NSJSP must be loaded using this application loader.
  -->

  <Loader
    className="com.tandem.servlet.catalina.loader.NSJSPWebappLoader"/>

  <!--
    NSJSP specific Standard Manager.
    The Manager does no session persistence.
  -->

  <!--
    Following is the Default Manager element configuration
  -->

  <Manager pathname=""
    className="com.tandem.servlet.catalina.session.NSJSPStandardManager"/>

  <!--
    Following is a sample Manager element configuration
  -->

  <!--
    <Manager
    className="com.tandem.servlet.catalina.session.NSJSPPersistentManager">
      <Store
    className="com.tandem.servlet.catalina.session.NonStopSQLJDBCStore"
      driverName="com.tandem.sqlmx.SQLMXDriver"
      connectionURL="jdbc:sqlmx:"
      sessionTable="<catalog>.<schema>.<tablename>"
      sessionIdCol="session_id"
      sessionProcessNameCol="process_name"
      sessionRecNumberCol="rec_number"
      sessionAppCol="app_name"
      sessionDataCol="session_data" sessionValidCol="valid"
      sessionMaxInactiveCol="maxinactiveinterval"
      sessionLastAccessedCol="lastaccessed"/>
    </Manager>
  -->

</Context>

```

4 Managing NSJSP

This chapter describes the tools that are used to manage NSJSP installations and the web applications deployed in those NSJSP installations. This chapter covers the following topics:

- [NSJSP Manager Application](#)
- [Admin Web Application](#)
- [Manager Web Application](#)
- [Operations Using the Command-line Interface](#)
- [Manual Deployment and Undeployment of Web Applications](#)
- [Comparison of the Management Applications](#)
- [Single Point of Management Using the NSJSP Manager](#)
- [The Architecture of the NSJSP Manager](#)

NSJSP Manager Application

Starting with the NSJSP 6.1 release, a new management web application, the NSJSP Manager, is provided. This section describes the following topics related to the NSJSP Manager application:

- [Overview](#)
- [NSJSP Manager Features](#)
- [NSJSP Security](#)
- [NSJSP Manager Operations](#)

Overview

The NSJSP Manager is a web-based, Graphical User Interface (GUI) tool that you can use to manage an NSJSP 6.1 installation within an iTP Secure WebServer environment.

The NSJSP Manager must be installed, before it can be used. The NSJSP `setup` script can be used to create an NSJSP Manager installation. Also, when creating an NSJSP installation, the `setup` script verifies if the NSJSP Manager has already been installed, and if not, the user is given the option to install the NSJSP Manager at that time. For more information on installing the NSJSP Manager application and NSJSP 6.1, see Chapter [2, Installing NSJSP](#).

The NSJSP Manager can manage server classes in a single iTP Secure WebServer PATHMON or in a Pathway domain with two PATHMONs when using an iTP Secure WebServer configured for online-upgrade.

The NSJSP Manager also enables you to deploy and manage web applications running in the default `localhost` or in any additional virtual Hosts specified within the `server.xml` file in an NSJSP 6.1 installation directory.

NSJSP Manager Features

The NSJSP Manager application enables you to do the following:

- Manage NSJSP 6.1 Server Classes
 - Start, freeze, thaw, and stop NSJSP 6.1 Server Classes
 - Display NSJSP 6.1 Server Class information:
 - System execution environment information
 - Configuration parameters
 - Server class process information
 - NSJSP connector statistics
 - Server class statistics

Note. Each NSJSP 6.1 installation has two server classes and with the NSJSP Manager, you can select and manage any server class, from any of the NSJSP 6.1 installations under the same ITP Secure WebServer installation.

- Manage web applications
 - List all deployed web applications
 - View details of web applications
 - Requests
 - Session information
 - HTTP method statistics
 - Context and deployment descriptors
 - Servlet mappings
 - Filters
 - Initialization parameters
 - Perform tasks, such as, start, stop, reload, and undeploy web applications.
 - Deploy web applications
- Access NSJSP and Apache Tomcat MBeans:
 - List the MBean tree structure
 - Select and view MBean attributes

- Compare MBean attributes across processes
- Modify MBean attributes

For more information on how to use the NSJSP Manager application, see the [NSJSP Manager Operations](#) on page 4-3.

NSJSP Security

To log in to the NSJSP Manager application, you require a user name and password. By default, a user called `admin` is used to log in to the NSJSP Manager application. The password assigned during the NSJSP Manager installation is the password for the `admin` user.

You can add new users by editing the `<NSJSP manager installation directory>/conf/ nsjsp-users.xml` file. All users must be assigned a role: `manager` or `admin`. The `manager` role enables you to manage or monitor the web applications installed in NSJSP. The `admin` role enables you to perform the operations using the Admin Web application. Both `manager` and the `admin` roles authorize you to use the NSJSP Manager application.

Note. The `admin` user has both `admin` and the `manager` roles assigned.

For more information on using the Admin Web application, see the [Admin Web Application](#) on page 4-56.

For more information on NSJSP security, see [Securing Web Applications](#) on page 7-1.

NSJSP Manager Operations

This section describes how to use the NSJSP Manager application to perform management tasks. This section covers the following topics:

- [Logging in to the NSJSP Manager Application](#)
- [Selecting the Server Class and Host](#)
- [NSJSP Manager Functions](#)
- [Viewing Information about Web Applications](#)
- [Managing Web Applications](#)
- [Viewing Server Class Information](#)
- [Performing Server Class Operations](#)
- [Viewing MBeans](#)
- [Managing MBeans](#)
- [Deploying Web Applications](#)

Logging in to the NSJSP Manager Application

To manage the NSJSP Server Classes that are created using the `setup` script, log in to the NSJSP Manager application.

To log in to the NSJSP Manager application, complete the following steps:

1. Enter the following URL:

```
http://IP address:Port number/manager/
```

For example:

```
http://15.148.2.1:1088/manager/
```

where,

15.148.2.1 specifies the *IP address* of the system on which the NSJSP Manager application is deployed.

1088 specifies the *Port number* of the iTP Secure WebServer.

The login page of the NSJSP Manager application appears.

[Figure 4-1](#) shows the NSJSP Manager application login page.

Figure 4-1. NSJSP Manager Application Login Page

2. Enter the user name and password.

Note. The default user name is admin. Use the password that you provided while running the `setup` script.

3. Click **Sign In**.

The NSJSP Manager Scope page appears.

By default, the **Scope** tab is enabled. To enable all the NSJSP Manager functions for a specific server class, you must select and set the server class and the Host.

Note. The NSJSP Manager application can only manage server classes that belong to NSJSP 6.1 installations.

Selecting the Server Class and Host

You must select the server class from the list of configured server classes. After selecting and setting the server class, the virtual Hosts that are configured for the server class in the `server.xml` file are listed. The default Host is `localhost` and that will be the only option unless the `server.xml` file is modified to include additional Hosts. Selecting a Host enables the NSJSP Manager application to manage all the web applications that are deployed on that Host.

Note.

- If a server class is not started, the list of Hosts cannot be fetched. To start the server class, you need to select a server class from the **Scope** tab and then start the server class using the **Server Class** tab. After the server class is successfully started, the list of Hosts is populated.
 - The NSJSP Manager cannot manage server classes from NSJSP 6.0 (or an earlier version) installation. Selecting a server class from an earlier version will result in an error being displayed on the page because earlier versions of NSJSP are not able to communicate with the NSJSP Manager application.
-

To select and set the server class and Host, complete the following steps:

1. From the **Server Class** list, select the server class name that you want to manage and click **Set**.

The **Server Class** tab is enabled and the **Host (in sever.xml)** list is displayed.

The following message is displayed:

```
Server class set to "server class name". Select a host, under
this server class, to manage.
```

Note. If you select the server class name that has already been set, the system displays the following message:

```
Server Class is already set to <server class name>.
```

2. From the **Host (in sever.xml)** list, select the Host name and click **Set**.

The following message is then displayed:

```
Host (in server.xml) set to "host name", with server class
"server class name".
```

For example,

```
Host (in server.xml) set to "localhost", with server class  
"SERVLETS".
```

where,

`localhost` specifies the *host name*.

`SERVLETS` specifies the *server class name*.

The **Applications**, **MBeans**, and **Deployment** tabs are now enabled.

Note. When you select the server class, only the **Server Class** tab that manages the server class is enabled. All other tabs, such as, **Applications** and **Deployment** are disabled.

The following scenario explains why only the **Server Class** tab is enabled immediately after selecting the server class.

The NSJSP operations might have shut down the server class from the command-line interface (CLI). To start this server class using the NSJSP Manager application, you must log on to the NSJSP Manager application and must select the server class under the **Scope** tab. A Host is not displayed for the selected server class since it was previously shutdown, and processes are not running on it. In such cases, the **Server Class** tab is enabled without selecting a Host, to permit starting any server class that was previously shutdown.

NSJSP Manager Functions

The main functions of the NSJSP Manager are organized as tabs in the user interface, which facilitates switching between the functions.

Each tab (set of functions) has multiple submenu items that are listed in a table within the tab.

All the tabs are enabled after logging in to the NSJSP Manager and selecting and setting the server class and the Host.

The following are the main functions of the NSJSP Manager:

- **Scope** – Used to set the scope for management, which requires selecting and setting the server class and the Host to be managed.

For information on selecting and setting a server class and Host, see [Selecting the Server Class and Host](#) on page 4-5.

- **Applications** – Enables the management of the deployed web applications. Provides screens for displaying configuration, status and statistics information for web applications deployed on the selected Host. It also provides operations for each application, such as, start, stop, reload, or undeploy.

For information on how to view the application details, see [Viewing Information about Web Applications](#) on page 4-8.

For information on how to perform operations on the web applications, see [Managing Web Applications](#) on page 4-26.

- **Server Class** – Enables display of the server class process information including status, connector and server class statistics, and configuration parameters. Also, provides TS/MP operations (for example, starting, freezing, thawing, and stopping a server class) for the server class in the selected scope.

For information on how to view the server class details, see [Viewing Server Class Information](#) on page 4-30.

For information on how to perform tasks on the server class, see [Performing Server Class Operations](#) on page 4-39.

- **MBeans** – Enables viewing all the NSJSP and Tomcat MBeans in NSJSP processes, comparing MBean attributes across processes, and modifying selected MBean attribute values in all or selected processes.

For information on how to view MBeans attributes, see [Viewing MBeans](#) on page 4-44.

For information on how to perform tasks, such as, comparing and modifying MBean attribute values, see [Managing MBeans](#) on page 4-46.

- **Deployment** – Enables the deployment of web applications.

For information on how to deploy web applications, see [Deploying Web Applications](#) on page 4-52.

The following are generic operations that you can perform using the NSJSP Manager:

- **Refresh Stats** – Obtains the latest status and statistical data from each NSJSP process.

Note. The NSJSP Manager application caches the information displayed on a screen. As a result, the data required for a particular screen is already present in the NSJSP Manager's cache. The NSJSP Manager application does not obtain the data from the individual NSJSP processes. The cache prevents unnecessary communication between the NSJSP Manager and the NSJSP processes. You can use the **Refresh Stats** option to force the NSJSP Manager application to obtain data from the individual NSJSP processes and thus refresh its cache.

- **Reset Stats** – Resets the statistics. After the reset operation is invoked, the NSJSP Manager application clears its internal cache and requests each NSJSP process to reset its statistics.

Note. Some statistical data is not reset. For example, the data in the Server Class Statistics page does not get reset since the data is obtained from PATHMON processes and is not provided by NSJSP processes.

The NSJSP Manager application displays the date and time when the last reset was performed. For example:

```
Last Reset: 03/19/10 12:49 PM (6 23:07:20)
```

where,

03/19/10 12:49 PM represents the time when the reset operation was performed.

6 23:07:20 represents the time elapsed since the last reset (6 = # of days, 23 = # hours, 07=mins, 20 seconds).

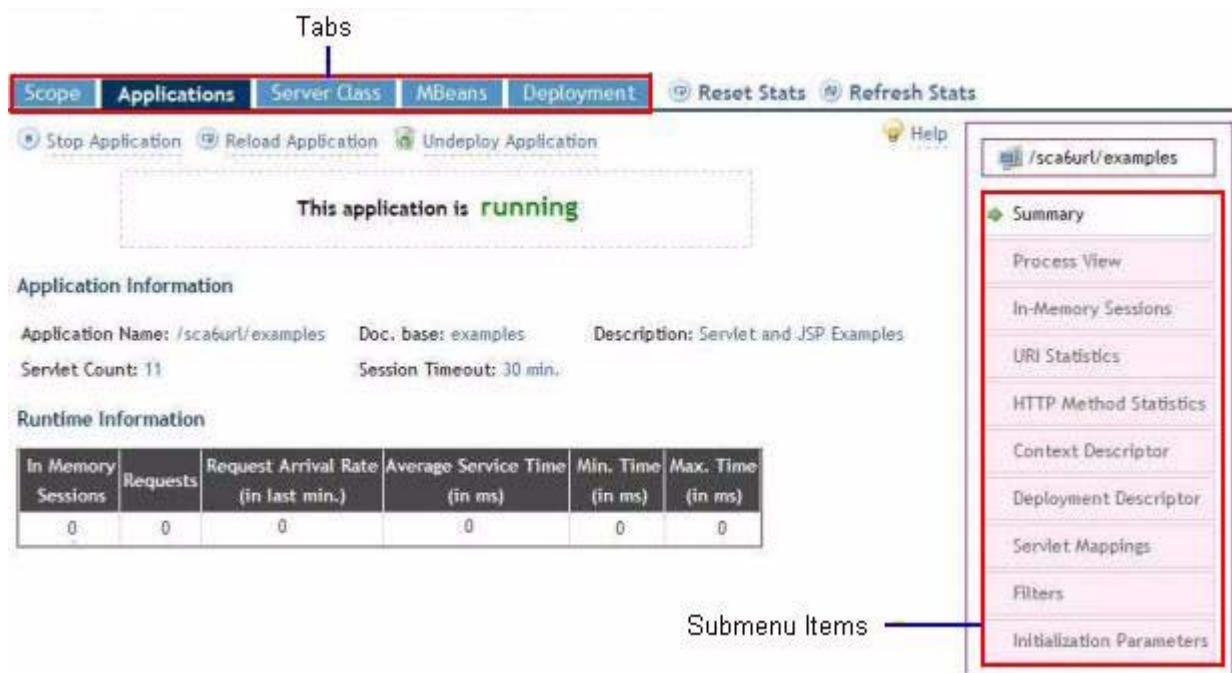
- **Help** – Includes brief information about the displayed page.

Note. The **Refresh Stats**, **Reset Stats**, and **Help** are displayed on each screen.

Note. You can sort the order of contents displayed in many tables by clicking the column heading for the column that will be sorted. The headings of columns that can be sorted are underlined.

[Figure 4-2](#) shows the NSJSP Manager application functions.

Figure 4-2. NSJSP Manager Application Functions



Viewing Information about Web Applications

The NSJSP Manager application enables viewing of the following:

- List of applications that are deployed in the Host under consideration, their status and description, the number of requests received, and the number of active sessions per application.
- Detailed information about each application, such as, number of servlets configured for an application, number of requests processed by an application, and processing time.


- Details of the applications for each NSJSP process, such as, name of the NSJSP process on which the application is running and rate at which requests are received by the application.
- Session details, such as its ID, the duration for which the session is idle.
- URI statistics, such as, the URI of an application and the time taken to process the URI.
- HTTP Method statistics, such as, a name of the HTTP method used to access an application and the number of requests processed for the HTTP method.
- Context configuration (`context.xml`) for an application, if present.
- Deployment descriptor (`web.xml`) details of an application.
- URL pattern, servlet name, and servlet class.
- List of filters that are configured for an application. The list includes the filters that are configured in the common deployment descriptor in the `<NSJSP 6.1 Installation Directory>/web.xml` file.

To view details of the applications deployed in the selected Host, click the **Applications** tab.

The Applications page appears. This page lists the applications that are deployed on the selected Host.

[Figure 4-3](#) shows a sample Applications page.

Figure 4-3. Applications Page

Scope Applications Server Class MBeans Deployment Reset Stats Refresh Stats				
				
Webapp Context Name	Deployment Status	Description	Request Count	Session Count
/sca6url	4 of 4	Welcome to NSJSP	0	0
/sca6url/admin	4 of 4	Tomcat Administration Application	0	0
/sca6url/bankapp	4 of 4	NSJSP BankDemo Sample Web Application	0	0
/sca6url/docs	4 of 4	Tomcat Documentation	0	0
/sca6url/examples	4 of 4	Servlet and JSP Examples	0	0
/sca6url/host-manager	4 of 4		0	0
/sca6url/manager	4 of 4	Tomcat Manager Application	0	0

You can perform the following operations using the Applications page:

- Click the webapp context name of an application to view the application summary.
- Click the application deployment status to view the application statistics by server class process.

- Click the request count value of an application to view the distribution of requests based on the URI.
- Click the session count value of an application to view all the in-memory sessions for the application across the processes in the selected server class.
- Click **Reset Stats** to reset the values of the **Request Count** column in the table.

[Table 4-1](#) lists the attributes displayed in the Applications page.

Table 4-1. Attributes in the Applications Page

Attribute	Description
Webapp Context Name	The context name of the application.
Deployment Status	Indicates whether the application is deployed on all the processes within the server class. The values displayed in this field indicate the number of processes on which the application is STARTED, and the total number of processes in the server class. For example, if the value is 6 of 8, the application is STARTED on six processes. The total number of processes is eight. A context can be in four states: STARTED, FAILED, STOPPED, and INITIALIZED. The INITIALIZED state indicates that the context is still starting.
Description	Briefly describes the application as mentioned in the <code>web.xml</code> application descriptor.
Request Count	The total number of user requests serviced by the application across all the processes of the server class.
Session Count	The total number of sessions of the application across all the processes that are currently in memory.

Viewing Application Summary

The NSJSP Manager application enables you to view detailed information about each application, such as, application name and runtime information. The runtime information is crucial because it provides the number of current requests that are accepted by an application, their processing time, and the frequency of the application being used.

To view the summary of an application, complete the following steps:

1. Click the **Applications** tab.

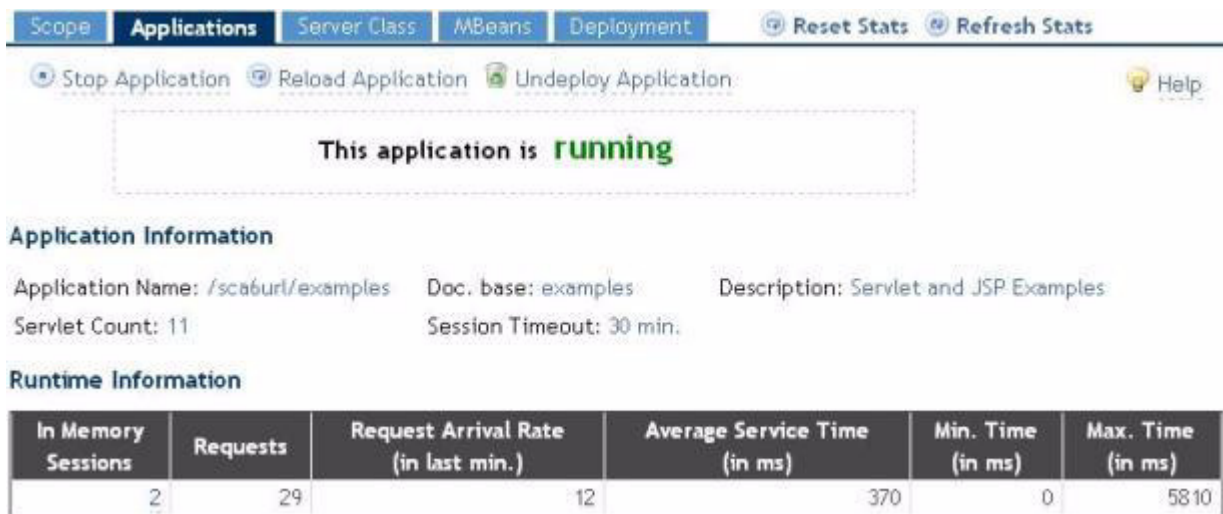
The Applications page appears.

2. Click the name of an application.

The Application Summary page for the selected application appears. The statistics displayed on this page are a summation of statistics gathered from all the processes in the NSJSP Server Class.

[Figure 4-4](#) shows a sample Application Summary page for the application, `/sca6url/examples`.

Figure 4-4. Application Summary Page



You can perform the following operations on the Application Summary page:

- Stop an application on all the processes in the selected server class using the **Stop Application** option. For more information on stopping the application, see [Stopping a Web Application](#) on page 4-26.
- Start an already stopped application using the **Start Application** option. For more information on starting the application, see [Starting a Web Application](#) on page 4-27.
- Reload an application. The **Reload Application** option reloads the entire context. This can be used to reload all the application resources while the application is still available for the users. For more information on reloading the application, see [Reloading a Web Application](#) on page 4-28.

Note. During a reload operation, all the in-memory sessions will be lost if there is no persistence manager (with a persistence store) configured. If a persistence manager is configured, the in-memory sessions are saved in the persistence store. These sessions are available even after the application is reloaded. The **Reload Application** option is displayed only if the application is currently running.

- Undeploy an application using the **Undeploy Application** option. This option removes the context and deletes all the application-related resources from the file system. For more information on undeploying the application, see [Undeploying a Web Application](#) on page 4-29.

- Click the value corresponding to the **Servlet Count:** field to view the list of servlet mappings configured for an application.

Note. A servlet can have multiple servlet mappings configured for it.

- Click a value from the **In Memory Sessions** column of the table to view all the sessions currently in memory for an application.
- Click **Reset Stats** to reset the value of the **Requests**, **Request Arrival Rate**, **Average Service Time**, **Min. Time**, and **Max. Time** columns in the table.

[Table 4-2](#) lists the operations displayed in the Applications Summary page.

Table 4-2. Operations in the Application Summary Page

Operation	Description
Stop Application or Start Application	<p>The Stop Application option stops the application on all the processes in this server class. You cannot access the application after stopping it. An attempt to access this application results in an HTTP error 404.</p> <p>The Stop Application option is displayed only if the application is currently running.</p> <p>The Start Application option starts an already stopped application. After a successful start, you can access the application. The Start Application option is displayed only if the application is currently stopped.</p>
Reload Application	<p>Stops and starts an application. If the changes made to the application need to be effective immediately, click the Reload Application option. This reloads the entire context. You can use this option to reload all the application resources while the application is still available. However, during a reload operation, all the in-memory sessions will be lost if there is no persistence manager (with a persistence store) configured. If a persistence manager is configured, the in-memory sessions are saved in the persistence store and will be available after the application is reloaded.</p> <p>The Reload Application option is displayed only if the application is currently running.</p>
Undeploy Application	<p>Removes the context and deletes all the application-related resources from the file system. This operation is irreversible.</p>

[Table 4-3](#) lists the attributes displayed in the Application Summary page.

Table 4-3. Attributes in the Application Summary Page

Attribute	Description
Application Information	
Application Name:	The context name of the application.
Servlet Count:	Number of servlets configured for the application.
Doc. base:	The root directory that contains all the application resources. It is relative to the application base (<code>appBase</code>) of the Host in which this application is deployed.
Session Timeout:	Maximum idle time allowed for a session. A session object will time out if it is not accessed for the specified duration. The unit of the Session Timeout value is minutes. If the value is -1, the session objects do not timeout.
Description:	The application description as mentioned in the <code>web.xml</code> application descriptor.
Runtime Information	
In Memory Sessions	Number of session objects that are currently in memory.
Requests	Number of requests that are processed by application servlets.
Request Arrival Rate (in last min.)	The number of user requests received in the last minute by the application.
Average Service Time (in ms)	Average time, in milliseconds, taken to process the user requests for the application.
Min time (in ms)	Minimum time taken to process a user request.
Max time (in ms)	Maximum time taken to process a user request.

[Table 4-4](#) lists the submenu items under the **Applications** tab.

Table 4-4. Submenu Items Under the Applications Tab (page 1 of 2)

Submenu Item	Description
Summary	Summary of applications, such as, the description of the application and runtime information of the application. For more information, see Viewing Information about Web Applications on page 4-8.
Process View	Details of the application for each NSJSP process within the selected server class. For more information, see Viewing Details of the Application for each NSJSP Process on page 4-14.
In-Memory Sessions	Number of session objects that are currently in memory. For more information, see Viewing Sessions for an Application on page 4-16.

Table 4-4. Submenu Items Under the Applications Tab (page 2 of 2)

Submenu Item	Description
URI Statistics	Uniform Resource Identifier (URI) access statistics for the application, such as, the list of application URIs accessed by users and the number of user requests received by the application. For more information, see Viewing the URI Statistics for an Application on page 4-18.
HTTP Method Statistics	Details of the Hypertext Transfer Protocol (HTTP) methods queried for the application. The HTTP Method Statistics page displays application statistics based on the HTTP method that is used to access the application resources. For more information, see Viewing HTTP Method Statistics on page 4-20.
Context Descriptor	Context descriptor details for the application. The Context Descriptor file denotes the configuration file for the application. For more information, see Viewing Context Descriptor Details on page 4-21.
Deployment Descriptor	Deployment descriptor details for the application. The Deployment Descriptor file is used to obtain information, such as, servlet mappings and authentication details, which are used to deploy the web applications. For more information, see Viewing Deployment Descriptor Details on page 4-22.
Servlet Mappings	Details of servlet mappings for the application. For more information, see Viewing Servlet Mappings on page 4-23.
Filters	Details of the filters for the application. Filters are Java components that allow transformations of payload and header information from the request into a resource and the response from a resource. A filter can use the information available in the user requests and the responses received from an application to perform tasks, such as, logging and auditing. For more information, see Viewing Filter Details on page 4-24.
Initialization Parameters	Initialization parameters that are used to initialize the application. For more information, see Viewing Initialization Parameters on page 4-25.

Viewing Details of the Application for each NSJSP Process

The application statistics for each NSJSP process includes detailed information, such as, the NSJSP process name and the number of requests processed by the application servlets.

To view the details of the applications for each NSJSP process, complete the following steps:

1. Complete the steps described in [Viewing Application Summary](#) on page 4-10.
2. Click **Process View**.

The details of the application for each NSJSP process are displayed on the Process View page.

You can perform the following operations on the Process View page:

- Click the value corresponding to the **Servlet Count**: field to view the list of servlet mappings configured for an application.

Note. A servlet can have multiple servlet mappings configured for it.

- Click the process name to display the Process Summary page, which displays the details, such as, process name, CPU and PIN of the process, Java Virtual Machine (JVM) heap statistics, and connector information.
- Click a value from the **In Memory Sessions** column of the table to view all the sessions currently in memory for an application.
- Click the requests value in the table to view the distribution of requests based on the URI.
- Click **Reset Stats** to reset the value of the **Requests**, **Request Arrival Rate**, **Average Service Time**, **Min. Time**, and **Max. Time** columns in the table.

[Figure 4-5](#) shows a sample output for the application, `/sca6url/examples`.

Figure 4-5. Application Summary Page After Clicking Process View



The Process View page displays statistics that indicate how the requests are distributed across processes and how the server class processes are processing the requests for the application.

[Table 4-5](#) lists the attributes displayed in the Process View page.

Table 4-5. Attributes in the Process View Page

Attribute	Description
Application Information	
Application Name:	The context name of the application.
Servlet Count:	Number of servlets configured in the application.
Doc. base:	The root directory that contains all the application resources. It is relative to the application base (<code>appBase</code>) of the Host in which this application is deployed.
Session Timeout:	Maximum idle time allowed for a session. A session object will time out if it is not accessed for the specified duration. The unit of the Session Timeout value is minutes. If the value is -1, the session objects do not timeout.
Description:	Application description as mentioned in the <code>web.xml</code> application descriptor.
Runtime Information	
Pathmon Name	Name of the PATHMON that started the process.
Process	Name of the NSJSP process on which the application is running.
CPU, PIN	Logical processor number and the Process Identification Number (PIN) for this process.
In Memory Sessions	Number of session objects across processes that are currently in memory.
Requests	Number of requests that are serviced by application servlets.
Request Arrival Rate (in last min.)	Number of user requests received in the last minute by the application.
Average Service Time (in ms)	Average time, in milliseconds, taken to service the user requests for the application.
Min. time (in ms)	Minimum time taken to service a user request.
Max. time (in ms)	Maximum time taken to service a user request.

Viewing Sessions for an Application

When you open an application, a session is created. All requests for the application belong to that session. The NSJSP Manager application displays the number of sessions that are currently active for an application.

To view the sessions for an application, complete the following steps:

1. Complete the steps described in [Viewing Application Summary](#) on page 4-10.
2. Click **In-Memory Sessions**.

The In-Memory Sessions page for the selected application appears. [Figure 4-6](#) shows a sample In-Memory Sessions page for the application, `/sca6url/examples`.

Figure 4-6. In-Memory Sessions Page

Session search criteria

Session id (RE)

Idle time from (sec) to

Age from (sec) to

In Memory Sessions

Session Timeout: 30 min.

3 items found, displaying all items.1

	Session ID	Idle Time	Age	Expiry Time
<input type="checkbox"/>	SZ11X\$7FD0BAE5EA39D0255256528C91AC7D03	00:00:16	00:00:16	Thu Apr 15 19:51:58 IST 2010
<input type="checkbox"/>	SZ11Y\$C8F111B11FF137941E8F2906C567908D	00:07:35	00:13:53	Thu Apr 15 19:44:39 IST 2010
<input type="checkbox"/>	SZ11Y\$BC66F7BD9FA48147403C7B1838872F01	00:01:21	00:01:21	Thu Apr 15 19:50:53 IST 2010

You can perform the following operations using the In-Memory Sessions page:

- Select or clear the check box corresponding to a session by clicking **Toggle**. The **Toggle** option is useful if all the sessions displayed on the In-Memory Session page need to be selected or removed.
- Expire the sessions from the available list. The **Expire** option must be used carefully because it is an irreversible operation. Sessions once expired cannot be restored by any other operation.
- If a persistent session manager is configured, you can persist the sessions, which are currently in memory, to the store using the **Persist** option.
- Search the sessions based on the following criteria:
 - Session ID: It accepts a regular expression (RE) for matching sessions. It is useful for searching sessions originating from a process. A search criterion, such as `\$Y31R.*` searches the list of all sessions in the process `$Y31R`.
 - Idle Time (sec): It filters out sessions that are idle. A session's idle time is the length of time during which a session has not been accessed.

- **Age (sec):** It is similar to the Idle Time except that this option filters the sessions based on the age. The age of a session indicates how long ago the session was created.

You can also use the **Apply search** option to search the sessions. This option appears only when you search for a session. Use the **Clear search** option to clear the search screen and return to the session list screen.

The In-Memory Sessions page displays a list of application sessions that are currently in any process memory.

[Table 4-6](#) lists the operations displayed in the In-Memory Sessions page.

Table 4-6. Operations in the In-Memory Sessions Page

Operation	Description
Toggle	Checks or unchecks each session.
Expire	Expires the sessions that are selected.
Persist	If a persistent session manager is configured, this option allows sessions that are currently in memory to be persisted to the store. Even after a session is persisted to a store, the session may continue to reside in memory.
Session search	Enables you to search for sessions using various criteria.

[Table 4-7](#) lists the attributes displayed in the In-Memory Sessions page.

Table 4-7. Attributes in the In-Memory Sessions Page

Attribute	Description
In Memory Sessions	
Session Timeout:	Maximum idle time allowed for a session. The unit of the Session Timeout value is minutes. If the value is -1, the session objects do not timeout.
Session ID	Identification name of the session.
Idle Time	Duration for which the session is idle. The value is expressed in the <code>hh:mm:ss</code> format.
Age	Duration from the time the session is created. The value is expressed in the <code>hh:mm:ss</code> format.
Expiry Time	Time at which the session is scheduled to expire.

Viewing the URI Statistics for an Application

The Uniform Resource Identifier (URI) Statistics page displays statistics related to the access of application resources (listed by URI). Although the list of URIs can be very long, it will provide a good indication of the URI usage pattern for the application.

To view the URI statistics for the application, complete the following steps:

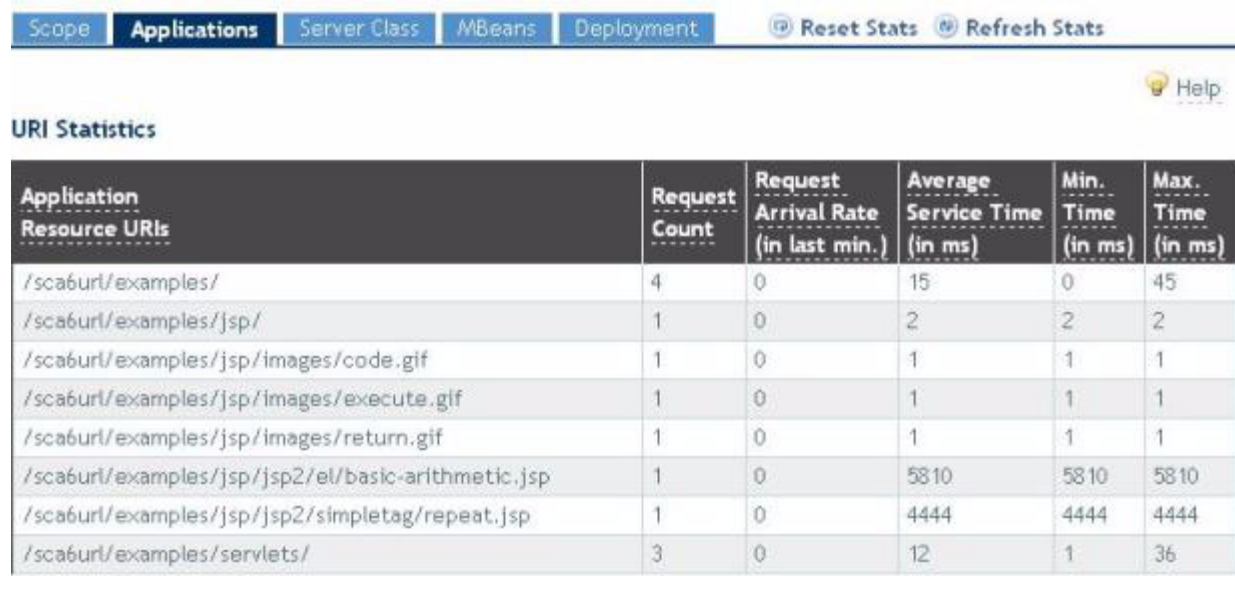
1. Complete the steps described in [Viewing Application Summary](#) on page 4-10.
2. Click **URI Statistics**.

The URI Statistics page appears. It displays access statistics of the application resources.

Note. The reset operation on the URI Statistics page resets the value of all the columns in the table.

[Figure 4-7](#) shows a sample URI Statistics page.

Figure 4-7. URI Statistics Page



Application Resource URIs	Request Count	Request Arrival Rate (in last min.)	Average Service Time (in ms)	Min. Time (in ms)	Max. Time (in ms)
/sca6url/examples/	4	0	15	0	45
/sca6url/examples/jsp/	1	0	2	2	2
/sca6url/examples/jsp/images/code.gif	1	0	1	1	1
/sca6url/examples/jsp/images/execute.gif	1	0	1	1	1
/sca6url/examples/jsp/images/return.gif	1	0	1	1	1
/sca6url/examples/jsp/jsp2/el/basic-arithmetic.jsp	1	0	5810	5810	5810
/sca6url/examples/jsp/jsp2/simpletag/repeat.jsp	1	0	4444	4444	4444
/sca6url/examples/servlets/	3	0	12	1	36

[Table 4-8](#) lists the attributes displayed in the URI Statistics page.

Table 4-8. Attributes in the URI Statistics Page (page 1 of 2)

Attribute	Description
Application Resource URIs	Uniform Resource Identifiers (URIs) of the application resources.
Request Count	Number of requests processed for the application resource that is indicated by the URI.
Request Arrival Rate (in last min.)	Number of requests processed for the resource in the last minute.

Table 4-8. Attributes in the URI Statistics Page (page 2 of 2)

Attribute	Description
Average Service Time (in ms)	Average time taken (in milliseconds) to process user requests for a URI.
Min. Time (in ms)	Minimum time taken (in milliseconds) to process a user request for a URI.
Max. Time (in ms)	Maximum time taken (in milliseconds) to process a user request for a URI.

Viewing HTTP Method Statistics

Any HTTP request received from NSJSP must be for one of the HTTP methods, such as, GET and POST. The page displays application statistics based on the HTTP method that is used to access the application resources.

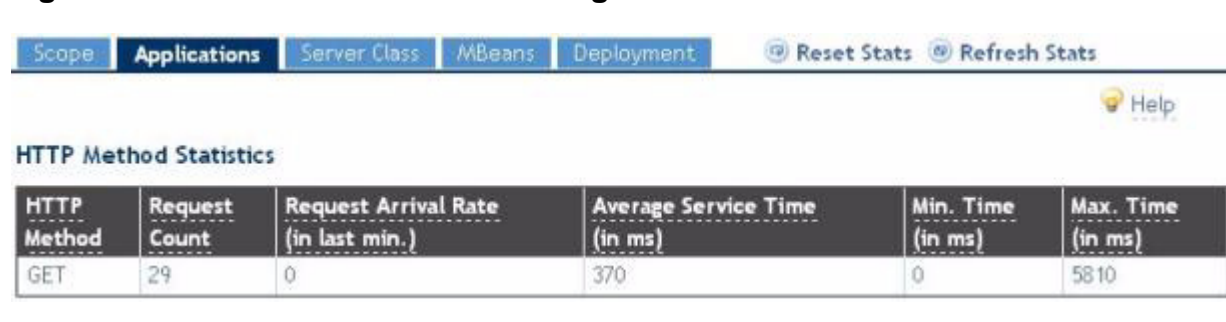
To view the HTTP method statistics, complete the following steps:

1. Complete the steps described in [Viewing Application Summary](#) on page 4-10.
2. Click **HTTP Method Statistics**.

The HTTP Method Statistics page for the selected application appears.

Note. The reset operation on the HTTP Method Statistics page resets the value of all the columns in the table.

[Figure 4-8](#) shows a sample HTTP Method Statistics page for the application, /sca6url/examples.

Figure 4-8. HTTP Method Statistics Page


HTTP Method	Request Count	Request Arrival Rate (in last min.)	Average Service Time (in ms)	Min. Time (in ms)	Max. Time (in ms)
GET	29	0	370	0	5810

[Table 4-9](#) lists the attributes displayed in the HTTP Method Statistics page.

Table 4-9. Attributes in the HTTP Method Statistics Page

Attribute	Description
HTTP Method	Name of the HTTP method.
Request Count	Number of requests processed for the HTTP method.
Request Arrival Rate (in last min.)	Number of requests processed for the HTTP method in the last minute.
Average Service Time (in ms)	Average time taken to process user requests for the HTTP method of the application.
Min. Time (in ms)	Minimum time taken to process a user request for the HTTP method of the application.
Max. Time (in ms)	Maximum time taken to process a user request for the HTTP method of the application.

Viewing Context Descriptor Details

The context descriptor is displayed only if the application explicitly defines its context using a context definition file, such as, `context.xml`.

To view the configuration in the context descriptor of an application, complete the following steps:

1. Complete the steps described in [Viewing Application Summary](#) on page 4-10.
2. Click **Context Descriptor**.

The Context Descriptor page for the selected application appears. [Figure 4-9](#) shows a sample Context Descriptor page for the application, `/sca6url/examples`.

Figure 4-9. Context Descriptor Page

You can use the **download** option to download the context definition file.

Viewing Deployment Descriptor Details

The Deployment Descriptor page displays the deployment descriptor for the selected application. The deployment descriptor is always in the `web.xml` file in the `WEB-INF` directory of the application.

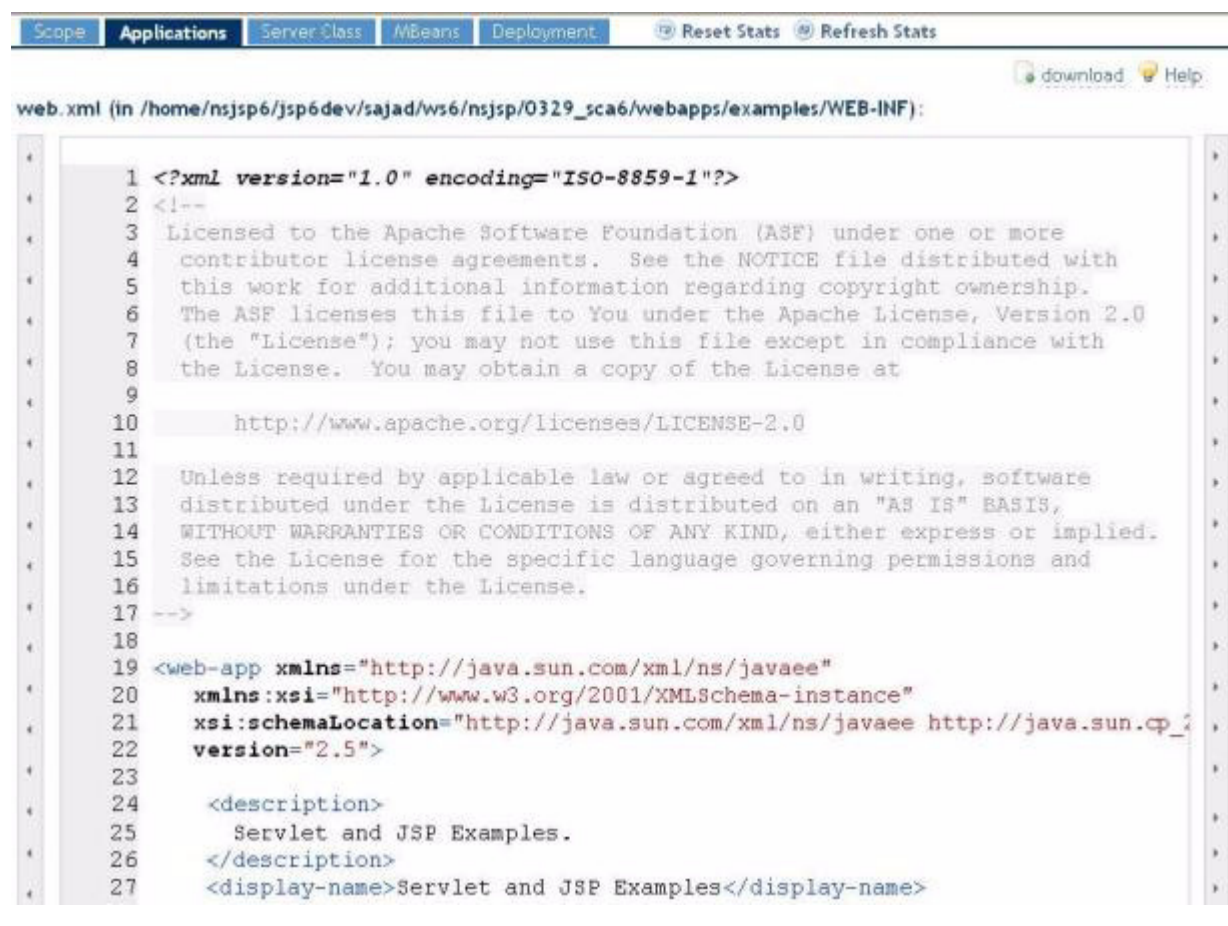
Note. There cannot be any application without a deployment descriptor.

To view the deployment descriptor details for the applications, complete the following steps:

1. Complete the steps described in [Viewing Application Summary](#) on page 4-10.
2. Click **Deployment Descriptor**.

The Deployment Descriptor page for the selected application appears. [Figure 4-10](#) shows a sample Deployment Descriptor page for the application, `/sca6url/examples`.

Figure 4-10. Deployment Descriptor Page



You can use the **download** option to download the deployment descriptor file.

Viewing Servlet Mappings

The deployment descriptor contains elements that define which servlet must process a request URL. The mapping between the URL and the servlet (configured to process the requests matching that URL) is called servlet mapping.

To view servlet mapping details for the applications, complete the following steps:

1. Complete the steps described in [Viewing Application Summary](#) on page 4-10.
2. Click **Servlet Mappings**.

The Servlet Mappings page for the selected application appears. It includes the definitions in the common deployment descriptor of the NSJSP installation, which is available at `<NSJSP 6.1 Installation Directory>/conf/web.xml`.

[Figure 4-11](#) shows a sample Servlet Mappings page for the application, `/sca6url/examples`.

Figure 4-11. Servlet Mappings Page



URL Pattern	Servlet Name	Servlet Class	Startup Order
*.jsp	jsp	org.apache.jasper.servlet.JspServlet	3
*.jspx	jsp	org.apache.jasper.servlet.JspServlet	3
/	default	org.apache.catalina.servlets.DefaultServlet	1
/CompressionTest	CompressionFilterTestServlet	compressionFilters.CompressionFilterTestServlet	
/jsp/chat/chat	ChatServlet	chat.ChatServlet	
/jsp/jsp2/misc/config.jsp	jsp	org.apache.jasper.servlet.JspServlet	3
/servletToJsp	servletToJsp	servletToJsp	
/servlets/servlet/CookieExample	CookieExample	CookieExample	
/servlets/servlet/HelloWorldExample	HelloWorldExample	HelloWorldExample	
/servlets/servlet/RequestHeaderExample	RequestHeaderExample	RequestHeaderExample	
/servlets/servlet/RequestInfoExample/*	RequestInfoExample	RequestInfoExample	
/servlets/servlet/RequestParamExample	RequestParamExample	RequestParamExample	
/servlets/servlet/SessionExample	SessionExample	SessionExample	

[Table 4-10](#) lists the attributes displayed in the Servlet Mappings page.

Table 4-10. Attributes in the Servlet Mappings Page

Attribute	Description
URL Pattern	Pattern of the Uniform Resource Locators (URLs), which will be serviced by the servlet.
Servlet Name	Logical name associated with the servlet.
Servlet Class	Fully qualified class name of the servlet.
Startup Order	The order in which the servlet initializes. Number 1 indicates that the servlet initializes first. No value indicates that a startup order is not specified.

Viewing Filter Details

Filters are Java components that allow the dynamic transformations of payload and header information from requests into resources and from the responses returned by resources. For example, a filter can use the information available in the user requests sent to the application and the responses received from the application to perform tasks, such as, logging and auditing.

Note. Filters are defined in the deployment descriptor (`web.xml`) of an application. You can also define the filters in the common deployment descriptor located in the `<NSJSP 6.1 Installation Directory>/conf` directory.

To view details of the filters for the applications, complete the following steps:

1. Complete the steps described in [Viewing Application Summary](#) on page 4-10.
2. Click **Filters**.

The Filters page for the selected application appears. [Figure 4-12](#) shows a sample Filters page for the application, `/sca6url/examples`.

Figure 4-12. Filters Page

Filter Name	Filter Class	Description
Compression Filter	compressionFilters.CompressionFilter	
Path Mapped Filter	filters.ExampleFilter	
Request Dumper Filter	filters.RequestDumperFilter	
Servlet Mapped Filter	filters.ExampleFilter	
Set Character Encoding	filters.SetCharacterEncodingFilter	

[Table 4-11](#) lists the attributes displayed in the Filters page.

Table 4-11. Attributes in the Filters Page

Attribute	Description
Filter Name	Logical name of the filter.
Filter Class	Fully qualified class name of the filter.
Description	Filter description as defined in the Deployment Descriptor.

Viewing Initialization Parameters

The Java servlet specification enables you to define the initialization parameters for each application. You can configure the initialization parameters using the `context-param` configuration element in the `web.xml` or the `Parameter` element nested under the `Context` element in the `context.xml` file.

To view the parameters of the applications, complete the following steps:

1. Complete the steps described in [Viewing Application Summary](#) on page 4-10.
2. Click **Initialization Parameters**.

The Initialization Parameters page for the selected application appears. [Figure 4-13](#) shows a sample Initialization Parameters page for an application.

Figure 4-13. Initialization Parameters Page

Param Name	Param Value	Source
attribute.value.role	manager	web.xml
contextConfigLocation	/WEB-INF/probe-servlet.xml	web.xml

[Table 4-12](#) lists the context initialization parameters.

Table 4-12. Context Initialization Parameters

Parameters	Description
Param Name	Name of the initialization parameter.
Param Value	Value set for the parameter.
Source	Location where the parameter is defined. The <code>web.xml</code> file is one of the locations where you can configure the parameters.

Managing Web Applications

Upon successful deployment of a web application, all the instances of that application on a Host are in the running state, by default. Following is the list of operations that you can perform on a web application:

- [Stopping a Web Application](#)
- [Starting a Web Application](#)
- [Reloading a Web Application](#)
- [Undeploying a Web Application](#)

Stopping a Web Application

To restrict users from accessing an application, you can stop the application using the **Stop Application** option in the NSJSP Manager application interface.

To stop the web application, complete the following steps:

1. Complete the steps described in [Viewing Application Summary](#) on page 4-10.
2. Click the application name that you want to stop.
3. Click **Stop Application**.

The application is stopped.

[Figure 4-14](#) shows the messages that are displayed after an application is stopped.

Figure 4-14. Application Summary Page Showing the Down Status

If you click the **Application** tab again, the status of the application is displayed as None.

After the application is stopped, it is no longer accessible to the users.

Starting a Web Application

The **Start Application** option of the NSJSP Manager application interface enables you to start the previously stopped application.

To start the web application, complete the following steps:

1. Complete the steps described in [Viewing Application Summary](#) on page 4-10.
2. Click the application name that you want to start.
3. Click **Start Application**.

The application is started.

[Figure 4-15](#) shows the messages that are displayed after an application is started.

Figure 4-15. Application Summary Page Showing the Running Status

The screenshot shows the NSJSP Manager interface with the **Applications** tab selected. At the top, there are tabs for **Scope**, **Applications**, **Server Class**, **MBeans**, and **Deployment**. To the right are buttons for **Reset Stats** and **Refresh Stats**. Below these are buttons for **Stop Application**, **Reload Application**, and **Undeploy Application**, along with a **Help** icon. A green message box states "Application started successfully." Below that, a larger green message box states "This application is **running**".

Application Information

Application Name: /sca6url/examples Doc. base: examples Description: Servlet and JSP Examples
 Servlet Count: 11 Session Timeout: 30 min.

Runtime Information

In Memory Sessions	Requests	Request Arrival Rate (in last min.)	Average Service Time (in ms)	Min. Time (in ms)	Max. Time (in ms)
0	0	0	0	0	0

If you click the **Applications** tab again, the status of the application is displayed as All NSJSPs.

You can now access the application.

Reloading a Web Application

The **Reload Application** option of the NSJSP Manager application interface reloads the entire context. It is used to reload all the application resources while the application is available to the users.

Note.

- During a reload operation, all the in-memory sessions will be lost if there is no persistence manager (with a persistence store) configured. If a persistence manager is configured, the in-memory sessions are saved in the persistence store. Thus, the in-memory sessions are available even after reloading the application.
- The **Reload Application** option is displayed only if the application is in the running state.

To reload the web application, complete the following steps:

1. Complete the steps described in [Viewing Application Summary](#) on page 4-10.
2. Click the application name that you want to reload.
3. Click **Reload Application**.

The application restarts.

[Figure 4-16](#) shows the messages that are displayed after an application is reloaded.

Figure 4-16. Application Summary Page Showing the Reloaded Status

The screenshot shows the NSJSP Manager interface. At the top, there are tabs for 'Scope', 'Applications' (selected), 'Server Class', 'MBeans', and 'Deployment'. To the right of these tabs are buttons for 'Reset Stats' and 'Refresh Stats'. Below the tabs, there are three main action buttons: 'Stop Application', 'Reload Application' (highlighted with a green border), and 'Undeploy Application'. A 'Help' icon is also present. In the center, a green message box states 'Application reloaded successfully.' Below this, another green message box states 'This application is running'. Underneath, the 'Application Information' section displays: 'Application Name: /sca6url/examples', 'Doc. base: examples', 'Description: Servlet and JSP Examples', 'Servlet Count: 11', and 'Session Timeout: 30 min.'. The 'Runtime Information' section contains a table with the following data:

In Memory Sessions	Requests	Request Arrival Rate (in last min.)	Average Service Time (in ms)	Min. Time (in ms)	Max. Time (in ms)
0	0	0	0	0	0

Undeploying a Web Application

The **Undeploy Application** option of the NSJSP Manager application interface removes the context and deletes all the application-related resources from the file system.

To undeploy the web application, complete the following steps:

1. Complete the steps described in [Viewing Application Summary](#) on page 4-10.
2. Click the application name that you want to undeploy.
3. Click **Undeploy Application**.

The dialog box appears with the following message:

This operation cannot be reversed. Do you really want to REMOVE /<application name>?

4. Click **OK** to remove the application from the applications list.
5. The application name is removed from the Host under consideration that you selected in the **Scope** tab.

Viewing Server Class Information

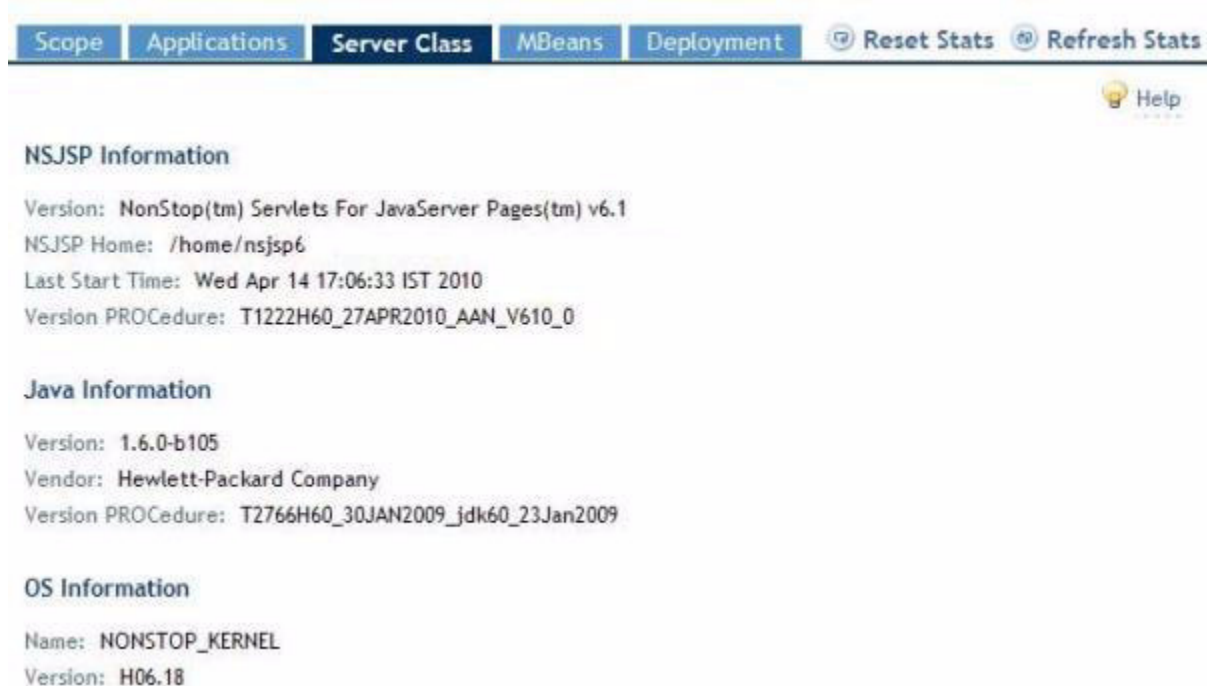
To view the details of an NSJSP Server Class, select the **Server Class** tab in the NSJSP Manager application. You can view the following information under the **Server Class** tab:

- NSJSP information
- Server class processes
- NSJSP connector statistics
- Configuration parameters
- Server class statistics
- Server class operations

To view information about a server class, click the **Server Class** tab.

The NSJSP Information page appears. [Figure 4-17](#) shows a sample NSJSP Information page.

Figure 4-17. NSJSP Information Page



The NSJSP Information page displays static information about the environment in which NSJSP is running.

[Table 4-13](#) lists attributes displayed in the NSJSP Information page.

Table 4-13. Attributes in the NSJSP Information Page

Attribute	Description
NSJSP Information	
Version:	NSJSP version installed on the system.
NSJSP Home:	Open System Services (OSS) path where the NSJSP installation is created.
Last Start Time:	Date and time at which the server class last started.
Version PROCedure	Version Procedure (VPROC) of NSJSP that is installed.
Java Information	
Version:	NSJ version that the NSJSP installation uses.
Vendor:	Name of the JVM vendor.
Version PROCedure:	Version Procedure (VPROC) of the Java used by the server class.
OS Information	
Name:	Name of the operating system on which NSJSP is running.
Version:	Version of the operating system on which NSJSP is running.

[Table 4-14](#) lists the submenu items under the **Server Class** tab.

Table 4-14. Submenu Items Under the Server Class (page 1 of 2)

Submenu Item	Description
NSJSP Information	Information about the NSJSP version, the version of the Java Virtual Machines (JVMs), and the HP NonStop operating system.
Server Class Processes	Information about NSJSP processes, such as, the name of the NSJSP processes in the server class and the number of user requests received by the NSJSP processes. For more information on how to view these details, see Viewing Details of NSJSP Processes on page 4-32.
NSJSP Connector Stats	Information about the NSJSP connectors for each process, such as, the number of threads used to serve user requests and the number of process threads allocated to serve user requests. For more information, see Viewing Details of the NSJSP Connector Statistics on page 4-33.

Table 4-14. Submenu Items Under the Server Class (page 2 of 2)

Submenu Item	Description
Configuration Parameters	TS/MP parameters that are used to deploy NSJSP processes. For more information, see Viewing Configuration Parameters on page 4-34.
Server Class Statistics	Aggregate Linkmon Statistics for the NSJSP Server Class. You can use the values displayed in this page for performance analysis and troubleshooting. For more information, see Viewing Server Class Statistics on page 4-36.
Server Class Operations	Options to stop, start, freeze, and thaw the server class. For more information, see Performing Server Class Operations on page 4-39.

Viewing Details of NSJSP Processes

The list of NSJSP processes that are running is displayed in the Server Class Processes page. The NSJSP processes provide details, such as, the processor on which the application is running, information about the \$RECEIVE queue, and high-level Java heap statistics. The output also indicates that if the processor is overloaded and if load balancing is required for the processor.

To view the details of the NSJSP processes, complete the following steps:

1. Click the **Server Class** tab.
2. Click **Server Class Processes**.

The Server Class Processes page appears, which displays the list of NSJSP Server Class processes that are currently running.

[Figure 4-18](#) shows a sample Server Class Processes page. This page displays some process-level statistics that can help tune the server class configuration parameters and the memory-related parameters in the JVM.

Figure 4-18. Server Class Processes Page

Pathmon Name	Process Name	CPU, PIN	Request Not Read	Request Not Replied	Request Open Depth	Max Memory (in Bytes)	Total Allocated Memory (in Bytes)	Used Memory (in Bytes)
\$ZSB6	\$X98V	0,1120	0	1	52	64,880,640	64,880,640 (100%)	20,780,136 (32%)
\$ZSB6	\$X98W	1,394	0	1	52	64,880,640	64,880,640 (100%)	20,025,656 (30%)

[Table 4-15](#) lists the attributes displayed in the Server Class Processes page.

Table 4-15. Attributes in the Server Class Processes Page

Attribute	Description
Pathmon Name	Identifies in which PATHMON a particular NSJSP process is running.
Process Name	Name of the NSJSP process.
CPU, PIN	CPU and PIN of the process.
Request Not Read	Number of requests that are yet to be read. These requests are currently in the \$RECEIVE queue.
Request Not Replied	Number of requests that are currently being processed.
Request Open Depth	It is also called the receive depth. It is the maximum the number of messages that can be read before replying to any read messages. In the NSJSP context, this number indicates the maximum number of requests that can be processed by a server class process in parallel. This is a static counter and the number is derived from the TANDEM_RECEIVE_DEPTH configuration parameter of the NSJSP Server Class.
Max Memory (in Bytes)	Maximum memory allowed for the JVM heap. The value is determined by the -Xmx command-line parameter for the JVM.
Total Allocated Memory (in Bytes)	The amount of heap that the JVM has allocated out of the maximum allowed heap. This component also displays the percentage of Total Allocated Memory with respect to Max Memory.
Used Memory (in Bytes)	The amount of heap in use out of the total heap currently allocated by the JVM.

Viewing Details of the NSJSP Connector Statistics

The connector statistics provides a good indication of the incoming request workload for individual NSJSP processes. Since all requests to all the deployed applications in a server class process have to pass through the connector, the connector counters provide a good indication of how many requests have been processed across all the applications deployed.

For more information on connectors, see Chapter [1, Introduction to NSJSP](#).

To view the details of the NSJSP connector statistics, complete the following steps:

1. Click the **Server Class** tab.
2. Click **NSJSP Connector Stats**.

The NSJSP Connector Stats page appears. [Figure 4-19](#) shows a sample NSJSP Connector Stats page.

Figure 4-19. NSJSP Connector Stats Page

Scope

Applications

Server Class

MBeans

Deployment

Reset Stats

Refresh Stats

Help

NSJSP Connector Stats

Process Name	Thread Used	Thread Free	Thread Total	Request Count	Request Arrival Rate (in last min.)	Average Service Time (in ms)	Min. Time (in ms)	Max. Time (in ms)
\$X98V	2	73	75	38	0	294	0	5814
\$X98W	1	74	75	21	0	390	0	7556

This page displays the activities of the NSJSP connector component. The connector activities indicate the incoming request workload for individual NSJSP processes.

[Table 4-16](#) lists the attributes in the NSJSP Connector Stats page.

Table 4-16. Attributes in the NSJSP Connector Statistics page

Attribute	Description
Process Name	Name of the NSJSP process.
Thread Used	Number of request processing threads created by the connector that are currently active. These threads are either waiting to process an incoming request or are already processing a request.
Thread Free	Number of threads that the connector can create, if required. This number does not include any threads that are already created.
Thread Total	Maximum number of request processing threads that the connector can create.
Request Count	Total number of requests processed by the connector.
Request Arrival Rate (in last min.)	Number of requests received by the NSJSP process in the last one minute.
Average Service Time (in ms)	Average time taken to process incoming requests. This value is expressed in milliseconds.
Min. Time (in ms)	Minimum time taken to process incoming requests. This value is expressed in milliseconds.
Max. Time (in ms)	Maximum time taken to process incoming requests. This value is expressed in milliseconds.

Viewing Configuration Parameters

The server class configuration parameters are used during initialization. The parameters that are displayed using the NSJSP Manager Application are defined in the `servlet.config` file. However, some of the parameters are displayed with their default values and are not required to be explicitly set. For example, the `AUTORESTART` parameter is shown with the default value of 3 and is not mentioned in

the `servlet.config` file. For more information on the server class configuration parameters and their values, see the *TS/MP System Management Manual*.

To view the parameters that are used to deploy NSJSP processes, complete the following steps:

1. Click the **Server Class** tab.
2. Click **Configuration Parameters**.

The Configuration Parameters page appears. [Figure 4-20](#) shows a sample Configuration Parameters page.

Figure 4-20. Configuration Parameters Page

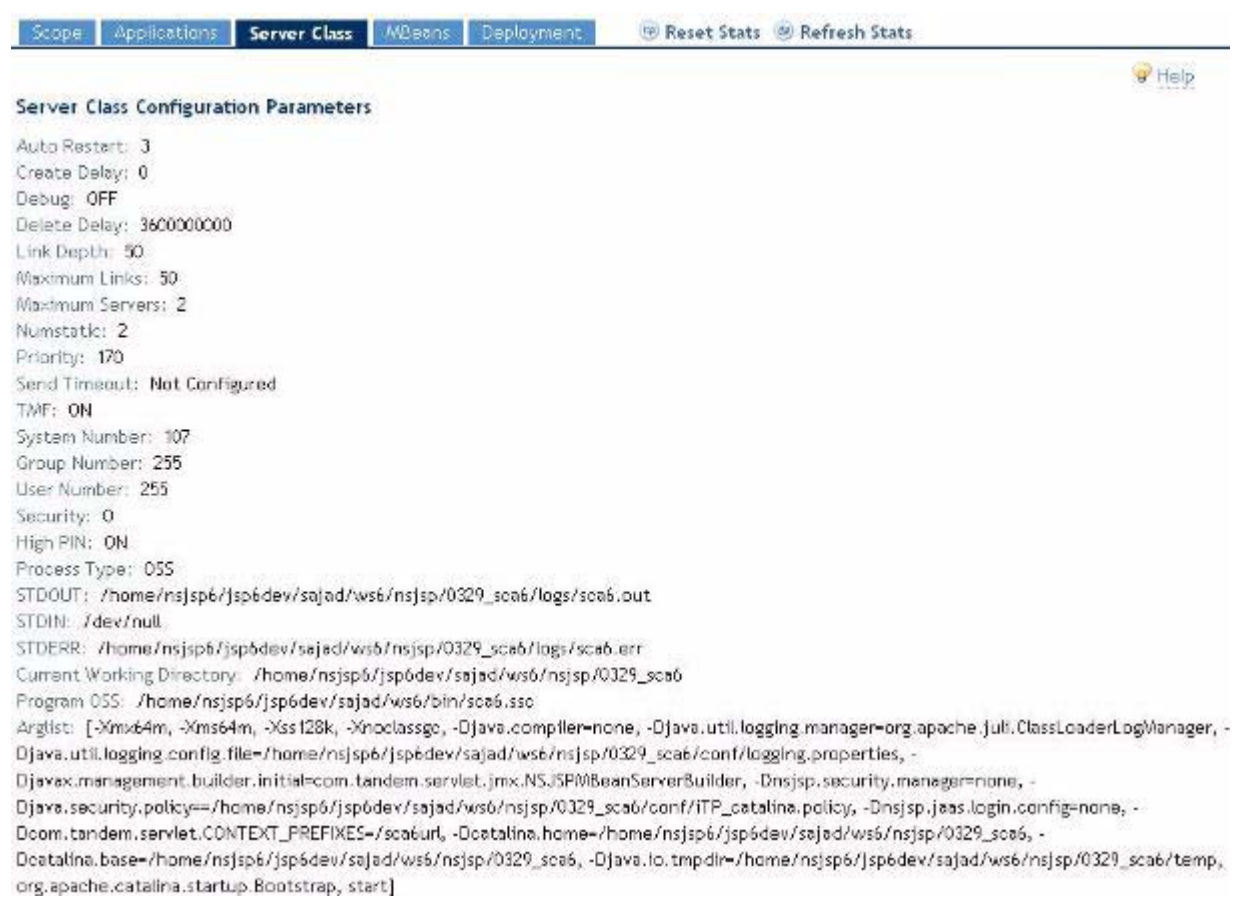


Figure 4-20. Configuration Parameters Page

Environment Variables	
Name	Value
BANK_CATALOG	bankctl.bankctl.secrets
CLASSPATH	/home/nsjsp6/jspidev/sajad/ws6/nsjsp/0329_sca6/bin/bootstrap.jar:/home/nsjsp6/jspidev/sajad/ws6/nsjsp/0329_sca6/bin/daemon.jar:/home/nsjsp6/jspidev/sajad/ws6/nsjsp/0329_sca6/bin/tomcat-juli.jar:/home/nsjsp6/jspidev/sajad/ws6/nsjsp/0329_sca6/bin/ns-logger.jar:/home/nsjsp6/jspidev/sajad/ws6/nsjsp/0329_sca6/bin/nsjspmbeanserver.jar:/usr/tandem/jdbc/ix/current/lib/jdbc
JAVA_HOME	/home/nsjsp6/jspidev/tools/java6.0/java
JRE_HOME	/home/nsjsp6/jspidev/tools/java6.0/java/jre
TANDEM_FILEMAPS_CONFIG	/home/nsjsp6/jspidev/sajad/ws6/nsjsp/0329_sca6/conf/filemaps.config
TANDEM_PATHMON_NAME	\$ZSB6
TANDEM_RECEIVE_DEPTH	50
_RLO_LIB_PATH	/home/nsjsp6/jspidev/sajad/ws6/nsjsp/0329_sca6/lib:/usr/tandem/jdbc/ix/current/lib:/usr/tandem/jdbc/ix/current/lib

[Table 4-17](#) lists the GUI server class configuration parameters.

Table 4-17. Server Class Configuration Parameters

Attribute	Description
Server Class Configuration Parameters	Lists the configuration parameters.
Environment Variables	
Name	Name of the environment variable set for the NSJSP process.
Value	Value of the environment variable set for the NSJSP process.

Viewing Server Class Statistics

The TS/MP subsystem provides the link manager statistics for the server class under consideration. If the server class is configured for online-upgrade (that is, configured in two different PATHMONs) in a Pathway domain configuration comprising two PATHMONs, the statistics are displayed for each PATHMON. The statistics includes the Server class link wait queue (Queue Info) and Input and Output operations (I/O Info).

To view the aggregate Linkmon statistics for the NSJSP Server Class, complete the following steps:

- 1. Click the **Server Class** tab.
- 2. Click **Server Class Statistics**.

The Server Class Statistics page appears. [Figure 4-21](#) shows a sample Server Class Statistics page.

Figure 4-21. Server Class Statistics Page

The Server Class Statistics page provides the following information:

- Server class link wait queue (Queue Info)

The link manager sends requests to the server class process through links. A request is placed in a link queue if it is not possible to immediately send the request to a server class process due to unavailability of a link. The request resides in the link wait queue until a free link is available or the link manager obtains a new link to a server class process from the PATHMON.

More number of requests residing in the link wait queue indicates a need of increasing the number of server class processes (NUMSTATIC) or the number of links per server class (MAXLINKS/LINKDEPTH). As a result, less number of requests will wait in the link wait queue, thereby improving the system response time.

- Input or output operations (I/O Info)

It is the information about I/O operations to and from a server class. A send operation is a message sent from the link manager to a server class process. A reply operation is a message received by the link manager from a server class process.

When an HTTPD process communicates with NSJSP, a single HTTPD message sent to an NSJSP process results in multiple send operations from the link manager to the process. This is because of the limitation on the maximum number of bytes that can be sent in one send operation. Similarly, a single HTTP response from the NSJSP Server Class process might result in multiple outgoing messages between the process and the link manager.

Note. The communication between an HTTPD process and an NSJSP Server Class process need not always happen through a link manager. If the session based load balancing is turned on, HTTPD processes communicate with the NSJSP Server Class processes directly through file system calls. In such a case, the link manager counters on this page are not incremented.

[Table 4-18](#) lists GUI the server class statistics.

Table 4-18. Server Class Statistics

Attribute	Description
Queue Info	
Request Count	Total number of requests for a link that were not satisfied immediately.
% Waits	Percentage of requests indicated by the Request Count value waited in the link wait queue before it was serviced.
Maximum Waits	Maximum number of requests that were queued in the link manager queue.
Average Waits	Average number of requests that were waiting.
Dynamic Links	Percentage of outstanding link requests that were serviced by dynamic links.
IO Info	
Send Request Count	Total number of requests sent to servers within the server class.
Send Maximum Size	Size, in bytes, of the largest amount of data transferred in a send.
Send Average Size	Size, in bytes, of the average amount of data transferred in a send.
Send IOs	Number of I/O operations sent to servers within the server class.
Reply Request Count	Not applicable
Reply Max Size	Size, in bytes, of the largest amount of data transferred in a reply from the server class.
Reply Average Size	Size, in bytes, of the average amount of data transferred in a reply from the server class.
Reply IOs	Not applicable

Note. Clicking Reset on this page does not reset the statistics displayed on this page. For information on how to reset these statistics, see the *TS/MP System Management Manual*.

Performing Server Class Operations

Under the **Server Class** tab, you can view the current status of the server class by clicking the **Server Class Operations** option.

The **Server Class** tab enables you to perform the following tasks:

- [Stop the NSJSP Server Class](#)
- [Start the NSJSP Server Class](#)
- [Freeze the NSJSP Server Class](#)
- [Thaw the NSJSP Server Class](#)

Note.

- If an NSJSP Server Class is configured for online upgrade, the NSJSP Manager application enables you to select a PATHMON on which you want to stop, start, freeze, or thaw the server class.
- Each server class operation has an equivalent PATHCOM or PDMCOM command, as follows:
 - The stop operation is equivalent to the `FREEZE <server class>` and `STOP <server class>` commands.
 - The start operation is equivalent to the `THAW <server class>` and `START <server class>` commands.
 - The freeze operation is equivalent to the `FREEZE <server class>` command.
 - The thaw operation is equivalent to the `THAW <server class>` command.

For more information on performing the server class operations using the PATHCOM commands, see [Server Class Operations](#) on page 4-86.

For more information on the PATHCOM commands, see the *TS/MP System Management Manual*.

Stop the NSJSP Server Class

To restrict access to the NSJSP installation, stop the NSJSP Server Class. Before stopping an NSJSP Server Class, ensure that no one is accessing that installation.

To stop the NSJSP Server Class, complete the following steps:

1. Click the **Server Class** tab.

The System Information page appears.

2. Click the **Server Class Operations**.

3. Perform one of the following steps depending on the number of PATHMONs configured:

- a. If the server class is configured on only one PATHMON, click **Stop**.

The following message is displayed:

The operation will make the server class unavailable for the user. Do you still want to stop the server class?

Click **OK**.

- b. If the server class is configured in a TS/MP Pathway domain, complete the following steps:

1. Select the **Pathmon Name**, which has the server class to be stopped.
2. Click **Stop**.

The following message is displayed:

The operation will make the server class unavailable for the user. Do you still want to stop the server class?

3. Click **OK**.

The following message is displayed for Steps 3a and 3b:

Stop Initiated. The operation may take a few minutes to complete.

The server class stops and you can no longer access any application running on the server class.

Figure 4-22 shows the Server Class Operations page after stopping the server class configured under the \$YSB6 PATHMON.

Figure 4-22. Server Class with the \$YSB6 PATHMON in the FROZEN State

The screenshot shows the 'Server Class' tab selected in the top navigation bar. Below the navigation bar, there are buttons for 'Stop', 'Thaw', and 'Freeze'. A green message box states: 'Stop Initiated. The operation may take a few minutes to complete.' Below this message, there is a table with the following data:

	Pathmon Name	Running Processes	State
<input type="radio"/>	\$YSB6	2	FROZEN
<input type="radio"/>	\$ZSB6	2	RUNNING

- ▲ **WARNING.** Wait for a minute or two for the operation to complete before performing any other operation on the NSJSP Manager application interface. The time required for an operation to complete depends on the number of requests serviced by the server class at the time of the stop operation and the number of server class processes that are running.

Start the NSJSP Server Class

Using the NSJSP Manager application, you can start an NSJSP Server Class that is in the STOPPED or THAWED state.

To start the NSJSP Server Class, complete the following steps:

1. Click the **Server Class** tab.
The System Information page appears.
2. Click the **Server Class Operations**.
3. Perform one of the following steps depending on the number of PATHMONs configured:
 - a. If the server class is configured on only one PATHMON, click **Start**.
 - b. If the server class is configured in a TS/MP Pathway domain, complete the following steps:
 1. Select the **Pathmon Name**, which has the server class to be started.
 2. Click **Start**.

The following message is displayed for Steps 3a and 3b:

Start Initiated. The operation may take a few minutes to complete.

Figure 4-23 shows the Server Class Operations page after starting the server class configured under the \$YSB6 PATHMON.

Figure 4-23. Server Class with the \$YSB6 PATHMON in the RUNNING State



The server class starts and you can access only those applications that are running on the server class.

Freeze the NSJSP Server Class

Using the NSJSP Manager application, you can freeze an NSJSP Server Class that is in the `RUNNING` or `THAWED` state. After performing a `FREEZE` operation, no new links will be assigned to the link manager, and the link manager will stop using the current links.

To freeze the NSJSP Server Class, complete the following steps:

1. Click the **Server Class** tab.

The System Information page appears.

2. Click the **Server Class Operations**.
3. Perform one of the following steps depending on the number of PATHMONs configured:

- a. If the server class is configured in one PATHMON, click **Freeze**.

The following message is displayed:

The operation will make the server class unavailable for the user. Do you still want to freeze the server class?

Click **OK**.

- b. If the server class is configured in a TS/MP Pathway domain, complete the following steps:

1. Select the **Pathmon Name**, which has the server class to be frozen.

2. Click **Freeze**.

The following message is displayed:

The operation will make the server class unavailable for the user. Do you still want to freeze the server class?

3. Click **OK**.

The following message is displayed for Steps [3a](#) and [3b](#):

Freeze successful.

In the `Frozen` state, no new links will be assigned to the link manager, and the link manager will stop using the current links.

[Figure 4-24](#) shows the Server Class Operations page after freezing the server class configured under the `$ZSB6` PATHMON.

Figure 4-24. Server Class with the \$ZSB6 PATHMON in the FROZEN State

Thaw the NSJSP Server Class

Using the NSJSP Manager application, you can thaw an NSJSP Server Class that is in the FROZEN or STOPPED state.

To thaw the NSJSP Server Class, complete the following steps:

1. Click the **Server Class** tab.
The System Information page appears.
2. Click the **Server Class Operations**.
3. Perform one of the following steps depending on the number of PATHMONs configured:
 - a. If the server class is configured in one PATHMON, click **Thaw**.
The following message is displayed:
Thaw successful
 - b. If the server class is configured in a TS/MP Pathway domain, complete the following steps:
 1. Select the **Pathmon Name**, which has the server class to be thawed.
 2. Click **Thaw**.

The following message is displayed and the server class state changes to Running as shown in [Figure 4-25](#):

Thaw successful

Figure 4-25. Server Class with the \$ZSB6 PATHMON in the RUNNING State

After changing the PATHMON status to **THAWED**, select the same PATHMON name and click **Start** to restart the server class.

Viewing MBeans

The NSJSP resources, such as, server, connector, and engine are instrumented by MBeans. For more information on MBeans, see [Appendix A, MBeans in the NSJSP Container](#).

You can view the list of MBeans running in each NSJSP process within an NSJSP Server Class.

To view MBeans, complete the following tasks:

1. Click the **MBeans** tab.

The NSJSP MBeans page appears. The **NSJSP Process** list is displayed from which you must select the NSJSP process name.

2. Select an NSJSP process from the **NSJSP Process** list and click **Load**.

The NSJSP process name is displayed in the **MBeans** frame.

3. To view the MBeans in the selected NSJSP process, expand the MBean tree in the **MBeans** frame.

A leaf node in the MBean tree denotes an MBean.

4. To view MBean attribute names and their values, click the leaf node for an MBean.

The MBean attributes and their corresponding values, object name, and a description of the MBean are displayed in the right pane.

[Figure 4-26](#) shows a sample NSJSP MBeans page.

Figure 4-26. NSJSP MBeans Page

NSJSP Process: \$X98V Load

MBeans

- \$X98V
 - Catalina
 - JMXImplementation
 - NSJSP
 - Cache
 - Connector
 - Controller
 - Deployer
 - localhost (selected)
 - Engine
 - GlobalRequestProcessor
 - Host
 - JspMonitor
 - Loader

Object Name: NSJSP:type=Deployer;host=localhost
Description:

[Compare across Processes](#) [Display Attribute Description](#)

Name	Value
configBaseName	/home/nsjsp6/jspdev/sajad/vel/nsjsp/0329_sca6/conf/NSJSP/localhost
configClass	org.apache.catalina.startup.ContextConfig
contextClass	org.apache.catalina.core.StandardContext
contextToBePrefixed	true
deployXML	true
modelerType	com.tandem.servlet.catalina.startup.NSJSPHostConfig
unpackWARs	true
xmlNamespaceAware	false
xmlValidation	false

You can perform the following operations using the NSJSP MBeans page:

- List the MBeans tree by selecting an NSJSP process from the list and clicking **Load**.
- List the attributes of an MBean by clicking a leaf node from the MBean tree.
- Compare the attributes of the MBean across NSJSP processes by clicking the **Compare across Processes** link. After clicking the link, the Compare - NSJSP MBeans page appears. For more information on comparing MBeans, see [Comparing MBean Attribute Values](#) on page 4-46.
- View the attribute names and values by clicking a leaf node from the MBean tree. You can also view the attribute names and their descriptions by clicking the **Display Attribute Description** link. This link toggles with the **Display Attribute Values** link based on the MBean that is displayed currently.

[Table 4-19](#) lists the parameters on the NSJSP MBeans page.

Table 4-19. Parameters for Viewing MBeans and their Attributes (page 1 of 2)

Parameter	Description
NSJSP Process	Lists all the NSJSP processes in the selected NSJSP Server Class. MBeans run in the NSJSP processes.

Table 4-19. Parameters for Viewing MBeans and their Attributes (page 2 of 2)

Parameter	Description
MBeans	
Left window	A navigation tree that lists all the MBeans running in the selected NSJSP process.
Right window	<p>Following are the fields displayed for a selected MBean:</p> <ul style="list-style-type: none"> ● Object Name - Object name of the MBean. ● Description - Description of the MBean. ● Compare Across Processes - A link to compare the attributes of the displayed MBean across processes. ● Display Attribute Description - A link to display attribute names and their descriptions. This option is enabled only when the attribute values are displayed. ● Display Attribute Values - A link to display attribute names and their values. This option is enabled only when the attribute descriptions are displayed. ● Name - An MBean attribute name. ● Value - An MBean attribute value. Values of MBean attributes provide information about the associated managed resource and may help in performance analysis and troubleshooting.

Managing MBeans

The MBeans tree displays the NSJSP MBeans and MBeans registered by the applications. You can compare MBean attribute values across NSJSP processes and you can modify some MBean attribute values. You can modify some of the attributes of the NSJSP, Catalina, and Users MBeans. You can also modify MBeans that are registered by the applications using the same procedure.

Comparing MBean Attribute Values

MBeans provide a mechanism to view the current state of the resource managed by the MBean. The resource state might include static information and dynamic information. To compare the static or dynamic information across NSJSP processes, use the compare MBean operation.

To compare the values of an MBean attribute, complete the following steps:

1. Click the **MBeans** tab.

The NSJSP MBeans page appears.

2. Click **Compare MBeans**.

The Compare - NSJSP MBeans page appears.

3. In the **MBean Object Name:** field, type the object name of the MBean whose attribute values you want to compare.

For example, if you enter the following string in the **MBean Object Name:** field and click **Compare All Attributes:**

```
NSJSP:j2eeType=Servlet,*
```

All MBeans of type `Servlet` that are configured in the NSJSP domain are displayed, as shown in [Figure 4-27](#).

Figure 4-27. List of MBeans

Compare MBeans across NSJSP Processes

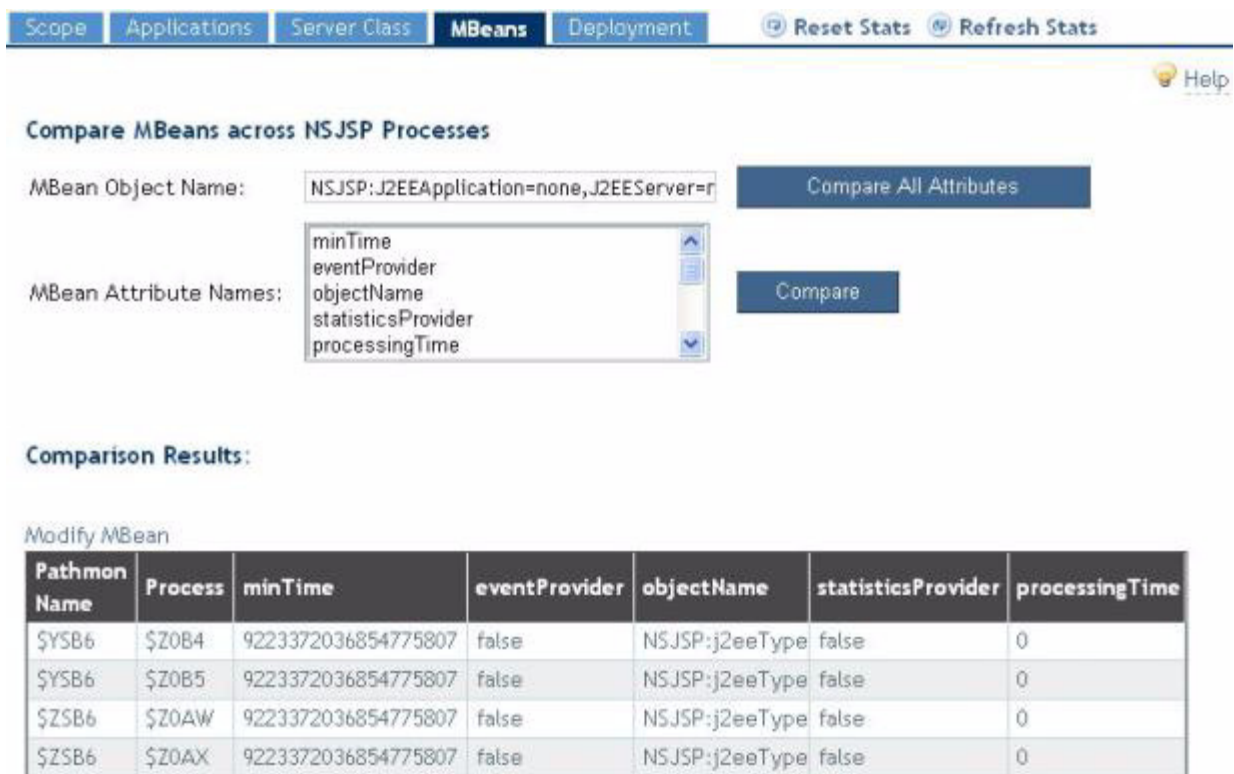
MBean Object Name: Compare All Attributes

The passed MBean Object Name matches with multiple MBeans listed below.
Select the MBean that you want to compare.

MBean Name
NSJSP: J2EEApplication=none, J2EEServer=none, WebModule=//localhost/sca6url, j2eeType=Servlet, name
NSJSP: J2EEApplication=none, J2EEServer=none, WebModule=//localhost/sca6url, j2eeType=Servlet, name
NSJSP: J2EEApplication=none, J2EEServer=none, WebModule=//localhost/sca6url/admin, j2eeType=Servlet
NSJSP: J2EEApplication=none, J2EEServer=none, WebModule=//localhost/sca6url/admin, j2eeType=Servlet
NSJSP: J2EEApplication=none, J2EEServer=none, WebModule=//localhost/sca6url/admin, j2eeType=Servlet
NSJSP: J2EEApplication=none, J2EEServer=none, WebModule=//localhost/sca6url/admin, j2eeType=Servlet
NSJSP: J2EEApplication=none, J2EEServer=none, WebModule=//localhost/sca6url/admin, j2eeType=Servlet
NSJSP: J2EEApplication=none, J2EEServer=none, WebModule=//localhost/sca6url/admin, j2eeType=Servlet
NSJSP: J2EEApplication=none, J2EEServer=none, WebModule=//localhost/sca6url/admin, j2eeType=Servlet
NSJSP: J2EEApplication=none, J2EEServer=none, WebModule=//localhost/sca6url/admin, j2eeType=Servlet
NSJSP: J2EEApplication=none, J2EEServer=none, WebModule=//localhost/sca6url/admin, j2eeType=Servlet
NSJSP: J2EEApplication=none, J2EEServer=none, WebModule=//localhost/sca6url/admin, j2eeType=Servlet

4. Click the MBean name that you want to compare.

After selecting an MBean, a list of all the attributes for that MBean across all the NSJSP processes is displayed, as shown in [Figure 4-28](#).

Figure 4-28. Compare - NSJSP MBeans Page


Scope Applications Server Class **MBeans** Deployment Reset Stats Refresh Stats

Help

Compare MBeans across NSJSP Processes

MBean Object Name: NSJSP:J2EEApplication=none,J2EEServer=r Compare All Attributes

MBean Attribute Names: minTime eventProvider objectName statisticsProvider processingTime Compare

Comparison Results:

Modify MBean

Pathmon Name	Process	minTime	eventProvider	objectName	statisticsProvider	processingTime
\$YSB6	\$Z0B4	9223372036854775807	false	NSJSP:j2eeType	false	0
\$YSB6	\$Z0B5	9223372036854775807	false	NSJSP:j2eeType	false	0
\$ZSB6	\$Z0AW	9223372036854775807	false	NSJSP:j2eeType	false	0
\$ZSB6	\$Z0AX	9223372036854775807	false	NSJSP:j2eeType	false	0

If you only want to compare a few attributes across all the NSJSP processes, select the attribute names from the **MBean Attribute Names:** field and click **Compare**. [Figure 4-29](#) shows the comparison result for the `modelerType` attribute across all the NSJSP processes.

Figure 4-29. Comparison Result for the modelerType Attribute

Scope Applications Server Class **MBeans** Deployment Reset Stats Refresh Stats

Help

Compare MBeans across NSJSP Processes

MBean Object Name: NSJSP:J2EEApplication=none,J2EEServer=r Compare All Attributes

MBean Attribute Names: engineName maxTime errorCount **modelerType** loadTime Compare

Comparison Results:

Modify MBean

Pathmon Name	Process	modelerType
\$YSB6	\$Z0B4	org.apache.catalina.core.StandardWrapper
\$YSB6	\$Z0B5	org.apache.catalina.core.StandardWrapper
\$ZSB6	\$Z0AW	org.apache.catalina.core.StandardWrapper
\$ZSB6	\$Z0AX	org.apache.catalina.core.StandardWrapper

Note. If you navigate the Compare - NSJSP MBeans page by clicking the **MBeans List** option and then clicking the **Compare across Processes** link, the object name and the attribute values for that MBean, across the NSJSP processes, are displayed on the page.

You can perform the following operations from the Compare - NSJSP MBeans page:

- Compare all the attribute values of a selected MBean across processes by specifying the object name of an MBean in the **MBean Object Name:** field and clicking **Compare All Attributes**.
- List the MBeans. MBean object names can be entered with wild card characters to search for all MBeans that match a domain and set of key-properties. The key-properties are the property-value pairs, which are represented as `name=value` in an object name. For more information on object names and key-properties, see [Appendix A, MBeans in the NSJSP Container](#).

The following are some examples:

- To list all MBeans under the domain `NSJSP`, enter the following string in the **MBean Object Name:** field and click **Compare All Attributes:**

`NSJSP:*`

- To list all the session managers configured in the application running in the `host=localhost`, enter the following string in the **MBean Object Name:** field and click **Compare All Attributes:**

`NSJSP:type=Manager,host=localhost,*"`

- Modify an MBean. The **Modify MBean** option is displayed only when the attributes of an MBean are displayed for all the processes.

[Table 4-20](#) lists the parameters in the Compare - NSJSP MBeans page.

Table 4-20. Parameters in the Comparing MBeans Across NSJSP Processes Page

Parameter	Description
MBean Object Name:	The name of the MBean whose attributes you want to compare across NSJSP processes.
Compare All Attributes	A link to compare all attributes of the MBean across NSJSP processes.
MBean Attribute Names:	A list of all the MBean attributes from which individual attributes can be selected for comparison.
Compare	A link to compare the selected attributes of the MBean across NSJSP processes.
Comparison Results:	Table that includes a comparison of MBean attribute values across NSJSP processes.
Modify MBean	A link to the Modify MBean page, which allows you to modify the values of some MBean attributes of the selected MBean.

Modifying MBean Attribute Values

To tune NSJSP as per your requirement, modify the MBean attribute values. The modify operation changes the corresponding value on the NSJSP processes that are in the running state. After modification, the NSJSP process behaves as per the modified value of the MBean attribute.

To modify a value of the MBean attribute, complete the following steps:

1. Click the **MBeans** tab.

The NSJSP MBeans page appears.

2. Click **Modify MBean**.

The Modify MBean page appears.

3. In the **MBean Object Name:** field, type the object name of the MBean whose attribute values you want to modify.
4. Click **Load Attribute list**.

The following fields are displayed on the Modify MBean page that must be set to modify an MBean attribute:

- **NSJSP Process Name:**
- **Attribute Name:**
- **New Attribute Value:**

The **Set New Attribute Value** button and the actual attribute values across NSJSP processes are also displayed on the Modify MBean page.

5. Select the NSJSP process name from the **NSJSP Process Name:** list. You can select one or multiple NSJSP processes. Only MBeans in the selected processes will have their attribute modified.
6. Select the attribute name that you want to modify from the **Attribute Name:** list.
7. In the **New Attribute Value:** field, enter the new attribute value.

Note. The attribute value you specify in the **New Attribute Value:** field depends on the attribute that you are modifying. For example, if the attribute `cacheMaxSize` needs to be modified, enter an integer value.

8. Click **Set New Attribute Value**.

The new value is assigned to the selected MBean attribute of the selected NSJSP processes. [Figure 4-30](#) shows a sample Modify MBean page.

Figure 4-30. Modify MBean Page

Scope

Applications

Server Class

MBeans

Deployment

Reset Stats

Refresh Stats

Help

Modify MBean

MBean Object Name:

NSJSP:type=Engine

Load Attribute list

NSJSP Process Name:

\$Z0AW

\$Z0AX

\$Z0B4

\$Z0B5

Attribute Name:

jvmRoute

New Attribute Value:

testRoute

Set New Attribute Value

Attribute values across NSJSP processes.

Pathmon Name	Process	jvmRoute	managedResource	baseDir	valveObjectNames	realm	name	defaultHost	modelerType
\$YSB6	\$Z0B4		Unavailable	/home/nsjsp	[Ljavaax.manager	Unavailable	NSJSP	localhost	org.apache.ca
\$YSB6	\$Z0B5		Unavailable	/home/nsjsp	[Ljavaax.manager	Unavailable	NSJSP	localhost	org.apache.ca
\$ZSB6	\$Z0AW		Unavailable	/home/nsjsp	[Ljavaax.manager	Unavailable	NSJSP	localhost	org.apache.ca
\$ZSB6	\$Z0AX		Unavailable	/home/nsjsp	[Ljavaax.manager	Unavailable	NSJSP	localhost	org.apache.ca

[Table 4-21](#) lists the parameters displayed in the Modify MBean page.

Table 4-21. Parameters in the Modify MBean Page

Parameter	Description
MBean Object Name:	Name of the MBean with the attribute that you want to modify.
NSJSP Process Name:	A list of NSJSP processes to which you may apply the new attribute value. You can select one or more NSJSP processes to have the selected attribute modified.
Attribute Name:	A list of MBean attributes. You select the MBean attribute that you want to modify.
New Attribute Value:	New attribute value that you want to assign.
Attribute values across NSJSP processes.	Table that lists the attribute values across NSJSP processes.

Deploying Web Applications

The application resources required for deploying an application can be present either on the workstation (where the browser is running) or on the NonStop server (where the NSJSP processes are running). Based on the location of the application resources, there are two types of web application deployments:

- Deployment from a server
- Deployment from a desktop

Note. NSJSP does not support context names, such as, `/my/context`, that include multiple `'/'` characters.

Deploying Web Applications from the Server

If an application resource already exists on the server in which NSJSP is running, use the server deployment option.

To deploy an application from the server, complete the following steps:

1. Click the **Deployment** tab.
The Application Deployment page appears.
2. Under **Web Application Deployment from Server**, complete the following steps:
 - a. In the **Context name** field, type the name of the context.

This is a required field.

- b. In the **Context Configuration File Location** field, you may enter the location of a context configuration file.

Note. The **Context Configuration File Location** field is required only if you want to deploy the application using a context configuration file. Otherwise, it can be skipped.

- c. In the **War or Directory Location** field, type the location of the `.war` file or the application directory path.
- d. Select one of the following check boxes based on your requirements:

- **Automatically Add Filemap**
- **Automatically Prefix Context Path**

See [Table 4-22](#) for detailed explanation of the above listed options.

By default, the **Automatically Prefix Context Path** check box is selected.

3. Click **Deploy**.

The web application is deployed in the selected server class and Host.

[Figure 4-31](#) shows the **Web Application Deployment from Server** window.

Figure 4-31. Web Application Deployment from Server

Scope Applications Server Class MBeans Deployment Refresh Stats

Help

Web Application Deployment from Server

Context name (ex. /dummy)

Context Configuration File Location (ex. /myContexts/Hello.xml)

War File or Webapp Directory Location (ex. /myWebapps/myapp.war, /myWebapps/myappdir/)

☐ Automatically Add Filemap

☒ Automatically Prefix Context name with URI name

Deploy

[Table 4-22](#) lists the attributes displayed under **Web Application Deployment from Server** on the Application Deployment page.

Table 4-22. Attributes on the Web Application Deployment from Server Page (page 1 of 2)

Attribute	Description
Context name	The context name to use for the application.
Context Configuration File Location	Absolute path of the application context file.
War or Webapp Directory Location	The application directory or the directory path and name of the .war file that you want to deploy.

Table 4-22. Attributes on the Web Application Deployment from Server

Page (page 2 of 2)

Attribute	Description
Automatically Add Filemap	Check box to enable the creation of a Filemap file entry for the application. The entries in the <code>filemaps.config</code> file are used by the Servlet Server Class during startup to determine the path to the application. Applications require a Filemap file entry. For more information on the context prefix for an application, see Chapter 3, Configuring NSJSP .
Automatically Prefix Context name with URI name	Check box to enable the application to be deployed along with the prefixed context name. Using this option you can prefix the context name with the URI name that was assigned to the NSJSP installation when the <code>setup</code> script was run. A filemap will already exist for the URI name. The <code><URI name></code> is specified during NSJSP installation, when the <code>setup</code> script is run. For example, if you set the context name to <code>/first</code> and select the Automatically Prefix Context name with URI name check box, the application will be deployed with the context name, <code>/<URI name>/first</code> . If you do not select the Automatically Prefix Context name with URI name check box, the application will be deployed with the context name <code>/first</code> and a filemap entry will be created for the application <code>/first</code> .
Deploy	Button used to deploy the application.

Deploying Web Applications from the Desktop

If an application is on the desktop and you need to deploy it, use the desktop deployment option.

Note. The application that you want to deploy must be packaged as a valid `.war` file.

To deploy an application from the desktop, complete the following steps:

1. Click the **Deployment** tab.
The Application Deployment page appears.
2. Under **Web Application Deployment from Desktop**, complete the following steps:
 - a. In the **Select a .war file to upload** field, click **Browse** to locate the `.war` file.
 - b. Type the context name of the application in the **Context name** field.

Note. The context name must be prefixed by `/`.

3. Click **Deploy**.

The web application is deployed in the selected server class and Host.

[Figure 4-32](#) shows the **Web Application Deployment from Desktop** window.

Figure 4-32. Web Application Deployment from Desktop

[Table 4-23](#) lists the attributes displayed under **Web Application Deployment from Desktop** on the Application Deployment page.

Table 4-23. Attributes on the Web Application Deployment from Desktop Page

Attribute	Description
Select a .war file to upload	The Browse... button by this field must be used to select the .war file that you want to deploy. The Browse... button allows you to view the desktop and to navigate the workstation file system to select the file to upload.
Context name	The context name to use for the application.
Deploy	Button used to deploy the application.

Admin Web Application

The Admin Web application is a web-based application that is associated with an NSJSP installation. It enables you to examine and modify parts of some configuration files, such as, `server.xml`, `context.xml`, and `nsjsp-users.xml`.

When you modify the configuration information using the Admin Web application, the changes made in the configuration files are persisted. This means that the changes will exist even after the NSJSP Server Class is restarted. However, when changes are made with the Admin Web application, the changes will be applied to the NSJSP processes only after the changes have been saved and then committed from the Admin Web application.

Note.

- HP recommends that you perform the `Commit Changes` operation only after making all changes to the configuration.
 - The Admin Web application can manage an NSJSP Server Class in only one PATHMON when NSJSP 6.1 is installed in an iTP Secure WebServer environment that is configured for online-upgrade.
-

This section describes the following topics:

- [Overview and Architecture](#)
- [Admin Web Application Features](#)
- [Login and Security Considerations](#)
- [Managing Admin Web Application Operations](#)

Overview and Architecture

An NSJSP 6.1 configuration can be changed by editing configuration files and then stopping and restarting the iTP Secure WebServer environment. However, it is also desirable to make changes dynamically without stopping the NSJSP processes. Such changes would also need to be made to the configuration files, if they need to be persisted. The Admin Web application provides this capability for a single PATHMON. It can update a configuration file with changes and distribute those changes to every running process in the NSJSP Server Class under a PATHMON. The Admin Web application runs in a separate NSJSP Admin Server Class from the NSJSP Servlet Server Class. The name of the Admin Server Class is derived from the NSJSP Server Class name. The Admin Server Class name is `<NSJSP server class name>-adm`.

Admin Web application requests are routed to the Admin Server Class (through a `filemap` created for the iTP Secure WebServer). The administrator browses objects and performs updates through the Admin Web application pages. When you select **Save** to complete an update, the save operation does not implement the changes in the NSJSP Server Class immediately. You must next click the **Commit Changes** button for the changes to actually be processed. The Admin Web application then updates the relevant configuration file and broadcasts all the saved changes to all running processes of the NSJSP Server Class under the same PATHMON.

Note. If the Admin Server Class fails to start or is stopped for some reason, the NSJSP Server Class continues to run. All web applications, except the Admin Web application and the Manager Web application, are still accessible.

[Figure 4-33](#) shows operator Admin operations received by the iTP Secure WebServer directed to the Admin Server Class. When you click **Commit Changes**, the Admin Server Class first updates the configuration file (for example, `server.xml`) and then broadcasts all the saved changes in a chunk to every running process of the NSJSP

Server Class, as shown in [Figure 4-34](#). As a result, any NSJSP processes subsequently created have the same configuration as the currently running processes.

Figure 4-33. Admin Operations followed by SAVE

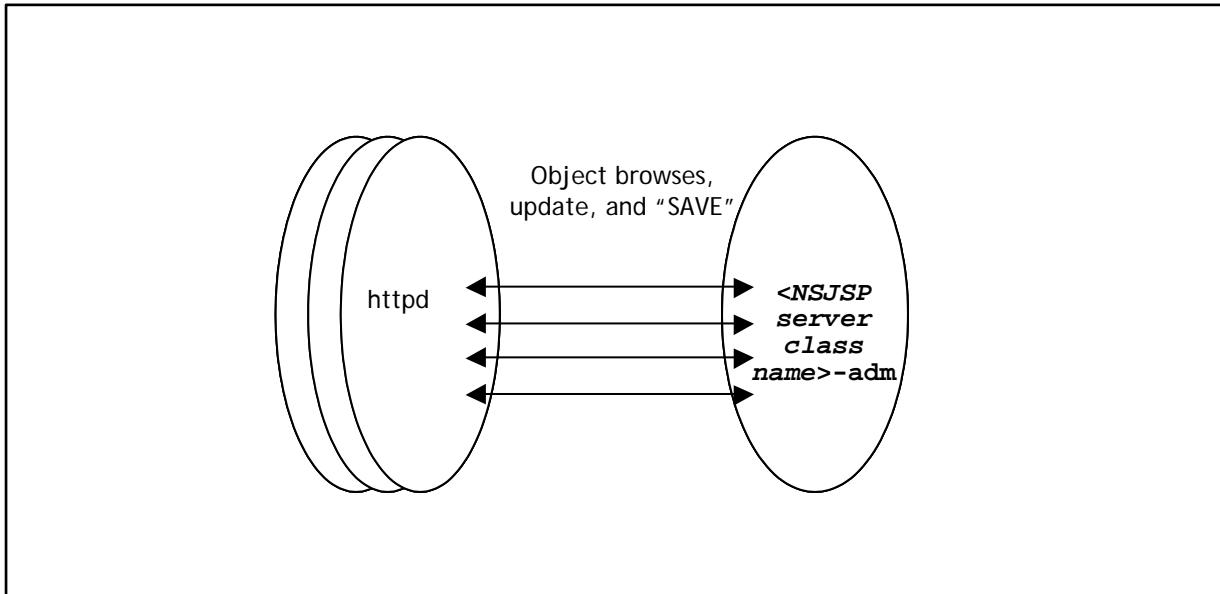
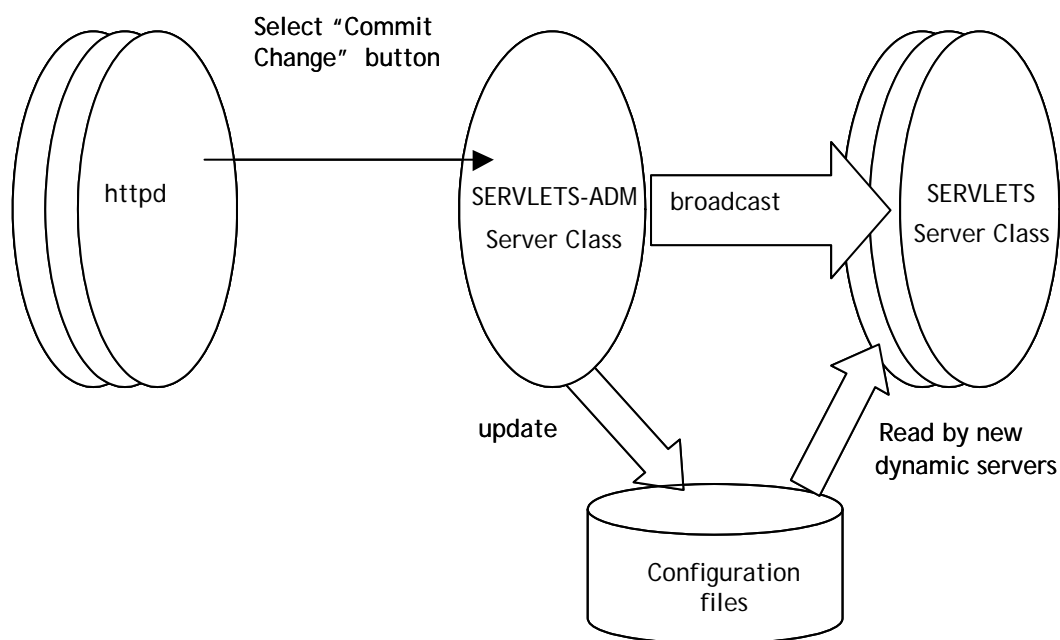


Figure 4-34. Operator Commit Changes Command



Admin Web Application Features

The Admin Web application includes a User Interface (UI) to perform the operations related to the NSJSP servlet container components, such as, Service and Host. The Admin Web application enables you to do the following:

- Modify the configuration files.
- Create new NSJSP servlet container components.
- Delete existing NSJSP servlet container components.
- Create users, groups, and roles for the users.
- Delete existing users, groups, and roles.
- Perform the `Commit Changes` operation for saved changes to propagate those changes to all the associated NSJSP Server Class processes in the PATHMON and update the configuration files.

Login and Security Considerations

The Admin Web application uses the FORM method for its login authentication and requires the admin role for access control, by default. The admin role is for users who need to perform Admin Web application operations.

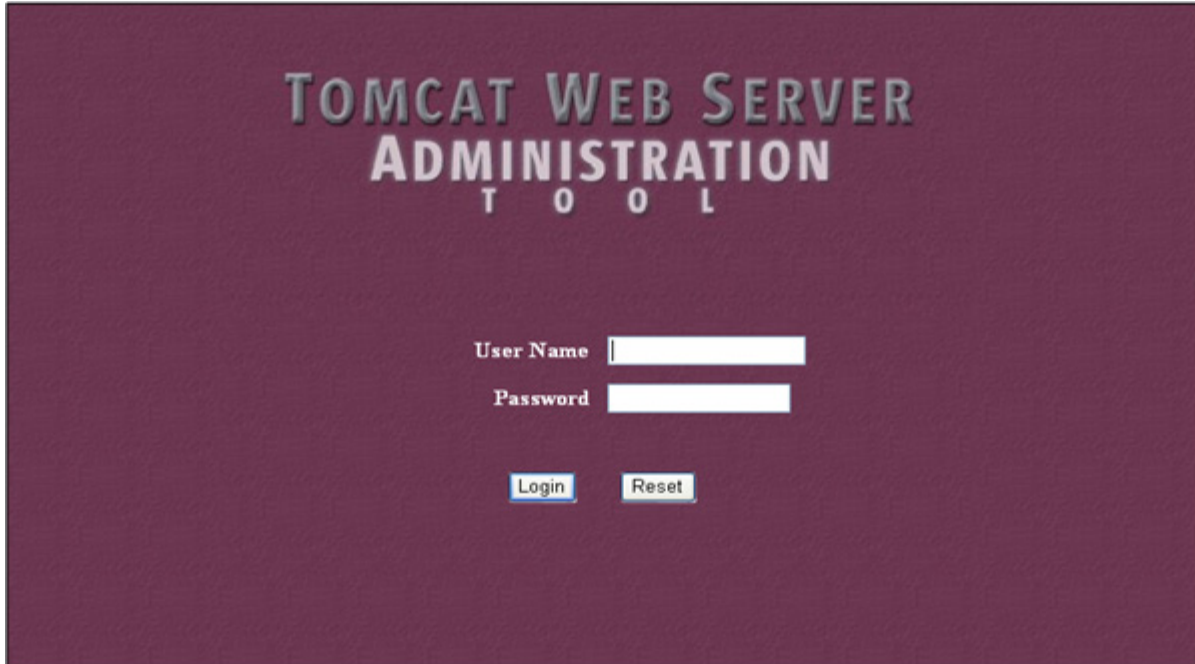
For a more secure environment, you could configure the Admin Web application in a private virtual Host. In addition, if you need remote access to the Admin Web application, you should consider configuring the Admin Web application to use the Secure Sockets Layer (SSL).

To access the NSJSP Admin Web application, you must enter a URL in the following format:

```
http://IP address:Port number/<URI for NSJSP 6.1  
Installation>/admin
```

Note. The URL for the NSJSP Admin Web application can be retrieved from the `<NSJSP 6.1 Installation Directory>/install.log` file.

[Figure 4-35](#) shows the login page of the NSJSP Admin Web application.

Figure 4-35. Admin Login pageThe image shows the Tomcat Web Server Administration login page. It has a dark red background. At the top, the text "TOMCAT WEB SERVER" is in a large, bold, serif font, with "ADMINISTRATION" below it in a slightly smaller, bold, serif font. Underneath "ADMINISTRATION", the letters "T O O L" are spaced out. In the center, there are two white input fields. The first is labeled "User Name" and the second is labeled "Password". Below the input fields are two buttons: "Login" and "Reset".

Note. To log in to the Admin Web application, enter the user name as admin and use the same password that you provided when the `setup` script was run.

An error message is displayed if you are not authorized to access the Admin Web application or if the authentication process fails.

After completing the authentication and authorization processes, the Admin Web Application home page is displayed as shown in [Figure 4-36](#).

Figure 4-36. Admin Web Application Home Page

The Admin Web application home page is divided into two panes: the left pane and the right pane. The left pane includes elements to manage the servlet container components and to perform tasks, such as, adding users and groups. These elements are displayed in a tree structure.

The left pane displays the following primary elements:

- Tomcat Server
- Resources
- User Definition

The primary elements can contain child elements. If a primary element has child elements, a key is displayed to the extreme left of the element name entry. The position of the key can be toggled by clicking on it. When the key points right, sub-elements are not displayed. When the key points down, sub-elements are displayed. The right pane displays the details of the element that is selected in the left pane. An element in the left pane is selected when you click the name of the element. The element has been selected, when the element name is displayed in bold text.

The Admin Web application also includes the following operations:

- **Save Changes** – The changes made before performing the `Commit Changes` operation are stored in the Admin Web application and not propagated to the NSJSP Server Class processes till you click **Commit Changes**.

- **Commit Changes** – This operation commits all the previously saved changes, notifies the running NSJSP processes of the changes and updates the configuration files.
- **Log Out** – Ends the current user session and requires a new login to access the Admin Web application.

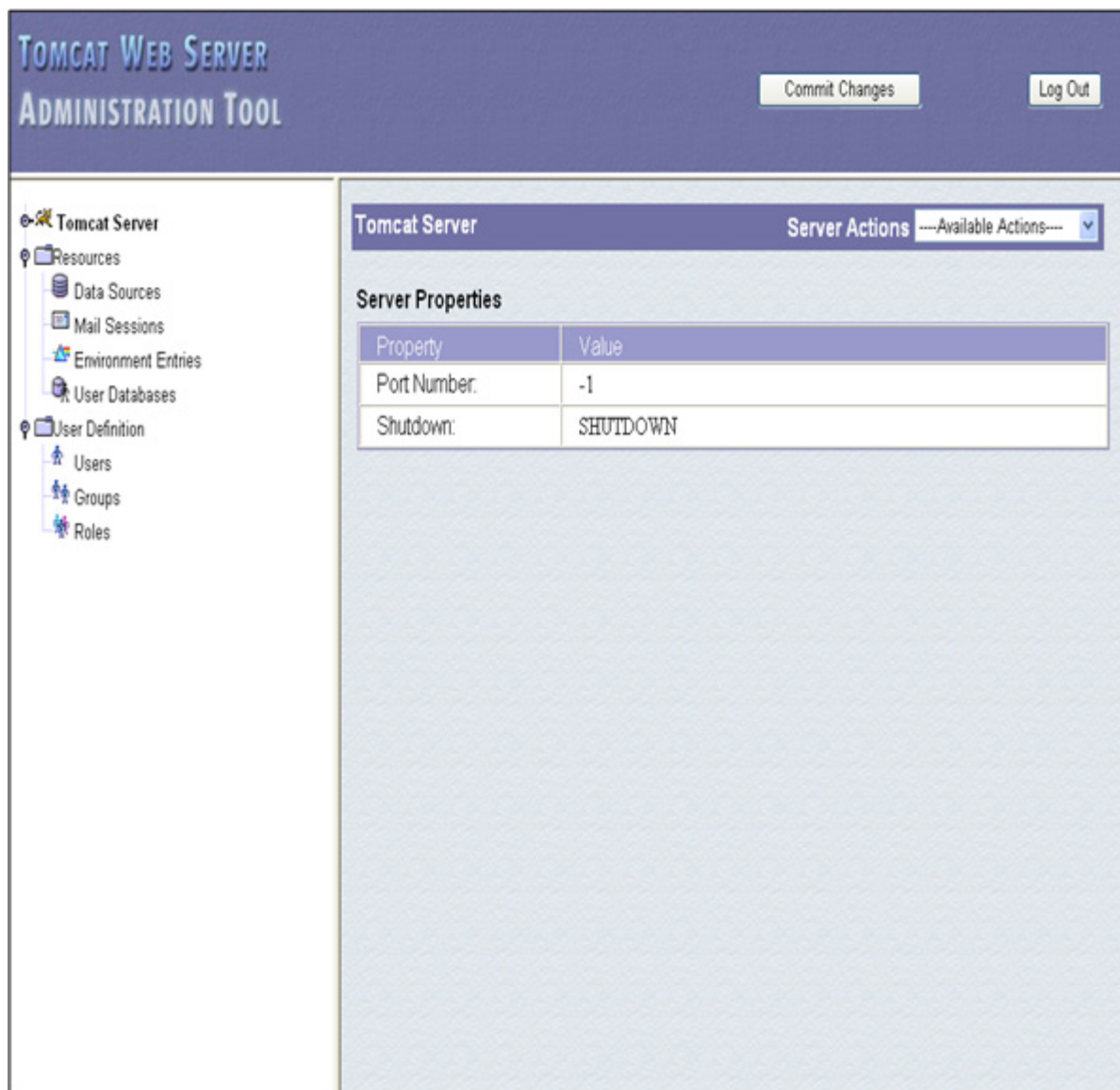
Managing Admin Web Application Operations

This section lists all the actions that you can perform using the Admin Web application. It discusses the following topics:

- [Administering the Server](#)
- [Administering a Service](#)
- [Administering a Connector](#)
- [Administering a Host](#)
- [Administering a Context](#)
- [Administering a Realm](#)
- [Administering a Valve](#)
- [Administering Resources](#)
- [Administering User Definitions](#)
- [Access Security Considerations](#)
- [Persisting Changes to the server.xml File and Context Files](#)

Administering the Server

Server represents the NSJSP container. After logging in to the Admin Web application, click the text **Tomcat Server** in the left pane. The **Server Actions** list and two properties of the **Server** element are displayed in the right pane, as shown in [Figure 4-37](#).

Figure 4-37. The Tomcat Server Element

You can select one of the following actions from the **Server Actions** list:

- Create New Service

- Delete Existing Services

△ **Caution.**

Although you can add more than one Service element, HP recommends that you retain only one Service element due to the following reasons:

- Each Service element requires a Connector element and NSJSP provides only one Connector implementation, which processes messages from the \$RECEIVE file. Two Connector definitions cannot use the same connector implementation. Therefore, the number of Service definitions is restricted to one.
- The NSJSP Manager can manage applications in only one service (that is the default NSJSP service).

The NSJSP Manager and the Admin Web application will stop working if you delete the default service.

For more information on the Server element, see the [Server Element](#) on page 3-40.

Administering a Service

A service represents the combination of one or more Connectors that share a single Engine to process incoming requests.

To administer a Service, you must click the **Service** name under **Tomcat Server** displayed in the left pane of the tree structure. The default Service entry is **Service (NSJSP)**. The name of each Service is displayed in parentheses.

[Figure 4-38](#) displays the **Service Actions** list, and a few properties of the NSJSP **Service** and **Engine** elements.

Figure 4-38. The Service Element

TOMCAT WEB SERVER ADMINISTRATION TOOL

Commit Changes Log Out

Tomcat Server

- Service (NSJSP)
- Resources
 - Data Sources
 - Mail Sessions
 - Environment Entries
 - User Databases
- User Definition
 - Users
 - Groups
 - Roles

Service (NSJSP) Service Actions Available Actions

Save Reset

Service Properties

Property	Value
Name:	NSJSP

Engine Properties

Property	Value
Name:	NSJSP
Default Hostname:	localhost

Save Reset

You can select one of the following actions from the **Service Actions** list:

- Create New Connector
- Delete Existing Connectors
- Create New Host
- Delete Existing Hosts
- Create New Valve

- Delete Existing Valves

△ **Caution.**

- HP recommends that you always have only one Connector element. Therefore, do not use the following actions:
 - Create New Connector
 - Delete Existing Connectors

For more information, see the Caution on page [4-64](#).

- Do not use the NSJSP Manager to manage Hosts, that are added using the Admin Web application, unless the Host is configured with a Request Tracker Valve in the `server.xml` file.
 - The Delete Existing Hosts action does not work in the NSJSP 6.1 release. However, you can manually remove a `<Host>` element from the `server.xml` file.
-

For more information on the Service element, see the [Service Element](#) on page 3-43.

For more information on the Engine element, see the [Engine Element](#) on page 3-50.

For more information on the Host element, see the [Host](#) on page 3-54.

Administering a Connector

A connector represents a communications end point on which requests are received from a client. NSJSP uses a NonStop-specific connector which works with the iTP Secure WebServer to process requests.

To administer a Connector, click the **Connector** under the appropriate **Service** name displayed in the left pane.

If the connector is not displayed, click the key that is located to the left of the **Service** icon. The key will then point down and the tree structure will be expanded to display the following sub-elements:

- **Connector (0)**
- **Host (localhost)**
- **Realm for Service (NSJSP)**

[Figure 4-39](#) displays the **Connector Actions** list and the properties of the **Connector** element in the right pane.

Figure 4-39. The Connector Element

TOMCAT WEB SERVER ADMINISTRATION TOOL

Commit Changes Log Out

Tomcat Server

- Service (NSJSP)
 - Connector (0)**
 - Host (localhost)
 - Realm for Service (NSJSP)
- Resources
 - Data Sources
 - Mail Sessions
 - Environment Entries
 - User Databases
- User Definition
 - Users
 - Groups
 - Roles

Connector (0) Connector Actions Available Actions

Save Reset

General	
Type:	HTTP
Scheme:	http
Enable DNS Lookups:	False
URI Encoding:	
Use Body Encoding For URI Query Parameters:	False
Allow TRACE Method:	False
IP Address:	
Secure:	False
Accept Count:	25
Compression:	off
Connection Linger (milliseconds):	-1
Connection Timeout (milliseconds):	0
Connection Upload Timeout (milliseconds):	300000
Default Buffer Size:	2048
Disable Upload Timeout:	True
Max KeepAlive Requests:	100
Max Spare Threads:	null
Max Threads:	75
Min Spare Threads:	null
Processor Thread Priority:	5
TCP No Delay:	True
X Powered By:	False
Ports	
Port Number:	0

Save Reset

For more information on the Connector element, see the [Connector Element](#) on page 3-45.

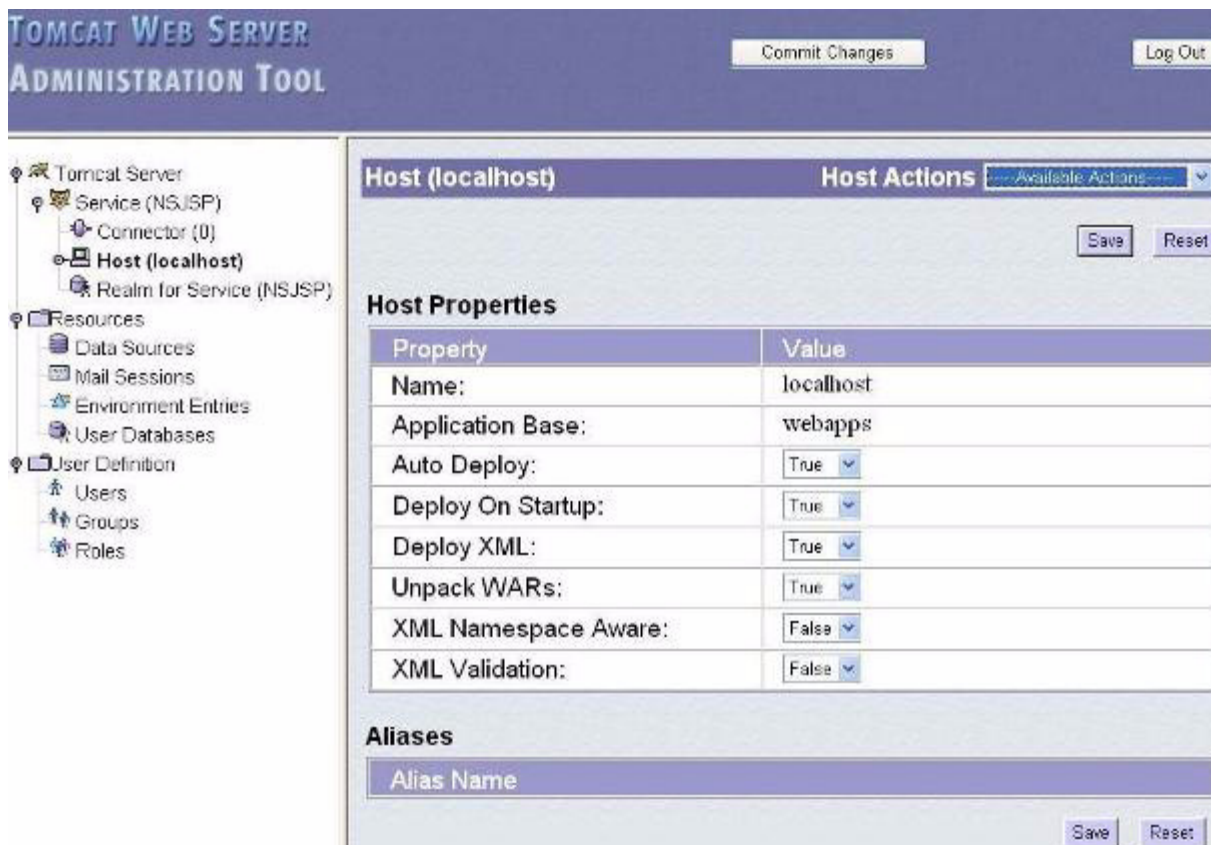
Note. The **Connector Actions** list includes the Delete This Connector action. If you perform this action, the NSJSP Server Class and the Admin Web application will become non functional.

Administering a Host

The Host represents a virtual Host. One or more Hosts are nested inside an Engine. To administer a Host, select **Host** under the appropriate **Service** displayed in the left pane.

[Figure 4-40](#) displays the **Host Actions** list and the properties of the **Host** element in the right pane.

Figure 4-40. The Host Element



The following actions are available under the **Host Actions** list:

- Create New Aliases
- Delete Aliases
- Create New Context
- Delete Existing Contexts
- Create New Valve
- Delete Existing Valves
- Delete This host

Note.

- An alias is the alias of a Host. A Host can have multiple aliases.
 - NSJSP 6.1 does not support the Create New Context and Delete Existing Contexts actions. Use the NSJSP Manager or the Manager Web application to create or delete a new context.
-

For more information on the Host element, see the [Host](#) on page 3-54.

Administering a Context

A context represents a web application. To administer a context, you must click the appropriate **Context** under the **Service** and the **Host** displayed in the left pane.

If there is no context displayed under the Host in the left pane, click the key that is located to the left of the Host name. The key will then point down and the tree structure will be expanded to display all **Context** and **Valve** sub-elements.

[Figure 4-41](#) displays the **Context Actions** list and the properties of the **Context** element in the right pane.

Figure 4-41. The Context Element

TOMCAT WEB SERVER ADMINISTRATION TOOL

Commit Changes Log Out

Tomcat Server

- Service (NSJSP)
 - Connector (0)
 - Host (localhost)
 - Context (/servlet_jsp)
 - Context (/servlet_jsp/admin)
 - Context (/servlet_jsp/bankapp)
 - Context (/servlet_jsp/docs)
 - Context (/servlet_jsp/examples)
 - Context (/servlet_jsp/host-manager)
 - Context (/servlet_jsp/manager)
 - Realm for Service (NSJSP)
- Resources
 - Data Sources
 - Mail Sessions
 - Environment Entries
 - User Databases
- User Definition
 - Users
 - Groups
 - Roles

Context (/servlet_jsp) Context Actions ---Available Actions---

Save Reset

Context Properties

Property	Value
Cookies:	True
Cross Context:	False
Document Base:	ROOT
Override:	False
Privileged:	False
Path:	/servlet_jsp
Reloadable:	False
Swallow Output:	False
Use Naming:	True
Prevent Jar Locking:	False
Prevent Locking Resources:	False

Loader Properties

Property	Value
Reloadable:	False

Session Manager Properties

Property	Value
Session ID Initializer:	com.tandem.servlet.catalina.session.NSJSPStandardManager@4
Maximum Active Sessions:	-1

Save Reset

You can perform one of the following actions using the **Context Actions** list:

- Create New User Realm
- Delete User Realms
- Create New Valve
- Delete Existing Valves
- Delete This Context

For more information on the Context element, see [Table 3-17, Attribute List for the Context Element](#).

The context is also associated with the NSJSP-specific loader and a Session Manager. Therefore, some attributes of the loader and the Session Manager are displayed along with the attributes of the Context element, as shown in [Figure 4-41](#).

Note. The `Session ID Initializer` attribute of the Session Manager will change the entropy of the context and not the Manager class.

Administering a Realm

A Realm represents a database of the information about authorized users, their passwords, and their assigned access roles. NSJSP supports the following types of Realms:

- JDBCRealm
- MemoryRealm
- UserDatabaseRealm
- DataSourceRealm
- JNDIRealm
- JAASRealm

[Figure 4-42](#) displays the properties of the MemoryRealm.

Figure 4-42. The Properties of MemoryRealm

The screenshot shows the Tomcat Web Server Administration Tool interface. On the left is a sidebar with a tree view containing the following items: Tomcat Server, Service (NSJSP), Connector (0), Host (localhost), Realm for Service (NSJSP), Resources, and User Definition. The main panel has a header bar with 'Tomcat WEB SERVER ADMINISTRATION TOOL' on the left, 'Commit Changes' and 'Log Out' buttons on the right, and a 'Label Actions' dropdown menu. Below the header, the main content area is titled 'Realm (MemoryRealm)'. It contains a table with two columns: 'Property' and 'Value'. The table has two rows: 'Type:' with value 'MemoryRealm' and 'Path Name:' with value 'conf/nsjsp-users.xml'. There are 'Save' and 'Reset' buttons to the right of the table.

Property	Value
Type:	MemoryRealm
Path Name:	conf/nsjsp-users.xml

The attribute list varies for each type of Realm. For more information on Realms and their attributes, see <http://tomcat.apache.org/tomcat-6.0-doc/config/realm.html>.

Administering a Valve

A Valve has distinct processing capabilities, such as, providing access, logging and request filtering.

NSJSP supports the following types of valves:

- AccessLogValve
- RemoteAddrValve

- RemoteHostValve
- RequestDumperValve
- SingleSignOn

For information on valves, see the [Valve Element](#) on page 3-58.

You can create a new valve by clicking one of the following elements in the left pane and selecting **Create New Valve** from the actions list:

- Service – A valve created at the Service level will be applicable to all the applications in the Engine.
- Host – A valve created at the Host level will be applicable to all the applications under that Host.
- Context – A valve created at the Context level will be applicable only to that Context.

[Figure 4-43](#) displays the properties of the AccessLogValve.

Figure 4-43. The Properties of AccessLogValve

TOMCAT WEB SERVER ADMINISTRATION TOOL

Commit Changes Log Out

Create New Valve Valve Actions Available Actions

Save Reset

Access Logger Properties

Property	Value
Type:	AccessLogValve
Directory:	logs
Pattern:	
Prefix:	access_log
Resolve Hosts:	False
Rotatable:	True
Suffix:	

Save Reset

For information on the valves, the different types of Valves and the attributes associated with each Valve, see

<http://tomcat.apache.org/tomcat-6.0-doc/config/valve.html>.

Administering Resources

Resources can be defined either globally or per web application under the **Context** element. The globally defined resources can be accessed using web applications by specifying links to the global resource in the web application itself.

The following are the types of resources:

- [Data Sources](#) (of type `javax.sql.DataSource`)
- [Mail Sessions](#) (of type `javax.mail.Session`)
- [Environment Entries](#) (of various Java language types such as `java.lang.Integer` and `java.lang.String`)
- [User Databases](#)
- [Resource Links](#)

For information on how to define the resources globally, see <http://tomcat.apache.org/tomcat-6.0-doc/config/globalresources.html>.

For information on how to define resources per web application, see <http://tomcat.apache.org/tomcat-6.0-doc/jndi-resources-howto.html>.

For information on how to create resource links, see <http://tomcat.apache.org/tomcat-6.0-doc/config/context.html#Resource%20Links>.

Data Sources

Data sources are resources that are used to perform database operations.

To create a data source, you must click **Data Sources** under **Resources** displayed in the left pane.

[Figure 4-44](#) displays the **Data Source Actions** list and the properties of the **Data Sources** in the right pane.

Figure 4-44. The Data Sources Element

TOMCAT WEB SERVER ADMINISTRATION TOOL

Commit Changes Log Out

Tomcat Server

- Service (NSJSP)
 - Connector (0)
 - Host (localhost)
 - Realm for Service (NSJSP)
- Resources
 - Data Sources**
 - Mail Sessions
 - Environment Entries
 - User Databases
- User Definition

Create New Data Source Data Source Actions Available Actions

Save Reset

Data Sources

Property	Value
JNDI Name:	MyDataSource
Data Source URL:	jdbc:sqlmx:
JDBC Driver Class:	com.tandem.sqlmx.SQLMXDriver
User Name:	
Password:	
Max. Active Connections:	4
Max. Idle Connections:	2
Max. Wait for Connection:	5000
Validation Query:	

Save Reset

You can perform one of the following actions using the **Data Source Actions** list:

- Create New Data Source
- Delete Data Sources

For more information on data sources, see

<http://java.sun.com/javase/6/docs/technotes/guides/jdbc/getstart/datasource.html>.

Mail Sessions

A Mail Session represents a Resource (under the child element of the Server called `GlobalNamingResources`) in the `server.xml` file.

The following sample code shows how the Mail Session is added as Resource in the `server.xml` file:

```
<Resource
name="MyMailSession"
type="javax.mail.Session"
mail.smtp.host="MyMailHost"
/>
```

[Figure 4-45](#) displays the properties after selecting the **Create New Mail Session** action from the **Mail Session Actions** list.

Figure 4-45. The Properties Displayed After Selecting the Create New Mail Session Action



The following are the actions available under the **Mail Session Actions** list:

- Create New Mail Session
- Delete Mail Session

Environment Entries

The `GlobalNamingResources` is an environment entry resource. For more information on `GlobalNamingResources`, see the [GlobalNamingResources Element](#) on page 3-42.

The following sample code shows how the environment variable is set in the `server.xml` file (under `GlobalNamingResources`):

```
<Environment
description=" "
name="MyEnvVar"
type="java.lang.Boolean"
```



```
value="true"
/>
```

[Figure 4-46](#) displays the properties after selecting the **Create New Env Entry** action from the **Environment Entry Actions** list.

Figure 4-46. The Properties Displayed After Selecting the Create New Env Entry Action



The following are the actions available under the **Environment Entry Actions** list:

- Create New Env Entry
- Delete Environment Entries

User Databases

A user database is a database that contains user information, such as user names, passwords, groups, and roles.

The default `server.xml` file defines the `org.apache.catalina.UserDatabase` resource, which is used by the `UserDatabaseRealm` nested in the `Engine` element. For more information on the `UserDatabaseRealm`, see the [UserDatabaseRealm](#) on page 7-19.

The following sample code shows how a user database is added to the `server.xml` file:

```
<Resource
description=" "
name="MyDB"
```

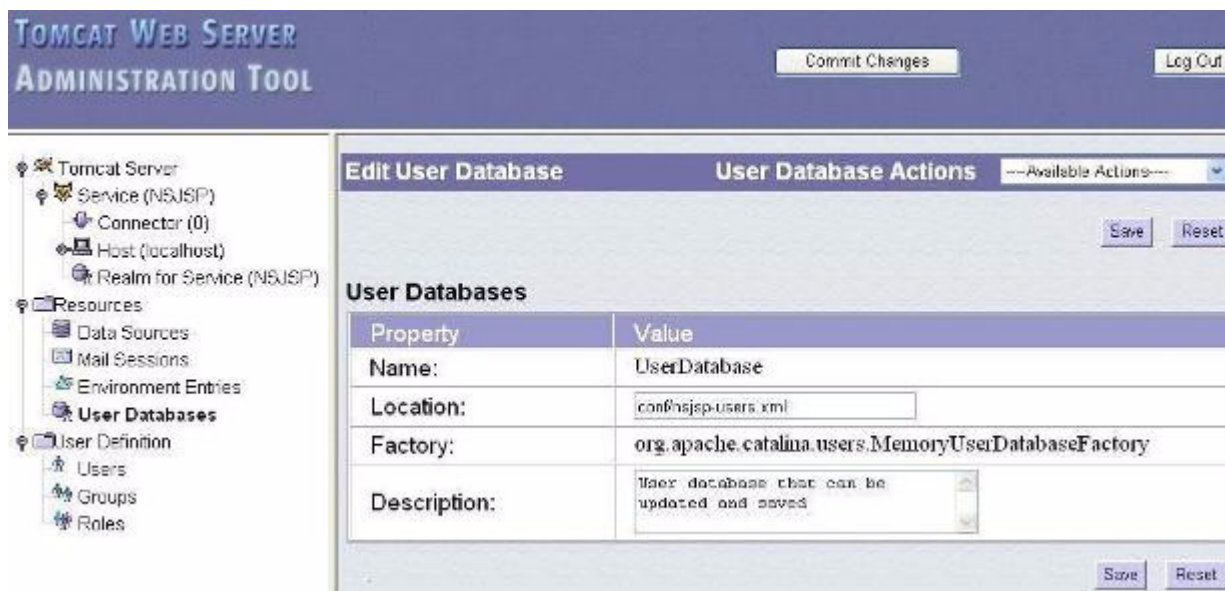
```

type="org.apache.catalina.UserDatabase"
factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
pathname="conf/nsjsp-users1.xml"
/>

```

[Figure 4-47](#) displays the properties of the default User Database.

Figure 4-47. The Properties of the Default User Database



You can perform one of the following actions using the **User Database Actions** list:

- Create New User Database
- Delete User Databases

Resource Links

A resource link is a link to a global resource in the global Java Naming and Directory Interface (JNDI) context.

[Figure 4-48](#) displays the properties after selecting the **Create New Resource Link** action from the **Resource Link Actions** list.

Figure 4-48. The Properties Displayed After Selecting the Create New Resource Link Action



You can perform one of the following actions using the **Resource Link Actions** list:

- Create New Resource Link
- Delete Resource Link

For more information on resource link, see

<http://tomcat.apache.org/tomcat-6.0-doc/config/globalresources.html>.

Administering User Definitions

Administering user definitions enables you to manage the users in the `UserDatabase` resource, which is defined in the `server.xml` file. In the default configuration, the user database is a memory database with data loaded from the `nsjsp-users.xml` file. Changes that are made to the default user database by the Admin Web application and saved, will be persisted to the `nsjsp-users.xml` file. All the NSJSP Server Class processes under the current `PATHMON` will be notified of the changes, when the `Commit Changes` operation is performed.

Note. This sub-section only applies to the default `UserDatabase` resource, which is a memory database implementation used in the `UserDataBase` realm defined for the NSJSP service and NSJSP engine. Administration of user definitions for other realms or implementations is not provided by the Admin Web application. For more information on the `UserDatabase`, see the [GlobalNamingResources Element](#) on page 3-42.

The user definition element includes the following sub-elements:

- Users
- Groups

- Roles

Users

A user is an entity that can be used to log in to user applications. A user can be assigned multiple roles.

[Figure 4-49](#) displays the properties of a selected user.

Figure 4-49. User Properties for the Admin User

The screenshot shows the Tomcat Web Server Administration Tool interface. On the left is a tree view with the following structure:

- Tomcat Server
 - Service (NSJSP)
 - Connector (0)
 - Host (localhost)
 - Realm for Service (NSJSP)
 - Resources
 - Data Sources
 - Mail Sessions
 - Environment Entries
 - User Databases
 - User Definition
 - Users** (selected)
 - Groups
 - Roles

The main panel is titled 'Edit Existing User Properties'. It has a 'User Actions' dropdown menu set to '---Available Actions---'. There are 'Save' and 'Reset' buttons. Below this is a 'User Properties' section with the following fields:

User Name:	admin
Password:
Full Name:	

Below the user properties is a table for assigned roles:

	Group Name	Description
<input checked="" type="checkbox"/>	admin	
<input checked="" type="checkbox"/>	manager	

At the bottom right of the role table are 'Save' and 'Reset' buttons.

You can perform one of the following actions using the **User Actions** list:

- Create New User
- Delete Existing Users
- List Existing Users

When the **Users** element is selected in the left pane, the **Users List** is displayed in the right pane. To display the properties of an individual user, click the user name in the **Users List**. The selected user's properties will be displayed, as shown in [Figure 4-49](#).

After creating a new user, the password entered for that user is saved as normal text in the configuration file called `nsjsp-users.xml` (in the default setup), which is not encrypted. The elements and actions under the **User Definition** only apply to the user database. Any Realm configuration can use the user database. In the default configuration, the user database is used by a Realm defined in the Engine element.

The Realm is configured to expect encrypted passwords. Therefore, while creating a new user, you must encrypt the password as per the Realm definition.

The Admin Web application itself does not encrypt the password.

Note. For a new user, password must be encrypted using the CLI. For information on setting a user password in the `nsjsp-users.xml` file, see Chapter [7, Migrating to NSJSP 6.1](#).

Groups List

A group is a set of one or more roles that has been assigned a name.

[Figure 4-50](#) displays the properties for a group.

Figure 4-50. New Group Properties

The screenshot shows the Tomcat Web Server Administration Tool interface. On the left is a tree view with the following structure:

- Tomcat Server
 - Service (NSJSP)
 - Connector (0)
 - Host (localhost)
 - Realm for Service (NSJSP)
 - Resources
 - Data Sources
 - Mail Sessions
 - Environment Entries
 - User Databases
 - User Definition
 - Users
 - Groups** (selected)
 - Roles

The main area is titled 'Create New Group Properties'. It contains a 'Group Properties' section with two input fields: 'Group Name' and 'Description'. Below this is a 'Group Actions' section with a table of roles:

	Role Name	Description
<input type="checkbox"/>	admin	
<input type="checkbox"/>	manager	

Buttons for 'Commit Changes', 'Log Out', 'Save', and 'Reset' are visible throughout the interface.

You can perform the following actions using the **Group Actions** list:

- Create New Group
- Delete Existing Groups
- List Existing Groups

When the **Groups** element is selected in the left pane, the **Groups List** is displayed in the right pane. To display the properties of an individual group, click the group name in the **Groups List**. The selected group's properties will be displayed, as shown in [Figure 4-50](#).

Roles List

Roles are used to control access to applications and resources. They may be directly assigned to users or roles can be assigned to groups, which in turn can be assigned to users.

[Figure 4-51](#) displays a Roles List.

Figure 4-51. Roles List



You can perform the following actions using the **Role Actions** list:

- Create New Role
- Delete Existing Roles
- List Existing Roles

When the **Roles** element is selected in the left pane, the **Roles List** is displayed in the right pane. To display the properties of an individual role, click the role name in the **Roles List**. The selected role's properties will be displayed.

Note. The Admin Web application currently supports only one role for a user. Therefore, if you select more roles, the user will be assigned only one role at a time. To assign more than one role for a user, you must edit the `nsjsp-users.xml` file manually.

Access Security Considerations

The NSJSP installation `setup` script prompts the user (with name `admin`) for a password. The `admin` user ID and password can be used to log on to the Admin Web application.

For information on how to change the `admin` user ID and password, see [Users](#) on page 4-80 and Chapter [7, Migrating to NSJSP 6.1](#).

Persisting Changes to the `server.xml` File and Context Files

When the `Commit Changes` operation is invoked, the Admin Server Class saves the existing `server.xml` file and any modified context definition files. It replaces these files with new files containing the changed configuration. The current version of the `server.xml` file is preserved and renamed using the following naming convention:

`server.xml.yyyy-mm-dd.hh-mm-ss`

where:

`yyyy-mm-dd` is the year-month-date.

`hh-mm-ss` is the hour-minutes-seconds (in 24-hour format).

`yyyy-mm-dd.hh-mm-ss` is the timestamp when the `Commit Change` operation is performed.

For example, `server.xml.2004-06-21.14-43-39` is created to preserve the configuration file before the `Commit Change` operation is performed at 2:43:39 PM on June 21st, 2004.

The preserved `server.xml` file is located in the `<NSJSP 6.1 Installation Directory>/conf/directory`.

The preserved context configuration file (with the naming convention `<context-name>.xml.yyyy-mm-dd.hh-mm-ss`) is located in the `<NSJSP 6.1 Installation Directory>/conf/[enginename]/[hostname]/[webappname].xml` file.

For example, the `<NSJSP 6.1 Installation Directory>/conf/NSJSP/localhost/bankapp.xml.2010-04-22.11-03-41` file is located in the `<NSJSP 6.1 Installation Directory>/conf/NSJSP/localhost` directory.

This subsection includes the following topics:

- [Contents of the server.xml File](#)
- [Roll Back a Commit Change Operation](#)

Contents of the `server.xml` File

When the Admin Server Class serializes its configuration back to the `server.xml` file, the following changes occur in addition to the configuration changes:

- All comments are removed.
- All contexts are serialized except those that were deployed using the `context.xml` file or those providing the `context.xml` file inside the `.war` or the `docBase` directory.
- All attributes are listed including those that were omitted (using default values).

Note. The attributes that are not applicable to NSJSP may not be serialized.

Note. HP recommends that you make changes to the `server.xml` file using a text editor.

Roll Back a Commit Change Operation

You can manually roll back the Commit Change operation, because the `server.xml` files are preserved.

Note.

- To roll back the Commit Changes operation, you must restart the NSJSP Server Class.
 - The `server.xml` and the `context.xml` files can be backed up but the `nsjsp-user.xml` file cannot be backed up.
-

Complete the following steps to roll back the Commit Change operation:

1. Stop the NSJSP Server Class.

Note. You can stop the NSJSP Server Class using the NSJSP Manager application or using the CLI.

2. Rename the current `server.xml` file to the `server.xml.tmp` file.
3. Rename the preserved `server.xml.yyyy-mm-dd.hh-mm-ss` file to the `server.xml` file.
4. If you have modified any setting for a particular context, which includes `<context name>.xml` file, complete the following steps to revert those context changes:
 - a. Rename the `<context name>.xml` file of that context (under the `<NSJSP 6.1 Installation Directory>/conf/NSJSP/localhost` folder) to the `<context name>.xml.tmp` file.
 - Rename the `<context-name>.xml.yyyy-mm-dd.hh-mm-ss` file to the `<context-name>.xml` file.

5. Start the server class that is in the STOPPED state.

Note. After each commit operation, the web applications that define an explicit context will be restarted because the context file is updated. This happens irrespective of whether the application's context was modified or not.

Manager Web Application

The Manager Web application was included in releases prior to NSJSP 6.1 to manage web applications hosted by NSJSP. The NSJSP Manager application introduced in NSJSP 6.1 supersedes the Manager Web application. Using the NSJSP Manager application, you can perform all the functions that the Manager Web application provides.

You can access the Manager Web application using this URL:

```
http://<host-name>:<port-number>/<NSJSP 6.1 URI name>/manager/html
```

For more information on the Manager Web application, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.0 System Administrator's Guide*.

Note. The Manager Web application is referred to as the old Manager application to differentiate it from the NSJSP Manager application that is introduced in the NSJSP 6.1 release.

Operations Using the Command-line Interface

This section discusses the following topics:

- [iTP Secure WebServer Operations](#)
- [Server Class Operations](#)

iTP Secure WebServer Operations

NSJSP is installed as part of an iTP Secure WebServer installation. Each NSJSP Server Class is defined to TS/MP during startup. The following configuration files are located in the *<NSJSP 6.1 Installation Directory>/conf* directory during the NSJSP installation:

- `servlet.config` (installation-specific and generic)
- `jdbc.config`
- `filemaps.config`
- `nsjspadmin.config`

When an NSJSP configuration file is modified, the HTTPD command-line utility must redefine the iTP Secure WebServer's TS/MP configuration. To redefine the TS/MP

configuration, you must cold start the iTP Secure WebServer. The iTP Secure WebServer must be cold started to create and add the NSJSP Server Class definitions to the iTP Secure WebServer TS/MP configuration.

Note. The HTTPD object in the *<iTP Installation Directory>/bin* directory can be used both as a command-line utility and as the object file for the HTTPD Server Class.

To perform a cold start, you must stop and then start the iTP Secure WebServer using the scripts provided in the *<iTP Installation Directory>/conf* directory. Restarting the iTP Secure WebServer is equivalent to stopping the iTP Secure WebServer and then doing a cold start.

To cold start the iTP Secure WebServer, see [Verifying the NSJSP Installation](#) on page 2-18.

If you modify NSJSP container files, such as `server.xml` and `logging.properties`, for the changes to take effect, restarting the NSJSP Server Class is sufficient. You need not cold start the iTP Secure WebServer.

The NSJSP Server Classes that are defined by the iTP Secure WebServer include configuration parameters, such as `MAXSERVERS` and `NUMSTATIC`, that are copied directly from the `servlet.config` file. If you modify the parameter values, you must update the `servlet.config` file, and must stop and then start the iTP Secure WebServer.

If the iTP Secure WebServer is configured for online-upgrade (that is, two PATHMONs are defined in a Pathway domain), two TS/MP configurations will be defined by the iTP Secure WebServer—one for each PATHMON. Each NSJSP Server Class will be defined for two PATHMONs. Because there is only one `servlet.config` file in the *<NSJSP 6.1 Installation Directory>/conf* directory, the iTP Secure WebServer will divide the value of the configuration parameters into two halves. Each half value will be assigned to two definitions for each NSJSP Server Class.

For example, if the `SERVLETS` Server Class has parameters `MAXSERVERS=4` and `NUMSTATIC=4` specified in the `servlet.config` file, the iTP Secure WebServer will create a `SERVLETS` Server Class for each PATHMON with the parameters `MAXSERVERS=2` and `NUMSTATIC=2`.

If a server class has a parameter `NUMSTATIC=1`, the server class for one PATHMON will have the parameter `NUMSTATIC=1` and the server class for the other PATHMON will have the parameter `NUMSTATIC=0`.

Server Class Operations

You can perform the following TS/MP Server Class operations using the command-line interface:

- [Stopping NSJSP](#)
- [Starting NSJSP](#)

Stopping NSJSP

Following are the two ways to stop the NSJSP Server Class:

- [Graceful Shutdown](#)
- [Forced Shutdown](#)

Graceful Shutdown

To stop the NSJSP Server Class gracefully, complete the following steps:

1. Obtain the PATHMON name from the `httpd.config` file located in the `<ITP Installation Directory>/conf` directory.
2. Start the PATHCOM program:

From an OSS prompt, PATHCOM can be started using the `gtac1` command as follows:

```
oss: gtac1 -p pathmon \$<PATHMON name>
```

This will start the PATHCOM program and open the specified PATHMON process. The "\" is required to ensure that the \$ sign remains part of the process name.

3. From a TACL prompt, PATHCOM can be started as follows:

```
TACL> PATHCOM $<PATHMON name>
```

If the correct PATHMON is not opened or another PATHMON needs to be opened, issue the PATHCOM command as follows:

```
= OPEN $<PATHMON name>
```

After executing the OPEN command, enter the following commands:

```
= FREEZE SERVER <server class name>
```

```
= STOP SERVER <server class name>
```

```
= STATUS SERVER <server class name>
```

```
= EXIT
```

In the following example, the PATHMON name is `$zweb` and the NSJSP Server Class name is `SCP1`.

```
TACL> PATHCOM $zweb
```

```
= FREEZE SERVER SCP1
```

```
= STOP SERVER SCP1
```

```
= STATUS SERVER SCP1
```

The STATUS SERVER command displays the following output for the SCP1 Server Class:

SERVER	#RUNNING	ERROR	INFO
SCP1	0		FROZEN

The processes of the SCP1 Server Class are stopped as indicated by the 0 in the #RUNNING column and the status of the SCP1 Server Class changes to FROZEN.

Note.

- Stop the NSJSP Server Class gracefully, to allow the server class processes to complete outstanding requests and to perform shutdown operations, before exiting.
 - You can use either PATHCOM or PDMCOM to execute server class operations, such as stop, start, thaw, and freeze.
-

Example of Stopping NSJSP Server Class Configured Under Two PATHMONs of iTP Secure WebServer

In this example, the iTP Secure WebServer is configured with two PATHMONs—\$zweb and \$yweb. The processes of the NSJSP Server Class run under these PATHMONs.

Enter the STATUS SERVER command to view the status of the SCP1 Server Class under the \$zweb and \$yweb PATHMONs:

```
TACL>> PDMCOM
PDMI 1>> OPEN $zweb,$yweb
PDMI 2>> STATUS SERVER SCP1
```

The following output is displayed:

```
PATHMON : \POS02.$ZWEB
SERVER          #RUNNING  ERROR  INFO
SCP1            2

PATHMON : \POS02.$YWEB
SERVER          #RUNNING  ERROR  INFO
SCP1            2
```

To stop the SCP1 Server Class running under the \$zweb PATHMON, enter the following commands:

```
PDMI 3>> OPEN $zweb
= FREEZE SERVER SCP1
= STOP SERVER SCP1
```

After you enter the STOP SERVER command, the SCP1 Server Class's processes running under the \$zweb PATHMON are stopped. To view the status of the SCP1 Server Class, enter the following commands:

```
PDMI 1>> OPEN $zweb,$yweb
```

```
PDMI 2>> STATUS SERVER SCP1
```

The following output is displayed:

```
PATHMON : \POS02.$ZWEB
```

```
SERVER          #RUNNING  ERROR  INFO
```

```
SCP1             0
```

```
PATHMON : \POS02.$YWEB
```

```
SERVER          #RUNNING  ERROR  INFO
```

```
SCP1             2
```

There are now only two SCP1 processes running under the \$yweb PATHMON and none under the \$zweb PATHMON.

Note.

- For information on the PDMCOM commands, see the *TS/MP Release Supplement Manual*.
 - For information on the PATHCOM commands, see the *TS/MP System Management Manual*.
-

Forced Shutdown

To stop the NSJSP Server Class immediately, run the iTP Secure WebServer `stop` script:

```
OSS: cd <iTP Installation Directory>/conf
OSS: ./stop
```

Running the `stop` script does not allow NSJSP processes to perform shutdown operations before shutting down. One such shutdown operation is to persist all the in-memory sessions to a persistent store, if configured.

Starting NSJSP

After configuring NSJSP and starting the iTP Secure WebServer, the NSJSP Server Classes will also be started. If the NSJSP Server Class TS/MP configuration has not been changed, but an NSJSP Server Class has been stopped, the NSJSP Server Class can be started without restarting the iTP Secure WebServer.

To start an NSJSP Server Class, run the `start` script from the `<iTP Installation Directory>/conf` directory in the iTP Secure WebServer environment:

```
OSS: cd <iTP Installation Directory>/conf
OSS: ./start
```

To start an NSJSP Server Class, complete the following steps:

1. Obtain the PATHMON name from the `httpd.config` file located in the `<ITP Installation Directory>/conf` directory.
2. At the TACL prompt, run the following commands:

```
TACL> PATHCOM $<PATHMON name>
= START SERVER <server class name>
```

Note. The THAW SERVER command only needs to be entered when a status show that the server class was frozen. If the THAW SERVER command is entered and the server class was not frozen, an error message is displayed with the following text:

```
ERROR - *1060* SERVER <server class name>,NOT FROZEN
```

After running the START SERVER command, the system displays the following message indicating that the server is started:

```
$YY66S: SERVER server_name, STARTED
```

Note.

- By default, an NSJSP Server Class includes four processes. The Admin Server Class includes only two processes.
 - You can use either PATHCOM or PDMCOM to execute server class operations such as stop, start, thaw and freeze.
 - Check the `<NSJSP 6.1 Installation name>.<date>.log` file to ensure that the NSJSP server has restarted.
 - You can use either PATHCOM or PDMCOM to execute server class operations such as stop, start, thaw and freeze.
-

Manual Deployment and Undeployment of Web Applications

You can control the deployment of applications by using attributes in the `server.xml` file. This section discusses the following topics:

- [Deploying Applications at Startup](#)
- [Deploying Applications on a Running NSJSP Server](#)

Deploying Applications at Startup

In the `server.xml` file, the location of web applications is specified by the `appBase` attribute of the Host.

Note. The default host is `localhost` and the default `appBase` is `<NSJSP_HOME>/webapps`.

To deploy a web application at startup, you must set the value of the Host's `deployOnStartup` attribute to `true`. If the `deployOnStartup` attribute is set to `false`, applications will not deploy at startup.

Even though applications are not deployed during startup (if `deployOnStartup` is set to `false`), the applications will get deployed when the auto deployer of the Host runs. You can control the auto deployer by using the `autoDeploy` attribute. For more information on the `autoDeploy` attribute, see Chapter [3, Configuring NSJSP](#).

Note. If the `deployOnStartup` and the `autoDeploy` attributes are set to `false`, you must deploy the applications using the NSJSP Manager.

The following is the deployment sequence:

1. All contexts in `<NSJSP_HOME>/conf/engineName/hostname` are deployed first. It is assumed that any XML file in the `<NSJSP_HOME>/conf/<engine_name>/<host_name>` directory contains a context element (and its associated subelements) for a single web application. The value of the `docBase` attribute of the context element will be the absolute path to a web application directory, or the absolute path of a `.war` file.

Note. The location of the `docBase` attribute defined in the XML context file must be outside the `appBase` directory to ensure that automatic deployment operates properly.

2. All `.war` files are then deployed. A `.war` file in the `appBase` directory that does not have a corresponding directory of the same name (without the `.war` extension) will be automatically expanded, unless the `unpackWARs` property is set to `false`. If you redeploy an updated `.war` file, you must delete the expanded directory before restarting NSJSP to ensure that the updated `war` file expands again.

Note. If the auto deployer is enabled, the updated `war` file will automatically expand after the last expanded directory is removed.

3. It is assumed that any subdirectory within the application base directory contains a web application and is deployed last.

Deploying Applications on a Running NSJSP Server

In addition to the deployment at startup, you can deploy web applications while NSJSP is running. If the value of the Host's `autoDeploy` attribute is set to `true`, the Host will attempt to deploy and update web applications dynamically, as required. The auto deployer first performs the procedure as described in [Deploying Applications at Startup](#) on page 4-90 and subsequently monitors web applications for the following changes:

- Updates to the `WEB-INF/web.xml` file – An update to the `WEB-INF/web.xml` file causes a reload of the web application.
- Deletion of `.war` files – Deleting a `.war` file causes the application to undeploy, and any associated expanded directory, context file, and work directory are removed. Subsequently, current user sessions will not be persisted.

- Deletion of a directory – Deleting a directory causes the application to undeploy, and any associated context file and work directory are removed. Current user sessions will not be persisted. Any associated `.war` file will not be deleted, and the application will be redeployed from the `.war` file when the auto deployer checks for changes.
- Deletion of a context file – Deleting a context file causes the application to undeploy, and any associated work directory is removed. Current user sessions will not be persisted. Any associated `.war` file and directory will not be deleted, and the application will be redeployed from the `.war` file (or from the directory if no `.war` file is present) when the auto deployer checks for changes.
- `.war` file update – Updating a `.war` file causes the application to undeploy, and any associated expanded directory, context file, and work directory are removed. Current user sessions will not be persisted.
- Directory update – Updating a directory causes the application to undeploy, and any associated context file and work directory are removed. Current user sessions will not be persisted. The application will be redeployed when the auto deployer checks for changes.
- Context file update – Updating a context file causes the application to undeploy, and any associated work directory is removed. Current user sessions will not be persisted. The application will be redeployed when the auto deployer checks for changes.

Comparison of the Management Applications

This section compares the NSJSP Manager, the old Manager, and the Admin Web applications. This section discusses the following topics:

- [Comparison of Architectures](#)
- [Comparison of Features](#)
- [Comparison of Management Application Access Roles](#)

Comparison of Architectures

This section compares the architectures of the management applications, as follows:

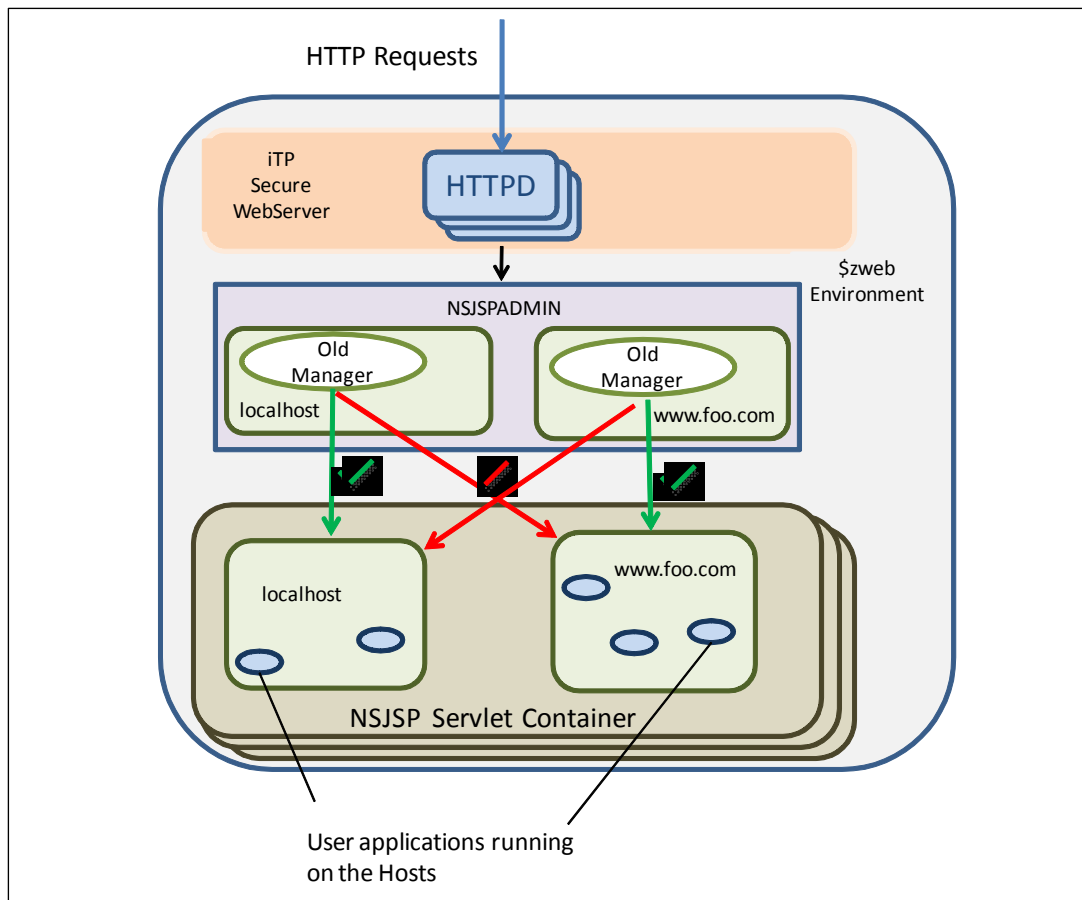
- [Old Manager Application](#)
- [Admin Web Application](#)
- [NSJSP Manager Application](#)
- [Differences in Architectures](#)

Old Manager Application

The old Manager application supports management of web applications on a Host. You can perform tasks, such as, deploy, start, and stop web applications on the Host where the old Manager application is running. If there are two or more Hosts within an NSJSP servlet container, an instance of the old Manager application is required on each Host.

[Figure 4-52](#) illustrates the architecture of the old Manager application.

Figure 4-52. Architecture of the Old Manager Application



In [Figure 4-52](#), the NSJSP servlet container includes two Hosts: `localhost` and `www.foo.com`. To manage web applications on each Host, the old Manager application is installed on both Hosts. The old Manager application on `localhost` can manage web applications on `localhost` and the old Manager application on `www.foo.com` can manage web applications on `www.foo.com`, as indicated by the green arrows. The old Manager application on `localhost` cannot manage web

applications on `www.foo.com`, and the old Manager application on `www.foo.com` cannot manage web applications on `localhost`, as indicated by the red arrows.

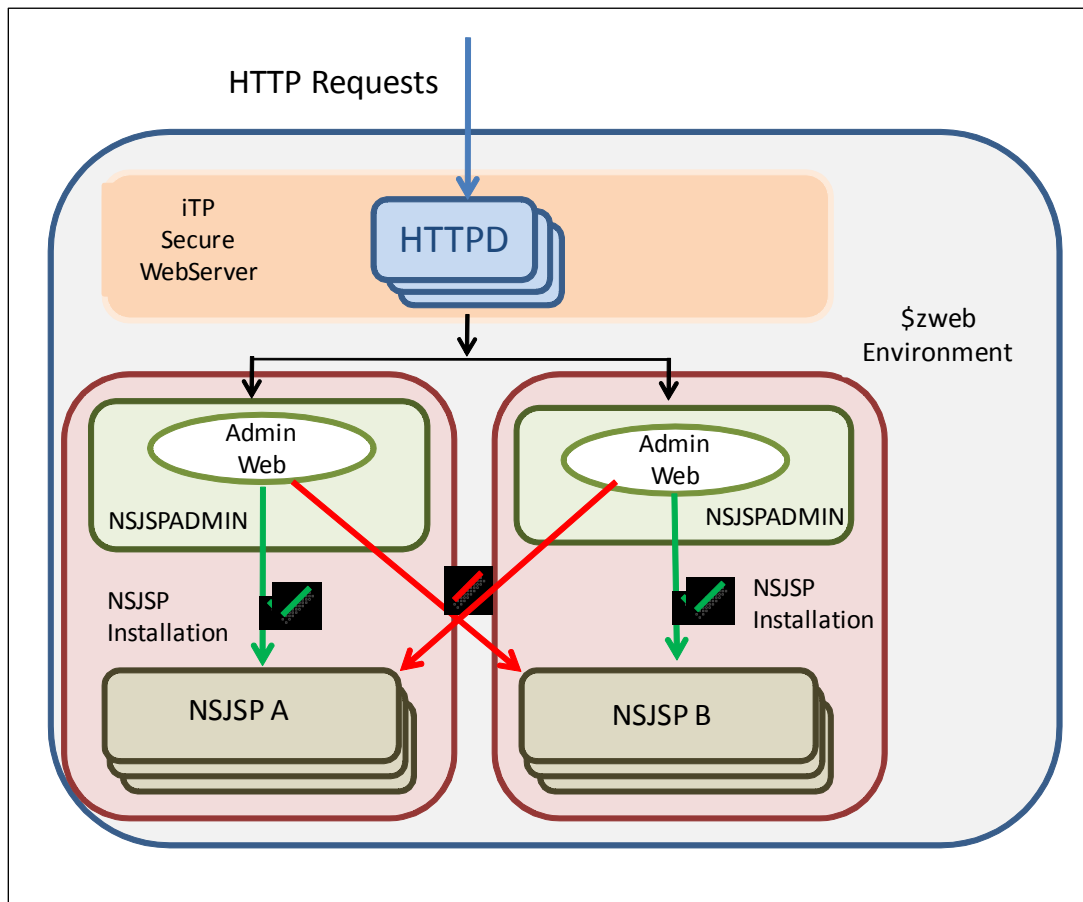
Note. The Manager Web application that was included in releases prior to NSJSP 6.1 is referred to as the old Manager application to differentiate it from the NSJSP Manager application that is introduced in the NSJSP 6.1 release.

Admin Web Application

The Admin Web application supports management of the NSJSP servlet container and some of its objects. You can perform tasks, such as, create and delete services, connectors, and Hosts on the NSJSP servlet container where the Admin Web application is installed. The Admin Web application can manage process instances of an NSJSP servlet container in only one NSJSP installation and in only one PATHMON environment.

Note. When you install NSJSP, the Admin Web application is installed by default.

[Figure 4-53](#) illustrates the architecture of the Admin Web application.

Figure 4-53. Architecture of the Admin Web Application

In [Figure 4-53](#), the \$zweb environment contains two NSJSP installations. You can manage NSJSP A and NSJSP B servlet containers and their objects using the Admin Web application in their respective NSJSP installations, as indicated by the green arrows. However, you cannot manage either NSJSP A or NSJSP B servlet containers and their objects using the Admin Web application in a different NSJSP installation, as indicated by the red arrows.

For more information on the architecture of the Admin Web application, see the [Admin Web Application](#) on page 4-56.

NSJSP Manager Application

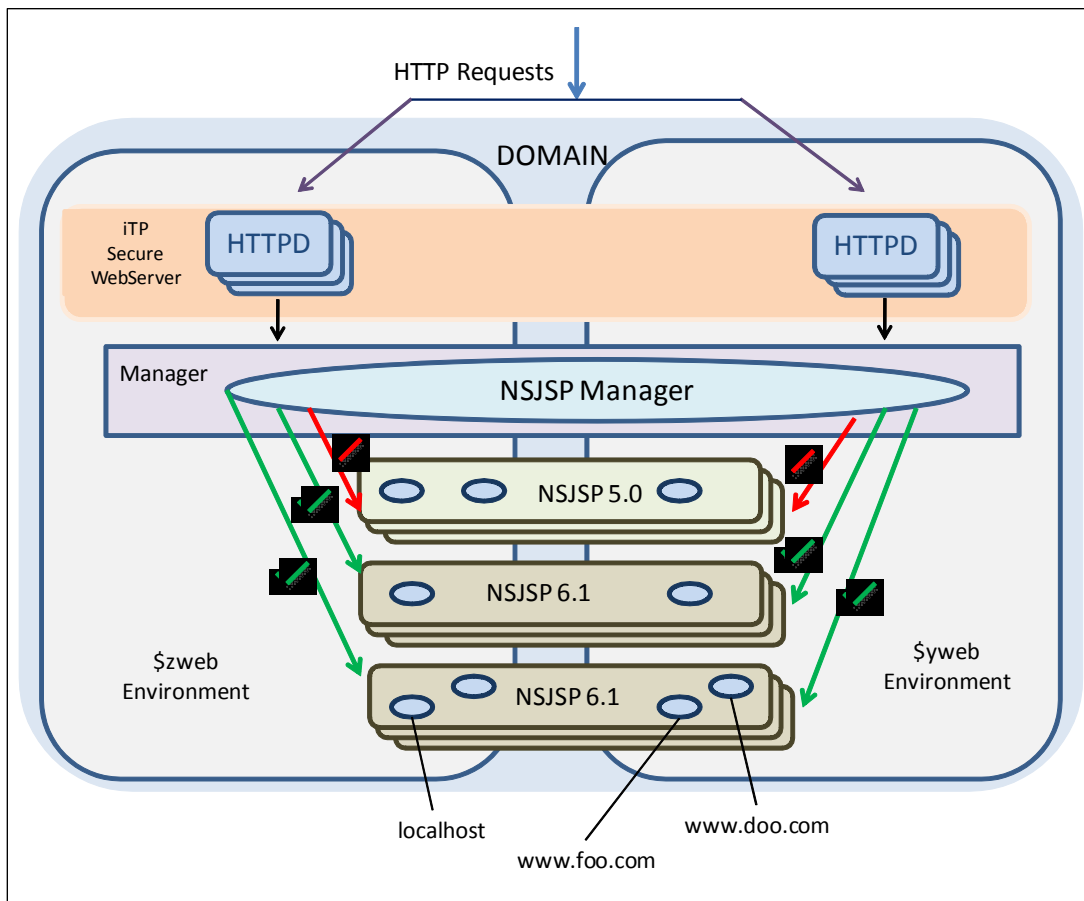
The NSJSP Manager application enables management of NSJSP 6.1 installations and the web applications deployed within them. The NSJSP Manager supports the following:

- Management of web applications running on multiple Hosts within a server class.

- Management of multiple NSJSP 6.1 installations within an iTP Secure WebServer environment.
- Management of NSJSP installations that are configured in a Pathway domain. The use of a Pathway domain implies that the iTP Secure WebServer was installed in an online-upgrade configuration, which has two PATHMONs.

[Figure 4-54](#) illustrates the architecture of the NSJSP Manager application.

Figure 4-54. Architecture of the NSJSP Manager Application



In [Figure 4-54](#), the \$zweb and \$yweb PATHMON environments include an instance of the NSJSP Manager application and three NSJSP installations: one NSJSP 5.0 installation and two NSJSP 6.1 installations. Using the NSJSP Manager, you can manage the two NSJSP 6.1 installations and the web applications running on their respective Hosts, as indicated by the green arrows in [Figure 4-54](#). However, you can manage only one NSJSP Server Class at a time. You cannot manage the NSJSP 5.0 installation with the NSJSP Manager, as indicated by the red arrows.

For more information on the architecture of the NSJSP Manager application, see the [NSJSP Manager Application](#) on page 4-1.

Differences in Architectures

The differences in the architectures of the three applications are as follows:

- Using the old Manager application, you can manage web applications running on only one Host.
- Using the Admin Web application, you can manage the servlet container and its objects within the same installation as the Admin Web application, but only within one PATHMON.
- Using the NSJSP Manager application, you can manage multiple NSJSP 6.1 installations. You can also manage all the web applications running on all the Hosts within a server class, and you can manage NSJSP 6.1 installations installed in a Pathway domain.

For a complete list of the features that these applications support, see the [Comparison of Features](#) on page 4-98.

Comparison of Features

[Table 4-24](#) compares the features of the NSJSP Manager, the Old Manager, and the Admin Web applications.

Table 4-24. Comparison of Features of NSJSP Manager, Old Manager, and Admin Web Applications (page 1 of 2)

Category	Feature	NSJSP Manager Application	Old Manager Application	Admin Web Application
Application Management	Displays the current applications running on the selected server class and Host.	Yes	Yes	No
	Enables you to deploy, undeploy, start, or stop applications.	Yes	Yes	No
	Displays URL statistics such as, links from the application that are accessed by users, and the number of user requests received by the applications.	Yes	No	No
	Displays information about the HTTP methods, such as, the HTTP methods used to access the application and the request count for each type of HTTP method.	Yes	No	No
	Allows web application management using ANT scripts.	Yes	Yes	No

Table 4-24. Comparison of Features of NSJSP Manager, Old Manager, and Admin Web Applications (page 2 of 2)

Category	Feature	NSJSP Manager Application	Old Manager Application	Admin Web Application
Server Class Information	Displays NSJSP processes and their status, such as, the processor on which the application is running and the amount of memory used.	Yes	No	No
	Displays the configuration parameters for the selected server class.	Yes	No	No
	Displays the NSJSP Server Class statistics. The statistics include the information related to the server class link wait queue (Queue Info) and Input and Output operations (IO Info).	Yes	No	No
	Displays the connector statistics for every NSJSP process. The connector statistics include information, such as, the number of threads that are in use, the number of threads that are free, and the total number of process threads allocated to serve requests.	Yes	Yes	No
	Enables control of server class operations by starting or stopping an NSJSP Server Class.	Yes	No	No
	Enables you to compare the values of a particular MBean attribute across all NSJSP processes.	Yes	No	No
MBean Operations	Displays the MBeans for every NSJSP process.	Yes	No	No
	Enables you to modify MBean values.	Yes	No	Yes
	Adds new configuration parameters.	No	No	Yes

Comparison of Management Application Access Roles

[Table 4-25](#) compares the roles that must be assigned to users to authorize them to access the management applications.

Table 4-25. Comparing Roles

Management Application	Role Required to Access the Management Application
Admin Web Application	admin – role for users who need to configure NSJSP.
Old Manager Web Application	manager – role for users who need to manage and monitor the web applications installed in NSJSP.
NSJSP Manager Web Application	admin and manager – both roles are authorized to use the NSJSP Manager.

Single Point of Management Using the NSJSP Manager

In releases prior to NSJSP 6.1, an iTP Secure WebServer environment included only one NSJSP installation. An NSJSP installation includes two server classes. An NSJSP Server Class can be configured to include multiple Hosts. Web applications are deployed in and run within these Hosts. To manage the web applications running in a Host, NSJSP included an application, called the Manager Web application. However, an instance of the Manager Web application could only manage applications running on a single Host. To manage web applications running on multiple Hosts, an instance of the Manager Web application was required for each Host.

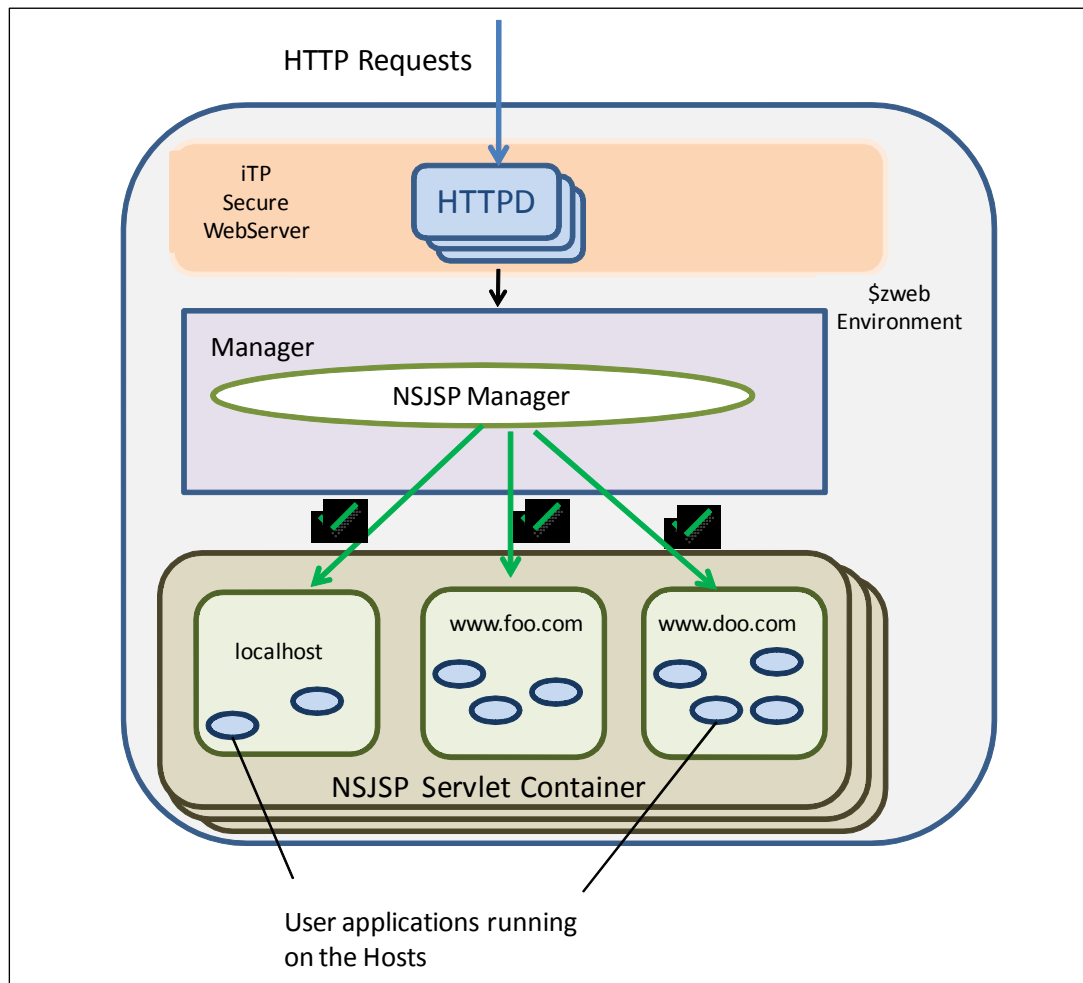
[Figure 4-52](#) illustrates the architecture of the old Manager application.

Starting with the NSJSP 6.1 release, the NSJSP Manager application is introduced to support single point of management in the following scenarios:

- [Multiple Hosts within a Server Class](#)
- [Multiple NSJSP Installations within an iTP Secure WebServer Environment](#)
- [NSJSP Installations in an iTP Secure WebServer Configured for Online-Upgrade](#)

Multiple Hosts within a Server Class

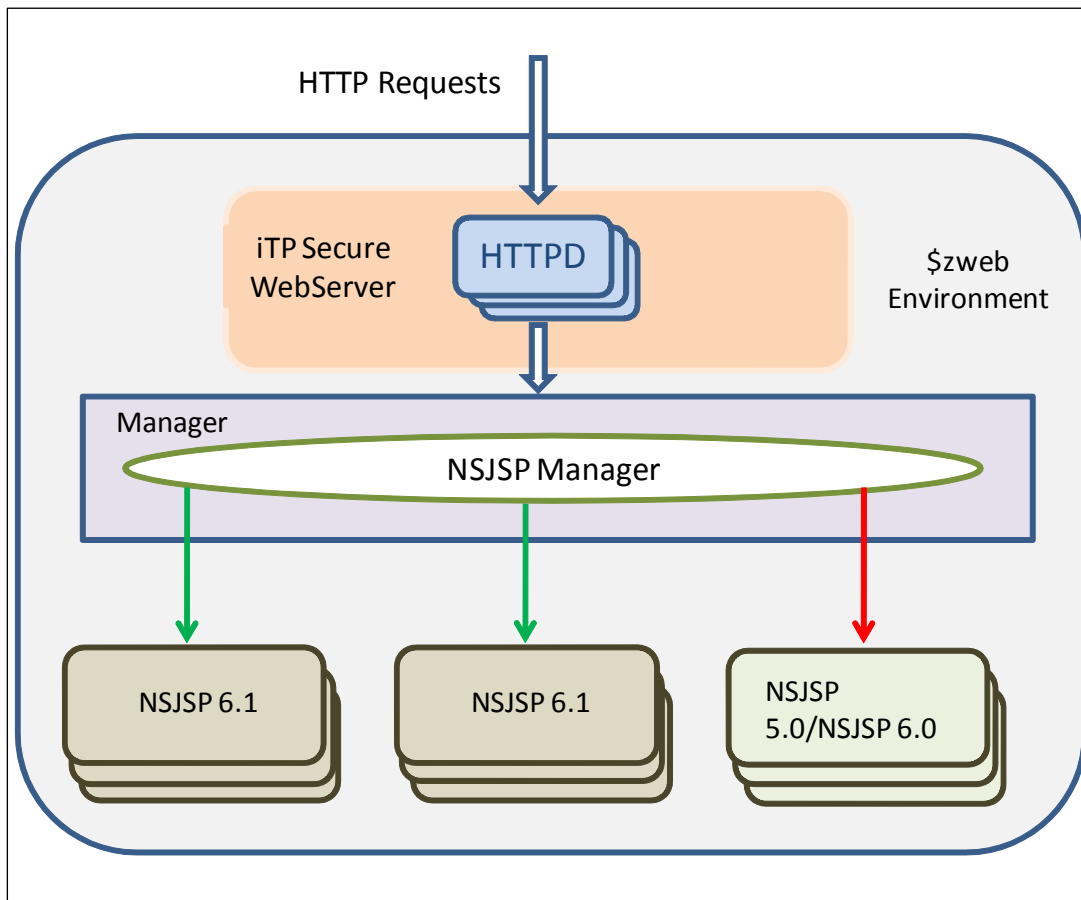
The NSJSP Manager application enables the management of web applications running in multiple Hosts within a server class.

Figure 4-55. NSJSP Manager Supporting Multiple Hosts

In [Figure 4-55](#), using a single instance of the NSJSP Manager, you can manage all the web applications running on the three Hosts: `localhost`, `www.foo.com`, and `www.doo.com`, as indicated by the green arrows.

Multiple NSJSP Installations within an iTP Secure WebServer Environment

Starting with the NSJSP 6.1 release, each iTP Secure WebServer environment can include multiple NSJSP installations. One iTP Secure WebServer environment can include multiple NSJSP 6.1 installations and one instance of an NSJSP 5.0 or NSJSP 6.0 installation. A single instance of the NSJSP Manager application enables the management of all NSJSP 6.1 installations within an iTP Secure WebServer environment. However, the NSJSP Manager is not designed to manage either NSJSP 5.0 or NSJSP 6.0 installations.

Figure 4-56. Multiple NSJSP Installations

In [Figure 4-56](#), a single instance of the NSJSP Manager application enables the management of all NSJSP 6.1 installations within an iTP Secure WebServer environment, as indicated by the green arrows. However, the NSJSP Manager is not capable of managing an NSJSP 5.0 or NSJSP 6.0 installation, as indicated by the red arrow.

Note. The NSJSP Manager only manages NSJSP 6.1 installations. The NSJSP Manager does not manage NSJSP 5.0 or NSJSP 6.0 installations.

NSJSP Installations in an iTP Secure WebServer Configured for Online-Upgrade

The NSJSP Manager also enables the management of NSJSP installations that are configured in a TS/MP Pathway domain. An iTP Secure WebServer can include up to two PATHMON environments. When an NSJSP installation is installed in an iTP Secure WebServer that includes two PATHMON environments, the NSJSP installation will be accessed by both PATHMON environments. This configuration is called a Pathway

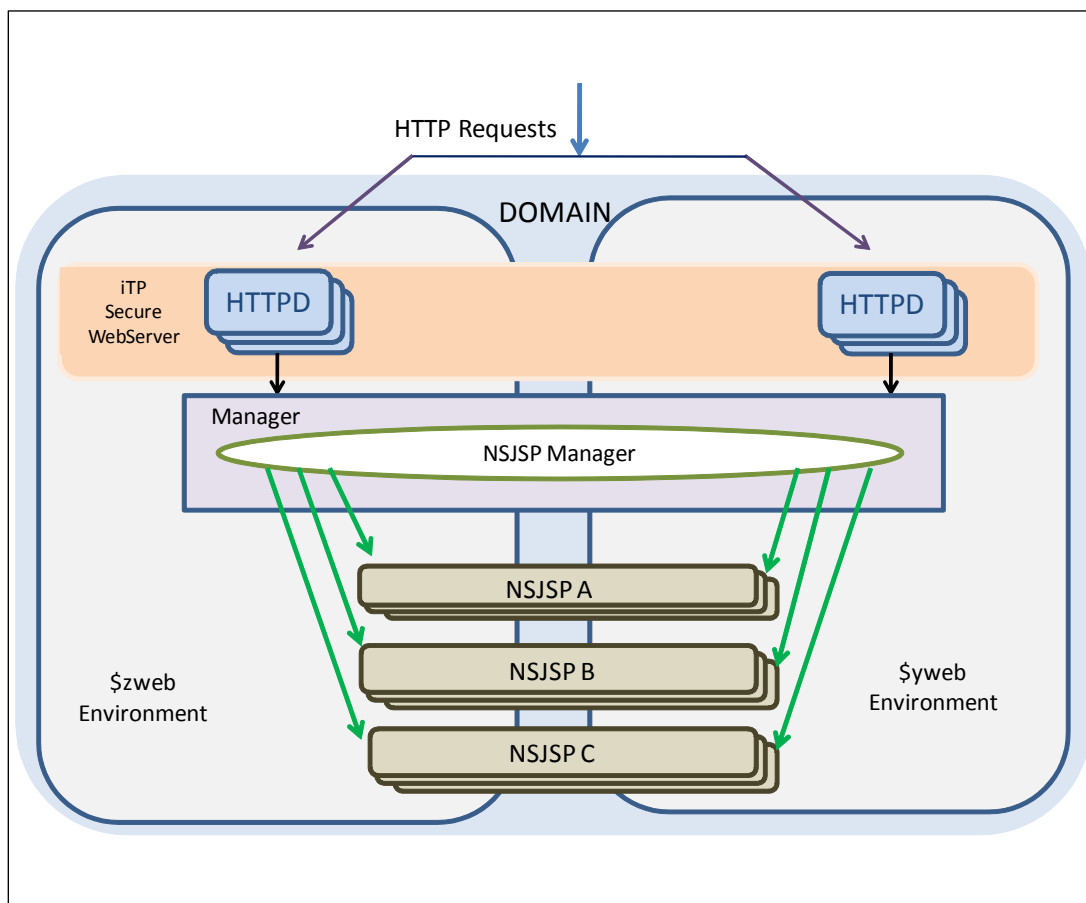
domain configuration. You can configure an NSJSP installation in a Pathway domain for online upgrades of web applications and servlet containers.

For more information on Pathway domain configuration, see the *TS/MP Release Supplement Manual*.

For more information on configuring for online-upgrade, see the *ITP Secure WebServer System Administrator's Guide*.

[Figure 4-57](#) illustrates NSJSP installations configured in a Pathway domain.

Figure 4-57. NSJSP in a Pathway Domain



In [Figure 4-57](#), a single instance of the NSJSP Manager enables the management of all NSJSP installations configured in the Pathway domain, as indicated by the green arrows.

Note. Although you can manage multiple NSJSP installations using the NSJSP Manager application, at any given time, you can manage only one NSJSP Server Class. You cannot manage multiple NSJSP Server Classes simultaneously. The NSJSP Manager application enables you to select the NSJSP Server Class that you want to manage.

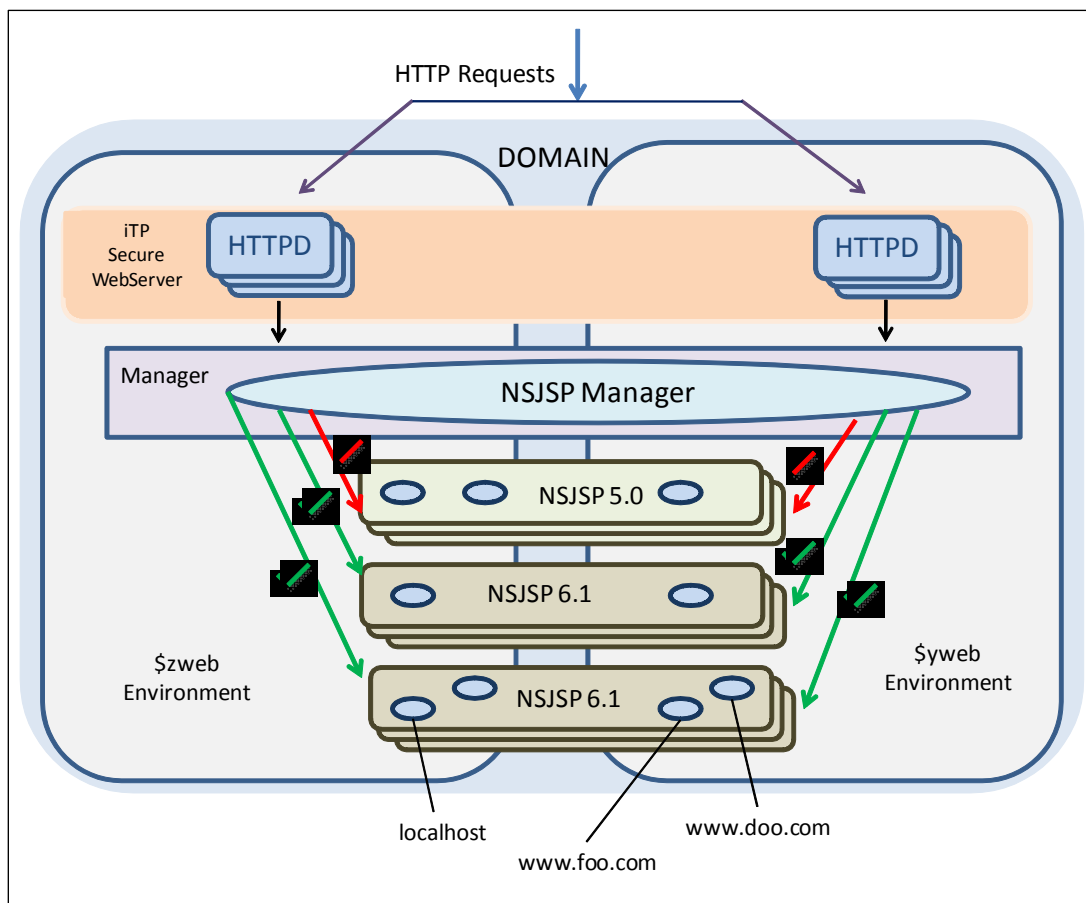
You can install the NSJSP Manager by selecting `Create an NSJSP Manager installation` when running the `NSJSP setup` script. Each iTP Secure WebServer environment can include one instance of the NSJSP Manager.

The Architecture of the NSJSP Manager

Upon successful installation of the NSJSP Manager, the NSJSP Manager runs in its own server class in the iTP Secure WebServer environment. You can use the NSJSP Manager application to manage all the NSJSP 6.1 installations within an iTP Secure WebServer installation. You can perform tasks, such as, deploying web applications and managing NSJSP installations.

[Figure 4-58](#) illustrates an NSJSP Manager application in an iTP Secure WebServer environment.

Figure 4-58. NSJSP Manager



In [Figure 4-58](#), the `$zweb` and `$yweb` PATHMON environments include an instance of the NSJSP Manager application and three NSJSP installations: one NSJSP 5.0 installation and two NSJSP 6.1 installations. Using the NSJSP Manager, you can

manage the two NSJSP 6.1 installations and the web applications running in their respective Hosts, as indicated by the green arrows. However, you can manage only one NSJSP Server Class at a time. You cannot manage the NSJSP 5.0 installation with the NSJSP Manager, as indicated by the red arrows.

This chapter addresses the following topics:

- [Logging Architecture](#)
- [Apache Tomcat Enhancements to the Logging Architecture](#)
- [NSJSP Enhancements to the Logging Architecture](#)
- [Logging Configuration](#)
- [Log Files Related to NSJSP](#)
- [Programming Considerations for Logging](#)
- [Commons Logging](#)

Logging Architecture

The logging architecture of NSJSP is based on the Java Platform, Standard Edition logging architecture. NSJSP provides a programmatic interface to the logging framework using Commons Logging.

For information on Commons Logging, see [Commons Logging](#) on page 5-30.

Additionally, NSJSP introduces its own enhancements related to logging.

The logging architecture includes the `java.util.logging` package that you can configure. The following Java logging architecture components are defined by the Java logging package, and enable logging related to the NSJSP Servlet Container objects and the web applications:

- [Loggers](#)
- [Handlers](#)
- [Formatters](#)
- [Log Manager](#)

Loggers

Loggers are named entities, which receive log messages. The name of a logger is usually the name of the Java package. The Java package includes classes whose log messages are handled by this logger. For example, a logger, called `java.awt` handles log messages originating from classes that belong to the `java.awt` package.

The namespace of the loggers is hierarchical. The hierarchy begins with the root logger, which is denoted by an empty string, `""`. The root logger is the parent logger and the other loggers are child loggers. Each child logger, in turn, can be a parent for other child loggers. For example, `com.foo` is the parent logger of `com.foo.bar`.

Additionally, `com.foo.bar1` and `com.foo.bar2` are also child loggers of `com.foo`, and are peer loggers of `com.foo.bar`.

Each log message is associated with a log level, which denotes the severity of the log. Similar to the log messages, a logger is associated with a log level. If no log level is associated with a logger, the logger inherits the log level from its nearest parent that is associated with a log level.

Each logger can be configured with a set of handlers. Based on the level of the log message that a logger receives and its own log level, the logger either forwards the message to the associated set of handlers for further processing, or it discards the log message. The logger can also send messages to its parent logger. The parent logger writes log records to its handlers and to all its parent loggers in the tree. By default, this option is disabled. However, you can enable the option by using the `useParentHandler` configuration property of the logger.

Handlers

A handler receives the log message forwarded by the logger. Based on the log level of the log message and its own log level, the handler either discards the log message or it publishes the log message to a variety of destinations. NSJSP provides handlers that can publish messages to destinations, such as, Event Management Service (EMS), `STDOUT`, and `STDERR`.

The Java logging package defines a set of handlers.

[Table 5-1](#) lists the handlers defined in the Java logging package.

Table 5-1. Handlers in the Java Logging Package

Handler	Description
<code>ConsoleHandler</code>	Publishes log messages to <code>system.err</code> .
<code>FileHandler</code>	Publishes log messages to a file.
<code>MemoryHandler</code>	Publishes logs to a circular buffer in memory.
<code>StreamHandler</code>	Publishes logs to a stream.
<code>SocketHandler</code>	Publishes logs to a TCP/IP socket.

You can configure each handler using the configuration properties that are specific to the handler.

Formatters

A formatter defines the format of log messages. Every handler is associated with a formatter. The Java logging package defines two formatters: `SimpleFormatter` and `XMLFormatter`.

Log Manager

The Log Manager tracks the global logging information.

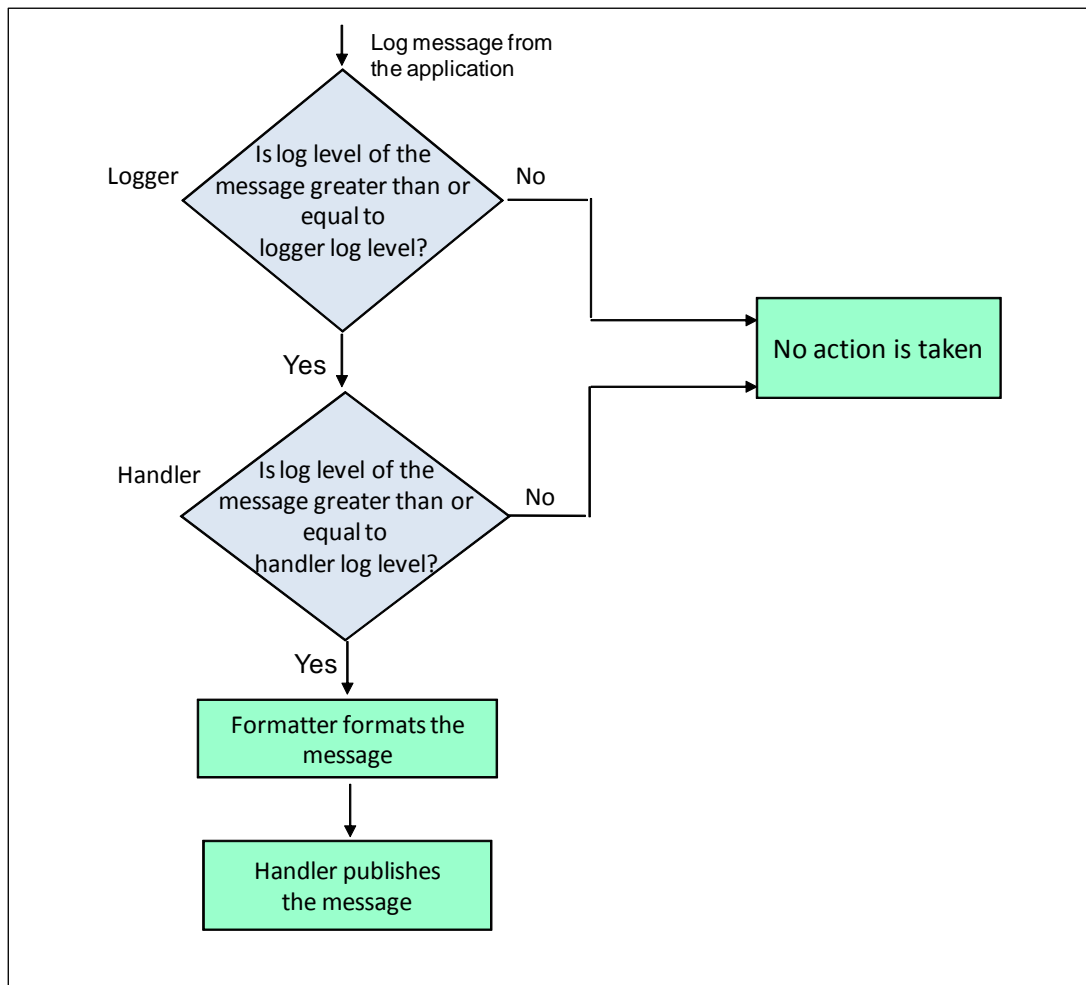
The Log Manager manages the following:

- The hierarchical namespace of logger objects. All named loggers are stored in this namespace.
- The logging control properties, which are simple key-value pairs that can be used by handlers and other logging objects.

The following is the sequence of events that occur when an application attempts to write a log message:

1. The application code sends the log message and its log level to a logger.
2. If the log level of the log message is lower than the log level of the logger, the log message is discarded. If the log level of the log message is greater than or equal to the log level of the logger, the logger forwards the log message to its associated handler.
3. If the log level of the log message is lower than the log level of the handler, the log message is discarded. If the log level of the log message is greater than or equal to the log level of the handler, the log message is formatted by a Formatter object associated with the handler.
4. The handler publishes the formatted log message.

[Figure 5-1](#) illustrates the logging work flow.

Figure 5-1. Logging Work Flow

Apache Tomcat Enhancements to the Logging Architecture

The following are the Apache Tomcat enhancements to the logging architecture:

- Configuring logging for each class loader: The default `java.util.logging` implementation available in the Java Development Kit (JDK) supports logging configurations to be defined per Java Virtual Machine (JVM). As a result, the following options are not allowed:
 - Independent logging configurations for user web applications hosted on the servlet container.
 - Servlet container logging configuration that is independent of logging configuration for web applications.

To overcome these inabilities, Tomcat replaces the `java.util.logging` implementation with a container-friendly implementation, called JULI.

JULI allows a logging configuration file for each class loader. Because each web application hosted on the servlet container has its own class loader, allowing a configuration file per class loader leads to each web application having its own logging configuration file. JULI offers this feature by providing the `ClassLoaderLogManager`, which conforms to the `java.util.logging` Log Manager specifications. The `ClassLoaderLogManager` extends the Log Manager of `java.util.logging`.

Along with the `ClassLoaderLogManager`, JULI also provides certain enhancements to the configuration parameters in the `logging.properties` file.

- Configuring multiple handlers with a single handler class: Tomcat enables you to configure multiple handlers using the same handler implementation class. For the same handler implementation class, the settings of one handler can be different from that of another handler.

For more information on configuring multiple handlers and loggers, see [Configuring Handlers](#) on page 5-10.

NSJSP Enhancements to the Logging Architecture

This section describes the NSJSP enhancements to the logging architecture.

NSJSP provides the following NonStop-specific classes along with JULI:

- [NSJSP Formatter](#)
- [NSJSP Log Handler](#)

NSJSP Formatter

The NSJSP Formatter class is a NonStop-specific class that is used to define the format in which messages must be published. The NSJSP Formatter class (`com.tandem.servlet.logging.NSJSPFormatter`) is an extension of the `java.util.logging.Formatter` class. It is commonly denoted as `NSJSPFormatter`.

For information on configuring the NSJSP Formatter, see [Configuring the NSJSP Formatter Class](#) on page 5-14.

NSJSP Log Handler

The NSJSP Log Handler class offers configuration properties for the log messages and enables you to configure the message format. The NSJSP Log Handler class (`com.tandem.servlet.logging.NSJSPLogHandler`) is commonly denoted as

NSJSPLogHandler. This class is an extension of the `java.util.logging.Handler` class.

In addition, the NSJSP Log Handler class offers a logging feature, called [Log Rollover](#).

For information on configuring the NSJSP Log Handler, see [Configuring Handlers](#) on page 5-10.

Log Rollover

The log rollover feature enables you to archive log files based on predefined criteria. The log rollover feature is introduced in the T1222^AAL SPR of the NSJSP 6.0 release.

Servlet containers and applications generate huge volumes of logs in a short period of time, leading to a difficulty in managing logs. Therefore, you can archive old log files. This is called log rollover. Log files roll over automatically based on the defined criteria, such as, the volume of log files or at regular intervals. As a result, managing the NSJSP log files is simplified. The following are the benefits of the log rollover feature:

- Manages and archives the log files depending on the volume and time interval.
- Supports user-defined directory into which log files must roll over.

This section describes the following:

- [Attributes Introduced to Configure Log Rollover](#)
- [Log Rollover Working](#)
 - [Log Rollover Based on File Size](#)
 - [Log Rollover Based on Timestamp](#)
 - [The archiveDirectory Attribute](#)

Attributes Introduced to Configure Log Rollover

NSJSP includes new attributes that enable you to configure log rollover. This section describes the new attributes.

The `logging.properties` file includes the following new attributes to support the log rollover feature:

- `maxFileSize`
Log rollover based on the `maxFileSize` attribute is called size-based rollover. The attribute specifies the threshold size of the file for rollover. The size of the file is measured in megabytes (MB). When the log file size exceeds the specified value, it rolls over.

For information on configuring the `maxFileSize` attribute, see [Configuring for Log Rollover Based on File Size](#) on page 5-16.

- `datePattern`

Log rollover based on the `datePattern` attribute is called timestamp-based rollover. The `datePattern` attribute enables you to configure the timestamp-based rollover. You can set the log files to roll over at regular recurring intervals, such as, after an hour, after a day, and after a week.

Note. The end of an hour is marked by the change in hour of the clock and not at the completion of 60 minutes. For example, at 17:40 hours on August 28th, 2008, if you configure the rollover of a log file to occur at the end of each hour, the first rollover occurs at 18:00 hours when the hour of the clock changes and not at 18:40 hours. The subsequent rollovers occur at 19:00 hours, 20:00 hours and so on.

The end of a day is marked by the change in date, which occurs at midnight, and not at the end of 24 hours from the time the log file is created. For example, at 17:00 hours on August 28th, 2008, if you configure the rollover of a log file to occur every day, the rollover occurs at midnight, when the date changes to August 29th, 2008 and not at 17:00 hours on August 29th, 2008.

For information on how to configure the `datePattern` attribute, see [Configuring for Log Rollover Based on Timestamp](#) on page 5-17.

- `archiveDirectory`

The `archiveDirectory` attribute enables you to specify the location to which the rolled over log files must be moved. Log files present in the specified location move to the configured `archiveDirectory` when the threshold is exceeded.

Note. If the `archiveDirectory` attribute value is not specified, the rolled over log file remains in the current directory. The current directory is the value of the `destination` property.

Note.

- The timestamp-based rollover is enabled by default. However, you can enable the size-based rollover.
- Although you can archive the log files based on the `maxFileSize` and `datePattern` attributes, only one of the two rollover attributes is supported at a time. If you set both attributes in the `logging.properties` file, the size-based rollover attribute overrides the timestamp-based rollover attribute. However, if the value of the `maxFileSize` attribute is invalid, and the value of the `datePattern` attribute is valid, log files roll over on the basis of the `datePattern` attribute value.
- After configuring for log rollover, you must restart NSJSP for the changes to take effect.

For information on how to configure the `archiveDirectory` attribute, see [Configuring the archiveDirectory Attribute](#) on page 5-19.

Log Rollover Working

This section describes the following topics:

- [Log Rollover Based on File Size](#)
- [Log Rollover Based on Timestamp](#)
- [The archiveDirectory Attribute](#)

Log Rollover Based on File Size

The `maxFileSize` attribute governs the size-based rollover of log files. You can configure a threshold file size for log rollover. This value must be greater than zero. If the file size exceeds the specified threshold, the log file rolls over. The base threshold, which is the lower limit to archive the log files, is 10 MB. If you set the `maxFileSize` attribute to a value less than 10 MB, the log file rolls over only when the size exceeds 10 MB. For example, if you set the value to 8 in the `logging.properties` file, the roll over occurs when the file size exceeds 10 MB, which is the minimum size of the log file to trigger a roll over. However, if you set the `maxFileSize` attribute to a value greater than 10, the log file rolls over only when its size exceeds the set value. If the value of the `maxFileSize` attribute is equal to or less than zero, it is considered an invalid value, and the size-based rollover feature is disabled. When the value of the `maxFileSize` attribute is invalid, an error message is logged to `STDERR`.

Note. The unit, MB is taken by default. You must specify only the file size.

Upon rollover, the log file name is appended with the date and time of rollover.

For example, if the user-defined name of the log file is `abc.log`, and it rolls over at 02.00.30 hours on August 30th, 2008, the rolled over log file is renamed to `abc.log.2008.08.30.02.00.30`, where `2008.08.30.02.00.30` is the timestamp. After `abc.log.2008.08.30.02.00.30` rolls over, a new log file is created. This log file becomes the current working log file, and is assigned the user-defined log file name, `abc.log`. Similarly, when this current working log file rolls over, the date and time are appended to the file name, and a new working log file is created, called `abc.log`. Therefore, the name of the current working file is always the user-defined file name.

Log Rollover Based on Timestamp

The `datePattern` attribute in the `logging.properties` file governs the timestamp-based rollover of log files. You can configure the log file to roll over at predefined time intervals using the `datePattern` attribute value. The `datePattern` attribute value must adhere to the format specified in the Java class documentation at: <http://java.sun.com/javase/6/docs/api/java/text/SimpleDateFormat.html>.

In timestamp-based rollover, when a log file is created, the log file name is appended with the current date, which is according to the format specified in the `java.text.SimpleDateFormat` Java class documentation.

For example, if a log file, called `nsjsp.log`, is created on August 30th, 2008, the complete log file name will be `nsjsp.log.2008.08.30`. The second part of the file name, `2008.08.30`, which corresponds to August 30th, 2008 is automatically appended to `nsjsp.log`.

Using the `datePattern` attribute in the `logging.properties` file, if you configure the log file to roll over on a daily basis, `nsjsp.log.2008.08.30` rolls over to the specified directory when the date changes to August 31st 2008. Subsequently, a new log file is created, called `nsjsp.log.2008.08.31`, and the logs are written in this file.

Similarly, if you configure the roll over to occur on an hourly basis, the log file name includes the date and time at which the log file is created along with the specified name. At the end of an hour, the log file rolls over to the specified directory and a new log file is created with the new time. For example, if the log file is created at 02.00.30 hours on August 30th, 2008, the name of the log file will be `nsjsp.log.2008.08.30.02.00.30`.

Note. If the `archiveDirectory` attribute value is set, the log file moves to the set location. Otherwise, the log file remains in the current directory, which is the value of the `destination` attribute.

The `archiveDirectory` Attribute

The `archiveDirectory` attribute value specifies the location where the log files will be moved upon roll over. The value of the `archiveDirectory` attribute can be either an absolute path or a relative path of the log file. If the `archiveDirectory` attribute value is a relative path, the log files are placed in a directory relative to the `${catalina.base}` directory. Also, you can configure an absolute path. Consequently, log files are placed in the specified absolute path.

If the `archiveDirectory` attribute value points to an invalid directory location or if the `archiveDirectory` attribute is not configured, the log files roll over in the current location that is denoted by the `destination` attribute value.

For information on configuring for log rollover, see [Configuring for Log Rollover](#) on page 5-15.

Logging Configuration

This section describes how to configure the following:

- [Configuring Handlers](#)
- [Configuring Loggers](#)
- [Configuring the NSJSP Formatter Class](#)
- [Configuring for Log Rollover](#)
- [Configuring the logging.properties File](#)
- [Configuring Logging for the NSJSP Container and Web Applications](#)

Configuring Handlers

You can configure each handler using the configuration properties that are specific to the handler.

NSJSP supports the following implementation classes:

- [FileHandler](#)
- [NSJSPLogHandler](#)

FileHandler

[Table 5-2](#) lists the `FileHandler` configuration properties.

Table 5-2. Configuration Properties of `FileHandler`

Configuration Property	Description	Default Value
<code>level</code>	<p>Specifies the log level. You can assign one of the following log levels:</p> <ul style="list-style-type: none"> ● <code>SEVERE</code> ● <code>WARNING</code> ● <code>INFO</code> ● <code>CONFIG</code> ● <code>FINE</code> ● <code>FINER</code> ● <code>FINEST</code> ● <code>OFF</code> ● <code>ALL</code> <p>Note: Assigning <code>OFF</code> causes all logging to be turned off. Assigning <code>ALL</code> causes all logging to be turned on.</p>	<code>INFO</code>
<code>Filter</code>	Specifies the filter class to be used.	<code>None</code>
<code>Formatter</code>	Specifies the formatter class to be used.	<code>SimpleFormatter</code>
<code>Encoding</code>	Specifies the character set encoding to be used.	Default platform encoding

The following properties are available in addition to the properties listed in [Table 5-2](#):

- `directory`

- `prefix`

Following is an example of the `FileHandler`:

```
#4admin.org.apache.juli.FileHandler.level = INFO
#4admin.org.apache.juli.FileHandler.directory =
${catalina.base}/logs
#4admin.org.apache.juli.FileHandler.prefix = admin.
```

NSJSPLogHandler

[Table 5-3](#) lists the configuration properties that can be specified for the `NSJSPLogHandler`.

Table 5-3. Configuration Properties of NSJSPLogHandler (page 1 of 2)

Configuration Property	Description	Default Value
<code>destination</code>	<p>Specifies the logging location. You can assign one of the following options:</p> <ul style="list-style-type: none"> ● <code>OSS file</code> ● <code>STDERR</code> ● <code>STDOUT</code> ● <code>EMS</code> <p>The Open System Services (OSS) file name is derived from the server class name. For example, if the server class name is <code>abc</code>, the OSS file name will be <code>abc.<date>.log</code>. The extension, <code>.log</code> is appended by the server.</p>	<code>OSS file</code>
<code>format</code>	Enables you to configure the message format.	<p>The following is the default format: <code>{DATE,date,EEE,MMM dd, HH:mm:ss}; {PROCESSNAME}; {LEVEL}; {SOURCE}; {MESSAGE}</code></p>
<code>maxFileSize</code>	<p>Specifies the file size (in MB) at which the file must roll over. The property is applicable only if the value assigned to the destination property is an OSS filename. You can assign any value above 10 MB.</p>	Disabled by default.

Table 5-3. Configuration Properties of NSJSPLogHandler (page 2 of 2)

Configuration Property	Description	Default Value
datePattern	Specifies the time interval at which the log file must roll over. For the complete list of options to enable rollover, see Configuring for Log Rollover Based on Timestamp on page 5-17.	'.'yyyy-MM-dd
archiveDirectory	Specifies the directory location where the rolled over log files reside. You can assign any directory location. If no value is specified, files rollover in the same directory where the logs are created.	\${catalina.base}/logs/archive
level	Specifies the log level. You can assign one of the following log levels: <ul style="list-style-type: none"> ● SEVERE ● WARNING ● INFO ● CONFIG ● FINE ● FINER ● FINEST ● OFF ● ALL <p>Note: Assigning OFF causes all logging to be turned off. Assigning ALL causes all logging to be turned on.</p>	INFO

The following is an example of the NSJSPLogHandler:

```

linsjsp.com.tandem.servlet.logging.NSJSPLogHandler.destination =
${catalina.base}/logs/abc

linsjsp.com.tandem.servlet.logging.NSJSPLogHandler.level = INFO

linsjsp.com.tandem.servlet.logging.NSJSPLogHandler.datePattern =
'.'yyyy-MM-dd

linsjsp.com.tandem.servlet.logging.NSJSPLogHandler.archiveDirectory =
${catalina.base}/logs/archive

```


In the given example, `${catalina.base}/logs/abc` is the destination of the log messages

where,

`abc` is the name of the server class. The name of the log file will be `abc.<date>.log`.

Multiple Handler Definitions Using the same Handler Class

JULI allows multiple handler definitions using the same handler class. You must configure multiple handlers as follows:

- The handler name must begin with a number and a prefix string.
- The prefix string must end with ". " .

Following are examples of handlers using the `org.apache.juli.FileHandler` implementation class:

- `1catalina.org.apache.juli.FileHandler`

where,

`1catalina.` - Denotes the prefix string including a number and ending with a ". " .

- `2localhost.org.apache.juli.FileHandler`

where,

`2localhost.` - Denotes the prefix string including a number and ending with a ". " .

Configuring Loggers

To configure a logger, you can associate the logger with a handler. As a result, the settings defined for the handler will apply. If no handler is associated with a logger, settings of the parent logger will apply. In addition, you can also assign a log level to the logger.

The following is an example of a parent logger configuration:

```
sample.bank.level=INFO
```

```
sample.bank.handlers=<handler1, handler2, handler 3>
```

where,

`sample.bank` - is the logger.

`level` - is the log level property.

`INFO` - is the log level value.

`<handler1, handler2, handler 3>` - are the associated handlers.

The following is an example of a child logger configuration whose parent logger is `sample.bank`:

```
sample.bank.servlet.level = INFO
sample.bank.servlet.useParentHandlers=true
sample.bank.servlet.handlers =
lbankapp.com.tandem.servlet.logging.NSJSPLogHandler
```

where,

`sample.bank.servlet` - is the logger.

`useParentHandlers` - if the value of this property is `true`, log messages will be sent to the parent logger, `sample.bank`.

`level` - is the log level property.

`INFO` - is the log level value.

`lbankapp.com.tandem.servlet.logging.NSJSPLogHandler` - is the associated handler.

Note. The associated handler can include its own log level. However, the log level assigned to the logger will apply. In the given example, the log level, `INFO` overrides the log level assigned to `lbankapp.com.tandem.servlet.logging.NSJSPLogHandler`.

Configuring the NSJSP Formatter Class

The NSJSP Formatter class uses the following default message format to write logs:

```
{DATE,date,EEE, MMM dd, HH:mm:ss}; {PROCESSNAME};
{LEVEL};{SOURCE}; {MESSAGE}
```

where, `{DATE,date,EEE, MMM dd, HH:mm:ss}`, `{PROCESSNAME}`, `{LEVEL}`, `{SOURCE}`, and `{MESSAGE}` are literals.

[Table 5-4](#) lists the predefined literals that enable you to configure the log message format.

Table 5-4. Literals in the Format Attribute

Literal	Data Type	Description
MESSAGE	String	The text message to be published.
LEVEL	String	The message severity as indicated by the application.
PROCESSNAME	String	The name of the JVM process.
PIN	Short	The PIN of the JVM process.
CPU	Short	The CPU in which the JVM process is running.
SOURCE	String	The class and method from where the message originates.
DATE	String	The date and time at which the message is created.

The following is a sample log message in the default message format:

```
Mon, Oct 22,
15:35:17;$Z4Z2;SEVERE;StandardContextSF#storeWithBackup; Cannot
move original context output file.
```

where,

Mon, Oct 22, 15:35:17 - is the date.

\$Z4Z2 - is the process name.

SEVERE - is the message severity.

StandardContextSF#storeWithBackup - is where the message originates.

Cannot move original context output file - is the log message.

When NSJSP Formatter is used with any other handler, it uses the default format. By default, NSJSPLogHandler uses the `com.tandem.servlet.logging.NSJSPFormatter` class.

The following syntax describes how you can configure NSJSP Formatter with other handlers, such as, `ConsoleHandler`:

```
java.util.logging.ConsoleHandler.formatter
=com.tandem.servlet.logging.NSJSPFormatter
```

Upon successful configuration, the `ConsoleHandler` will use the `NSJSPFormatter` to format messages.

Configuring for Log Rollover

This section describes the procedures to configure for log rollover.

- [Configuring for Log Rollover Based on File Size](#)
- [Configuring for Log Rollover Based on Timestamp](#)

- [Configuring the archiveDirectory Attribute](#)

Configuring for Log Rollover Based on File Size

This section includes the syntax and examples that describe how to configure for rollover based on file size.

The following is the syntax to configure `maxFileSize`-based rollover:

```
com.tandem.servlet.logging.NSJSPLogHandler.maxFileSize = file size
```

where, *file size* is the threshold size of the log file that triggers the rollover.

The following are examples to configure `maxFileSize`-based rollover:

- `com.tandem.servlet.logging.NSJSPLogHandler.maxFileSize = 6`
In the given example, although the set value is 6, the log file rolls over when the file size exceeds 10 MB, which is the base threshold that triggers the rollover.
- `com.tandem.servlet.logging.NSJSPLogHandler.maxFileSize = 20`
In the given example, the log file rolls over when the file size exceeds 20 MB.

[Table 5-5](#) describes the behavior of log files in `maxFileSize`-based rollover.

Table 5-5. Behavior of Log Files in `maxfilesize`-based Rollover

Log File Destination	Complete Path of Log File	archiveDirectory	New Location After Rollover	Comments
abc.log (Default location)	<NSJSP_HOME>/abc.log	Not specified	<NSJSP_HOME>/abc.log.<timestamp>	When the threshold is exceeded, the log file rolls over, and the timestamp is appended to the file name. The file continues to remain in the same location.
/sample/abc.log (User-defined location)	/sample/abc.log	Not specified	/sample/abc.log.<timestamp>	
abc.log (Default location)	<NSJSP_HOME>/abc.log	/archive/logs/	/archive/logs/abc.log.<timestamp>	The log file rolls over to the archiveDirectory when the threshold is exceeded, and the timestamp is appended to the file name.
/sample/abc.log (User-defined location)	/sample/abc.log	/archive/logs/	/archive/logs/abc.log.<timestamp>	

Configuring for Log Rollover Based on Timestamp

This section includes the syntax and examples that describe how to configure for rollover based on timestamp.

The following is the syntax to configure the `datePattern`-based rollover:

```
com.tandem.servlet.logging.NSJSPLogHandler.datePattern =
'.'datePattern value
```

where, *datePattern value* is the time interval at which the rollover occurs and '.' improves the readability of the file name.

The `datePattern` attribute value must follow certain guidelines for date and time pattern strings.

[Table 5-6](#) describes the date and time pattern strings you can use to configure for log rollover.

Table 5-6. Timestamp Attributes

Alphabet	Date or Time Component	Examples
G	Era designator	AD
y	Year	1992;92
M	Month in a year	July; Jul; 07
w	Week in a year	25
W	Week in a month	3
D	Day in year	190
d	Day in month	12
F	Day of week in month	3
E	Day in week	Monday; Mon
a	am/pm marker	PM
H	Hour in day (0-23)	0
k	Hour in day (1-24)	24
K	Hour in am/pm (0-11)	0
h	Hour in am/pm (1-12)	12
m	Minute in hour	30
s	Second in minute	45
S	Millisecond	900
z	Time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	-0800

Using combinations of the date and time pattern strings, you can configure timestamp-based rollover.

[Table 5-7](#) describes the different time intervals at which rollover occurs.

Table 5-7. Time Intervals for Rollover

Attribute	Description
"YYYY"	Rollover occurs after every year from the time the log file is created. The change in the value of <code>YYYY</code> triggers the rollover.
"YYYY.MM"	Rollover occurs after every month from the time the log file is created. The change in the value of <code>MM</code> triggers the rollover.
"YYYY.MM.W"	Rollover occurs after every week in the month from the time the log file is created. The change in the value of <code>w</code> triggers the rollover.
"YYYY.MM.dd"	Rollover occurs after every day from the time the log file is created. The change in the value of <code>dd</code> triggers the rollover.
"YYYY.ww"	Rollover occurs after every week in the year from the time the log file is created. The change in the value of <code>ww</code> triggers the rollover.
"YYYY.MM.dd.hh"	Rollover occurs after every hour from the time the log file is created. The change in the value of <code>hh</code> triggers the rollover.
"YYYY.MM.dd.hh.mm"	Rollover occurs after every minute from the time the log file is created. The change in the value of <code>mm</code> triggers the rollover.
"YYYY.MM.dd.hh.a"	Rollover occurs at midday (12:00 hours) and at midnight (00:00 hours), when AM changes to PM or vice-versa. The change in the value of <code>a</code> triggers the rollover.

The following are examples to configure the `datePattern`-based rollover:

- `com.tandem.servlet.logging.NSJSPLogHandler.datePattern = ' . 'yyyy.MM.dd`

In the given example, the log file rolls over every day, when the value of `dd` changes.

- `com.tandem.servlet.logging.NSJSPLogHandler.datePattern = ' . 'yyyy.MM.dd.hh`

In the given example, the log file rolls over every hour, when the value of `hh` changes.

In the given examples, `' . '` improves the readability of the file name.

[Table 5-8](#) describes the behavior of log files in `datePattern`-based rollover.

Table 5-8. Behavior of Log Files in datepattern-based Rollover

Log File Destination	Complete Path of Log File	archiveDirectory	New Location After Rollover	Comments
abc.log (Default location)	<NSJSP_HOME>/abc.log.<timestamp>	Not specified	<NSJSP_HOME>/abc.log.<timestamp>	When NSJSP starts, the timestamp is appended to the log file name.
/sample/abc.log (User-defined location)	/sample/abc.log.<timestamp>	Not specified	/sample/abc.log.<timestamp>	When the threshold is exceeded, the log file rolls over, and it continues to remain in the same location.
abc.log (Default location)	<NSJSP_HOME>/abc.log.<timestamp>	/archive/logs/	/archive/logs/abc.log.<timestamp>	When NSJSP starts, the timestamp is appended to the log file name. The log file rolls over to the
/sample/abc.log (User-defined location)	/sample/abc.log.<timestamp>	/archive/logs/	/archive/logs/abc.log.<timestamp>	archiveDirectory when the threshold is exceeded.

Configuring the archiveDirectory Attribute

This section includes the syntax and examples that describe how to configure the `archiveDirectory` attribute.

The following is the syntax to configure the `archiveDirectory`:

```
com.tandem.servlet.logging.NSJSPLogHandler.archiveDirectory =
archive directory
```

where, *archive directory* is the location to which the log files rollover.

The following is an example to configure the `archiveDirectory`:

```
com.tandem.servlet.logging.NSJSPLogHandler.archiveDirectory =
tempLogs
```

In the given example, the log file rolls over to the `tempLogs` directory. The complete path of the `archiveDirectory` is `<NSJSP_HOME>/tempLogs`.

Configuring the logging.properties File

This section describes the contents of the logging.properties file.

You can use the logging.properties file to configure log-related settings, such as, which logs must be written, where the logs must be written, and when the logs must rollover. The file contains java.util.logging elements, such as loggers and handlers, and various attributes to which you can assign values to control the logs.

You can specify the location of the logging.properties file by configuring the Java system property, java.util.logging.config.file. By default, the directory location is <NSJSP 6.1 installation directory>/conf.

The logging.properties file contains handlers and loggers.

Handlers in the logging.properties File

The following lines indicate the handlers defined in the logging.properties file:

```
handlers = 1nsjsp.com.tandem.servlet.logging.NSJSPLogHandler,
2console.com.tandem.servlet.logging.NSJSPLogHandler,
1catalina.org.apache.juli.FileHandler,
2localhost.org.apache.juli.FileHandler,
3manager.org.apache.juli.FileHandler,
4admin.org.apache.juli.FileHandler, 5host-
manager.org.apache.juli.FileHandler
```

In the given example, the handlers attribute is assigned seven values. Each of the seven handlers has its own settings. However, among the seven handlers, only two handler implementation classes are used. They are

com.tandem.servlet.logging.NSJSPLogHandler and org.apache.juli.FileHandler. These handler implementation classes are loaded during the NSJSP startup.

The following line sets the handlers property for the root logger. The root logger is an empty string. The complete attribute name is <root logger>.handlers. This is the default handler. If no defined handler meets the requirements to enable logging, the default handler is used.

```
.handlers = 1nsjsp.com.tandem.servlet.logging.NSJSPLogHandler
```

The following lines denote the configuration settings of the handler, called

1nsjsp.com.tandem.servlet.logging.NSJSPLogHandler:

```
1nsjsp.com.tandem.servlet.logging.NSJSPLogHandler.destination =
${catalina.base}/logs/nsjsp
1nsjsp.com.tandem.servlet.logging.NSJSPLogHandler.level = INFO
1nsjsp.com.tandem.servlet.logging.NSJSPLogHandler.datePattern =
'. 'yyyy-MM-dd
1nsjsp.com.tandem.servlet.logging.NSJSPLogHandler.archiveDirecto
ry = ${catalina.base}/logs/archive
```


The following list describes the settings of

`1nsjsp.com.tandem.servlet.logging.NSJSPLogHandler:`

- Destination - `${catalina.base}/logs/nsjsp`
- Log level - INFO
- Rollover interval - `'.'yyyy-MM-dd`
- Archive directory for the logs that rollover - `${catalina.base}/logs/archive`

The following lines denote the configuration settings of the handler, called

`1catalina.org.apache.juli.FileHandler:`

```
1catalina.org.apache.juli.FileHandler.level = INFO
1catalina.org.apache.juli.FileHandler.directory =
${catalina.base}/logs
1catalina.org.apache.juli.FileHandler.prefix = catalina
```

The following list describes the settings of

`1catalina.org.apache.juli.FileHandler:`

- Log level - INFO
- Log directory - `${catalina.base}/logs`
- Prefix properties - `catalina`

Note. The `FileHandler` implementation class does not support log rollover.

The following lines denote the configuration settings of the handler, called

`2localhost.org.apache.juli.FileHandler:`

```
2localhost.org.apache.juli.FileHandler.level = INFO
2localhost.org.apache.juli.FileHandler.directory =
${catalina.base}/logs
2localhost.org.apache.juli.FileHandler.prefix = localhost
```

The following list describes the settings of

`2localhost.org.apache.juli.FileHandler:`

- Log level - INFO
- Log directory - `${catalina.base}/logs`
- Prefix properties - `localhost`

The following lines denote the configuration settings of the handler, called

`3manager.org.apache.juli.FileHandler:`

```
3manager.org.apache.juli.FileHandler.level = INFO
```

```
3manager.org.apache.juli.FileHandler.directory =  
${catalina.base}/logs
```

```
3manager.org.apache.juli.FileHandler.prefix = manager
```

The following list describes the settings of

`3manager.org.apache.juli.FileHandler:`

- Log level - INFO
- Log directory - `${catalina.base}/logs`
- Prefix properties - `manager`

The following lines denote the configuration settings of the handler, called

`4admin.org.apache.juli.FileHandler:`

```
4admin.org.apache.juli.FileHandler.level = INFO
```

```
4admin.org.apache.juli.FileHandler.directory =  
${catalina.base}/logs
```

```
4admin.org.apache.juli.FileHandler.prefix = admin
```

The following list describes the settings of

`4admin.org.apache.juli.FileHandler:`

- Log level - INFO
- Log directory - `${catalina.base}/logs`
- Prefix properties - `admin`

The following lines denote the configuration settings of the handler, called

`5host-manager.org.apache.juli.FileHandler:`

```
5host-manager.org.apache.juli.FileHandler.level = INFO
```

```
5host-manager.org.apache.juli.FileHandler.directory =  
${catalina.base}/logs
```

```
5host-manager.org.apache.juli.FileHandler.prefix = host-manager
```

The following list describes the settings of

`5host-manager.org.apache.juli.FileHandler:`

- Log level - INFO
- Log directory - `${catalina.base}/logs`
- Prefix properties - `host-manager`

The following lines denote the configuration settings of the handler, called

```
2console.com.tandem.servlet.logging.NSJSPLogHandler:
```

```
2console.com.tandem.servlet.logging.NSJSPLogHandler.level = INFO
```

```
2console.com.tandem.servlet.logging.NSJSPLogHandler.destination  
= STDOUT
```

The following list describes the settings of

```
2console.com.tandem.servlet.logging.NSJSPLogHandler:
```

- Log level - INFO
- Destination - STDOUT

The location of `STDOUT` is defined in the server class definition.

Loggers in the logging.properties File

This section describes the loggers used in the `logging.properties` file.

The following lines denote the log level and handler properties of the logger, called

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost]:
```

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].level  
= INFO
```

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].handl  
ers = 2localhost.org.apache.juli.FileHandler
```

The logger writes log messages about the Host component, `localhost`, which is located within the engine component, `NSJSP`.

The following lines denote the log level and handler of the logger, called

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/ser  
vletsj/manager]:
```

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/ser  
vletsj/manager].level = INFO
```

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/ser  
vletsj/manager].handlers = 3manager.org.apache.juli.FileHandler
```

The logger writes log messages about the context component, `/servletsj/manager`. The context component, which refers to the Manager Web application, is located within the Host component, `localhost`. The `localhost` component, in turn, is located within the engine component, `NSJSP`.

The following lines denote the log level and handler properties of the logger, called

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/ser  
vletsj/admin]:
```

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/ser  
vletsj/admin].level = INFO
```

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/ser  
vletsj/admin].handlers = 4admin.org.apache.juli.FileHandler
```

The logger writes log messages about the context component, `/servletsj/admin`. The context component, which refers to the Admin Web application, is located within the Host component, `localhost`. The `localhost` component, in turn, is located within the engine component, `NSJSP`.

The following lines denote the log level and handler properties of the logger, called `org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/servletsj/host-manager]`:

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/servletsj/host-manager].level = INFO
```

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/servletsj/host-manager].handlers = 5host-manager.org.apache.juli.FileHandler
```

The logger writes log messages about the context component, `/servletsj/host-manager`. The context component, which refers to the NSJSP Manager application, is located within the Host component, `localhost`. The `localhost` component, in turn, is located within the engine component, `NSJSP`.

All logs originating from the Host component and the three web applications are handled by the JULI File Handler. Each of the four handlers that are defined by the File Handler class (`2localhost.*`, `3manager.*`, `4admin.*`, `5host-manager.*`) open four different files as indicated by the prefix attribute of the handlers.

Any log originating outside the Host component and the three web applications will be handled by the `1NSJSP.*NSJSPLogHandler` handler, which is associated with the root logger.

Configuring Logging for the NSJSP Container and Web Applications

This section describes the procedures to configure logging for the NSJSP servlet container and web applications running on the server class Host.

Each server class includes a `logging.properties` file, which is located in `<NSJSP 6.1 installation directory>/conf`. You can configure the `logging.properties` file to manage logging related to the server class components and the web applications running on the server class Host. When the `logging.properties` file is configured, and NSJSP is restarted, all logs will be written in the newly created file, called `<server class>.<date>.log`. This log file will be created in `<NSJSP_HOME>/logs`.

Additionally, each web application running on the server class Host can include its own `logging.properties` file, which is located in `<NSJSP 6.1 installation directory>/webapps/<application name>/WEB-INF/classes`. You can configure this file to manage logging related to the specific web application.

Configuring the web application-specific `logging.properties` file, which is located in `<NSJSP 6.1 installation directory>/webapps/<application name>/WEB-INF/classes` is not a mandatory procedure. It is required only to

configure logging that is specific to the application. If an application-specific `logging.properties` file is not present, the settings in the generic `logging.properties` file, located in `<NSJSP 6.1 installation directory>/conf` apply.

The settings in the web application-specific `logging.properties` file override the settings in the generic `logging.properties` file, which is located in `<NSJSP 6.1 installation directory>/conf`.

Configuring Logging for Web Applications

This section describes the procedure to configure logging for web applications.

To configure the `logging.properties` file specific to an application, complete the following steps:

1. Go to the following directory location:

```
<NSJSP 6.1 installation directory>/webapps/<application  
name>/WEB-INF/classes
```

2. Create the `logging.properties` file and assign the required values.
3. Save and close the file.
4. Restart NSJSP.

The configuration settings will become effective.

The following example illustrates logging configuration for a web application:

```
handlers = lbankapp.com.tandem.servlet.logging.NSJSPLogHandler
#This is the default handler

.handlers = lbankapp.com.tandem.servlet.logging.NSJSPLogHandler

#####
# Handler specific properties.
# Describes specific configuration info for Handlers.
#####

#####
#Destination can have values like STDOUT,STDERR and EMS. With
these three values roll over will be disabled

#With the value of datePattern set to yyyy-MM-dd the files will
roll over each day

#To enable size based roll over un-comment the following line.
This will roll over the log file once it reaches 10MB

#lnsjsp.com.tandem.servlet.logging.NSJSPLogHandler.maxFileSize =
10

#The rolled over log files are placed in the archiveDirectory
(logs/archive)

#If archive directory is not mentioned the rolled over files
will be present in the same directory as the log file

#####
lbankapp.com.tandem.servlet.logging.NSJSPLogHandler.destination
= ${catalina.base}/logs/bankapp

lbankapp.com.tandem.servlet.logging.NSJSPLogHandler.level = INFO

lbankapp.com.tandem.servlet.logging.NSJSPLogHandler.maxFileSize
= 10

lbankapp.com.tandem.servlet.logging.NSJSPLogHandler.archiveDirec
tory = ${catalina.base}/logs/archive

#####
# Facility specific properties.
# Provides extra control for each logger.
#####
```

```
sample.bank.servlet.level = INFO

sample.bank.servlet.handler =
lbankapp.com.tandem.servlet.logging.NSJSPLogHandler

sample.bank.util.level = SEVERE

sample.bank.util.handler =
lbankapp.com.tandem.servlet.logging.NSJSPLogHandler
```

Log Files Related to NSJSP

This section describes the log files related to NSJSP.

The default configuration of NSJSP creates five log files in the `<NSJSP_HOME>/logs` directory per NSJSP installation. Following are the log files:

- Two files with the extension, `out`. These `out` files are the `STDOUT` files defined in the two server classes that are configured for each installation.
- Two files with the extension, `err`. These `err` files are the `STDERR` files defined in the server classes that are configured for each installation.
- A log file with the extension, `log`. This log file is created by the NSJSP logging framework, JULI.

The `out` and `err` Log Files

All log messages from the Java Virtual Machine (JVM) are directed to either the `out` file or the `err` file. For example, if an error in the arguments is passed to the JVM through the `Arglist` configuration parameter, the JVM writes the message to the `STDERR` file. In NSJSP, the `STDERR` file is configured as the `err` file. When the server is starting up, the Java Native Interface (JNI) code of NSJSP writes messages to the `out` file. Startup messages are written to the `out` file, because these messages are generated before the JULI framework is initialized.

The complete names of the `out` and `err` log files are derived from the names of the server classes to which they belong. The `out` file of the Servlet Server Class is `<Servlet Server Class name>.out`. The `err` file of the Servlet Server Class is `<Servlet Server Class name>.err`.

Similarly, the `out` file of the Admin Server Class is `<Admin Server Class name>.out`. The `err` file of the Admin Server Class is `<Admin Server Class name>.err`.

The `out` and `err` log files of the Servlet Server Class are defined in the `servlet.config` file. The `out` and `err` log files of the Admin Server Class are defined in the `nsjspadmin.config` file.

You cannot configure the settings of the `out` and `err` files. In addition, the `out` and `err` files do not support log rollover.

Log File Created by JULI

When the NSJSP server is running and the JULI framework is initialized, all log messages are written in the log file created by the JULI logging framework.

Note. Under certain conditions, log messages are also written to the `err` file. For example, when the internal buffers of the NSJSP JNI library are full, log messages are written in the log file created by the JULI logging framework and in the `err` file.

The log file created by JULI is the default destination of logs. However, you can change the default destination of logs.

For information on log file configuration, see [Configuring the logging.properties File](#) on page 5-20.

The name of the log file is derived from the Servlet Server Class name. For example, if the name of the Servlet Server Class is ABC, the name of the log file is

ABC.<date>.log

where,

ABC - is the name of the Servlet Server Class.

<date> - is the date when NSJSP starts. The date is in yyyy-mm-dd format, and is automatically added.

This log file supports log rollover. By default, date-based rollover is enabled. However, unlike the `out` and `err` files, you can modify the settings of this log file.

Programming Considerations for Logging

To use JULI logging framework, you must be aware of the following classes:

- [LogFactory](#)
- [Log](#)

LogFactory

The `LogFactory` class is used to identify an underlying logging implementation. It provides an entry point for the applications to obtain the appropriate implementation of the `Log` interface. The `LogFactory` implementation in JULI is similar to the `LogFactory` implementation in Commons Logging, except that the `LogFactory` has a hard-coded implementation for `java.util.logging`.

For information on the `LogFactory` implementation in Commons Logging, see <http://commons.apache.org/logging/>.

The following is an example of the `getLog` method of the `LogFactory` class (`org.apache.juli.logging.Log`) to obtain the most suitable implementation of

the `Log` interface (`org.apache.juli.logging.Log`) for the `NSJSPInputBuffer.class`:

```
protected static org.apache.juli.logging.Log log =
org.apache.juli.logging.LogFactory.getLog(NSJSPInternalInputBuff
er.class);
```

Log

In JULI, the implementation of the `Log` interface is for the underlying logging implementation of `java.util.logging`. JULI implements the `Log` interface of Commons logging for the `java.util.logging` logging framework.

The following is an example of the `Log` interface implementation for the `MyTestClass.class` from the logging configuration file, `logging.properties`:

```
Log logger = LogFactory.getLog(MyTestClass.class)
```

The `Log` interface of Commons Logging defines methods such as `error`, `fatal`, `trace`, and so on. The underlying logging implementation, `java.util.logging`, defines its own logging levels.

[Table 5-9](#) shows the mapping between Commons Logging methods and `java.util.logging` levels.

Table 5-9. Mapping of Logging Methods

java.util.logging.level	Commons Logging Log Method
SEVERE	error, fatal
WARNING	warn
INFO	info
FINE	debug
FINER	trace

Note. The priority of log level defined in [Table 5-9](#) is in the descending order. The log level `SEVERE` has the highest priority and log level `FINER` has the lowest priority.

The `java.util.logging` levels are defined in the `logging.properties` configuration file. The Commons Logging methods are used in the user application.

In the following sample of code, the `import` statements are used to import the `LogFactory` and the `Log` classes of the Commons Logging. The `LogFactory` class creates the logger, which in turn used to log messages based on the log levels such as `info`, `warning`, and so on.

```
....
....
import org.apache.juli.logging.LogFactory;
import org.apache.juli.logging.Log;
....
```

```

.....
public class SampleLoggingClass
{
private static Log _logger =
LogFactory.getLog(SampleLoggingClass.class);
.....
.....
public void sampleMethod()
{
.....
.....
if(_logger.isDebugEnabled())
{
// Compose the message
_logger.debug(message);
}

_logger.error(error message);
}
}

```

Commons Logging

The applications use different logging implementations, such as, `log4j` and `java.util.logging`. The logging implementations do not follow any particular standard. The applications written using one logging implementation must be modified to switch to any other logging implementation. To overcome this drawback, the actual logging implementation must be abstracted from the application using the logging implementation. Commons Logging is the abstraction between the application and the actual logging implementation.

The Commons Logging package is an ultra-thin bridge between different logging implementations. An application that is programmed to use the commons-logging application programming interface (API) can be used with any logging implementation at runtime.

Commons Logging is one of the components of an Apache Commons project that provides a layer of abstraction over many popular logging implementations. Using Commons Logging, you can program to an interface rather than to an implementation.

The main components of Commons Logging are as follows:

- `Log` interface – It provides an interface that is intended to be an independent abstraction of the underlying logging implementation.
- `LogFactory` – It detects the underlying logging implementation and creates the log instances for the logging implementation detected.

For more information about Commons Logging, see <http://commons.apache.org/logging/>.

6

Debugging NSJSP

This chapter discusses the various ways of debugging applications deployed in NSJSP. Debugging using Java Debugger tool and Eclipse platform have been described elaborately in this chapter.

This chapter discusses the following topics:

- [Debugging using Java Debugger tool](#)
- [Debugging using Eclipse platform](#)

Debugging using Java Debugger tool

Use the `jdb` command to start the Java Debugger tool and communicate with the web application to be debugged. You need to add `jdb` debugging parameters in the `servlet.config` file in `<NSJSP_HOME>/conf` directory. This can be performed in the following two ways:

1. Connecting to a specified port number

This is achieved by adding arguments in the `Arglist` of the `servlet.config` file in `<NSJSP_HOME>/conf` directory as shown:

- `-Xdebug -Xnoagent -Djava.compiler=none
-Xrunjdw:server=y,transport=dt_socket,suspend=n,
address=<debug-port>`
- Set `Numstatic` value to 1

Attach the Java Virtual Machine to the Java Debugger (`jdb`) using the following command:

```
jdb -attach <debug-port>
```

2. Using process attaching connector

This is achieved by adding arguments in the `Arglist` of the `servlet.config` file in `<NSJSP_HOME>/conf` directory as shown:

- `-Xdebug -Xnoagent -Djava.compiler=none
-Xrunjdw:server=y,transport=dt_socket,suspend=n`
- This connector is uniquely identified as `com.sun.jdi.ProcessAttach`. Any NSJSP process can be connected using this `jdb`'s `ProcessAttach` as given below:

```
jdb -connect com.sun.jdi.ProcessAttach:pid=<oss_pid>
```

For example:

```
jdb -connect com.sun.jdi.ProcessAttach:pid=218431829
```

where,

218431829 is the process ID (PID) of the NSJSP process to be debugged.

Note:

1. You can open multiple `jdb` sessions and connect to any NSJSP processes in separate `jdb` sessions.
2. There is no limit for the Numstatic value for this option.
3. `jdb` provides all debug options like setting breakpoints, watch a field value, continue, step in, step out and so on.

For more information on debugging, see *Debugging Java Programs* in *Implementation specific* section of *NonStop Java 6.0 Programmer's Pages Manual* and, the *jdb: Java Debugger* section in *NonStop Java 6.0 Tools Reference Pages Manual*.

Debugging using Eclipse platform

This section describes the usage of open source Eclipse platform to debug web applications deployed in NSJSP.

To debug web applications, complete the following steps:

1. Add the debugging parameters in the `Arglist` of `servlet.config` file in `<NSJSP_HOME>/conf` directory. This can be performed in the following three ways:

- a. Debugging web application with Numstatic equal to 1 with single debug port

If you intend to debug web application with single port, make the following changes to the `servlet.config` file in `<NSJSP_HOME>/conf` directory:

- Add the following arguments to the `Arglist` of the `servlet.config` file.

```
Arglist -Xdebug -Xnoagent -Djava.compiler=none
-Xrunjdwp:server=y,transport=dt_socket,suspend=n,
address=<debug-port>
```

- Set the Numstatic and Maxservers values to 1

For more information on java command-line options, which are used for debugging see the *Debugging Java Programs* section in *NonStop Java 6.0 Programmer's Pages Manual*.

- b. Debugging web application with Numstatic greater than 1 with dynamic debug ports

If you intend to debug web application with dynamically allocated ports, make the following changes to the `servlet.config` file in `<NSJSP_HOME>/conf` directory.

- Add the following arguments in the Arglist of `servlet.config` file:
`-Xdebug -Xnoagent -Djava.compiler=none`
`-Xrunjdw:server=y,transport=dt_socket,suspend=n`
- Set the Numstatic and Maxservers values to greater than or equal to 1

During NSJSP startup, when the port number is not specified in the `servlet.config` file, the debug ports are dynamically allocated to individual NSJSP processes. These allocated ports are captured in `<Servlet Server Class name>.out` file present in `<NSJSP_HOME>/logs` directory.

For example:

For Numstatic equal to 4, the ports allocated is displayed in the `<Servlet Server Class name>.out` file as shown:

```
Listening for transport dt_socket at address:4129
Listening for transport dt_socket at address:4131
Listening for transport dt_socket at address:4130
Listening for transport dt_socket at address:4133
```

These debug ports, can now be connected from Eclipse.

When Numstatic is greater than 1, the ports used for debugging are generated dynamically and therefore cannot be customized in the `servlet.config` file.

- Debugging web application with Numstatic greater than 1 with debug port range

If you intend to debug a specific port range with Numstatic and Maxservers value greater than 1, make the following changes to the `servlet.config` file in `<NSJSP_HOME>/conf` directory:

- Add the following arguments in the Arglist of `servlet.config` file:
`Arglist -Xdebug -Xnoagent -Djava.compiler=none -`
`Xrunjdw:server=y,transport=dt_socket,suspend=n,address=<min-debug-port> - <max-debug-port>`
- Set the Numstatic and Maxservers value to 1

This option is supported in the *NonStop Server for Java T2766H60^ACA SPR* onwards.

2. Launch Eclipse and select Debug Configuration window. Connect to the NSJSP process using the port as mentioned in `<Servlet Server Class name>.out` file.
3. The debug configurations can be created for individual NSJSP process. However, if you intend to debug for specific NSJSP processes, say process `$Y1BX`, to identify the debug port associated with the process to be used for Eclipse connection, complete the following steps:

1. Go to Guardian shell and execute the following commands:

- `scf`
- `assume $zztcp`
- `listopen mon *`

This lists all the NSJSP processes and their respective debug port numbers.

For example: 2122 is the debug port for nsjsp process \$Y1BX:

\HEMAN.\$Y1BX	0,732	8	2	TCP	2122
---------------	-------	---	---	-----	------

4. In Eclipse, set the necessary breakpoints. For more info, see the Eclipse website <http://www.eclipse.org/> for more details.
5. Open the web application from the browser. Once the page is loaded, the web application is suspended at the breakpoint's set. Switch to Eclipse IDE to select **Debug perspective** and continue the debugging.

7 Migrating to NSJSP 6.1

This chapter compares the migration-related characteristics of NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1. This chapter also provides the considerations before migrating user applications from NSJSP 5.0 or NSJSP 6.0 to NSJSP 6.1. You can migrate your applications after you have installed and configured NSJSP 6.1.

This chapter addresses the following topics:

- [Comparison of NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1](#)
- [Considerations for Migrating Web Applications from NSJSP 5.0 to NSJSP 6.1](#)
- [Considerations for Migrating Web Applications from NSJSP 6.0 to NSJSP 6.1](#)
- [Migrating the Session Store](#)
- [Migrating to NSJSP Manager Application in NSJSP 6.1](#)
- [Support for Multiple NSJSP Installations in a Single ITP Secure WebServer Environment](#)

Comparison of NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1

Before migrating your web applications to NSJSP 6.1, you must understand the similarities and differences between the NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1 versions, and then make necessary modifications to either the installation or the application or both. This section compares the following aspects of the different NSJSP releases: installation, configuration, management, logging, and some miscellaneous characteristics. The comparison discussed in the subsequent sections are with respect to the default installation of NSJSP.

Comparing Installation Properties in NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1

[Table 7-1](#) compares the installation properties of NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1.

Table 7-1. Comparison of Installation Properties of NSJSP Versions (page 1 of 3)

Factors	NSJSP 5.0	NSJSP 6.0	NSJSP 6.1
NSJSP installation directory	By default, NSJSP is installed in the <code><TPWS_INSTALL_HOME>/servlet_jsp</code> directory. You cannot change the location of the installation directory.	By default, NSJSP is installed in the <code><TPWS_INSTALL_HOME>/servlet_jsp</code> directory. You cannot change the location of the installation directory.	You can install NSJSP in a directory location of your choice.
URI to access the NSJSP installation	<code>servlet_jsp</code> (default URI name)	<code>servlet_jsp</code> (default URI name), users have the ability to manually modify the URI. You can manually modify the URI using the <code>CONTEXT_PREFIXES</code> command-line option. For more information on this command-line option, see Chapter 3, Configuring NSJSP .	A default URI name is not available. You can specify the URI name during installation. However, a unique URI name must be assigned to each installation.
Server classes created during installation	<ul style="list-style-type: none"> ● <code>servlet</code> (Servlet Server class) ● <code>nsjspadmin</code> (Admin Server class) 	Same as in NSJSP 5.0	<p>You can specify the name of the Servlet Server Class during installation. The name of the Admin Server Class is automatically derived from the Servlet Server Class name as follows:</p> <ul style="list-style-type: none"> ● <code><server class_name></code> (Servlet Server Class) ● <code><server class_name>-adm</code> (Admin Server class)

Table 7-1. Comparison of Installation Properties of NSJSP Versions (page 2 of 3)

Factors	NSJSP 5.0	NSJSP 6.0	NSJSP 6.1
Ability to contain multiple NSJSP installations in an iTP Secure WebServer environment	<p>No</p> <p>Allows single installation of NSJSP 5.0. If NSJSP 5.0 is already installed and you try to install NSJSP again, the script overwrites the existing installation.</p>	<p>No</p> <p>Allows a single installation of NSJSP 6.0. If NSJSP 6.0 is already installed and you try to install NSJSP again, the script overwrites the existing installation.</p>	<p>Yes</p> <p>Allows multiple installations of NSJSP 6.1 in an iTP Secure WebServer environment.</p> <p>NSJSP 6.1 can co-exist with either NSJSP 5.0 or NSJSP 6.0.</p> <p>NOTE:</p> <p>Each installation must be present in a unique directory location.</p>
Directory structure	<p>bin/</p> <p>common/</p> <p>classes/</p> <p>lib/</p> <p>endorsed/</p> <p>conf/</p> <p>NSJSP/</p> <p>backup/</p> <p>lib/</p> <p>logs/</p> <p>server/</p> <p> classes/</p> <p> lib/</p> <p>nsjsp_webapps</p> <p>webapps/</p> <p>work/</p> <p>deployer/</p> <p>shared/</p> <p>temp/</p>	<p>bin/</p> <p>conf/</p> <p>lib/</p> <p>logs/</p> <p>webapps/</p> <p>work/</p> <p>deployer/</p> <p>temp/</p>	<p>Same as in NSJSP 6.0</p>

Table 7-1. Comparison of Installation Properties of NSJSP Versions (page 3 of 3)

Factors	NSJSP 5.0	NSJSP 6.0	NSJSP 6.1
Online-upgrade feature in iTP Secure Web-Server	Not aware of the online-upgrade feature of the iTP Secure WebServer	Same as in NSJSP 5.0	The <code>setup</code> script identifies the online-upgrade feature and configures NSJSP accordingly. For more information, see Chapter 1, Introduction to NSJSP .
Ability to upgrade an existing version of NSJSP to a new SPR version	No	No	Yes
Option to install a management application	Installs the Manager Web application in the <code>NSJSPADMIN</code> server class. (The Manager Web application is denoted as Old Manager application in this guide)	Same as in NSJSP 5.0	Contains an option to install the new NSJSP Manager. The installation script also installs the Old Manager application. While the NSJSP Manager is installed only once in an iTP Secure WebServer, the Old Manager is installed with each installation of NSJSP 6.1. NOTE: The Manager Web application included in NSJSP 5.0 and NSJSP 6.0 is referred to as Old Manager application in this guide.

Comparing Configuration Properties in NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1

The following tables enable you to compare the default container configuration elements of NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1:

- [Table 7-2](#) discusses the differences in the NSJSP 5.0 and NSJSP 6.1 `server.xml` files.
- [Table 7-3](#) discusses the differences in the NSJSP 6.0 and NSJSP 6.1 `server.xml` files.
- [Table 7-4](#) discusses the differences in the NSJSP 5.0 and NSJSP 6.1 `servlet.config` files.

- [Table 7-5](#) discusses the differences in the NSJSP 6.0 and NSJSP 6.1 `servlet.config` files.

The [Considerations for Migrating Web Applications from NSJSP 5.0 to NSJSP 6.1](#) section discusses the `conf/context.xml` file that is included in NSJSP 6.0.

[Table 7-2](#) lists the differences in the NSJSP 5.0 and NSJSP 6.1 `server.xml` files.

Table 7-2. Differences in the NSJSP 5.0 and NSJSP 6.1 `server.xml` Files

Element/Attribute	Default Value in NSJSP 5.0	Default Value in NSJSP 6.1
Server		
debug	0	Not Available
port	Not Available	-1
Listeners	<ul style="list-style-type: none"> ● <code>com.tandem.servlet.catalina.mbeans.NSJSPServerLifecycleListener</code> ● <code>org.apache.catalina.mbeans.GlobalResourcesLifecycleListener</code> 	<ul style="list-style-type: none"> ● <code>org.apache.catalina.core.JasperListener</code> ● <code>org.apache.catalina.mbeans.ServerLifecycleListener</code> ● <code>org.apache.catalina.mbeans.GlobalResourcesLifecycleListener</code> ● <code>org.apache.catalina.storeconfig.StoreConfigLifecycleListener</code> ● <code>com.tandem.servlet.NSJSPLifecycleListener</code>
GlobalNamingResources		

Table 7-2. Differences in the NSJSP 5.0 and NSJSP 6.1 server.xml Files

Element/Attribute	Default Value in NSJSP 5.0	Default Value in NSJSP 6.1
Environment	<pre> name="simpleValue" type="java.lang.Integer" value="30" <Resource name="UserDatabase" auth="Container" type="org.apache.catalina .UserDatabase" description="User database that can be updated and saved"> </Resource> <ResourceParams name="UserDatabase"> <parameter> <name>factory</name> <value>org.apache.catalin a.users.MemoryUserDatabas eFactory</value> </parameter> <parameter> <name>pathname</name> <value>conf/tomcat- users.xml</value> </parameter> </ResourceParams> </pre>	<pre> <Resource name="UserDatabase" auth="Container" type="org.apache.catalina .UserDatabase" description="User database that can be updated and saved" factory="org.apache.catal ina.users.MemoryUserDatab aseFactory" pathname="conf/nsjsp- users.xml" readonly="false"> </pre>
Service		
name	name="NSJSP-iTPWebServer"	name="NSJSP"
Connector	<pre> className="com.tandem.ser vlet.coyote.tomcat5.NSJSP CoyoteConnector" minProcessors="5" maxProcessors="75" acceptCount="25" maxThreads="75" minSpareThreads="5" maxSpareThreads="25" enableLookups="false" redirectPort="443" debug="0" connectionTimeout="0" disableUploadTimeout="tru e" </pre>	<pre> protocol="HTTP/1.1" maxThreads="75" </pre>

Table 7-2. Differences in the NSJSP 5.0 and NSJSP 6.1 `server.xml` Files

Element/Attribute	Default Value in NSJSP 5.0	Default Value in NSJSP 6.1
Engine	<code>name="NSJSP"</code> <code>defaultHost="localhost"</code> <code>debug="0"</code>	<code>com.tandem.servlet.JMXCon nectionListener</code> <code>name="NSJSP"</code> <code>backgroundProcessorDelay= "60"</code> <code>defaultHost="localhost"</code>
Realm	<code>className="org.apache.cat alina.realm.UserDatabaseR ealm"</code> <code>debug="0"</code> <code>resourceName="UserDatabas e"</code>	<code>className="org.apache.cat alina.realm.UserDatabaseR ealm"</code> <code>resourceName="UserDatabas e"</code> <code>digest="MD5"</code> <p>Note: In NSJSP 5.0, passwords were in clear text. In NSJSP 6.1, passwords are encrypted using the MD5 digest algorithm.</p>
Host	<code>name="localhost"</code> <code>debug="0"</code> <code>appBase="webapps"</code> <code>unpackWARs="true"</code> <code>autoDeploy="false"</code> <code>xmlValidation="false"</code> <code>xmlNamespaceAware="false"</code>	<code>name="localhost"</code> <code>appBase="webapps"</code> <code>unpackWARs="true"</code> <code>autoDeploy="true"</code> <code>xmlValidation="false"</code> <code>xmlNamespaceAware="false"</code> <code>configClass="com.tandem.s ervlet.catalina.startup.N SJSPContextConfig"</code> <p>For information on <code>autoDeploy</code> and <code>configClass</code>, see The <code>server.xml</code> File on page 3-37.</p>
Valve Element within the Host	Not available. Only a sample valve configuration is included.	<code>Valve</code> <code>className="com.hp.tandem. nsjsp.valves.RequestTrack erValve"</code> <p>For information on the Valve element, see Valve Element on page 3-58.</p>

The reasons why the values of some attributes in NSJSP 6.1 are different from those in NSJSP 5.0, as described in [Table 7-2](#), are as follows:

`debug` – Because NSJSP 6.1 uses a different logging mechanism, the `debug` flag, which was related to logging severity in NSJSP 5.0 is not valid in NSJSP 6.1.

For more information on logging severity in NSJSP 6.1, see Chapter [5, Logging in NSJSP](#).

`port` – Although the behavior of the server component with respect to the `port` attributes has not changed between NSJSP 5.0 and NSJSP 6.1, NSJSP 6.1 requires the `port` attribute to be explicitly set to -1.

`Listeners` – The `NSJSPServerLifeCycleListener` is replaced by the `ServerLifeCycleListener`. The functionality of the two listeners is the same. For more information on the new listeners in NSJSP 6.1, see [Child Elements Nested in the Server Element](#) on page 3-41.

`Environment Resource Name` – The parameters mentioned as `ResourceParams` in NSJSP 5.0 are mentioned as attributes of the Resource definition in NSJSP 6.1.

`Connector` – The changes in NSJSP 6.1 are as follows:

- The `minProcessors` and `maxProcessors` properties are not included in NSJSP 6.1. These properties refer to the protocol processors and the minimum and the maximum number of processors that the connector needs to create. In NSJSP 6.1, the protocol processors are created only if required. After the protocol processors are created, the processors are cached internally and reused.
- The `acceptCount` property has no relevance in NSJSP 6.1 as the messages are obtained through the `$RECEIVE` file. Messages are queued in the `$RECEIVE` file and no internal queue for messages is maintained.
- The `minSpareThreads` and `maxSpareThreads` attributes are not relevant in NSJSP 6.1. Starting with the NSJSP 6.0 release, you can configure the connector to use an `Executor` element for threads. You can use the `minSpareThreads` and `maxSpareThreads` attributes in the `Executor` element. However, the default configuration does not use the `Executor` element. For more information about the `Executor`, see <http://tomcat.apache.org/tomcat-6.0-doc/config/executor.html>.
- In NSJSP 6.1, the default value of `enableLookups` is `false`, which is the same as in the NSJSP 5.0 configuration. Therefore, this property is not mentioned explicitly in the connector configuration.
- The `connectionTimeout` is not relevant in NSJSP 6.1 as it uses a custom `ServerSocket` class, which is extended from the `java.net.ServerSocket`, to create socket objects for messages received on `$RECEIVE`.

[Table 7-3](#) lists the differences in the NSJSP 6.0 and NSJSP 6.1 `server.xml` files.

Table 7-3. Differences in the NSJSP 6.0 and NSJSP 6.1 `server.xml` Files (page 1 of 2)

Element/Attribute	Default Value in NSJSP 6.0	Default Value in NSJSP 6.1
The GlobalNamingResources Element	<pre><Resource name="UserDatabase" auth="Container" type="org.apache.catalina .UserDatabase" description="User database that can be updated and saved" factory="org.apache.catal ina.users.MemoryUserDatab aseFactory" pathname="conf/nsjsp- users.xml" /></pre>	<pre><Resource name="UserDatabase" auth="Container" type="org.apache.catalin a.UserDatabase" description="User database that can be updated and saved" factory="org.apache.cata lina.users.MemoryUserDat abaseFactory" pathname="conf/nsjsp- users.xml" readonly="false" /></pre> <p>Note: In NSJSP 6.1, the memory user data base is read-only. If it is read-only, you cannot use the Admin Web application to add or modify users in the memory database. To enable the Admin Web application to add and modify the users, the read-only attribute is set to <code>false</code>.</p>

Table 7-3. Differences in the NSJSP 6.0 and NSJSP 6.1 `server.xml` Files (page 2 of 2)

Element/Attribute	Default Value in NSJSP 6.0	Default Value in NSJSP 6.1
Connector	<code>protocol="HTTP/1.1"</code> <code>connectionTimeout="0"</code> <code>acceptCount="25"</code> <code>maxThreads="75"</code> Note: The <code>acceptCount</code> property has no relevance in NSJSP 6.1 as the messages are obtained through the <code>\$RECEIVE</code> file. Messages are queued in the <code>\$RECEIVE</code> file and no internal queue for messages is maintained. The <code>connectionTimeout</code> is not relevant in NSJSP 6.1 as it uses a custom <code>ServerSocket</code> class, which is extended from the <code>java.net.ServerSocket</code> , to create socket objects for messages received on <code>\$RECEIVE</code> .	<code>protocol="HTTP/1.1"</code> <code>maxThreads="75"</code>
Listener within the Connector Engine	Not available	<code>com.tandem.servlet.JMXConnectionListener</code> <code>name="NSJSP"</code> <code>backgroundProcessorDelay="60"</code> <code>defaultHost="localhost"</code>
Host	<code>name="localhost"</code> <code>appBase="webapps"</code> <code>unpackWARs="false"</code> <code>autoDeploy="false"</code> <code>xmlValidation="false"</code> <code>xmlNamespaceAware="false"</code> <code>configClass="com.tandem.servlet.catalina.startup.NSJSPPContextConfig"</code>	<code>Host name="localhost"</code> <code>appBase="webapps"</code> <code>unpackWARs="true"</code> <code>autoDeploy="true"</code> <code>xmlValidation="false"</code> <code>xmlNamespaceAware="false"</code> <code>"</code> <code>configClass="com.tandem.servlet.catalina.startup.NSJSPPContextConfig"</code> For information on <code>autoDeploy</code> and <code>unpackWARs</code> , see The <code>server.xml</code> File on page 3-37. <code>Valve</code> <code>className="com.hp.tandem.nsjsp.valves.RequestTrackerValve"</code>

Note. `Valve className` is a new attribute introduced in NSJSP 6.1.

[Table 7-4](#) lists the differences in the NSJSP 5.0 and NSJSP 6.1 `servlet.config` files.

Table 7-4. Differences in the NSJSP 5.0 and NSJSP 6.1 `servlet.config` Files (page 1 of 2)

	NSJSP 5.0	NSJSP 6.1	Effect
Maxservers	5	4	For more information on the properties, see The Installation-Specific servlet.config File on page 3-7.
Numstatic	2	4	
Maxlinks	250	50	
Linkdepth	25	50	
TANDEM_RECEIVE _DEPTH	25	50	

Command-Line Arguments

Table 7-4. Differences in the NSJSP 5.0 and NSJSP 6.1 `servlet.config` Files (page 2 of 2)

NSJSP 5.0	NSJSP 6.1	Effect
–	Not applicable in NSJ 1.5 and NSJ 6.0	
Xbootclasspath /a:\$env(JAVA_HOME)/lib/tools.jar		
Not available	– Dcom.tandem.servlet.CONTEXT_PREFIXES=/urlB For more info, see Chapter 3, Configuring NSJSP .	
Filemap /servlet \$server_object code	Filemap /urlB \$server_object code	● Filemap /servlet was included in NSJSP 5.0 for backward compatibility with earlier versions. Versions earlier than NSJSP 5.0 used the /servlet URI to map requests to an NSJSP Server Class. However, it is not relevant in NSJSP 6.1.
Filemap /servlet_jsp \$server_object code		● In NSJSP 5.0, the URI was always <code>servlet_jsp</code> . In NSJSP 6.1, you can specify the URI during installation.

Note. The generic `servlet.config` file is introduced in the NSJSP 6.1 release. For more information on the `servlet.config` file, see [Chapter 3, Configuring NSJSP](#).

[Table 7-5](#) lists the differences in the NSJSP 6.0 and NSJSP 6.1 `servlet.config` files.

Table 7-5. Differences in the NSJSP 6.0 and NSJSP 6.1 `servlet.config` Files

	NSJSP 6.0	NSJSP 6.1	Effect
Maxservers	12	4	
Maxlinks	250	50	
Linkdepth	25	50	
TANDEM_RECEIVE_DEPTH	25	50	For more information on the properties, see The Installation-Specific <code>servlet.config</code> File on page 3-7.
Command-Line Arguments			
	-Xmx64m	-Xmx64m	
	-Xss128k	-Xss128k	
	-Xnoclassgc	-Xnoclassgc	
		-Xms64m	
		For more information on -Xms64m, see The Installation-Specific <code>servlet.config</code> File on page 3-7.	
	java.compiler= =none \	Not applicable. For more information on why the attribute is removed, see The Installation-Specific <code>servlet.config</code> File on page 3-7.	
	Filemap /servlet_jsp \$server_objec tcode	Filemap /urlB \$server_objec tcode	In NSJSP 6.0, the URI was always <code>servlet_jsp</code> . In NSJSP 6.1, you can specify the URI during installation.

Difference in the NSJSP 6.0 and NSJSP 6.1 `web.xml` files

The `web.xml` file in NSJSP 6.0 and NSJSP 6.1 are similar. However, the NSJSP 6.1 `web.xml` file contains a new filter, called `StaticContentFilter`.

Note. NSJSP 5.0 does not include the `web.xml` file.

Difference in the NSJSP 6.0 and NSJSP 6.1 `conf/context.xml` files

The `conf/context.xml` file in NSJSP 6.0 and NSJSP 6.1 are similar. However, the NSJSP 6.1 `conf/context.xml` file includes a sample configuration of the persistent session manager.

Note. NSJSP 5.0 does not include the `conf/context.xml` file. However, in NSJSP 5.0, contexts were handled differently. For more information on contexts, [Context Definition](#) on page 7-20.

Note. If you have customized any container configuration file in NSJSP 5.0 or NSJSP 6.0, you might want to change the corresponding container configuration file in NSJSP 6.1.

Comparing Management Properties in NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1

[Table 7-6](#) compares the management properties in NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1.

Table 7-6. Differences and Similarities in Management Properties of NSJSP Versions

Factors	NSJSP 5.0	NSJSP 6.0	NSJSP 6.1
Server classes related to management in NSJSP	NSJSPADMIN	Same as NSJSP 5.0.	NSJSPADMIN and MANAGER Each installation of NSJSP contains the NSJSPADMIN server class and only one MANAGER server class in an iTP Secure WebServer environment.
Management application	Admin Web and Manager Web	Same as NSJSP 5.0	Admin Web, Manager Web, and NSJSP Manager The Admin Web and Manager Web applications are carried forward from the previous NSJSP versions.
Support for deploying web applications from a remote desktop using the command-line interface (CLI)	Yes Ant scripts are used to deploy web applications	Same as NSJSP 5.0	Same as NSJSP 6.0

HP recommends that you use the NSJSP manager application that provides many new features. For a comparison of the different manager applications provided by NSJSP 6.1, see Chapter [4, Managing NSJSP](#)

Comparing Logging Infrastructure in NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1

[Table 7-7](#) compares the logging properties in NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1.

Table 7-7. Differences and Similarities in NSJSP Versions

Factors	NSJSP 5.0	NSJSP 6.0	NSJSP 6.1
How is Logging achieved?	Uses <code><Logger></code> elements to configure application-specific logging.	Uses the JULI logging framework with log rollover feature. Starting from NSJSP 6.0, the <code><Logger></code> elements are not supported.	Same as in NSJSP 6.0
Default directory location of log files	<code><iTP_WebServer_HOME>/logs</code>	<code><NSJSP_HOME>/logs</code>	Same as in NSJSP 6.0
Location where logging can be configured for the servlet container components	<code>iTP_server.xml</code> file in the <code><NSJSP_HOME>/conf</code> directory	<code>logging.properties</code> in the <code><NSJSP_HOME>/conf</code> directory	Same as in NSJSP 6.0
Location where logging can be configured for user applications	<code>META-INF/context.xml</code>	<code>Application-specific logging.properties</code>	Same as in NSJSP 6.0
Classes required for Programming	<ul style="list-style-type: none"> ● <code>org.apache.commons.logging.LogFactory</code> ● <code>org.apache.commons.logging.Log</code> 	<ul style="list-style-type: none"> ● <code>org.apache.juli.logging.LogFactory</code> ● <code>org.apache.juli.logging.Log</code> 	Same as in NSJSP 6.0

Logging Configuration of Servlet Container Components

In addition to the comparison related to logging in [Table 7-7](#), this section discusses the details of the logging configuration of servlet container components in NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1.

Logging Configuration in NSJSP 5.0

In NSJSP 5.0, logging is configured in the `iTP_server.xml` file. The `iTP_server.xml` file includes default configurations for the Engine and the localhost. The default configuration of the Engine component is as follows:

```
<Logger
className="com.tandem.servlet.catalina.logger.NSJSPFileLogger"
prefix="NSJSP_Catalina." suffix=".log" timestamp="true"/>
```

This configuration creates a log file called `NSJSP_catalina.<date>.log` in the `<NSJSP_HOME>/logs` folder.

Note. In the default configuration of NSJSP 5.0, this file is not created as the Logger definition of the Engine is overridden by the Logger configuration in `localhost`, which is one of the child elements of Engine.

The default configuration of the `localhost` is as follows:

```
<Logger
className="com.tandem.servlet.catalina.logger.NSJSPFileLogger"
directory="logs" prefix="localhost." suffix=".log"
timestamp="true"/>
```

According to the Logger configuration, the `localhost.<date>.log` file is created in the `<NSJSP_HOME>/logs` folder. Log messages related to the Engine and all its child elements are logged to the `localhost.<date>.log` file.

In the default configuration of NSJSP 5.0, log messages generated by components other than Engine and its child components are written in the `STDOUT` file, which is the `servlet.log` file, in the `<iTP WebServer Home>/logs` folder.

Note. The Servlet Server Class includes the `STDOUT` file, which is defined as the `servlet.log` and the `STDERR` file, which is defined as `servlet_error.log`. Both these files are available in the `<iTP WebServer Home>/logs` folder. The NSJSPADMIN Server Class includes both `STDOUT` and `STDERR`, which are defined as `nsjspadmin.log` file in the `<iTP WebServer Home>/logs` folder.

Note. The Admin Web application, Manager Web application, and the Host-Manager do not include a default logging configuration.

Logging Configuration in NSJSP 6.0

In NSJSP 6.0, all the NSJSP servlet container-specific logs are published in the `STDOUT` file, which is `servlet.log` by default, and all the logs related to Admin, Manager and Host-Manager are published in their respective log files.

Logging is configured in the `logging.properties` file. There is no explicit configuration for the Engine and the Host elements in the `logging.properties` file. As a result, logging is handled by the root logger. The root logger defines an `NSJSPLogHandler` that logs all messages to the `STDOUT` file. The `STDOUT` file is configured as the `servlet.out` file in the `<NSJSP_HOME>/logs` directory. The following is the root logger configuration:

```
.handlers = com.tandem.servlet.logging.NSJSPLogHandler
```

NSJSP 6.0 includes the following default configuration for the Admin Web application:

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/ser
vlet_jsp/admin].level = INFO
```

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/ser
vlet_jsp/admin].handlers = 4admin.org.apache.juli.FileHandler
```

According to the `4admin.org.apache.juli.FileHandler` configuration, all the Admin-related logs are published in the `admin.<date>.log` file in the `<NSJSP_Home>/logs` folder.

NSJSP 6.0 includes the following default configuration for the Manager:

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/ser
vlet_jsp/manager].level = INFO
```

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/ser
vlet_jsp/manager].handlers =
3manager.org.apache.juli.FileHandler
```

According to the `3manager.org.apache.juli.FileHandler` configuration, all the manager-related logs are published in the `manager.<date>.log` file in the `<NSJSP_Home>/logs` folder.

NSJSP 6.0 includes the following default configuration for the Host-Manager:

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/ser
vlet_jsp/host-manager].level = INFO
```

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].[/ser
vlet_jsp/host-manager].handlers = 5host-
manager.org.apache.juli.FileHandler
```

According to the `5host-manager.org.apache.juli.FileHandler` configuration, all the Host-Manager logs are published in the `host-manager.<date>.log` file in the `<NSJSP_Home>/logs` folder.

Logging Configuration in NSJSP 6.1

In NSJSP 6.1, all logs related to the NSJSP servlet container are published in the `<server class name>.<date>.log` file. The logs related to the Admin, Manager, and the Host-Manager are also published in the same file.

Logging is configured in the `logging.properties` file.

The default configuration of the localhost is as follows:

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].level
= INFO
```

```
org.apache.catalina.core.ContainerBase.[NSJSP].[localhost].handl
ers = 1nsjsp.com.tandem.servlet.logging.NSJSPLogHandler
```

All logs related to the localhost are created in `<NSJSP_HOME>/logs/<NSJSP Server Class name>.<date>.log`. For example, logs related to the localhost are created in `<NSJSP_HOME>/logs/SC1.2010-05-03.log`.

There is no explicit logging definition for the Engine component. Because the root logger is associated with `lnsjsp.com.tandem.servlet.logging.NSJSPLogHandler`, the root logger publishes all messages to the same log file as the localhost. The following is the root logger configuration:

```
.handlers = lnjsjp.com.tandem.servlet.logging.NSJSPLogHandler
```

For example, the logs are created in `<NSJSP_HOME>/logs/SC1.2010-05-03.log`.

The logging configuration for the Admin, Manager, and the Host-Manager use the same handler as the logger for localhost and root logger. As a result, all log messages related to these web applications are published in the same file as the container logs, that is, `<server class name>.date.log`.

For more information on log files and default handler, see Chapter [5, Logging in NSJSP](#).

Comparing Miscellaneous Properties

[Table 7-8](#) compares the miscellaneous properties of NSJSP 5.0, NSJSP 6.0, and NSJSP 6.1.

Table 7-8. Miscellaneous Properties of NSJSP Versions

Factors	NSJSP 5.0	NSJSP 6.0	NSJSP 6.1
Supported specifications	Java Servlets 2.4	Java Servlets 2.5	Java Servlets 2.5
	JavaServer Pages (JSP) 2.0	JavaServer Pages (JSP) 2.1	JavaServer Pages (JSP) 2.1

Considerations for Migrating Web Applications from NSJSP 5.0 to NSJSP 6.1

[Table 7-9](#) lists the prerequisites of NSJSP 5.0 and NSJSP 6.1.

Table 7-9. Prerequisites of NSJSP 5.0 and NSJSP 6.1

Operating System/Product	NSJSP 5.0	NSJSP 6.1
NonStop operating system	H06.03 or later	J06.04 or later J-series, or H06.15 or later H-series
iTP Secure WebServer	6.0 SPR ACA or later	iTP Secure Webserver 7.0 (T8996H02 or T8997H02) or later
NonStop Server for Java (NSJ)	4.2 or later	NSJ 5.1 (SPR ABS or later of T2766H51) or NSJ 6.0 (SPR ABP or later of T2766H60)
JDBC/MX	Not applicable	T2 or T4 driver

The NSJSP 6.1 web applications are available in the `<iTP WebServer Home>/<server class>/webapps` directory. The NSJSP 5.0 or NSJSP 6.0 web applications are located in the `<iTP Installation Directory>/servlet_jsp/webapps` directory.

The following are the considerations while migrating web applications from NSJSP 5.0 to NSJSP 6.1:

- [Default Context](#)
- [Context Definition](#)
- [Session Manager Configuration](#)
- [Shared Application Resources](#)
- [Container-Specific Resources](#)
- [Application Modifications](#)
- [Compiling the Application](#)
- [Session Store](#)
- [Deploying Web Applications in NSJSP 6.1](#)

Default Context

Default context is the context definition of applications that do not define the context explicitly. You can define the context of applications in the `META-INF/context.xml` file of the application.

In NSJSP 5.0, the default context definition for applications in a host is defined using the `DefaultContext` child element of either the `Engine` or the `Host`, in the `itp_server.xml` file.

In NSJSP 6.1, the default context for applications hosted in the entire NSJSP servlet container is defined in the `conf/context.xml` file.

Note. The default configuration does not define any `DefaultContext`.

Context Definition

The context definition of NSJSP 5.0 and NSJSP 6.1 are similar. Most attributes included in NSJSP 5.0 are valid in NSJSP 6.1. [Table 7-10](#) lists the attributes that were included in NSJSP 5.0, but are not applicable in NSJSP 6.1.

Table 7-10. Context Definition Attributes in NSJSP 5.0 and NSJSP 6.1

Attribute	NSJSP 5.0	NSJSP 6.1
managerChecksFrequency	Frequency of session expiry and related manager operations.	Not Applicable This is replaced by the processExpiresFrequency attribute of the Manager element.
ResourceParams	Used to define the configuration parameters for the resource specified in the <Resource> definition. For example, <pre><Resource name="jdbc/EmployeeDB" auth="Container" type="javax.sql.DataSource" description="Employees Database for HR Applications"/></pre> <pre><ResourceParams name="jdbc/EmployeeDB"> <parameter> <name>driverClassName</name> <value>org.hsql.jdbcDriver</value> </parameter> <parameter> <name>driverName</name> <value>jdbc:HypersonicSQL:database</value> </parameter></pre>	Not Applicable Resource parameters are configured as attributes of the resource in the <Resource> definition. For example, <pre><Resource name="jdbc/EmployeeDB" auth="Container" type="javax.sql.DataSource" description="Employees Database for HR Applications" driverClassName=" org.hsql.jdbcDriver" driverName=" jdbc:HypersonicSQL:database"/></pre>

Session Manager Configuration

In NSJSP 5.0, you can configure the session manager either in the Default Context or in the application-specific context definition, in the `META-INF/context.xml` file. In NSJSP 6.1, you can configure the Manager either in the `conf/context.xml`, which is the equivalent of Default Context, or in the application-specific context in the `META-INF/context.xml` file of the application.

[Table 7-11](#) compares the attributes of Manager that are either not relevant or have a different meaning in NSJSP 6.1.

Table 7-11. Session Manager Configuration

Attribute	NSJSP 5.0	NSJSP 6.1
<code>checkInterval</code>	Valid for both in-memory manager and the persistent manager.	Valid only in the persistent manager. In the standard manager, the sessions expire during the background process checks.

Shared Application Resources

In NSJSP 5.0, the `<NSJSP_HOME>/shared` folder can contain application resources, such as, class files and `jar` files that are shared by all web applications. By default, NSJSP 6.1 does not have a shared folder in the default installation. However, you can create a folder and place all resources common to all web applications in that folder. If you create the shared folder, the `conf/catalina.properties` file must be modified to indicate the location of the shared folder.

The following is an example of an entry in the properties file:

```
shared.loader=${catalina.home}/shared,${catalina.home}/shared/*.jar
```

HP recommends that all application-specific resources be bundled with the application and not be placed in the shared folder. This is to ensure that the application resources are independent of the container.

Container-Specific Resources

In NSJSP 5.0, all the servlet container-specific `jar` files were included in the `server` folder. The `jar` files common to web applications and the container were included in the `common/lib` folder. NSJSP 6.1 includes only the `<NSJSP_HOME>/lib` folder. The `<NSJSP_HOME>/lib` folder includes the common and the container-specific libraries.

In NSJSP 5.0, the Admin Web and the Manager Web applications were included in the `server/webapps` folder. In NSJSP 6.1, the Admin Web and the Manager Web applications are included in the `webapps` folder of the installation.

Although there is a difference in the directory location of the files and folders between NSJSP 5.0 and NSJSP 6.1, no changes are required for these web applications during

migration. Also, the Admin Web and Manager Web applications provide the same functionality in NSJSP 5.0 and in NSJSP 6.1. As a result, there is no need to migrate these applications from NSJSP 5.0 to NSJSP 6.1.

Application Modifications

This section discusses the modifications related to applications that you must consider while migrating from NSJSP 5.0 to NSJSP 6.1.

The Logger element

The following are the differences between NSJSP 5.0 and NSJSP 6.1:

- Logging in NSJSP 5.0 was configured using the Logger element. If the application that you want to migrate to NSJSP 6.1 has a context explicitly defined in the `META-INF/context.xml` file, and if the context definition includes a Logger element, you must remove the Logger element because NSJSP 6.1 does not support the Logger element. Logging in NSJSP 6.1 is handled by the JULI framework. If the application that you want to migrate to NSJSP 6.1 contains a Logger element, you must explicitly define logging for the application.
- In NSJSP 6.1, you can define application-specific logging using the `logging.properties` file. Equivalent attributes of the directory and the prefix attributes of the Logger element are available in the Handler configuration in JULI. The suffix attribute of the Logger has no equivalent in JULI.
- The Logger element offers the timestamp attribute option to specify the date and time at which the log message is written. In JULI, the default behavior is to log all messages with a timestamp.
- The format in which the messages are logged can be configured using the format attribute of the Handler. For more information on the Handler component and how to configure Handlers, see Chapter [5, Logging in NSJSP](#).

The Logging API

If the application uses the `log` method of the `javax.servlet.ServletContext` interface to log application messages, you need not make any changes to the application with respect to the Logging API.

If the application uses a custom logging implementation, such as Log4J, you need not make any changes to the application.

NSJSP 5.0 is bundled with the Apache Commons Logging (`commons-logging-api.jar` is located in the `<NSJSP_HOME>/bin` folder). All the internal classes of NSJSP use the `LogFactory` and `Log` classes of Commons Logging. In NSJSP 6.1, they are replaced by the `LogFactory` and the `Log` classes in the JULI framework. Although similar to the NSJSP 5.0 classes, the NSJSP 6.1 `LogFactory` and `Log` classes are bundled in a different Java package. If the application uses these classes, you must change the package definition from `org.apache.commons.logging` to `org.apache.juli.logging`.

Compiling the Application

NSJSP 5.0 implements the Java Servlets 2.4 specifications whereas NSJSP 6.1 implements the Java Servlet 2.5 specifications. You must compile the application with the NSJSP 6.1 library `jar` files to ensure that the Java Servlet API used in the application is supported in NSJSP 6.1. The library files are located in the `lib` folder of the installation.

Session Store

If the application uses HTTP sessions and the application or the NSJSP servlet container is configured for persistent sessions, you must migrate the session store along with the application.

For information on migrating the session store, see [Migrating the Session Store](#) on page 7-27.

Note. In NSJSP 5.0, the persistent manager in the application's `context.xml` file enables you to configure the application for persistent sessions.

For more information on persistent sessions, see Chapter [3, Configuring NSJSP](#).

Deploying Web Applications in NSJSP 6.1

After making the changes to the application, you must deploy the application in NSJSP 6.1. Although you can manually deploy an application in NSJSP 6.1 by copying the application resources, such as the `.war` file, to an appropriate location, such as the `webapps` folder in NSJSP 6.1, HP recommends that you deploy the application in NSJSP 6.1 using the NSJSP Manager.

For information on deploying web applications using the NSJSP Manager, see Chapter [4, Managing NSJSP](#).

You can use the following methods to migrate applications from NSJSP 5.0 to NSJSP 6.1:

Application Deployed Using a `context.xml` File

If an application is deployed using its context definition file, HP recommends that you deploy the application using the NSJSP Manager. An application is deployed using the `context.xml`, as follows:

Unlike in NSJSP 5.0, the `path` attribute of the context element in the context definition is ignored in NSJSP 6.1. The `path` is derived from the name of the context definition file. For example, if the name of the context definition file is `foo.xml`, the `path` of the context is `/foo`.

The NSJSP Manager provides an option to deploy an application using its context definition file. In this case, you can mention the context path and the manager derives the name of the context definition file from the path specified. For example, if the value of `path` is `/foo` and the full path to context definition file is

`/usr/webmstr/webapps/myapp.xml`, the NSJSP Manager copies the `myapp.xml` file to the appropriate location in NSJSP 6.1 with the name `foo.xml` and then deploys the application based on the contents of `foo.xml`. The directory where the context definition file is copied will always be `<NSJSP_HOME>/conf/<engine_name>/<host_name>`. In this example, `foo.xml` is copied to the above location.

Note. In NSJSP 5.0, the path defined in the context definition file is not used. Also, the path is not derived from the name of the file.

Application Deployed Using a .war File

Deploying a `.war` file can be achieved without using the NSJSP Manager. Copy the `.war` file to the folder defined by the `appBase` of the Host element in which the application must be deployed. In the default configuration, `appBase` of the only Host element is the `webapps` folder in the `<NSJSP_HOME>` directory.

A `.war` file can also be deployed using the NSJSP Manager. You can also provide the context path.

NSJSP 6.1 handles the context definition (`META-INF/context.xml`) differently than NSJSP 5.0. NSJSP 6.1, extracts the `context.xml` file from the `.war` file and copies the `context.xml` to the `<NSJSP_HOME>/conf/<engine_name>/<host_name>` directory with the name `<context_name>.xml`. For example, if `foo.war` is deployed and contains a `META-INF/context.xml` file, NSJSP 6.1 copies the `context.xml` file to `foo.xml` to the above directory. With subsequent restarts of NSJSP, this application will be deployed from the context definition file rather from the `.war` file. You can also modify the context during run time using the Admin application.

Application Deployed Using a Directory

Copy the entire application directory to the folder specified in `appBase` of the Host element. In the default configuration, `appBase` of the only Host element is the `webapps` folder in the `<NSJSP_HOME>` directory.

An application directory can also be deployed using the NSJSP Manager. The NSJSP Manager offers the additional flexibility of providing the context path.

NSJSP 6.1 handles the context definition (`META-INF/context.xml`) differently than NSJSP 5.0. NSJSP 6.1 copies the `META-INF/context.xml` file of the application and copies the `context.xml` in the

`<NSJSP_HOME>/conf/<engine_name>/<host_name>` directory with the name `<context_name>.xml`. For example, if `foo.war` is deployed and contains a `META-INF/context.xml` file, NSJSP 6.1 copies the `context.xml` file to `foo.xml` in the specified directory. With subsequent restarts of NSJSP, this application will be deployed from the context definition file rather from the application directory. You can also modify the context during run time using the Admin application.

Considerations for Migrating Web Applications from NSJSP 6.0 to NSJSP 6.1

[Table 7-12](#) lists the prerequisites of NSJSP 6.0 and NSJSP 6.1.

Table 7-12. Prerequisites of NSJSP 6.0 and NSJSP 6.1

Operating System/Product	NSJSP 6.0	NSJSP 6.1
NonStop operating system	H06.04 or later	J06.04 or later J-series, or H06.15 or later H-series
iTP Secure WebServer	6.0 SPR ACA or later	iTP Secure Webserver 7.0 (T8996H02 or T8997H02) or later
NonStop Server for Java (NSJ)	5.0 or later	NSJ 5.1 (SPR ABS or later of T2766H51) or NSJ 6.0 (SPR ABP or later of T2766H60)
JDBC/MX	T2 driver for SQL/MX or SQL/MP	T2 or T4 driver

You need not make any changes to the web applications to migrate to NSJSP 6.1. However, in the container, there are some changes in the application deployment behavior with respect to the context definitions.

If an application defines its own context definition in `META-INF/context.xml` of the application, NSJSP 6.1 copies the `context.xml` to `<NSJSP_HOME>/conf/<engine_name>/<host_name>` with the name `<context name>.xml`. For example, if `foo.war` is deployed and if the `.war` files contain `META-INF/context.xml`, this file is extracted from the `.war` and is then copied as `foo.xml`. During subsequent restarts, the context for this application is read from `foo.xml` rather than from the `META-INF/context.xml` of the `.war` file. This behavior allows the context to be dynamically modified and persisted using the Admin Web application.

Although applications can be deployed by copying the application resources, such as, `.war` file or the application directory, to the `webapps` folder of the NSJSP 6.1 installation, HP recommends that you deploy the applications using the NSJSP Manager as it enables you to specify the context name of the application that you are deploying.

For information on the default files where logs are written, see [Logging Configuration of Servlet Container Components](#) on page 7-15.

Migrating the Session Store

The session store configured in NSJSP 5.0 does not work in NSJSP 6.1, because there are differences in the field sizes in the table definition of the JDBC store. However, behavior of the session store in NSJSP 6.0 and NSJSP 6.1 is the same.

To migrate sessions from NSJSP 5.0 or NSJSP 6.0 to NSJSP 6.1, complete the following steps:

1. Run one of the following commands to create the NSJSP 6.1 session store:
 - `NSJSP_HOME/conf/nsjsp_createSessionStore_mp.sql` to create the persistent session store for SQL/MP.
 - `NSJSP_HOME/conf/nsjsp_createSessionStore_mx.sql` to create the persistent session store for SQL/MX.
2. Run the `NSJSP_HOME/conf/nsjsp_migrateSessionStore` script to migrate data from NSJSP 5.0 or NSJSP 6.0 to NSJSP 6.1.

The `nsjsp_migrateSessionStore` script migrates the data from SQL/MP to SQL/MX or SQL/MP to SQL/MP.

Note. Although NSJSP 6.1 implements the servlet 2.5 and JSP 2.1 specifications, it supports the servlet 2.4 specification.

In NSJSP 6.1, the Manager and Admin Web applications have the contexts `/<deployment_name>/manager` and `/<deployment_name>/admin` respectively, unlike `/manager` and `/admin` in NSJSP 5.0.

Migrating to NSJSP Manager Application in NSJSP 6.1

Starting with the NSJSP 6.1 release, a new application called the NSJSP Manager is introduced. HP recommends that you use this application to manage web applications, server classes, and MBeans. For more information on this application, see Chapter [4, Managing NSJSP](#).

Support for Multiple NSJSP Installations in a Single iTP Secure WebServer Environment

The NSJSP installation script is enhanced to support multiple NSJSP installations in a single iTP Secure WebServer environment. The following enhancements were made to the installation script to support multiple NSJSP installations:

- Directory locations of NSJSP installations

In releases prior to NSJSP 6.1, NSJSP is installed in the default path (*<iTP Installation Directory>/servlet_jsp*). This default NSJSP location is derived from the location of the iTP Secure WebServer, and it cannot be modified. A directory location can include only one NSJSP installation. As a result, you can have only one NSJSP installation in an iTP Secure WebServer environment. Starting with the NSJSP 6.1 release, you can install NSJSP 6.1 in a directory location of your choice. Besides, you can have multiple NSJSP 6.1 installations in the same iTP Secure WebServer environment. However, each installation must be present in a unique directory location and be assigned a unique name.

- Directory location of configuration files related to the iTP Secure WebServer

In releases prior to NSJSP 6.1, during the NSJSP installation, the following iTP Secure WebServer related configuration files were created in *<iTP Installation Directory>/conf*:

- jdbc.config
- nsjspadmin.config
- servlet.config
- filemaps.config

Starting with the NSJSP 6.1 release, each NSJSP 6.1 installation includes its own set of configuration files. The configuration files are no longer present in the iTP Secure WebServer location except for the generic *servlet.config* file. Instead, the files are present in the same location as the respective NSJSP installation. The location of these NSJSP 6.1 installation-specific configuration files is *<NSJSP 6.1 Installation Directory>/conf*.

The following sample command displays the iTP Secure WebServer-related configuration files:

```
<NSJSP 6.1 Installation Directory>/conf: ls
filemaps.config      jdbc.config    nsjspadmin.config
servlet.config
```

When NSJSP 6.0 is installed in the iTP Secure WebServer environment, the NSJSP 6.0 configuration files, including *servlet.config*, are located in the *<iTP Installation Directory>/conf* directory. During the installation of NSJSP 6.1 in this environment, a generic *servlet.config* file is created in the location of *<iTP Installation Directory>/conf* directory. This file is generic to all the NSJSP installations in the environment, and is different from the instance-specific *servlet.config* files located in the NSJSP instances. The generic *servlet.config* file links the iTP Secure WebServer with all the instance-specific *servlet.config* files. Additionally, it includes the locations of the associated NSJSP installations. The existing *servlet.config* that includes the configuration details of NSJSP 6.0 is modified and renamed.

Example

This example shows how the new generic `servlet.config` file is created and how the existing `servlet.config` that includes NSJSP 6.0 specific information is renamed:

[Figure 7-1](#) shows the contents of the sample *<iTP Installation Directory>/conf* directory contents after installing NSJSP 6.0.

Figure 7-1. Sample conf Directory Created After NSJSP 6.0 Installation

```
<ITPWS_INSTALL_HOME>/conf: ls
filemaps.config          rollstarth.sample
httpd.config             sampleservers.config
httpd.config.sample      sampleservers.config.sample
jdbc.config              servlet.config
mime-types.config        start
mime-types.config.sample start.sample
nsjspadmin.config        stop
restart                  stop.sample
restart.sample           trace
restarth                 trace.sample
restarth.sample          trcon
rollover                 trcon.sample
rollover.sample          vcache
rollstarth               vcache.sample
|
```

The `servlet.config` file includes the configuration information for NSJSP 6.0.

[Figure 7-2](#) shows the contents of the sample *<iTP Installation Directory>/conf* directory after installing NSJSP 6.1.

Figure 7-2. Sample conf Directory Contents After NSJSP 6.1 Installation

```
<ITPWS_INSTALL_HOME>/conf: ls
filemaps.config          sampleservers.config
httpd.config             sampleservers.config.sample
httpd.config.sample      servlet-6.0.13.config
jdbc.config              servlet.config
mime-types.config        servlet.config.2009-11-05:12.02.42
mime-types.config.sample start
nsjspadmin.config        start.sample
restart                  stop
restart.sample           stop.sample
restarth                 trace
restarth.sample          trace.sample
rollover                 trcon
rollover.sample          trcon.sample
rollstarth               vcache
rollstarth.sample        vcache.sample
```

A new `servlet.config` file is created that is generic to NSJSP 6.0 and NSJSP 6.1. The existing NSJSP 6.0 `servlet.config` file containing the NSJSP 6.0

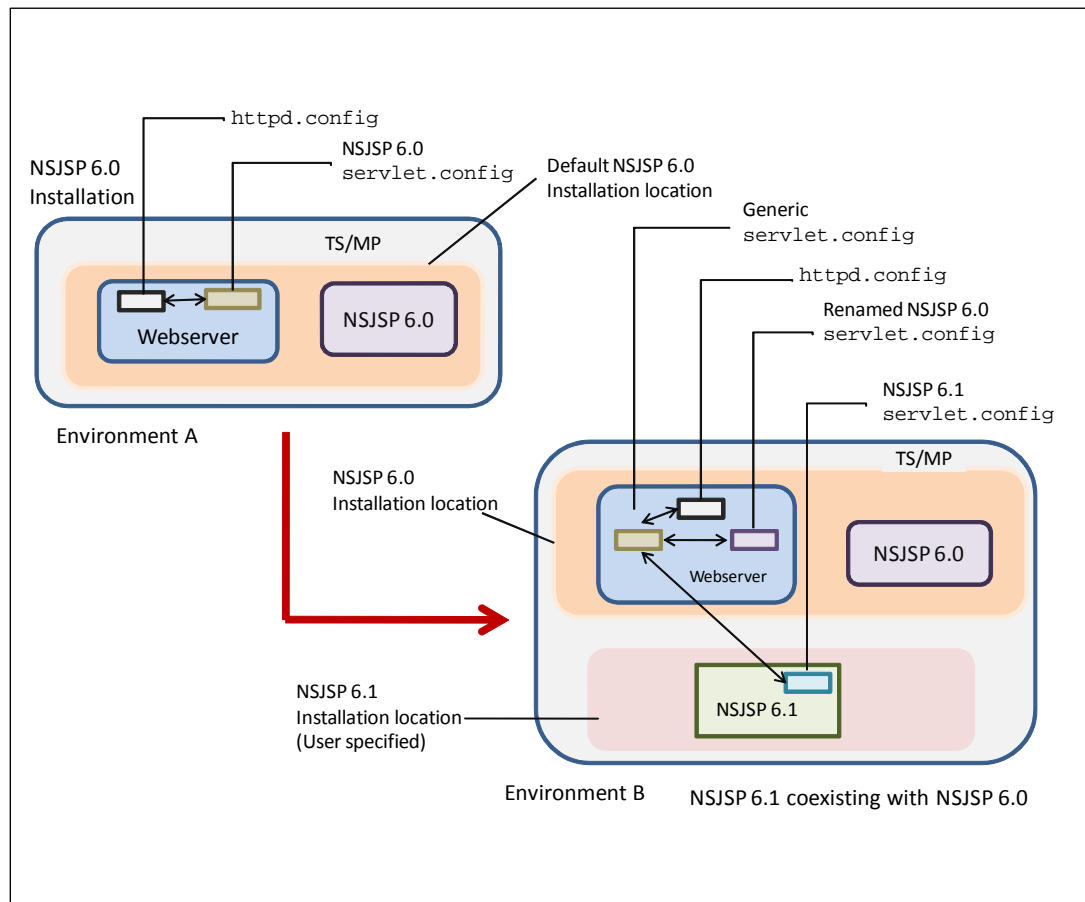
configuration details is modified and renamed to `servlet-6.0.13.config`. Additionally, the existing NSJSP 6.0 `servlet.config` file is backed up in a file, called `servlet.config.2009-11-05:12.02.42`.

The sequence of events is as follows:

1. The NSJSP 6.0 `servlet.config` file is modified and renamed to `servlet-6.0.13.config`, and it continues to exist in the same location.
2. A new file, called `servlet.config`, is created in the location of *<iTP Installation Directory>/conf* directory. The newly created `servlet.config` file is generic to NSJSP 6.0 and NSJSP 6.1.
3. The NSJSP 6.1 instance-specific `servlet.config` file is created in the *<NSJSP 6.1 Installation Directory>/conf* directory.
4. The generic `servlet.config` available in the *<iTP Installation Directory>/conf* directory is linked to the NSJSP 6.0 and NSJSP 6.1 `servlet.config` files.
5. The complete locations of the NSJSP 6.0 and NSJSP 6.1 deployments are listed in the generic `servlet.config` file.

After successful installation, NSJSP 6.0 and NSJSP 6.1 operate with the iTP Secure WebServer. However, the new NSJSP Manager Web application cannot manage the NSJSP 6.0 installation but they can co-exist in the iTP Secure WebServer environment.

[Figure 7-3](#) shows the installation locations of NSJSP 6.0 and NSJSP 6.1, and how the `servlet.config` files are linked.

Figure 7-3. NSJSP 6.0 and NSJSP 6.1 Installation Locations

In [Figure 7-3](#), Environment A represents an iTP Secure WebServer environment that includes an NSJSP 6.0 installation. Both NSJSP 6.0 and the iTP Secure WebServer exist in the same location. The NSJSP 6.0 `servlet.config` file is present in the iTP Secure WebServer, and is linked to the `httpd.config` file.

When you install NSJSP 6.1, Environment A transforms to Environment B. While installing NSJSP 6.1, you must specify a new path. Consequently, NSJSP 6.1 is installed in the specified path, and it includes the NSJSP 6.1 `servlet.config` file. Additionally, the existing NSJSP 6.0 `servlet.config` file in the iTP Secure WebServer is renamed, and a new generic `servlet.config` file is created. The generic `servlet.config` file is linked to the renamed NSJSP 6.0 `servlet.config` file, the NSJSP 6.1 `servlet.config` file, and the `httpd.config` file.

Note. Multiple NSJSP installations are supported only if NSJSP 6.1 is installed after installing NSJSP 6.0. If you attempt to install NSJSP 6.0 in an iTP Secure WebServer environment that already includes NSJSP 6.1, NSJSP 6.0 overwrites NSJSP 6.1.

An NSJSP 6.1 installation does not support NSJSP 5.0 installation and supports only NSJSP 6.0 installation. Therefore, you must refrain from installing NSJSP 6.1 with NSJSP 5.0.

This chapter discusses security considerations to secure data transfer from a web browser to the web server. The process includes validating a user, verifying whether a user has access to a particular web resource, and preventing malicious code from disrupting the NSJSP servlet container.

This chapter includes the following topics:

- [Securing Web Applications](#)
- [Establishing a Secure Link](#)
- [Authenticating a User](#)
- [Authorizing a User](#)
- [Validating the Sender](#)
- [Java Security Manager](#)
- [Manager Web Application and NSJSP Manager Security](#)

Securing Web Applications

When data flows between a web browser and a web server, and the link established between the web browser and the web server is not secure, the link is susceptible to attacks that can lead to data theft. Configuring a secure link between the web browser and the web server ensures that the data flow is encrypted and provides authentication mechanisms using certificates. For more information about the secure link that can be established between a web browser and a web server, see [Establishing a Secure Link](#) on page 8-2.

When the link between the web browser and the web server is secured, the next level of security is to authenticate users who access the web applications. For more information about various methods that NSJSP uses to authenticate a user, see [Authenticating a User](#) on page 8-3.

When the user is authenticated, a web application performs various checks to ensure that the user is authorized to access the requested web resource. For more information about authorization, see [Authorizing a User](#) on page 8-29.

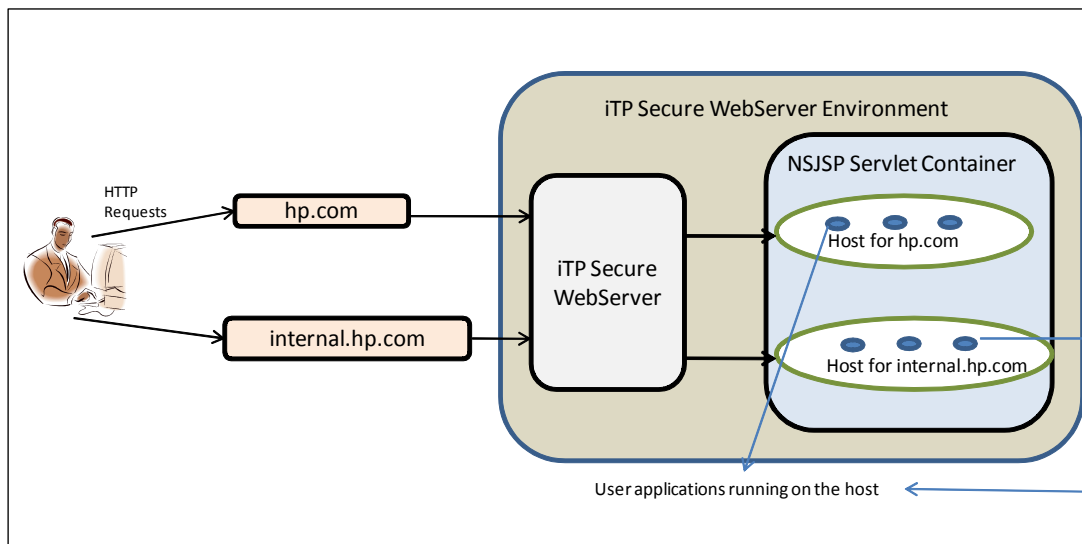
NSJSP provides certain security features to prevent malicious or erroneous code (such as invoking `system.exit()` in a JSP) from affecting the NSJSP container. For more information about how the security manager can prevent such malicious code from being executed using the Java Security manager, see [Java Security Manager](#) on page 8-35.

In addition to these various methods of securing web applications, this chapter also discusses other security features that can filter (allow or prevent) requests originating from a specific host or with a specified URL pattern. For more information on these security features, see [Validating the Sender](#) on page 8-33.

This chapter also discusses how security can be implemented in Web applications (Admin Web, Manager Web, and NSJSP Manager Web applications) that are used for administration and monitoring purposes. For more information on securing these Web applications, see [Manager Web Application and NSJSP Manager Security](#) on page 8-41.

[Figure 8-1](#) illustrates the security considerations discussed in this section.

Figure 8-1. Flow of User Request



Establishing a Secure Link

A secure link is required between the web browser and the web server to ensure that a secure channel is created for information exchange and to ensure protection against eavesdropping.

Secure Sockets Layer (SSL) is a protocol that is used to establish a secure and encrypted link between two nodes in a network so that data passed between the nodes is secure. In HP NonStop servers, the SSL standard is implemented by the iTP Secure WebServer. For information on how to configure iTP Secure WebServer for secure transport using SSL, see the *iTP Secure Webserver System Administrator's Guide*.

You need not perform any additional configuration tasks in NSJSP to enable NSJSP to handle secure requests. The iTP Secure WebServer acts as the front-end WebServer for NSJSP. NSJSP does not handle the SSL protocol but obtains sufficient information from the iTP Secure WebServer about the request delivered over the SSL protocol. The request can contain certificates (public key certificates) that NSJSP can use to authenticate. For example, based on the web application configuration, NSJSP can authenticate the user using the X.509 certificate that is delivered over the secure link.

Authenticating a User

The process of authentication involves obtaining user credentials and validating them against a database of user credentials. In NSJSP, this database is called a Realm. A web application need not always authenticate a user using a password. The user can also be authenticated using a certificate presented by the user. Because iTP Secure WebServer supports X.509 certificates, a user can also be authenticated in NSJSP using the X.509 certificates. A web application is configured to specify the method of obtaining the user credentials. When user credentials are obtained, the user is authenticated against the credentials stored in the Realm.

This section discusses the various configuration alternatives for obtaining user credentials. The process to validate the user credentials is discussed in [Realms](#) on page 8-7.

The following configurations can be used to obtain user credentials:

- [HTTP Basic Authentication](#)
- [HTTP Digest Authentication](#)
- [Form-Based Authentication](#)
- [HTTPS Client Authentication](#)

HTTP Basic Authentication

HTTP basic authentication is the authentication mechanism defined in the HTTP/1.0 specification. When a user tries to access a secured resource, NSJSP requests the web browser to obtain the username and password. The web client obtains the username and the password from the user and sends them back. NSJSP then authenticates the user and if the user is authorized to access the resource, NSJSP provides access to the secured resource. For more information on authorization, see [Authorizing a User](#) on page 8-29.

The HTTP basic authentication is not a secure authentication protocol because user passwords are sent in the simple base64 encoding format. A secured transport layer, such as Secure Socket Layer (SSL), provides a more secure connection.

You can configure a web application for the HTTP basic authentication by setting the `auth-method` element in the application's `web.xml` (located in the `<NSJSP_HOME>/webapps/<application_directory>/WEB-INF` directory for a web application deployed in the `<application_directory>`) file to `BASIC`:

```
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Realm_Name</realm-name>
</login-config>
```

where, *Realm_Name* is the name of the Realm. For more information on Realms, see [Realms](#) on page 8-7.

When a user attempts to access a web application that is configured for the HTTP basic authentication, a login page as shown in [Figure 8-2](#) appears and the user is prompted to enter the username and password.

Figure 8-2. Login Page for HTTP Basic Authentication



HTTP Digest Authentication

Similar to the HTTP basic authentication method, the HTTP digest authentication method authenticates a user based on a username and a password. However, the authentication is performed by sending the password in an encrypted form that is more secure than the simple base64 encoding used by the HTTP basic authentication.

You can configure a web application for digest authentication by setting the `auth-method` element in the `web.xml` (located in the `<NSJSP_HOME>/webapps/<application_directory>/WEB-INF` directory for a web application deployed in the `<application_directory>`) file to `DIGEST`:

```
<login-config>
    <auth-method>DIGEST</auth-method>
    <realm-name>Realm_Name</realm-name>
</login-config>
```

where, *Realm_Name* is the name of the Realm. For more information on Realms, see [Realms](#) on page 8-7.

When a user attempts to access a web application that is configured for the HTTP digest authentication, the logon page as shown in [Figure 8-3](#) appears and the user is prompted to enter the username and password.

Figure 8-3. Logon Page for HTTP Digest Authentication



Form-Based Authentication

The form-based authentication method provides a mechanism for a web application developer to control the display of the login screen. The display of the logon page cannot be varied using the web browser's inbuilt authentication methods, such as BASIC and DIGEST.

You can configure a web application for form-based authentication by setting the `auth-method` in the `<NSJSP_HOME>/web.xml` file to `FORM`. The web application deployment descriptor contains entries for a login form and an error page:

```
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/login.jsp</form-login-page>
    <form-error-page>/login.jsp?authFailure=true
```

```
</form-error-page>
</form-login-config>
</login-config>
```

The login form must contain HTML fields for entering a username and a password, and these fields must be named as `j_username` and `j_password`, respectively.

For the form-based authentication to function properly, you must always set the `action` attribute in the login form to `j_security_check`. The following is a sample form coding for an HTML logon page:

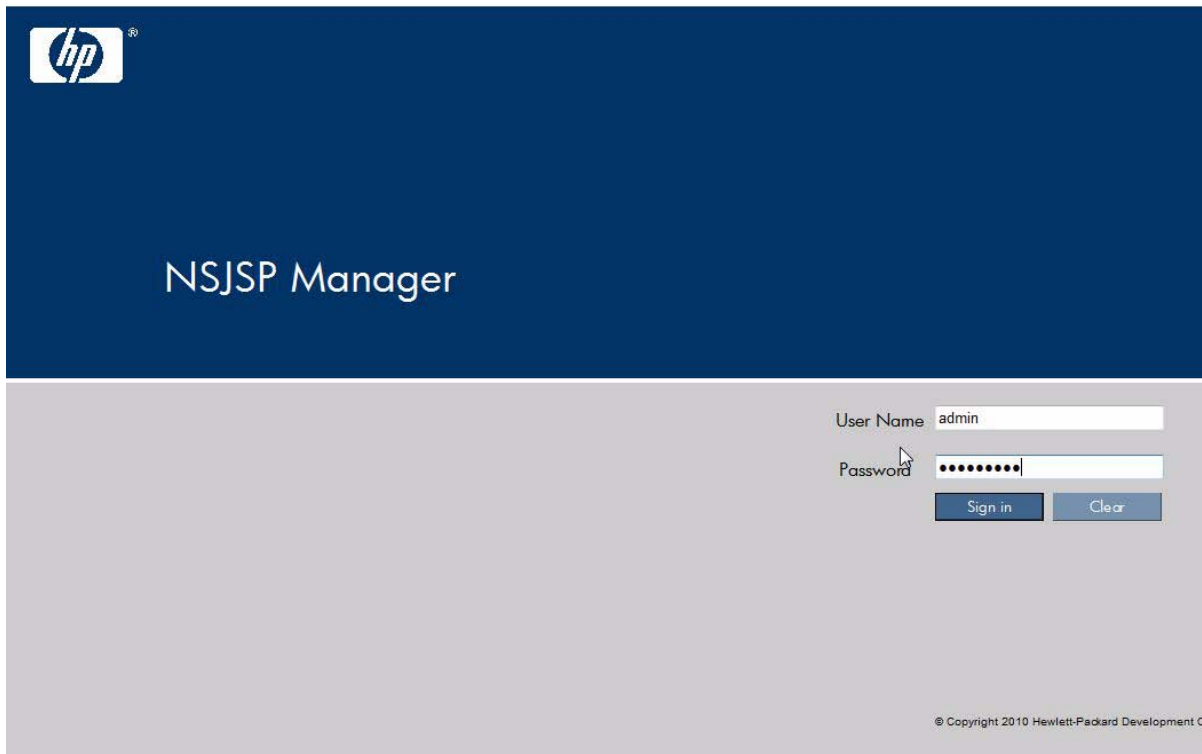
```
<form method="POST" action="j_security_check">
  <input type="text" name="j_username">
  <input type="password" name="j_password">
</form>
```

The following sequence of steps occur when a user attempts to access a protected web resource using form-based authentication:

1. The login form associated with the security constraint is sent to the web browser and the URL path triggering the authentication is stored by the NSJSP container.
2. The user is prompted to enter the username and password in the login form.
3. The web browser sends the login form to the server.
4. The NSJSP container attempts to authenticate the user using the information from the form:
 - a. If authentication fails, the container returns an error page.
 - b. If authentication succeeds, the authenticated user's credential is checked to verify if it has been assigned a role, which is authorized to access the web resource.
5. If the user is authorized, the web browser is redirected to the web resource using the stored URL path in the NSJSP container.

Similar to the HTTP basic authentication method, the form-based authentication method also lacks security because the password is transmitted as plain text. A secured transport layer, such as SSL, can ease some of these concerns.

[Figure 8-4](#) shows the logon page for a form-based authentication.

Figure 8-4. Logon Page for a Form-Based Authentication

hp

NSJSP Manager

User Name admin

Password

Sign in Clear

© Copyright 2010 Hewlett-Packard Development C

HTTPS Client Authentication

The end user authentication using HTTPS (HTTP over SSL) requires the client to possess a Public Key Certificate (PKC). Both NSJSP and the iTP Secure WebServer support X.509 version 3 certificates. A web application can be configured for HTTPS client authentication by setting the type of authentication in the `web.xml` file to `CLIENT-CERT`.

Realms

Although the Java Servlet specification describes a portable mechanism for applications to declare their security requirements (in the `web.xml` deployment descriptor), a portable API that defines the interface between a servlet container and the associated user and role information is not available. Therefore, a database of usernames and passwords is required to validate the users. A Realm is such a database of usernames and passwords that identify valid users of a web application (or a set of web applications) with a list of the roles associated with each valid user. Roles are similar to groups in UNIX-like operating systems because access to specific web

application resources is granted to all users possessing a particular role. There can be any number of roles associated with a username.

Note. The Java Servlet specification defines declarative elements, such as `security-constraint`, `auth-constraint`, `user-data-constraint` as part of the deployment descriptor to define an application's security requirements.

In many cases, however, it is desirable to connect a servlet container to an existing authentication database or an existing authentication mechanism. Therefore, NSJSP defines a Java interface (`org.apache.catalina.Realm`) that can be implemented by plug in components to establish a connection for authentication.

The following plug-in components support connections to various sources of authentication information:

- **MemoryRealm:** Accesses authentication information stored in an in-memory object collection, which is initialized from an XML document (`<NSJSP_HOME>/conf/nsjsp-users.xml`).
- **JDBCRealm:** Accesses authentication information stored in a relational database, accessed using a Java Database Connectivity (JDBC) driver.
- **DataSourceRealm:** Accesses authentication information stored in a relational database, accessed using a Java Naming and Directory Interface (JNDI) named JDBC DataSource.
- **JNDIRealm:** Accesses authentication information stored in the Lightweight Directory Access Protocol (LDAP) accessible directory server, accessed using a JNDI provider.
- **UserDataBaseRealm:** Accesses authentication information stored in a JNDI resource, which is an XML file by default.
- **JAASRealm:** Accesses authentication information through the Java Authentication and Authorization Service (JAAS) framework.

Apart from these standard plug-in components, `NSJSPLockoutRealm` and `CombinedRealm` are other implementations, (that also implement the Realm interface) which provide the Realm functionality by using one or more of the previously mentioned plug-ins.

Realm elements can be configured as child elements of any of the following elements:

- **The Engine element:** The Engine element is defined in the `<NSJSP_HOME>/conf/server.xml` file. In this case, the Realm applies to all contexts (web applications) in all the Hosts configured under the Engine element. Any Realm definition in the Host or Context element overrides the Realm definition in the Engine element. In the default NSJSP configuration, a `UserDataBaseRealm` is configured under the Engine element and is used to authenticate users in the Admin and the Manager Web applications.
- **The Host element:** Host elements are defined in the `<NSJSP_HOME>/conf/server.xml` file. This Realm definition applies to all

contexts (web applications) configured in the `Host`. Any Realm definition in the `Context` overrides the `Host` definition.

- The `Context` element: Realms can be defined as child elements of a context (web application) in the `context.xml` file. The `context.xml` file for each application is located in the `META-INF` folder of the web application base directory, which will be located in the `<NSJSP_HOME>/webapps` directory. The `Context` element can also be defined in additional locations that are described at <http://tomcat.apache.org/tomcat-6.0-doc/config/context.html>. A Realm definition in a `Context` overrides any Realm definitions in either the `Host` or `Engine` element.

The following sections discuss different types of Realms:

- [JNDIRealm](#)
- [MemoryRealm](#)
- [JDBCRealm](#)
- [UserDatabaseRealm](#)
- [JAASRealm](#)
- [DataSourceRealm](#)
- [CombinedRealm](#)
- [NSJSPLockOutRealm](#)

JNDIRealm

It is important to have a basic understanding of JNDI and LDAP before attempting to understand the concept of the JNDIRealm.

The JNDIRealm is an implementation of the Realm interface that looks up users in a LDAP directory server accessed by a JNDI provider (typically, the standard LDAP provider, which implements the JNDI API classes). The JNDIRealm supports many approaches for using an LDAP directory for authentication. For more information on JNDI, see <http://java.sun.com/javase/6/docs/technotes/guides/jndi/index.html>.

When the JNDIRealm is configured in NSJSP, the JNDIRealm connects to the directory server, authenticates the user, and fetches the roles associated with the user if the authentication is successful. This section discusses these actions in detail.

This section discusses the following topics:

- [Connecting to the Directory](#)
- [Selecting the Directory Entry for the User](#)
- [Authenticating the User in a JNDIRealm Configuration](#)
- [Assigning Roles to a User](#)
- [JNDIRealm Attributes](#)

Connecting to the Directory

The `connectionURL` configuration attribute in the `JNDIRealm` defines the Realm's connection to the directory server and the JNDI provider defines the format for this URL. Usually, an LDAP URL specifies the domain name of the directory server to connect, and optionally the port number and distinguished name (DN) of the required root naming context.

The `alternateURL` configuration attribute can be used in case of multiple providers so that if a socket connection cannot be established to the provider on the `connectionURL`, the `alternateURL` can be used.

While making a connection, in order to search the directory and to retrieve user and role information, the Realm authenticates itself to the directory with the username and password specified by the `connectionName` and `connectionPassword` properties. If these properties are not specified, the connection is anonymous.

Selecting the Directory Entry for the User

Each user that can be authenticated must be represented in the directory by an individual entry. This entry corresponds to an element in the initial `DirContext` defined by the `connectionURL` attribute. The user entry must have an attribute containing the username that is presented for authentication.

Often the distinguished name of the user's entry contains the username presented for authentication but is otherwise the same for all users. In this case, the `userPattern` attribute can be used to specify the DN with a `{0}` marking where the username must be substituted.

Otherwise, the `JNDIRealm` must search the directory to find a unique entry containing the username. You can configure the following attributes to search the username:

- `userBase` - Specifies the entry that is the base of the subtree containing users. If not specified, the search base is the top-level context.
- `userSubtree` - Specifies the search scope. Set this attribute to `true`, if you want to search the entire subtree rooted at the `userBase` entry. You can set this attribute to `false`, if you want to perform a single-level search that includes only the top level.
- `userSearch` - Specifies the pattern that indicates the LDAP search filter after substituting the username.

Authenticating the User in a JNDIRealm Configuration

The user can be authenticated using the following methods depending on the `JNDIRealm` configuration:

- Bind Mode

By default, the Realm authenticates a user by binding to the directory with the DN of the entry for a user and the password provided by the user. If this bind succeeds, the user is considered to be authenticated.

For security reasons, a directory may store a digest of the user's password rather than the plain text version. In this case, as part of the bind operation, the directory automatically computes the correct digest of the plain text password before validating it against the stored value. Therefore, in the bind mode, the Realm is not involved in digest processing. The digest attribute is not used, and will be ignored if the attribute is set.

- **Comparison Mode**

In the comparison mode, a Realm can retrieve the password stored in the directory and compare the password explicitly with the value presented by the user. You can configure this mode by setting the `userPassword` attribute to the name of a directory attribute in the user's entry that contains the password.

The comparison mode poses some disadvantages. The `connectionName` and `connectionPassword` attributes must be configured to allow the Realm to read users' passwords in the directory. Due to security reasons, it is not preferable for a Realm to read users' passwords. Many directory implementations do not allow even the directory manager to read these passwords. Additionally, the Realm must handle password digests itself, including variations in the algorithms and different methods of representing password hashes in the directory. However, the Realm might sometimes need access to the stored password, for instance to support HTTP Digest Access Authentication.

Assigning Roles to a User

The JNDIRealm supports the following methods to represent roles in the directory:

Note. You can use a combination of both these methods to represent a role.

- **Roles as explicit directory entries**

You can represent roles by specifying explicit directory entries. A role entry is usually an LDAP group entry with one attribute containing the name of the role and another attribute specifying the distinguished names or usernames of users in that role.

You can use the following attributes to configure a directory search to find role names corresponding to the authenticated user:

- `roleBase` - Specifies the base entry for the role search. If this attribute is not specified, the search base is the top-level directory context.
- `roleSubtree` - Specifies the search scope. Set this attribute to `true`, if you want to search the entire subtree rooted at the `roleBase` entry. The default value (`false`) requests a single-level search, including only the top level.

- `roleSearch` - Specifies the LDAP search filter for selecting role entries. It optionally includes pattern replacements for either the distinguished name (`{0}`) or the username (`{1}`) of the authenticated user, or both of them.
- `roleName` - Specifies the attribute in a role entry containing the name of that role.
- Roles as an attribute of the user entry

You can specify role names as the values of an attribute in the user's directory entry. Use `userRoleName` to specify the name of this attribute.

JNDIRealm Attributes

[Table 8-1](#) lists the attributes that can be used in the JNDIRealm.

Table 8-1. Attributes in the JNDIRealm (page 1 of 3)

Attribute	Description
<code>adCompat</code>	Specifies whether the JNDIRealm must ignore exceptions. The Microsoft Active Directory (AD) often returns referrals. When iterating over <code>NamingEnumerations</code> , these iterations lead to <code>PartialResultExceptions</code> . If you want JNDIRealm to ignore those exceptions, set this attribute to <code>true</code> . There is no stable way to detect if the exceptions arrived from an AD referral. The default value for <code>adCompat</code> is <code>false</code> .
<code>alternateURL</code>	Specifies the alternate URL to use if the JNDIRealm cannot make a socket connection to the provider at the <code>connectionURL</code> .
<code>authentication</code>	Specifies the type of authentication. The values are <code>none</code> , <code>simple</code> , <code>strong</code> or a provider-specific definition. If a value is not specified, the provider's default value is used.
<code>commonRole</code>	Specifies a role name assigned to each successfully authenticated user in addition to the roles retrieved from LDAP. If this attribute is not specified, only the roles retrieved from LDAP are used.
<code>connectionName</code>	Specifies the directory username when establishing a connection to the directory for LDAP search operations. If this attribute is not specified, the JNDIRealm makes an anonymous connection that is sufficient unless you specify the <code>userPassword</code> property.
<code>connectionPassword</code>	Specifies the directory password while establishing a connection to the directory for LDAP search operations. If this attribute is not specified, the JNDIRealm makes an anonymous connection that is sufficient unless you specify the <code>userPassword</code> property.
<code>connectionTimeout</code>	Specifies the timeout (in milliseconds) while establishing the connection to the LDAP directory. If this attribute is not specified, a timeout value of 5000 (5 seconds) is used.

Table 8-1. Attributes in the JNDIRealm (page 2 of 3)

Attribute	Description
<code>connectionURL</code>	Specifies the connection URL that is passed to the JNDI driver while establishing a connection to the directory.
<code>contextFactory</code>	Specifies the fully qualified Java class name of the factory class that is used to acquire the JNDI InitialContext. By default, JNDIRealm considers that the standard JNDI LDAP provider is utilized.
<code>derefAliases</code>	Specifies how aliases must be dereferenced during search operations. The permitted values are <code>always</code> , <code>never</code> , <code>finding</code> , and <code>searching</code> . If this attribute is not specified, <code>always</code> is used.
<code>digest</code>	Specifies the digest algorithm to be applied to the plain text password offered by the user before comparing it with the value retrieved from the directory. The valid values for <code>digest</code> are those accepted for the algorithm name by the <code>java.security.MessageDigest</code> class. If this attribute is not specified, the plain text password is considered to be retrieved. The <code>digest</code> attribute is not required unless <code>userPassword</code> is specified.
<code>protocol</code>	Specifies the security protocol. If this attribute is not specified, the protocol defined by the provider is used by default.
<code>referrals</code>	Specifies whether to follow referrals. The permitted values are <code>ignore</code> , <code>follow</code> , or <code>throw</code> . For more information on referrals, see <code>javax.naming.Context.REFERRAL</code> . The Microsoft Active Directory often returns referrals. To follow AD referrals, set <code>referrals</code> to <code>follow</code> . Warning: If your DNS server is not part of an AD, the LDAP client library might try to resolve your domain name in the DNS to find another LDAP server.
<code>roleBase</code>	Specifies the base directory entry for searching roles. If this attribute is not specified, the top-level element in the directory context is used.
<code>roleName</code>	Specifies the name of the attribute that contains role names in the directory entries found by a role search. Additionally, you can use the <code>userRoleName</code> property to specify in the user's entry, the name of an attribute that contains additional role names. If the <code>roleName</code> attribute is not specified, a role search does not occur and roles are obtained only from the user's entry.
<code>roleSearch</code>	Specifies the LDAP filter expression used for performing role searches. You can use <code>{0}</code> to substitute the distinguished name of the user and <code>{1}</code> to substitute the username. If this attribute is not specified, a role search does not occur and roles are taken only from the <code>userRoleName</code> attribute in the user's entry.

Table 8-1. Attributes in the JNDIRealm (page 3 of 3)

Attribute	Description
<code>roleSubtree</code>	Specifies whether multiple levels under the node specified by <code>roleBase</code> must be searched. You can set this attribute to <code>true</code> to search the entire subtree of the element specified by the <code>roleBase</code> property, for role entries associated with the user. The default value (<code>false</code>) results in searches of the top level subtree.
<code>userBase</code>	Specifies the base element for user searches performed using the <code>userSearch</code> expression. This attribute is not used if you are using the <code>userPattern</code> expression.
<code>userPassword</code>	Specifies the name of the attribute in the user's entry containing the user's password. If you specify a value for this attribute, the JNDIRealm binds to the directory using the values specified by <code>connectionName</code> and <code>connectionPassword</code> properties, and retrieves the corresponding attribute for comparison with the value specified by the user. If you do not specify this value, the JNDIRealm attempts a simple bind to the directory using the DN of the user's entry and the password presented by the user. If the bind is successful, the user is considered to be authenticated.
<code>userPattern</code>	Specifies the pattern for the distinguished name of the user's directory entry. A <code>{0}</code> marks where the actual username must be inserted. You can use this attribute instead of <code>userSearch</code> , <code>userSubtree</code> and <code>userBase</code> when the distinguished name contains the username and is otherwise the same for all users.
<code>userRoleName</code>	Specifies the name of an attribute in the user's directory entry that contains zero or more values for the names of roles assigned to this user. Additionally, you can use the <code>roleName</code> attribute to specify the name of an attribute to be retrieved from individual role entries, which were obtained by searching the directory. If this attribute is not specified, all the roles for a user are derived from the role search.
<code>userSearch</code>	Specifies the LDAP filter expression to use, when searching the user's directory entry. A <code>{0}</code> marks where the actual username must be inserted. You can use this attribute (along with the <code>userBase</code> and <code>userSubtree</code> properties) instead of <code>userPattern</code> to search the directory for the user's entry.
<code>userSubtree</code>	Specifies whether multiple levels under the node specified by <code>userBase</code> must be searched. You can set this attribute to <code>true</code> if you want to search the entire subtree of the element specified by the <code>userBase</code> property for the user's entry. The default value (<code>false</code>) results in searching only the top level subtree. This attribute is not used if you use the <code>userPattern</code> expression.

MemoryRealm

The MemoryRealm is a simple demonstration implementation of the Realm interface and is not intended for production use. During startup, the MemoryRealm loads information about all users, and their corresponding roles, from an XML document (by default, this document is loaded from `<NSJSP_HOME>/conf/nsjsp-users.xml`). Changes made to the `<NSJSP_HOME>/conf/nsjsp-users.xml` file will be effective only after NSJSP is restarted.

This section discusses the following topics:

- [MemoryRealm File Format](#)
- [Attributes in the MemoryRealm](#)

MemoryRealm File Format

The users file (by default, `<NSJSP_HOME>/conf/nsjsp-users.xml`) must be an XML document, with a root element called `<tomcat-users>`. The format specified by this file is the MemoryRealm file format. Nested inside the root element of this file is a `<user>` element for each valid user, consisting of the following attributes:

- `name` - Specifies the username with which the valid user must log in.
- `password` - Specifies the password with which the valid user must log in (in plain text format if the `digest` attribute is not set in the `<Realm>` element or digested appropriately as described in [HTTP Digest Authentication](#) on page 8-4).
- `roles` - Specifies the comma-separated list of the role names associated with this user.

Attributes in the MemoryRealm

[Table 8-2](#) lists the attributes that can be used in the MemoryRealm.

Table 8-2. MemoryRealm Attributes

Attribute	Description
<code>digest</code>	Specifies the digest algorithm that must be used to store passwords in non-plain text formats. The valid values for this attribute are those that are accepted for the algorithm name by the <code>java.security.MessageDigest</code> class. If this attribute is not specified, passwords are stored in plain text.
<code>pathname</code>	Specifies the absolute or relative (to <code><NSJSP_HOME></code>) pathname to the XML file containing the user information.

JDBCRealm

The JDBCRealm is an implementation of the Realm interface that obtains user information from a relational database accessed through a JDBC driver. The

JDBCRealm provides substantial configuration flexibility to adapt to existing table and column names, if your database structure conforms to the following requirements:

- **Users table:** There must be a table that contains one row for every user that the JDBCRealm must validate. The `users` table must contain at least the following two columns (it might contain more if applications require them):
 - **Username:** The username to be validated by NSJSP when the user logs in.
 - **Password:** The password to be validated by NSJSP when the user logs in. This value may be in plain text or digested. For more information on digested password, see [Digested Passwords](#) on page 8-28.
- **User roles table:** There must also be a table that contains one row for every valid role that is assigned to a particular user. A user can have zero, one, or more valid roles. The user roles table must contain at least the following two columns (it may contain more if applications require them):
 - **Username:** The username to be validated by NSJSP (the same value as specified in the `users` table).
 - **Role name:** The role name of a valid role associated with this user.

The JDBCRealm queries the database each time it is requested to authenticate a user. Therefore, any changes to the database are immediately reflected in the information used to authenticate users.

When a user is authenticated, the user (including the user's associated roles) information is cached within NSJSP for the duration of a user login session. For form-based authentication, the cached information lasts till the session times out or is invalidated; for basic and digest authentication, the cached information lasts till the user closes the browser. Changes to the database information for an authenticated user are not reflected until the next login by that user.

Attributes in the JDBCRealm

[Table 8-3](#) lists the attributes supported by the JDBCRealm.

Table 8-3. JDBCRealm Attributes (page 1 of 2)

Attribute	Description
<code>connectionName</code>	Specifies the database username to use when establishing the JDBC connection. This attribute is not relevant when using the Type 2 SQL/MX (or SQL/MP) driver.
<code>connectionPassword</code>	Specifies the database password that must be used when establishing the JDBC connection. This attribute is not relevant when using the Type 2 SQL/MX (or SQL/MP) driver.
<code>connectionURL</code>	Specifies the connection URL that must be passed to the JDBC driver when establishing a database connection.

Table 8-3. JDBCRealm Attributes (page 2 of 2)

Attribute	Description
<code>digest</code>	Specifies the name of the <code>MessageDigest</code> algorithm used to encrypt user passwords stored in the database. If this attribute is not specified, user passwords are by default stored in plain text. For more information on digest passwords, see Digested Passwords on page 8-28.
<code>digestEncoding</code>	Specifies the character set for encoding digests. If this attribute is not specified, the platform default will be used.
<code>driverName</code>	Specifies the fully qualified Java class name of the JDBC driver to connect to the authentication database.
<code>roleNameCol</code>	Specifies the name of the column, in the <code>user roles</code> table, which contains a role name assigned to the corresponding user. This is a mandatory attribute and has no default value.
<code>userCredCol</code>	Specifies the name of the column, in the <code>users</code> table, which contains the user's credentials (that is, password). If a value for the <code>digest</code> attribute is specified, this component considers the passwords to have been encoded with the specified algorithm. Otherwise, they will be considered to be in plain text. This is a mandatory attribute and has no default value.
<code>userNameCol</code>	Specifies the name of the column, in the <code>users</code> and <code>user roles</code> table, that contains the user's username. This is a mandatory attribute and has no default value.
<code>userRoleTable</code>	Specifies the name of the <code>user roles</code> table, which must contain columns specified by the <code>userNameCol</code> and the <code>roleNameCol</code> attributes. This is a mandatory attribute and has no default value.
<code>userTable</code>	Specifies the name of the <code>users</code> table, which must contain columns named by the <code>userNameCol</code> and the <code>userCredCol</code> attributes. This is a mandatory attribute and has no default value.

[Example 8-1](#) shows some `mxci` statements to create tables in SQL/MX for use in a JDBCRealm.

Example 8-1. Creating SQL/MX Tables for use in a JDBCRealm

```
create catalog mycatalog;
set catalog mycatalog;
create schema myschema;
set schema myschema;

create table users (
user_name char(15) not null,
user_pass varchar(45) not null,
primary key (user_name));

create table roles (
role_name char(20) not null,
role_desc varchar(250),
primary key (role_name));

create table userrole (
user_name char(15) not null,
role_name char(20) not null,
primary key(user_name, role_name));

insert into users values ('tomcat', 'tomcatpassword');
insert into users values ('user1', 'user1password');
insert into users values ('user2', 'user2password');
insert into roles values ('admin', 'The System Administrator');
insert into roles values ('manager', 'NSJSP Applications Manager');
insert into roles values ('role2', 'Another role');
insert into userrole values ('tomcat', 'admin');
insert into userrole values ('tomcat', 'manager');
insert into userrole value ('user1', 'manager');
```

[Example 8-2](#) shows a Realm definition for querying user information from an SQL/MX database using a JDBC Type 2 driver and the database schema listed in [Example 8-1](#).

Example 8-2. Sample Realm Configuration

```
<Realm className="org.apache.catalina.realm.JDBCRealm"
driverName="com.tandem.sqlmx.SQLMXDriver"
connectionURL="jdbc:sqlmx:"
userTable="mycatalog.myschema.users"
userNameCol="user_name"
userCredCol="user_pass"
userRoleTable="mycatalog.myschema.userrole"
roleNameCol="role_name"/>
```

Although the `roles` table in [Example 8-1](#) is not used in the Realm configuration in [Example 8-2](#), the `roles` table could be used to contain the description of each role.

[Example 8-1](#) uses a `char(15)` field for the `user_name` column in the `users` table. If you configure a web application to use the client-certificate based authentication method, this column size (15 characters) is not sufficient because the value stored in the `user_name` column must be the `Subject` from the client-certificate.

Note. `Subject` is the name of a field in a client certificate.

If the contents of the `Subject` field exceeds the NonStop SQL maximum allowable primary key limit of 256 characters, alter the `user_name` field to the appropriate size and define a new primary key within the NonStop SQL limits.

UserDatabaseRealm

For testing or limited production use, when the username, password, and roles are typically loaded from an XML document and any changes or additions need to be persisted to the XML document, then the `UserDatabaseRealm` is appropriate. In such cases, the web application should provide an implementation of the interface `org.apache.catalina.UserDatabase`. The methods of this interface also enable creating and deleting users and roles. The `UserDatabaseRealm` is a `Realm` implementation that complies with the `org.apache.catalina.UserDatabase` interface.

The default installation of NSJSP provides an implementation of the `org.apache.catalina.UserDatabase` resource that loads the data from `<NSJSP_HOME>/conf/nsjsp-users.xml`. The implementation class is `org.apache.catalina.user.MemoryUserDatabase`. The `UserDatabase` is configured as a global JNDI resource and a JNDI object creating factory class `MemoryUserDatabaseFactory`, (that implements the `javax.naming.spi.ObjectFactory` interface) which is used to obtain an instance of `MemoryUserDatabase`.

[Example 8-3](#) shows the `UserDatabase` resource definition in the `GlobalNamingResources` section. It also shows a `UserDatabaseRealm` definition that uses the `UserDatabase` resource in the `Engine` element.

Example 8-3. Sample UserDatabase Definition

```
...
...
<GlobalNamingResources>
    <Resource name="UserDatabase" auth="Container"
        type="org.apache.catalina.UserDatabase"
        description="User database that can be updated and saved"
        factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
        pathname="conf/nsjsp-users.xml" />
</GlobalNamingResources>
...
<Engine name="NSJSP" defaultHost="localhost">
    <RealmclassName="org.apache.catalina.realm.UserDatabaseRealm"
        resourceName="UserDatabase" digest="MD5" />
...
...
...
```

Attributes in the UserDatabaseRealm

[Table 8-4](#) lists the attributes that can be used in the `UserDatabaseRealm`.

Table 8-4. UserDatabaseRealm Attributes

Attribute	Description
<code>resourceName</code>	Specifies the name of the resource that this Realm will use to obtain user, password, and role information.
<code>digest</code>	Specifies the name of the <code>MessageDigest</code> algorithm used to encode user passwords stored in the database. If this attribute is not specified, by default, user passwords are stored in plain text. For more information on digested passwords, see Digested Passwords on page 8-28.

JAASRealm

The `JAASRealm` is an implementation of the `Realm` interface that authenticates users through the Java Authentication and Authorization Service (JAAS) framework. For

more information on JAAS, see

<http://java.sun.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html>

NSJSP provides the required JAAS infrastructure to accept NonStop usernames and to authenticate users in the Safeguard subsystem. This means NonStop usernames and passwords can be used to authenticate users and groups in the Safeguard subsystem that can be used as user roles to authorize access to application resources.

The JAAS framework provides the `LoginModule` interface that must be implemented by authentication service providers. NSJSP provides an implementation of the `LoginModule` interface in the `com.tandem.servlet.jaas.NonStopLoginModule` class. The `NonStopLoginModule` authenticates the user using the Safeguard subsystem and fetches the groups to which the logged in user belongs. The `LoginModule` class must be registered with the JAAS framework. This is done by providing the JAAS configuration file (`<NSJSP_HOME>/conf/iTP_jaas.config`) through the `-Djava.security.auth.login.config` command-line argument. The following sample `iTP_jaas.config` file shows the configuration of the `NonStopLoginModule`:

```
NonStopUserDB {
    com.tandem.servlet.jaas.NonStopLoginModule  REQUIRED  debug=false;
};
```

where, `NonStopUserDB` is the name used by the JAAS framework to uniquely identify the `LoginModule` that must be used for authentication.

With the use of `NonStopLoginModule` as the `LoginModule`, users can login with NonStop usernames in any of the following forms:

- NonStop Username: `super.webmstr`
- NonStop Group, User: `255,20`
- NonStop User ID: `65305`
- Safeguard alias name: `webman`

On successful authentication of a NonStop user, the `NonStopLoginModule` returns the groups to which the user or alias belongs. For example, if the Safeguard alias `webman` (for NonStop user `SUPER.WEBMSTR`) is configured as belonging to groups `SUPER`, `SOFTWARE` and `WEB`, then on successful authentication, the roles returned for `webman` are:

- `SUPER`
- `SOFTWARE`
- `WEB`
- `SUPER.WEBMSTR` (the actual NonStop username)

Attributes in the JAASRealm

[Table 8-5](#) lists the attributes that can be used in the JAASRealm.

Table 8-5. JAASRealm Attributes

Attribute	Description
<code>appName</code>	Specifies the name of the application as configured in your login configuration file.
<code>userClassNames</code>	Specifies a comma-separated list of the names of classes that you have created for your user Principals.
<code>roleClassNames</code>	Specifies a comma-separated list of the names of the classes that you have created for your role Principals.
<code>useContextClassLoader</code>	Instructs the JAASRealm to use the context class loader for loading the user-specified <code>LoginModule</code> class and associated Principal classes. The default value is <code>true</code> . To load classes using the container's classloader, specify <code>false</code> . If the <code>NonStopLoginModule</code> is used, this value can be set to <code>false</code> since the class is available with the container class loader.
<code>digest</code>	Specifies the digest algorithm used to store passwords in non-plain text formats. Valid values are those accepted for the algorithm name by the <code>java.security.MessageDigest</code> class. If not specified, passwords are stored in plain text.

Configuring Authentication Using JAASRealm

To configure authentication using JAASRealm and the `NonStopLoginModule`, complete the following steps:

1. Open the `servlet.config` file in `<NSJSP_HOME>/conf` and complete the following steps:
 - a. Comment the following entry in the `servlet.config` file by inserting a `#` at the beginning of the line:


```
set NSJSP_JAAS_CONFIG -Dnsjsp.jaas.login.config=none.
```
 - b. Uncomment the following entry by removing `#` in the beginning of the entry:


```
set NSJSP_JAAS_CONFIG -Djava.security.auth.login.config==
$env(JAAS_CONFIG_FILE
```
2. Configure security constraints in the web application deployment descriptor (`web.xml`) to allow access to application resources for users belonging to

NonStop groups. The following sample code provides access to users belonging to SUPER and SYSSW groups:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>My Appl's Secure Pages
  </web-resource-name>
    <description>Security constraint for resources in the
      secure directory</description>
    <url-pattern>/secure/*</url-pattern>
    <http-method>POST</http-method>
    <http-method>GET</http-method>
  </web-resource-collection>
  <auth-constraint>
    <description>only let the system user login
    </description>
    <role-name>SYSSW</role-name>
    <role-name>SUPER</role-name>
  </auth-constraint>
</security-constraint>
```

3. Configure the JAASRealm:

```
<Realm className="org.apache.catalina.realm.JAASRealm"
  appName="NonStopUserDB"
  userClassNames="com.tandem.servlet.jaas.
    NonStopUserPrincipal"
  roleClassNames="com.tandem.servlet.jaas.
    NonStopRolePrincipal"
  useContextClassLoader="false" />
```

For information on configuring a Realm, see [Realms](#) on page 8-7.

DataSourceRealm

The DataSourceRealm is an implementation of the Realm interface that queries users in a relational database accessed through a JNDI named JDBC DataSource.

The database schema of the table containing usernames and passwords, and the table linking usernames with roles is the same as for the JDBCRealm. To create the required tables, see [Example 8-1](#) in the [JDBCRealm](#) on page 8-15.

The `DataSourceRealm` queries the database each time it attempts to authenticate a user. Therefore, changes made to the database will be immediately reflected in the information used to authenticate users.

After a user has been authenticated, the user (and the user's associated roles) information is cached within NSJSP for the duration of a user login session. For form-based authentication, the cached information lasts till the session times out or is invalidated; for basic and digest authentication, the cached information lasts till the user closes the browser. Changes to the database information for an already authenticated user are not reflected until the next login by the user.

Attributes in the `DataSourceRealm`

[Table 8-6](#) lists the attributes that can be used in the `DataSourceRealm`.

Table 8-6. `DataSourceRealm` Attributes

Attribute	Description
<code>dataSourceName</code>	Specifies the name of the JNDI JDBC <code>DataSource</code> for this Realm.
<code>digest</code>	Specifies the name of the <code>MessageDigest</code> algorithm used to encode user passwords stored in the database. If not specified, user passwords are considered to be stored in plain text.
<code>localDataSource</code>	When the Realm is nested inside a <code>Context</code> element, this allows the Realm to use a <code>DataSource</code> defined for the Context rather than a global <code>DataSource</code> . If this attribute is not specified, the default is <code>false</code> and the Realm uses a global <code>DataSource</code> . A global <code>datasource</code> is defined in the <code>GlobalNamingResources</code> element under the <code>Server</code> element.
<code>roleNameCol</code>	Specifies the name of the column, in the <code>user roles</code> table, which contains a role name assigned to the corresponding user.
<code>userCredCol</code>	Specifies the name of the column, in the <code>users</code> table, which contains the user's credentials (that is, password). If a value for the <code>digest</code> attribute is specified, this component considers that passwords have been encoded with the specified algorithm. Otherwise, they are considered to be in plain text.
<code>userNameCol</code>	Specifies the name of the column, in the <code>users</code> and <code>user roles</code> table that contains the user's username.
<code>userRoleTable</code>	Specifies the name of the <code>user roles</code> table, which must contain columns named by the <code>userNameCol</code> and <code>roleNameCol</code> attributes.
<code>userTable</code>	Specifies the name of the <code>users</code> table, which must contain columns named by the <code>userNameCol</code> and <code>userCredCol</code> attributes.

By default, NSJSP uses the Apache Commons Database connection pool (DBCP) library to create data source objects.

[Example 8-4](#) shows how to define a JNDI datasource to connect to SQL/MX using the JDBC Type 2 driver. The JNDI resource can either be configured in `server.xml` or in the application-specific `context.xml` file. For more information on configuring JNDI resources, see <http://tomcat.apache.org/tomcat-6.0-doc/jndi-resources-howto.html>.

Example 8-4. Defining a Global JNDI Datasource

```
<GlobalNamingResources>
  <Resource name="jdbc/authority"
    auth="Container"
    type="javax.sql.DataSource"
    driverClassName="com.tandem.sqlmx.SQLMXDriver"
    url="jdbc:sqlmx: "
    maxActive="20"
    maxIdle="10"
    maxWait="-1"/>
</GlobalNamingResources>
```

For a complete set of attributes that can be used to configure a data source, see <http://commons.apache.org/dbcp/configuration.html>.

[Example 8-5](#) shows a sample `DataSourceRealm` configuration that uses a global `DataSource`.

Example 8-5. Sample DataSourceRealm Configuration

```
<Realm className="org.apache.catalina.realm.DataSourceRealm"
  dataSourceName="jdbc/authority"
  userTable="mycatalog.myschema.users"
  userNameCol="user_name"
  userCredCol="user_pass"
  userRoleTable="mycatalog.myschema.userrole"
  roleNameCol="role_name"/>
```

Note. Because the `localDataSource` attribute is not used in [Example 8-5](#), the Realm searches the datasource using the `jdbc/authority` name defined in `GlobalNamingResources`.

CombinedRealm

The `CombinedRealm` is an implementation of the Tomcat 6 Realm interface that authenticates users through one or more sub-Realms.

The `CombinedRealm` provides the ability to combine multiple Realms of the same or different types. The `CombinedRealm` can be used to authenticate against different sources, provide fallback in case one Realm fails or for any other purpose that requires multiple Realms.

Sub-Realms are defined by nesting `Realm` elements inside the `Realm` element that defines the `CombinedRealm`. Authentication will be attempted against each Realm in the order they are listed. Successful authentication against any Realm is sufficient to authenticate the user.

[Example 8-6](#) shows how to configure a `UserDataBaseRealm` and a `DataSourceRealm` within a `CombinedRealm`.

Example 8-6. Configuring a `UserDataBaseRealm` and `DataSourceRealm` Within a `CombinedRealm`

```
<Realm className="org.apache.catalina.realm.CombinedRealm" >
  <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
    resourceName="UserDatabase"/>
  <Realm className="org.apache.catalina.realm.DataSourceRealm"
    dataSourceName="jdbc/authority"
    userTable="users"
    userNameCol="user_name"
    userCredCol="user_pass"
    userRoleTable="user_roles"
    roleNameCol="role_name"/>
</Realm/>
```

NSJSPLockOutRealm

The `NSJSPLockOutRealm` is used to provide the option of locking out a user if there are many failed authentication attempts in a given period of time. This Realm is implemented by the

`com.tandem.servlet.catalina.realm.NSJSPLockOutRealm` class.

To ensure correct functioning, there is a reasonable degree of synchronization built into the Realm, across the server class instances. This means that each server class instance is aware of the total number of failed authentication attempts for a user, even though the authentication attempts might have occurred in different server class instances. This Realm uses a disk file to record authorization attempts across server class instances, and the user records on failed authentication attempts are persisted to a disk file and are available across server class restarts.

The `NSJSPLockOutRealm` does not require modification to the underlying Realms or the associated user storage mechanisms. It achieves this by recording all failed login attempts, including those for users that are not defined. Storing failed login attempts of

even users that do not exist could result in a large number of user records getting cached, especially when the authentication of invalid users is deliberate, such as in a denial of service attack. To prevent unlimited growth of the cache, use the `cacheSize` attribute to indicate the maximum number of user records that may be cached.

Sub-Realms are defined by nesting the Realm elements inside the Realm element that defines the `LockOutRealm`. Authentication will be attempted against each Realm in the order they are listed. Successful authentication against any Realm will be sufficient to authenticate the user.

Attributes in the NSJSPLockOutRealm

[Table 8-7](#) lists the attributes in the NSJSPLockOutRealm.

Table 8-7. NSJSPLockOutRealm Attributes

Attribute	Description
<code>cacheRemovalWarningTime</code>	If a failed user cannot be added to the cache because there is insufficient memory to accommodate the failed user, one of the existing entries will be removed. If the removed entry has been in the cache for a lesser interval of time than the time configured in this attribute, then a warning message is generated.
<code>cacheSize</code>	Specifies the maximum number of user slots to hold failed user authentication attempts. Over a period of time, the cache will grow to the size specified by <code>cacheSize</code> and may not shrink. The default size is 1000 users.
<code>failureCount</code>	Specifies the number of times in a row that a user has to fail authentication to be locked out. The default value is 5.
<code>lockOutTime</code>	Specifies the time (in seconds) that a user is locked out after too many authentication failures. The default value is 300 (5 minutes).

[Example 8-7](#) shows how to configure an NSJSPLockOutRealm that uses the `UserDatabaseRealm` to authenticate users.

Example 8-7. Configuring an NSJSPLockOutRealm

```
<Realm className="com.tandem.servlet.catalina.realm.NSJSPLockOutRealm"
  failureCount="3"
  lockOutTime="3600">
  <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
    resourceName="UserDatabase"/>
</Realm/>
```

In [Example 8-7](#), users are locked out for 1 hour (3600 seconds) after 3 failed authentication attempts.

Digested Passwords

For each standard Realm implementation, the user's password (by default) is stored in plain text. In most environments, this situation is not acceptable because casual observers of the authentication data can collect enough information to log on successfully and impersonate other users. To avoid this problem, the standard Realm implementations support the concept of digesting user passwords. Digesting passwords causes the stored version of passwords to be encoded in a form that is not easily reversible but which the Realm implementation can still use for authentication.

You can enable digested passwords by specifying the `digest` attribute in the `Realm` element. The value for this attribute must be one of the digest algorithms supported by the `java.security.MessageDigest` class (such as `SHA` or `MD5`).

When you specify this option, the contents of the password that are stored in the Realm must be the digested version of the plain text password, as digested by the specified algorithm. When the `authenticate()` method of the Realm is called, the (plain text) password specified by the user is digested by the same algorithm, and the result is compared with the value returned by the Realm. A match means that the user is authorized.

To calculate the digested value of a plain text password, you can use the following techniques:

- If you are writing an application that needs to calculate digested passwords dynamically, call the static `Digest()` method of the `org.apache.catalina.realm.RealmBase` class, passing the plain text password and the digest algorithm name as arguments. This method returns the digested password.
- A command-line utility (`nsjsp_digestPassword`) is available to calculate the digested password.
Enter the following at the command-line prompt:

```
<NSJSP_HOME>/conf/nsjsp_digestPassword {digest algorithm}  
{cleartext-password}
```

The digested version of this plain text password is returned to the standard output.

Single Sign-On

You can use the single sign-on feature when you wish to provide users the ability to sign on to any one of the web applications associated with your virtual host, and then have their identity recognized by all other web applications on the same virtual host.

This feature is provided in the form of a valve called `SingleSignOn`. The single sign-on facility operates according to the following rules:

- All web applications configured for this virtual host must share the same Realm. This means you can nest the `Realm` element inside this `Host` element (or the

surrounding `Engine` element), but not inside a `Context` element for one of the involved web applications.

- If users access only unprotected resources in any of the web applications on this virtual host, they will not be challenged to authenticate themselves.
- If users access a protected resource in any web application associated with this virtual host, users will be challenged to authenticate, using the login method defined for the web application currently being accessed.
- After authentication, the roles associated with this user will be utilized for access control decisions for all the associated web applications, without challenging the user to authenticate themselves to each application individually.
- When the user logs out of one web application (for example, by invalidating the corresponding session, if form-based authentication is used), the user's sessions in all web applications will be invalidated. Any subsequent attempt to access a protected resource in any application will require the user to authenticate again.
- The single sign-on feature utilizes HTTP cookies to transmit a token that associates each request with the saved user identity, so it can only be utilized in client environments that support cookies.

To configure single sign-on for the applications within a host, configure the `SingleSignOn` valve as a child of the `Host` element where the single sign-on feature should be implemented:

```
<Host>
...
  <Valve
    className="org.apache.catalina.authenticator.SingleSignOn"
  />
...
</Host>
```

Authorizing a User

While the authentication process establishes the identity of the user, the authorization process determines if the user is allowed to access a secured resource. You can authorize a user to access specific resources by using the `security-constraint` element in an application-specific deployment descriptor, which is an XML document named `.../WEB-INF/web.xml`.

Security constraints are a declarative method of defining the protection for web content. A security constraint associates authorization or user data constraints, or both, with HTTP operations on web resources. Therefore, you can configure a web application to secure an associated set of web resources, which may include HTTP operations and URL patterns either by authorizing a user or by checking the transport layer connection used to access the resources or both.

The security constraint, which is represented by the `security-constraint` element in a deployment descriptor, consists of the following elements that are used to secure a resource:

- `web-resource-collection`
- `auth-constraint`
- `user-data-constraint`

While the `web-resource-collection` is a required element, `auth-constraint` and `user-data-constraint` elements are optional elements.

The following sections discuss the different elements in the `security-constraint` element:

- [Web Resource Collection](#)
- [Authorization Constraint](#)
- [User Data Constraint](#)

Web Resource Collection

HTTP operations and web resources to which a security constraint applies (that is, the constrained requests) are identified by one or more web resource collections. A web resource collection (identified by the `web-resource-collection` element) consists of the following elements:

- URL patterns - Specified by the `url-pattern` element in the `web.xml` file.
- HTTP methods - Specified by the `http-method` element in the `web.xml` file.

The following is a sample `web-resource-collection` from a `web.xml` file:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Domain Manager User URLs
      </web-resource-name>
    <url-pattern>*.htm</url-pattern>
    <url-pattern>*.html</url-pattern>
    <url-pattern>*.jsp</url-pattern>
    <url-pattern>*.xml</url-pattern>
    <url-pattern>*.help</url-pattern>
    <http-method>POST</http-method>
    <http-method>GET</http-method>
  </web-resource-collection>
```

```

...
...
...
</security-constraint>

```

Authorization Constraint

An authorization constraint establishes a requirement for authentication and specifies the authorization roles permitted to perform constrained requests. A user must be a member of at least one of the specified roles to be permitted to perform the constrained requests. The special role name `*` specifies all the role names defined in the deployment descriptor.

Note. Security roles referenced by a web application are identified by specifying the `security-role` element in the `web.xml` file.

An authorization constraint that does not specify any roles indicates that access to the constrained requests is not permitted under any circumstances. An authorization constraint contains the `role-name` element.

The following is a sample authorization constraint (`<auth-constraint>`) element from a `web.xml` file:

```

<security-constraint>
  <web-resource-collection>
    ...
    ...
  </web-resource-collection>
  <auth-constraint>
    <description>only let the system user login</description>
    <role-name>admin</role-name>
    <role-name>manager</role-name>
  </auth-constraint>
</security-constraint>

```

User Data Constraint

A user data constraint establishes a requirement that the constrained requests are received over a protected transport layer connection. The strength of the required protection is defined by the value of the transport guarantee. A `user-data-constraint` element contains a `transport-guarantee` element, which specifies

the strength of the transport layer protection. By default, the `transport-guarantee` element is not defined in the `web.xml` file.

[Table 8-8](#) lists the types of transport guarantee that can be defined in the `web.xml` file.

Table 8-8. Types of Transport Guarantee

Type	Description
INTEGRAL	Establishes a requirement for content integrity.
CONFIDENTIAL	Establishes a requirement for confidentiality.
NONE	Indicates that the container must accept the constrained requests when received on any connection including an unprotected connection.

The transport guarantee can ensure that certain resources are always requested over a secure link. If the transport guarantee is set to either `INTEGRAL` or `CONFIDENTIAL`, the constrained resource must be requested over a secure transport, such as HTTPS. The iTP Secure WebServer is configured for secure transport through the `httpd.stl.config` file located in the `<iTP WebServer Home>/conf` directory.

The following is a sample definition of the `user-data-constraint` element from a `web.xml` file:

```
<security-constraint>
    <web-resource-collection>
        ...
        ...
    </web-resource-collection>
    <auth-constraint>
        ...
        ...
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Note. For a complete listing of elements that can be used in the deployment descriptor, see the Java Servlet Specification Version 2.5.

Validating the Sender

While NSJSP uses the user's credentials to authenticate and authorize a user, it also provides `valve` elements to validate the origin of a client request before the request is serviced. The `valve` element allows or denies requests originating from certain client's hosts based on the client's host name or IP address.

A `valve` element represents a component that will be inserted into the request processing pipeline for the associated Catalina container (`Engine`, `Host`, or `Context`).

The iTP Secure WebServer also provides features to either deny or allow access to resources. For more information, see the `Region` directive, and the `DenyHost` and `AllowHost` commands for the `Region` directive in the *iTP Secure Webserver System Administrator's Guide*.

Note. Even though NSJSP provides the capability to allow or deny requests originating from certain client hosts, HP recommends that you configure such criteria in the iTP Secure WebServer rather than in NSJSP.

The following sections discuss the two valves that are available with NSJSP to restrict access based on the client's host name and IP address:

- [Remote Host Filter](#)
- [Remote Address Filter](#)

Remote Host Filter

The Remote Host Filter valve compares the hostname of the host that sent the request against one or more regular expressions, and either allows the request to continue or refuses to process the request from that host. The syntax for the regular expressions is specified in the `java.util.regex` class. For more information on this class, see <http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html>.

You can associate a Remote Host Filter valve with any Catalina container (`Engine`, `Host`, or `Context`). The Remote Host Filter must accept any request presented to the associated container for processing before passing on the request.

[Table 8-9](#) lists the attributes that can be used to configure a Remote Host Filter valve.

Table 8-9. Remote Host Filter Attributes

Attribute	Description
classname	Specifies the Java class name for this valve. You must set this attribute to <code>org.apache.catalina.valves.RemoteHostValve</code> .
allow	Specifies a comma-separated list of regular expression patterns with which the remote hostname is compared. If you specify this attribute, the hostname from an incoming request must match an expression for the request to be accepted. If you do not specify this attribute, all requests are accepted unless the remote hostname matches a <code>deny</code> pattern.
deny	Specifies a comma-separated list of regular expression patterns with which the remote hostname is compared. If you specify this attribute, the hostname from an incoming request must not match any expression for the request to be accepted. If you do not specify this attribute, all requests are accepted based on the <code>allow</code> attribute.

Remote Address Filter

The Remote Address Filter valve compares the IP address of the host that sends a request against one or more regular expressions, and either allows the request to continue or refuses to process the request from the host. You can associate a Remote Address Filter with any Catalina container (`Engine`, `Host`, or `Context`). The Remote Address Filter must accept any request presented to the associated container for processing before passing on the request.

[Table 8-10](#) lists the attributes that can be used to configure a Remote Address Filter valve.

Table 8-10. Remote Address Filter Attributes

Attribute	Description
classname	Specifies the Java class name for this valve. You must set this attribute to <code>org.apache.catalina.valves.RemoteAddrValve</code> .
allow	Specifies a comma-separated list of regular expression patterns with which the remote client's IP address is compared. If you specify this attribute, the remote client's IP address from an incoming request must match an expression for the request to be accepted. If you do not specify this attribute, all requests are accepted unless the remote client's IP address matches the <code>deny</code> pattern.
deny	Specifies a comma-separated list of regular expression patterns with which the remote client's IP address is compared. If you specify this attribute, the remote client's IP address from an incoming request must not match any expression for the request to be accepted. If you do not specify this attribute, all requests are accepted based on the <code>allow</code> attribute.

Java Security Manager

The Java security manager can be used to restrict access to system resources, such as Java Virtual Machine (JVM) properties, methods, and sockets, thus safeguarding application data and services, and ensuring the security and reliability of NSJSP.

This section explains some of the features of the Java security manager that are used in NSJSP.

Note. The complete details of the Java security manager are beyond the scope of this section.

This section contains the following topics:

- [Configuring the Java Security Manager](#)
- [Securing NSJSP Resources Using the permissions Directive](#)
- [Package Protection in NSJSP](#)
- [Troubleshooting the Java Security Manager](#)

Configuring the Java Security Manager

The security policies implemented by the Java Security Manager are configured in the `iTP_catalina.policy` file located in the `<NSJSP_HOME>/conf` directory.

The `iTP_catalina.policy` file replaces any system `java.policy` file. The `iTP_catalina.policy` file contains a default set of security policies to be enforced (by the JVM) when NSJSP is run with the `java.security.manager` option. You can assign additional permissions to individual web applications by adding additional grant entries in the `iTP_catalina.policy` file.

Entries in the `iTP_catalina.policy` file use the standard format for `java.policy` files, as shown in [Example 8-8](#):

Example 8-8. Java Policy File Entry

```
// Example policy file entry

grant [signedBy <signer>,) [codeBase <code source>] {
    permission <class> [<name> [, <action list>]];
};
```

The `signedBy` and `codeBase` entries are optional when granting permissions. Comment lines begin with `//` and end at the end of the current line. The `codeBase` is in the form of a URL, which can use the `${java.home}` and the `${catalina.home}` properties, which are expanded to the directory paths defined for them by the `JAVA_HOME`, `CATALINA_BASE`, and `CATALINA_HOME` environment variables.

The default `iTP_catalina.policy` file contains all the grant entries in the standard `catalina.policy` file and additional entries for the NSJSP container, as shown in [Example 8-9](#):

Example 8-9. Policy File Entry for the NSJSP Container (page 1 of 2)

```
// These permissions apply to the nsjsp-logging code
grant codeBase "file:${catalina.home}/bin/nsjsp-logging.jar" {
    permission java.security.AllPermission;
};

grant codeBase "file:${catalina.home}/bin/nsjsp_bootstrap.jar" {
    permission java.security.AllPermission;
};

...

// These permissions apply to the servlet API classes
// and those that are shared across all class loaders
// located in the "common" directory. Need 3 different directory
// paths as the java Security Manager can't handle symbolic
// links within a directory tree.
grant codeBase "file:${catalina.home}/common/classes/-" {
    permission java.security.AllPermission;
};

grant codeBase "file:${catalina.home}/common/endorsed/-" {
    permission java.security.AllPermission;
};

grant codeBase "file:${catalina.home}/common/lib/-" {
    permission java.security.AllPermission;
};

// These permissions apply to the container's core code, plus
// any additional libraries installed in the "server" directory.
grant codeBase "file:${catalina.home}/server/classes/-" {
    permission java.security.AllPermission;
};

grant codeBase "file:${catalina.home}/server/lib/-" {
    permission java.security.AllPermission;
};

grant codeBase "file:${catalina.home}/server/nsjsp_webapps/-" {
    permission java.security.AllPermission;
};

// ===== JDBC DRIVERS CODE PERMISSIONS =====
grant codeBase "file:/usr/tandem/jdbcMx/current/lib/jdbcMx.jar"
{
    permission java.security.AllPermission;
};

grant codeBase "file:/usr/tandem/jdbcMp/current/lib/sqlmp.jar" {
    permission java.security.AllPermission;
};
```

Example 8-9. Policy File Entry for the NSJSP Container (page 2 of 2)

```
// These permissions are granted to the NSJSP balancer web
// application.
grant codeBase "file:${catalina.home}/webapps/balancer/WEB-
INF/lib/catalina-balancer.jar" {
    permission java.lang.reflect.ReflectPermission
    "suppressAccessChecks";
};

// These permissions are granted by default to all web
// applications. In addition, a web application will be given a
// read FilePermission and JndiPermission for all files and
// directories in its document root.
grant {
    ...
    // NSJSP Specific properties to allow read access
    permission java.util.PropertyPermission
    "com.tandem.servlet.*", "read";
    permission
    java.util.PropertyPermission "org.apache.commons.logging.*",
    "read";
    ...
};
```

Starting NSJSP with the Java Security Manager

After you configure the `iTP_catalina.policy` file, NSJSP can start with the Java Security Manager using the command-line arguments `java.security.manager` and

`java.security.policy==${env(<NSJSP_HOME>)/conf/iTP_catalina.policy}` in the `servlet.config` file.

[Example 8-10](#) shows the configuration to start NSJSP with the Java Security Manager.

Example 8-10. Starting NSJSP with the Java Security Manager

```

#
# NSJSP Java2 System policy file and Java2 VM option.
#
# Note: the "double" equalto signs "==" is not a typo!! This informs
the JVM
# to use this file exclusively and that all others are to be ignored.
#
set  env(JVM_POLICY_FILE)  $env(NSJSP_HOME)/conf/iTP_catalina.policy
set  NSJSP_SECMGR_POLICY -Djava.security.policy==$env(JVM_POLICY_FILE)
#
# By default, the JVM is run without a security manager.
#
# set  NSJSP_SECMGR -Dnsjsp.security.manager=none
#
# If you wish to run with a security Manager, uncomment the next
# statement ("set  NSJSP_SECMGR ...").
#
set  NSJSP_SECMGR -Djava.security.manager

...

#
# This is the actual Arglist used to startup the NSJSP Container.
#
Arglist -Xmx64m -Xss128k -Xnoclassgc \
    -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager \
    -Djava.util.logging.config.file=$env(NSJSP_HOME)/conf/logging.properties \
    -Djavax.management.builder.initial=com.tandem.servlet.jmx.NSJSPMBean
ServerBuilder \
    $NSJSP_SECMGR \
    $NSJSP_SECMGR_POLICY \
    $NSJSP_JAAS_CONFIG \
    -Dcatalina.home=$env(NSJSP_HOME) \
    -Dcatalina.base=$env(NSJSP_HOME) \
    -Djava.io.tmpdir=$env(NSJSP_HOME)/temp \
    org.apache.catalina.startup.Bootstrap start

```

In [Example 8-10](#), you can notice that the following entry (the highlighted entry in the example) is uncommented so that NSJSP runs with the Java Security Manager:

```
set  NSJSP_SECMGR -Djava.security.manager
```

This entry is always commented out in the `servlet.config` file and by default, NSJSP runs without the Java Security Manager. The following default definition in the `servlet.config` file informs NSJSP not to run with a Java Security Manager:

```
set  NSJSP_SECMGR -Dnsjsp.security.manager=none
```

Securing NSJSP Resources Using the permissions Directive

The permission attribute in the `iTP_catalina.policy` file represents access permission to a system resource. A system resource could mean a particular method in a class, resources such as disk files or system property values. When NSJSP is running with Java Security Manager, the web application code must be provided explicit permissions to execute secure code.

By default, all web applications are granted read permission to all the file resources under the web application's base directory. This enables the application to read the static resources. Permissions are granted to access the resources either through JNDI or through classes in the `java.io` package.

The following sample `iTP_catalina.policy` file shows permissions to read system properties:

```
grant {
    // Required for JNDI lookup of named JDBC DataSource's and
    // javamail named MimePart DataSource used to send mail
    permission java.util.PropertyPermission "java.home", "read";
    permission java.util.PropertyPermission "java.naming.*", "read";
    permission java.util.PropertyPermission "javax.sql.*", "read";
    ...
}
```

As the `grant` directive does not specify any `codeBase`, the `grant` directive applies to every `codeBase`. This means that all container libraries and web applications are granted permission to read system properties, such as `java.home`.

The following sample `grant` directive grants permissions to all the `jar` files present in the `${catalina.home}/lib` directory, which in NSJSP translates to `<NSJSP_HOME>/lib`:

```
grant codeBase "file:${catalina.home}/lib/-" {
    permission java.security.AllPermission;
};
```

Providing all permissions is the same as running without the Java Security Manager for the `codeBase` in consideration.

While running NSJSP under a Java Security Manager, with the default security policy, some JSP examples throw a security exception. For example, running `examples/jsp/jsp2/simpletag/hello.jsp` throws the following exception:

```
access: access denied (java.lang.RuntimePermission accessDeclaredMembers)
java.lang.Exception: Stack trace
    at java.lang.Thread.dumpStack(Thread.java:1158)
    at
    java.security.AccessControlContext.checkPermission(AccessControlContext.java:253)
        at java.security.AccessController.checkPermission
        (AccessController.java:427)
        at java.lang.SecurityManager.checkPermission
        (SecurityManager.java:532)
        at java.lang.SecurityManager.checkMemberAccess(SecurityManager.java:1662)
    )
    ...
    ...
```

This exception is thrown because the user applications are not provided the `accessDeclaredMembers` permission. This permission can be granted to every `codeBase` by adding the following permission in a `grant` block:

```
permission java.lang.RuntimePermission "accessDeclaredMembers";
```

You must be careful before granting the `accessDeclaredMembers` permission to any `codeBase`.

Note. HP recommends that you understand the impact of providing certain security permissions. You can find a list of all security permissions provided by Java at <http://java.sun.com/javase/6/docs/technotes/guides/security/permissions.html>.

When NSJSP is run with the security manager with the default security policy file, all web applications are prevented from executing code, such as `System.exit()`.

Package Protection in NSJSP

As discussed in [Securing NSJSP Resources Using the permissions Directive](#) on page 8-38, NSJSP provides security to prevent malicious applications from gaining access to NSJSP internal classes. Apart from executing methods of some internal classes, it is possible to gain entry into internal classes by defining classes in the same package as the NSJSP internal classes. For example, all classes within the same package have access to protected resources (for example, methods and variables) of other classes in the package.

You can specify internal NSJSP packages that must be protected against package definition and access. The protection can be configured in the `catalina.properties` file in the `<NSJSP_HOME>/conf` directory using the following `package.access` and `package.definition` properties:

- `package.access`: This property can be used to restrict access to classes in certain packages. For example, if the value of this property is `java.io, java.net`, then access to classes in these packages is prevented unless permissions are granted using the `accessClassInPackage` target name of the `java.lang.RuntimePermission`.

By default, the value is not set. However, the following entry is available in the `catalina.properties` file to grant access to specified packages and is commented by default:

```
#package.access=sun.,org.apache.catalina.,org.apache.coyote.,
org.apache.tomcat.,org.apache.jasper.,sun.beans.
```

- `package.definition`: This property can be used to restrict class definitions to certain packages.

By default, none of the packages are restricted and none of the class loaders call `checkPackageDefinition`. However, the following entry is available in the `catalina.properties` file to restrict access to specified packages and is commented by default:

```
#package.definition=sun.,java.,org.apache.catalina.,org.apach
e.coyote.,org.apache.tomcat.,org.apache.jasper.
```

Troubleshooting the Java Security Manager

The security manager can be configured to write debug logs by using the `java.security.debug` property. This can be useful in identifying the call and permission that caused a security exception from being thrown. All logs are written to the `STDOUT` file. By default, the file referenced by the `STDOUT` server class attribute is located in the `<NSJSP_HOME>/logs` directory. For more information on the `java.security.debug` property options, see <http://java.sun.com/javase/6/webnotes/trouble/TSG-VM/html/envvars.html>.

In addition to these options, the value `all` prints a large amount of information (the size of which can grow to several MBs). HP recommends that you use the `access,failure` value. Following is a sample definition:

```
-Djava.security.debug=access,failure
```

Additionally, printing too many log messages may result in slower NSJSP startup time.

[Example 8-11](#) shows how to set the Java Security Manager debug information in the `servlet.config` file.

Example 8-11. Setting the Java Security Debug Information

```
Arglist -Xmx64m -Xss128k -Xnoclassgc \
-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager \
-Djava.util.logging.config.file=$env(NSJSP_HOME)/conf/logging.properties \
-Djavax.management.builder.initial=com.tandem.servlet.jmx.NSJSPMBeanServerBuilder \
-Djava.security.debug=access,failure \
$NSJSP_SECMGR \
$NSJSP_SECMGR_POLICY \
$NSJSP_JAAS_CONFIG \
-Dcatalina.home=$env(NSJSP_HOME) \
-Dcatalina.base=$env(NSJSP_HOME) \
-Djava.io.tmpdir=$env(NSJSP_HOME)/temp \
org.apache.catalina.startup.Bootstrap start
```

Manager Web Application and NSJSP Manager Security

NSJSP provides several Web Applications for administering and monitoring purposes. The Admin Web application in NSJSP enables you to administer container objects and manage resources, such as users and roles. The following Manager Web applications in NSJSP enable you to manage web applications in an NSJSP deployment:

- Manager Web Application
- NSJSP Manager

The Admin and Manager Web Applications are security-sensitive applications and proper security constraints must be implemented so that only authorized users are allowed access to these Web Applications.

For more information on the Admin and Manager Web Applications, see Chapter [4, Managing NSJSP](#).

The following sections discuss how security can be implemented in these Web Applications:

- [Using Realms to Implement Security](#)
- [Monitoring Server Classes and Hosts](#)

Using Realms to Implement Security

When a user attempts to access a Manager, NSJSP Manager, or Admin Web application, the user's credentials are verified and validated using the Realms repository. For a detailed description on the usage of Realms for implementing security in a web application, see [Realms](#) on page 8-7.

Monitoring Server Classes and Hosts

Starting with NSJSP 6.1, a user can manage multiple NSJSP installations in one iTP WebServer installation. Each NSJSP installation can have multiple hosts and each host can contain multiple applications. Each installation of NSJSP is associated with a unique server class. All the hosts might not need access to all the server classes. Therefore, a security constraint may be implemented so that only certain hosts are visible for a particular server class.

The NSJSP Manager enables you to monitor hosts and server classes as defined in the `<NSJSP_MANAGER_HOME>/conf/host-access.properties` file. You can use the **Scope** tab in the NSJSP Manager Web Application to select a Host in a specific server class. Hosts and the server class displayed in the NSJSP Manager depends on the definition in the `<NSJSP_HOME>/webapps/ROOT/WEB-INF/host-access.properties` file.

[Example 8-12](#) provides sample definition in the `host-access.properties` file.

Example 8-12. Sample `host-access.properties` File

```
manager=*:*  
admin=*.*
```

[Example 8-12](#) indicates that all users with role `manager` can manage all server classes (indicated by the first `*` in `*:*`) and all the Hosts in those server classes (indicated by the second `*` in `*:*`).

[Example 8-13](#) shows different formats to define a role in the `host-access.properties` file.

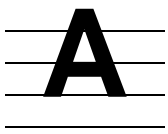
Example 8-13. Role Definitions

```
role1=sc1:host1|sc2:host2|*:host3|sc4:*|sc2:host5  
role2=sc3|sc4
```

The first entry in [Example 8-13](#) indicates that all users with role `role1` can manage:

- Server class 1 (`sc1`) and Host 1 in that server class (`sc1:host1`)
- Server class 2 (`sc2`) and Host 2 in that server class (`sc2:host2`)
- All server classes and Host 3 in any of those server classes (`*:host3`)
- Server class 4 (`sc4`) and all the Hosts in that server class (`sc4:*`)
- Server class 2 (`sc2`) and Host 5 in that server class (`sc2:host5`)

The second entry in [Example 8-13](#) indicates that all users with role `role2` can manage server class 3 (`sc3`) and all Hosts under `sc3`, and server class 4 (`sc4`) and all Hosts under `sc4`.



MBeans in the NSJSP Container

This appendix describes NSJSP MBeans, how the MBeans are represented in the NSJSP Manager application, and lists the MBeans that are commonly used in NSJSP.

This appendix addresses the following topics:

- [Prerequisites](#)
- [Overview](#)
- [Object Names and Attributes of MBeans](#)
- [MBeans Representation in NSJSP Manager](#)
- [Commonly Used MBeans in NSJSP](#)

Prerequisites

To understand the concept of MBeans and how they are used to obtain information about the servlet container components, you must be aware of the following:

- Java Management Extensions (JMX) technology and how it can be used to manage Java resources, such as, a Java application, a JDBC connection, or a servlet.
- NSJSP servlet container components, such as, Server, Service, Engine, Connector, and Host. A knowledge of these components is required to understand how these components are instrumented by MBeans.

For information on how JMX and MBeans facilitate the management of Java resources deployed on a Java Virtual Machine (JVM), see <http://java.sun.com/javase/6/docs/technotes/guides/jmx/index.html>.

Overview

Starting with the NSJSP 6.0 release, NSJSP components (such as, Server, Connector, Engine, Host, and Contexts) are instrumented by MBeans. Thus, the NSJSP components can be managed using the JMX technology. You can use MBeans to obtain information about the NSJSP components.

Note. NSJSP extensively uses dynamic MBeans. The dynamic MBeans allow NSJSP to define the management interface at run time. NSJSP uses the `mbeans-descriptors.xml` file to identify the MBean's management interface. Most of the MBeans have the management interface defined at run time. While using the NSJSP Manager to view the MBean attribute, some MBeans display the attributes, such as `managedResource` and `modelerType`, which indicates that the resource is being managed by the MBean.

The following are typical uses of MBeans in NSJSP:

- You can use MBeans to obtain the state of the component it instruments. The information obtained using MBeans can be of the following types:
 - Information that is constant and does not change with time. For example, the MBean that instruments the Host component has the `appBase` attribute, which represents the directory where the web applications are placed. The MBean attributes are explained in [Object Names and Attributes of MBeans](#) on page A-2.
 - Information that denotes the current state of the component, and can change with time. For example, the MBean instrumenting the Servlet by name JSP, has the `processingTime` attribute that represents the average time taken by the Servlet to process JSP requests. The MBeans attributes are explained in [Object Names and Attributes of MBeans](#) on page A-2.
- You can also use MBeans to alter the state of the component that it instruments. For example, you can modify the value of the `maxInactiveInterval` variable. For more information on `maxInactiveInterval`, see [Configuring In-Memory Sessions](#) on page 3-76.

Note. Similar examples are included later in this chapter.

Object Names and Attributes of MBeans

MBeans that are registered with an MBean server are grouped using namespace, called domain. Each MBean has a unique identifier, called the object name. The object name consists of a domain followed by a key-properties list.

The key-properties list with the domain identifies a specific MBean within an MBean server. The key-properties list consists of a minimum of one and might be more property-value pairs.

Note. The MBean server stores all the MBeans in a JVM.

The following are examples of object names:

- `NSJSP:name=http-0,type=ThreadPool`

where,

`NSJSP` represents the domain followed by the key-properties list.

The key-properties list includes the following property-value pairs:

`name=http-0` and `type=ThreadPool`

- `NSJSP:type=Manager,path=/sca6url/examples,host=localhost`

where,

`NSJSP` represents the domain.

`type=Manager, path=/sca6url/examples, and host=localhost` represent the property-value pairs.

- `Catalina:type=NamingResources, resourcetype=Context, path=/sca6url, host=localhost`

where,

`Catalina` represents the domain.

`type=NamingResources, resourcetype=Context, path=/sca6url, and host=localhost` represent the property-value pairs.

- `JMImplementation:type=MBeanServerDelegate`

where,

`JMImplementation` represents the domain.

`type=MBeanServerDelegate` represent the property-value pairs.

The object name might include the property-value pairs that have unusual values, such as, `none` and `URI`. In the following example, the `J2EEApplication` and the `J2EEServer` properties have the value `none`:

```
NSJSP:type=JspMonitor,name=jsp,WebModule=//localhost/sca6url/examples,J2EEApplication=none,J2EEServer=none
```

More information is provided on page [A-7](#).

For more information on the object name, see

<http://java.sun.com/javase/6/docs/api/javax/management/ObjectName.html>.

An MBean provides the state and details of the managed components in the form of attributes. Each attribute is denoted by a unique attribute name. You can view and change the state of the component by modifying the value of the attributes. For example, the attribute, called `maxThreads` of the MBean instrumenting the Connector component denotes the number of threads that the Connector can spawn. By modifying the value of the `maxThreads` attribute, the number of threads that the Connector is allowed to spawn can be changed.

Note.

- The NSJSP Manager application enables you to view and modify the values of the MBean attributes. However, it does not allow you to invoke methods on MBeans. For more information on the NSJSP Manager application, see Chapter [4, Managing NSJSP](#).
 - You cannot modify those MBeans whose `writable` parameter is set to `false`.
-

You can view all the MBeans in the NSJSP Server Class processes using the NSJSP Manager application. The MBeans are organized in a tree structure based on the MBean object names.

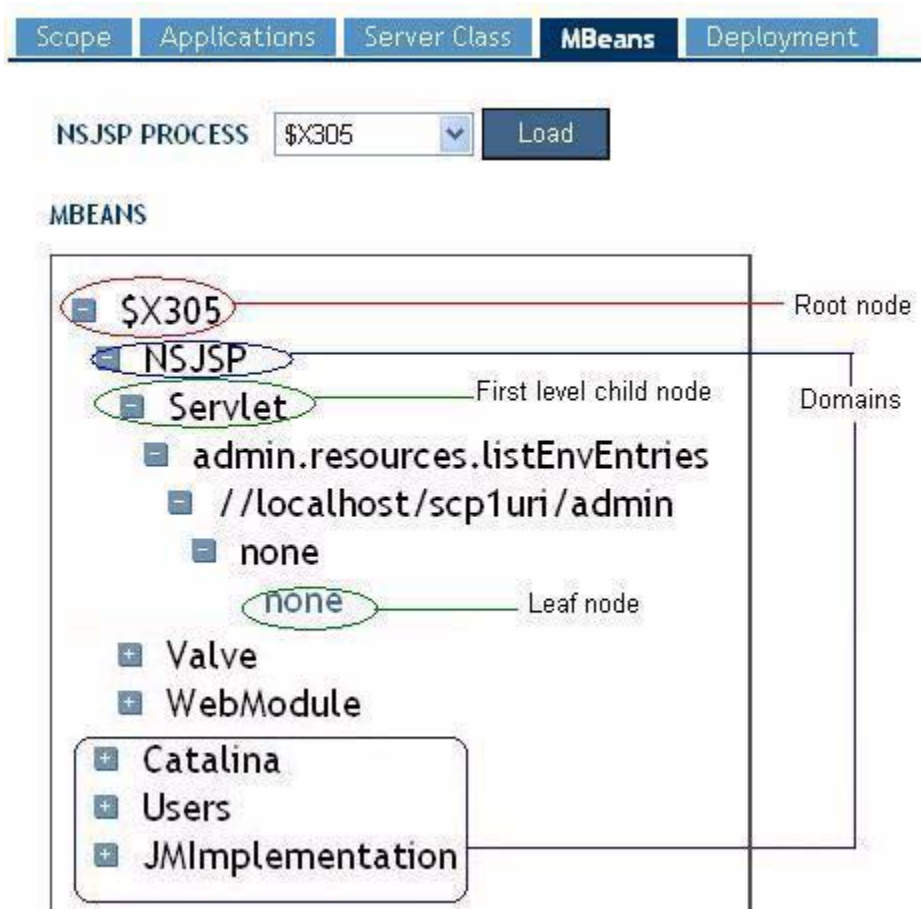
The following section explains how the MBeans are organized in a tree structure.

MBeans Representation in NSJSP Manager

For ease of navigation, MBeans are organized in a tree structure comprising multiple tree nodes. A tree structure begins with a root node and ends with a leaf node. The root node of the tree denotes the process name and the first level child nodes denote the different domains registered with the MBean server. Each domain can contain multiple tree nodes.

[Figure A-1](#) illustrates a tree view of an MBean.

Figure A-1. Tree View of an MBean



The tree structure of MBeans is constructed using the domain of the MBean and the property-value pairs. In [Figure A-1](#), NSJSP, Catalina, Users, and JMImplementation denote the domains within the MBean server in the \$X305 process. The NSJSP domain contains nodes, such as, Servlet, Valve, and WebModule. All MBeans under the Servlet node have the property `j2eeType` with the value `Servlet`. However, all MBeans under the Valve node have the property `type` with the value `Valve`. Although Servlet and Valve are peer nodes within the

same domain, the properties (`j2eeType` for `Servlet` and `type` for `Valve`) of the key-properties list are not the same.

[Figure A-2](#) shows a tree view of the following MBean under the `Servlet` node:

```
NSJSP:j2eeType=Servlet,name=HTMLManager,WebModule=//localhost/sca6url/manager,J2EEApplication=none,J2EEServer=none
```

where,

NSJSP is the domain.

`j2eeType=Servlet`, `name=HTMLManager`, `WebModule=//localhost/sca6url/manager`, `J2EEApplication=none`, and `J2EEServer=none` are the property-value pairs.

Figure A-2. Tree View of an MBean Under the `servlet` Node

NSJSP PROCESS

MBEANS

- [-] \$X305
 - [-] NSJSP
 - [-] Servlet
 - [-] jsp
 - [-] //localhost/sca6url/manager
 - [-] none
 - none

Object Name:
NSJSP:j2eeType=Servlet,name=HTMLManager,
WebModule=//localhost/sca6url/manager,
J2EEApplication=none,J2EEServer=none

Description: Wrapper that represents an individual
servlet definition

[Compare across Processes](#) [Display Attribute Description](#)

NAME	VALUE
classLoadTime	6
engineName	NSJSP
errorCount	0
eventProvider	false
loadTime	13
maxTime	0
minTime	9223372036854775807
modelerType	org.apache.catalina.core. StandardWrapper
objectName	NSJSP:j2eeType=Servlet, name=HTMLManager, WebModule=//localhost/sca6url/manager, J2EEApplication=none,J2EEServer=none
processingTime	0
requestCount	0
stateManageable	false

[Figure A-3](#) shows a tree view of the following MBean under the `valve` node:

NSJSP:type=Valve,name=RequestTrackerValve,host=localhost

where,

NSJSP is the domain.

type=Valve, name=RequestTrackerValve, and host=localhost are the
property-value pairs.

Figure A-3. Tree View of an MBean Under the valve Node

NSJSP PROCESS

MBEANS

- \$X305
 - NSJSP
 - + Servlet
 - Valve
 - RequestTrackerValve
 - localhost

Object Name: NSJSP:type=Valve,
name=RequestTrackerValve,host=localhost

Description: Tracks the request related statistics and figures.

[Compare across Processes](#) [Display Attribute Description](#)

Name	Value
className	com.hp.tandem.nsjsp.valves.RequestTrackerValve
containerName	NSJSP:type=Host,host=localhost
modelerType	com.hp.tandem.nsjsp.valves.RequestTrackerValve

As an exception, some MBeans might have properties with value `none`. For example, the following MBean represents a servlet (`j2eeType=Servlet`) called `jsp` (`name=jsp`) that is running in the context `/sca6url`, which is configured under the Host, called `localhost` (`WebModule=//localhost/sca6url`) and has two other properties (`J2EEApplication` and `J2EEServer`) whose values are `none`:

```
NSJSP:j2eeType=Servlet,name=jsp,WebModule=//localhost/sca6url,
J2EEApplication=none,J2EEServer=none
```

When this MBean is expanded in the tree using the NSJSP Manager application, the leaf node and the parent of the leaf node have the value `none`. The NSJSP container tries to find a valid value for each of the property. If it does not find a value, the property is displayed with value `none`.

[Figure A-4](#) displays a tree view of the MBean, which is described in the example, with the leaf node and its immediate parent node as `none`.

Figure A-4. Node with Value none

NSJSP PROCESS: \$X305

MBEANS

- \$X305
 - NSJSP
 - Servlet
 - jsp
 - //localhost/sca6url
 - none (highlighted with a red box and the text "none" below it)

Parent node and leaf node with value none

Object Name:
NSJSP:j2eeType=Servlet,name=jsp,
WebModule=//localhost/sca6url,
J2EEApplication=none,J2EEServer=none

Description: Wrapper that represents an individual servlet definition

[Compare across Processes](#) [Display Attribute Description](#)

NAME	VALUE
classLoadTime	6
engineName	NSJSP
errorCount	0
eventProvider	false
loadTime	13
maxTime	0
minTime	9223372036854775807
modelerType	org.apache.catalina.core. StandardWrapper
objectName	NSJSP:j2eeType=Servlet,name=jsp, WebModule=//localhost/sca6url, J2EEApplication=none,J2EEServer=none
processingTime	0
requestCount	0
stateManageable	false

The leaf node is a hyperlink. Clicking the leaf node displays the values of the MBean attributes.

[Figure A-5](#) displays the MBean attributes and their associated values.

Figure A-5. Values of the MBean Attributes

Object Name:

NSJSP:j2eeType=Servlet,name=Manager,WebModule=//localhost/sca6url/manager,
J2EEApplication=none,J2EEServer=none

Description: Wrapper that represents an individual servlet definition

[Compare across Processes](#) [Display Attribute Description](#)

Name	Value
classLoadTime	0
engineName	NSJSP
errorCount	0
eventProvider	false
loadTime	0
maxTime	0
minTime	9223372036854775807
modelerType	org.apache.catalina.core.StandardWrapper
objectName	NSJSP:j2eeType=Servlet,name=Manager, WebModule=//localhost/sca6url/manager, J2EEApplication=none,J2EEServer=none
processingTime	0
requestCount	0
stateManageable	false
statisticsProvider	false

To view the description of the attributes, click **Display Attribute Description** link.
[Figure A-6](#) shows the description of the MBean attributes.

Figure A-6. Description of the MBean Attributes

[Compare across Processes](#) [Display Attribute Values](#)

Name	Description
classLoadTime	Class loading time
engineName	Fully qualified class name of the managed object
errorCount	Error count
eventProvider	Event provider support for this managed object
loadTime	Load time
maxTime	Maximum processing time of a request
minTime	Minimum processing time of a request
modelerType	Type of the modeled resource. Can be set only once
objectName	Name of the object
processingTime	Total execution time of the servlet's service method
requestCount	Number of requests processed by this wrapper
stateManageable	State management support for this managed object
statisticsProvider	Performance statistics support for this managed object

Commonly Used MBeans in NSJSP

This section lists the important MBeans and their attributes. You can use the following MBeans to manage the Java resources that are deployed on a JVM:

- Thread Pool - Represents the connector thread pool.

The object name is:

`NSJSP:name=http-0,type=ThreadPool`

[Table A-1](#) lists only the important attributes associated with Thread Pool. For more information about the Thread Pool, see Chapter [3, Configuring NSJSP](#).

Table A-1. Attributes Associated with Thread Pool

Attributes	Attribute Description	Type	Writable
maxThreads	Introspected attribute maxThreads.	Integer	True
currentThreadsBusy	Introspected attribute currentThreadsBusy threadPriority Introspected attribute threadPriority.	Integer	False
threadPriority	—	Integer	True

Note. The NSJSP Manager application does not display the contents in the columns **Type** and **Writable** as shown in [Table A-1](#). The values for the `writable` parameter are true or false. The default value is true. The attributes, which can be changed, do not have the `writable` parameter included.

- **Host** - Represents the configuration parameters of a Host called `localhost`.

The object name is:

```
NSJSP:host=localhost,type=Host
```

[Table A-2](#) lists only the important attributes associated with Host. For more information about the Host attributes, see the [Host](#) on page 3-54.

Table A-2. Attributes Associated with Host

Attributes	Attribute Description	Type	Writable
<code>autoDeploy</code>	The auto deploy flag for this Host name - Unique name of this Host <code>unpackWARs</code> - Unpack WARs property.	Boolean	True
<code>unpackWARs</code>	—	Boolean	True
<code>name</code>	—	String	False

- **Request Dumper** - Switches on or switches off the knob to dump incoming requests.

The object name is:

```
NSJSP:type=Valve,name=NSJSPRequestDumperValve
```

[Table A-3](#) lists only the important attributes associated with Request Dumper. For more information about the Request Dumper attributes, see the [Request Dumper Valve](#) on page 3-59.

Table A-3. Attributes Associated with Request Dumper

Attributes	Attribute Description	Type	Writable
<code>dumperOn</code>	Indicates the knob that switches on or switches off the dumper valve.	Boolean	True
<code>recordLength</code>	Indicates the maximum number of bytes that is to be dumped for each request.	Integer	True

- **Application Context** - The MBean representing a context with name `first` and deployed in the Host called `localhost`.

The object name is:

```
NSJSP:J2EEApplication=none,J2EEServer=none,j2eeType=WebModule
,name=//localhost/first (application name)
```

[Table A-4](#) lists only the important attributes associated Application Context. For more information about the Application Context attributes, see [Table 3-17, Attribute List for the Context Element](#), on page 3-61.

Table A-4. Attributes Associated with Application Context

Attributes	Attribute Description	Type	Writable
workDir	Indicates the work directory for this context.	String	True

- JSP Statistics - Provides statistics related to all the JSP pages in an application.

The following example of an MBean object name refers to the statistics for executing JSPs in the `/Servlets/examples` context running in the Host called `localhost`:

```
NSJSP:j2eeType=Servlet,name=jsp,WebModule=//localhost/servlets/examples,J2EEApplication=none,J2EEServer=none
```

[Table A-5](#) lists only the important attributes associated with the JSP Statistics.

Table A-5. Attributes Associated with JSP Statistics

Attributes	Attribute Description	Type	Writable
minTime	Indicates the minimum time to process a JSP.	Integer	False
maxTime	Indicates the maximum time to process a JSP.	Integer	False
processingTime	Indicates the processing time that an NSJSP process takes to process all the JSPs.	Integer	False

Note. This Appendix lists only those MBeans that are specific to NSJSP and might be frequently used. To view the complete list of MBeans and the attributes associated with each MBean under a particular domain, log on to the NSJSP Manager Application. In the Compare - NSJSP MBeans page, type “*domain: **” string in the **MBean Object Name:** field and click **Compare All Attributes**. The list of MBeans for the specified domain will be displayed.

Glossary

This glossary defines terms used both in this manual and in other HP manuals. Both industry-standard terms and HP-specific terms are included.

Admin Server Class. Refers to one of the server classes configured with an installation of NSJSP. Each installation of NSJSP results in 2 server classes. One server class will host and process requests for user application and the other is used by the Admin application. The server class that is used by the Admin Web application is referred to as the Admin Server Class. With NSJSP 6.1, the name of the Servlet Server Class is prompted for during installation and the Admin Server Class is called `<svc>-adm` where `svc` is the name of the corresponding Servlet Server Class provided by the user during installation.

Admin Web application. It is a web-based application that is associated with an NSJSP installation and enables you to examine and modify parts of some configuration files.

authentication. The process of identifying an individual, usually based on a username and password. In security systems, authentication is distinct from authorization, which is the process of giving individuals access to system objects based on their identity. Authentication merely ensures that the individual is who he or she claims to be, but says nothing about the access rights of the individual.

Apache Tomcat. A Java servlets and JSP container that is developed by the Apache Software Foundation.

availability. The amount of time an application running on a Tandem system can be used effectively by a user of that application.

browser. A graphical user interface (GUI) used to access sites on the World Wide Web. Netscape, Internet Explorer, Mosaic, and Lynx are commonly used browsers.

CCITT (International Telegraph and Telephone Consultative Committee). A division of the United Nations International Telecommunications Union that coordinates standards-setting activities.

CGI. See [Common Gateway Interface \(CGI\)](#)

CERN. The European Laboratory for particle physics. The originators of the [HyperText Transport Protocol \(HTTP\)](#) and [HyperText Markup Language \(HTML\)](#) concepts.

ClassLoaderLogManager. Container specific Log Manager, which conforms to the java.util.logging Log Manager specifications.

client deployer. The client deployer environment enables you to run the ant script in the remote workstation.

CommerceNet. A consortium that was formed in Silicon Valley to promote electronic commerce over the Internet.

Common Gateway Interface (CGI). A standard protocol used as the interface between web servers and the programs these servers use to process requests from web clients.

Commons Logging. A component of Apache Commons project that provides a layer of abstraction over many popular logging implementations.

connection. The path between two protocol modules that provides reliable stream delivery service. In the Internet, a connection extends from a [Transmission Control Protocol \(TCP\)](#) module on one machine to a TCP module on another machine.

connector. Apache Tomcat component that enables Apache Tomcat to function as a web server.

container. Apache Tomcat component that processes servlets and JSPs.

cookie. A message given to a Web browser by a Web server. The browser stores the message in a text file. The message is then sent back to the server each time the browser requests a page from the server. The main purpose of cookies is to identify users and possibly prepare customized Web pages for them.

data sources. Resources that are used to perform database operations.

deployment descriptor. The `web.xml` file that contain resource definitions such as MIME types, mapping of requests to servlets, access control and servlet initialization parameters.

disk files. Standard POSIX or Guardian style disk files. The file names of POSIX disk files comply with the POSIX specifications.

distinguished name (DN). The complete name of a directory entry, consisting of the Relative Distinguished Name (RDN) of the entry and the RDNs of its superior entries.

DN. See [distinguished name \(DN\)](#)

DNS. See [Domain Name Server \(DNS\)](#).

Document Type Definition (DTD). A DTD states what tags and attributes are used to describe content in an SGML document, where each tag is allowed, and which tags can appear within other tags. For example, in a DTD one could say that LIST tags can contain ITEM tags, but ITEM tags cannot contain LIST tags. In some editors, when authors are inputting information, they can place tags only where the DTD allows. This ensures that all the documentation is formatted the same way.

Domain Name Server (DNS). A method for naming resources. The basic function of the DNS is to provide information about network objects by answering queries.

domain. Namespace used to group MBeans that are registered with an MBean server.

DTD. ISee [Document Type Definition \(DTD\)](#).

EJB. ISee [Enterprise JavaBeans \(EJB\)](#)

Enterprise JavaBeans (EJB). Enterprise JavaBeans (EJB) is a Java API developed by Sun Microsystems that defines a component architecture for multi-tier client/server systems. EJB systems allow developers to focus on the actual business architecture of the model, rather than worry about endless amounts of programming and coding needed to connect all the working parts. This task is left to EJB server vendors. Developers just design (or purchase) the needed EJB components and arrange them on the server. Because EJB systems are written in Java, they are platform independent. Being object oriented, they can be implemented into existing systems with little or no recompiling and configuring.

Ethernet. A popular [local area network \(LAN\)](#) technology invented at the Xerox Corporation Palo Alto Research Center. An Ethernet itself is a passive coaxial cable; the interconnections all contain active components. Ethernet is a best-effort delivery system that uses CSMA/CD technology. Xerox Corporation, Digital Equipment Corporation, and Intel Corporation developed and published the standard for 10 Mbps Ethernet.

fault tolerance. The ability of a HP NonStop system to continue processing despite the failure of any single software or hardware component within the system.

File Transfer Protocol (FTP). The Internet standard, high-level protocol for transferring files from one machine to another. Usually implemented as application-level programs, FTP uses the [TELNET](#) and [Transmission Control Protocol \(TCP\)](#) protocols. The server side requires a web client to supply a login identifier and password before it will honor requests.

Filter. Provides fine-grained control over what gets logged, beyond the control provided by log levels. The logging APIs support a general-purpose filter mechanism that allows application code to attach arbitrary filters to control logging output.

Formatter. Provides support for formatting LogRecord objects. This package includes two formatters, SimpleFormatter and XMLFormatter, for formatting log records in plain text or XML respectively. As with Handlers, additional Formatters may be developed by third parties.

FTP. See [File Transfer Protocol \(FTP\)](#).

gateway. A special-purpose, dedicated computer that attaches to two or more networks and routes packets from one to the other. In particular, an Internet gateway routes [Internet Protocol \(IP\)](#) datagrams among the networks to which it is connected. Gateways route packets to other gateways until they can be delivered to the final destination directly across one physical network. The term is loosely applied to any machine that transfers information from one network to another, as in mail gateway.

GESA. See [Gigabit Ethernet ServerNet Adapter \(GESA\)](#).

Gigabit Ethernet ServerNet Adapter (GESA). A single-port ServerNet adapter that provides 1000 megabits/second (Mbps) data transfer rates between HP NonStop™ systems and Ethernet LANs. A GESA can be directly installed in slots 51 through 54 of an I/O enclosure and slots 53 and 54 of a processor enclosure.

Two versions of the GESA are available:

- 3865 GESA-C (T523572): a single-port copper version compliant with the 1000 Base-T standard (802.3ab)
- 3865 GESA-F (T523572): a single-port fiber version compliant with the 1000 Base-SX standard (802.z)

Handler. Exports LogRecord objects to a variety of destinations including memory, output streams, consoles, files, and sockets. A variety of Handler subclasses exist for this purpose. Additional Handlers may be developed by third parties and delivered on top of the core platform.

hierarchical routing. Routing based on a hierarchical addressing scheme. Most Internet routing is based on a two-level hierarchy in which an Internet address is divided into a network portion and a host portion. Gateways use only the network portion until the datagram reaches a gateway that can deliver it directly. Subnetting introduces additional levels of hierarchical routing.

high-availability. Continuous availability of service that NSJSP offers.

HyperText Markup Language (HTML). The tagging language used to format HyperText documents on the World Wide Web. It is built on top of Standard Generalized Markup Language (SGML).

HyperText Transport Protocol (HTTP). The communications protocol used for transmitting data between servers and web clients (browsers) on the World Wide Web.

IEEE. See [Institute of Electrical and Electronics Engineers \(IEEE\)](#).

Institute of Electrical and Electronics Engineers (IEEE). An international industry group that develops standards for many areas of electrical engineering and computers.

Internet address. The 32-bit address assigned to hosts that want to participate in the Internet using TCP/IP. Internet addresses are the abstraction of physical hardware addresses, just as the Internet is an abstraction of physical networks. Actually assigned to the interconnection of a host to a physical network, an Internet address consists of a network portion and a host portion. The partition makes routing efficient.

Internet Protocol (IP). The Internet standard protocol that defines the Internet datagram as the unit of information passed across the Internet and that provides the basis for the Internet connectionless, best-effort packet delivery service.

Internet. Physically, a collection of packet-switching networks interconnected by gateways, along with protocols that allow them to function logically as a single, large, virtual network. When written in uppercase, INTERNET refers specifically to the DARPA Internet and the TCP/IP protocols it uses.

interoperability. The ability of software and hardware on multiple machines from multiple vendors to communicate meaningfully.

IP. See [Internet Protocol \(IP\)](#).

IPSetup. It is the Windows application that enables you to transfer IP software from the CD-ROM or other IP delivery medium (DVD) to a NonStop system.

iTP Secure WebServer. Creates Java Servlets that utilize the database and transaction services infrastructure of the HP NonStop server.

<iTP WebServer Home>. Refers to the directory where iTP Secure WebServer is installed.

J2EE. See [Java 2 Platform Enterprise Edition \(J2EE\)](#)

Java 2 Platform Enterprise Edition (J2EE). J2EE is a platform-independent, Java-centric environment from Sun for developing, building, and deploying Web-based enterprise applications online. The J2EE platform consists of a set of services, APIs, and protocols that provide the functionality for developing multitiered, Web-based applications.

Java Database Connectivity (JDBC). The Java standard for access to relational database like SQL/MX.

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.

JavaServer Page (JSP). Server side technology that enables you to develop and maintain dynamic web pages. It extends the functionality of web-based applications by providing dynamic content from a web server to a client browser over the Hypertext Transfer Protocol (HTTP).

Java Servlets. A server-side Java program that any World Wide Web browser can access. It inherits scalability and persistence from the Pathway CGI server that manages it. The Java class named servlets executes in server environments such as World Wide Web servers. The Servlet API is defined in a draft standard by Sun Microsystems.

Java Thread. A part of a program that can execute independently of other parts. Operating systems that support multithreading enable programmers to design programs whose threaded parts can execute concurrently.

java.util.logging. The java.util.logging is a logging implementation provided as part of the Java 2 Platform, Standard Edition (J2SE).

JDBC. See [Java Database Connectivity \(JDBC\)](#).

Java Management Extensions (JMX). Java technology that offers tools to manage and monitor different types of resources such as an application.

JNDI. See [Java Naming and Directory Interface \(JNDI\)](#).

Joint Photographic Expert Group (JPEG). An image format used to transmit graphics on the World Wide Web (WWW).

JPEG. See [Joint Photographic Expert Group \(JPEG\)](#).

JSP. See [JavaServer Page \(JSP\)](#)

JULI. The logging mechanism in NSJSP, which is inherited from Tomcat.

key database file. The file in which you maintain keys you generated using the keyadmin command with either the -mkpair or -keydb argument. These are the keys you use to generate certificates for software encryption. Compare [WID keyfile](#).

Key Ex change Key (KEK). An encryption key used to encrypt other keys.

LDAP. See [Lightweight Directory Access Protocol \(LDAP\)](#).

Level. Defines a set of standard logging levels that can be used to control logging output. Programs can be configured to output logging for some levels while ignoring output for others.

Lightweight Directory Access Protocol (LDAP). An application protocol for querying and modifying directory services running over TCP/IP wherein a directory is a set of objects with attributes organized in a logical and hierarchical manner.

local area network (LAN). Any physical network technology that operates at high speed (usually from tens of megabits per second to several gigabits per second) over short distances (up to a few thousand meters).

Logger. The main entity on which applications make logging calls.

LogFactory. It detects the underlying logging implementation and creates the log instances for the logging implementation detected.

Log interface. A Commons Logging component, which is an independent abstraction of the underlying logging implementation.

Log Manager. Tracks the global logging information, which includes a hierarchical namespace of the loggers.

logging.properties. A configuration file that enables you to initialize logging configuration.

Log Record. Used to pass logging requests between the logging framework and individual log handlers.

MBean attributes. A pattern in which an MBean provides the state and details of the managed resource.

MBean object name. It is a unique identifier of an MBean.

MBean server. A repository that stores all the MBeans in a JVM.

Migration. Transitioning the web applications from one NSJSP version to another NSJSP version.

Netscape. See [browser](#).

NonStop Kernel. The HP operating system, which consists of core and system services. The operating system does not include any application program interfaces (APIs).

nowait mode. In Guardian file-system operations and in some APS operations, the mode in which the called procedure initiates an input/output (I/O) operation but does not wait for it to complete before returning control to the caller. In order to make the called procedure wait for the completion of the operation, the application calls a separate procedure. Compare [wait mode](#).

NonStop Servlets for JavaServer Pages (NSJSP). NonStop Servlets for JavaServer Pages (NSJSP) are platform-independent server-side programs that programmatically extend the functionality of web-based applications by providing dynamic content from a webserver to a client browser over the HTTP protocol.

NSJSP Log Handler. The NSJSP Log Handler class offers configuration attributes for the log messages and enables you to configure the message format.

NSJSP Manager application. It is a web-based, Graphical User Interface (GUI) tool that you can use to manage an NSJSP 6.1 installation within an iTP Secure WebServer environment.

NSJSP servlet container. A servlet container is an executable program that supports servlet execution. A servlet container provides an environment in which you can deploy, execute, and manage web applications based on servlets or Java Server Pages (JSPs).

<NSJSP_HOME>. Refers to the directory where NSJSP is installed.

object name. It is the unique identifier of an MBean.

Open System Services (OSS). An open system environment available for interactive or programmatic use with the NonStop Kernel operating system. Processes that run in the OSS environment use the OSS application program interface (API); interactive users of the OSS environment use the OSS shell for their command interpreter.

OSS applications. POSIX compliant applications.

OSS. See [Open System Services \(OSS\)](#).

packet. The unit of data sent across a packet-switching network. While some Internet literature uses it to refer specifically to data sent across a physical network, other literature views the Internet as a packet-switching network and describes IP datagrams as packets.

PATHMON process. The central controlling process for a NonStop TS/MP application.

Pathway. The former name of NonStop TS/MP, a product providing transaction services for persistent, scalable, transaction-processing applications.

Pathway domain. A logical domain having multiple replicated Pathway environments grouped together.

physical layer. Layer 1 in the OSI Reference Model. This layer establishes the actual physical connection between the network and the computer equipment. Protocols at the Physical Layer include rules for the transmission of bits across the physical medium and rules for connectors and wiring.

process. A running entity that is managed by the operating system, as opposed to a program, which is a collection of code and data. When a program is taken from a file on a disk and run in a processor, the running entity is called a process.

protocol. A formal description of the message formats and rules two or more machines must follow to exchange messages. Protocols can describe low-level details of machine-to-machine interfaces (for example, the order in which the bits from a byte are sent across a wire) or high-level exchanges between application programs (for example, the way in which two programs transfer a file across the Internet). Most protocols include both intuitive descriptions of the expected interactions and more formal specifications using finite state-machine models.

QIO subsystem. A product that provides buffers and control blocks for protocol processes, including TCP/IP, TLAM, and NonStop IPX/SPX running on the same processor.

Realm. Represents a database of the information about authorized users, their passwords, and their assigned access roles.

Request for Comments (RFC). The name of a series of notes that contain surveys, measurements, ideas, techniques, and observations, along with proposed and accepted Internet protocol standards. RFCs are edited but not referenced. They are available across the Internet.

RFC. See [Request for Comments \(RFC\)](#).

sandbox. A protected memory space wherein a program cannot access outside resources such as file or network services.

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.

Secure Sockets Layer (SSL). A protocol for private communication on the World Wide Web and authentication of a web server by a web client.

servlet mapping. It is the mapping between the URL and the servlet (configured to process the requests matching that URL).

server. A process or set of processes that satisfy requests from web clients in a client-server environment.

server class. A grouping of duplicate copies of a single server program, all of which execute the same object program.

server process. A process that implements requests for an application and returns replies to the requester.

server programs. In NonStop TS/MP, programs that handle the data manipulation and data output activities for online transaction processing applications. Server programs are designed to receive request messages from requester programs; perform the desired operations, such as database inquiries or updates, security verifications, numerical calculations, or data routing to other computer systems; and return reply messages to requester programs.

servlet. A server-side Java program that any World Wide Web browser can access. It inherits scalability and persistence from the Pathway CGI server that manages it. The Java class named `servlets` executes in server environments such as World Wide Web servers. The Servlet API is defined in a draft standard by Sun Microsystems.

Servlet Server Class. Refers to one of the server classes configured with an installation of NSJSP. Each installation of NSJSP results in two server classes. One server class hosts and processes requests for user application and the other is used by the Admin application. The server class that will host and process requests for user application is referred to as the Servlet Server Class. With NSJSP 6.1, the name of the Servlet Server Class is prompted for during installation.

session. A session, also called an HTTP session, provides the means to associate an HTTP Client and an HTTP Server. This association or session, persists over multiple connections or requests or both during a given time period. Sessions are used to maintain the state and also user identity across multiple requests and connections. An example of such a state is the contents of a shopping cart, which is stored in a session object.

session object. Also called as a Java object. It is a medium to interact with client and server.

setup script. An utility used to install NSJSP.

Simple Mail Transfer Protocol (SMTP). The Internet standard protocol for transferring e-mail messages from one machine to another. SMTP specifies how two mail systems interact, and specifies the format of control messages the two mail systems exchange to transfer mail.

Single Point Management. Ability to manage multiple NSJSP server classes and user applications using a single management application.

SSL. See [Secure Sockets Layer \(SSL\)](#).

STDERR. A value of the destination attribute. Logs can be written in STDERR.

STDOUT. A value of the destination attribute. Logs can be written in STDOUT.

subnet address. An extension of the Internet addressing scheme that allows a site to use a single Internet address for multiple physical networks. Outside of the site using subnet addressing, routing continues as usual by dividing the destination address into an Internet portion and local portion. Gateways and hosts inside a site using subnet addressing interpret the local portion of the address by dividing it into a physical network portion and host portion.

subsystem. The software or hardware or both facilities that provide users with access to a set of communications services.

TACL. See Tandem Advanced Command Language (TACL).

Tandem Advanced Command Language (TACL). The user interface to the Tandem NonStop Kernel in the Guardian environment. The TACL product is both a command interpreter and a command language.

Transmission Control Protocol (TCP). The Internet standard transport-level protocol that provides the reliable, full-duplex stream service on which many application protocols depend. TCP allows a process on one machine to send a stream of data to a process on another. It is connection-oriented, in the sense that before transmitting data participants must establish a connection. Software implementing TCP usually resides on the operating system and uses the [Internet Protocol \(IP\)](#) to transmit information across the Internet. It is possible to terminate (shut down) one direction of flow across a TCP connection, leaving a one-way (simplex) connection. The Internet protocol suite is often referred to as TCP/IP because TCP is one of the two most fundamental protocols.

TELNET. The Internet standard protocol for remote terminal connection service. TELNET allows a user at one site to interact with remote timesharing systems at another site just as if the user's terminal is connected directly to the remote machine. That is, the user invokes a TELNET application program that connects to a remote machine, prompts for a login ID and password, and then passes keystrokes from the user's terminal to the remote machine and displays output from the remote machine on the user's terminal.

TLD. See [Top-Level Domain \(TLD\)](#)

Top-Level Domain (TLD). Refers to the suffix attached to Internet domain names. There are a limited number of predefined suffixes, and each one represent a top-level domain. Current top-level domains include:

- com – Commercial businesses; this is the most common TLD
- gov – U.S. government agencies
- edu – Educational institutions, such as universities
- org – Organizations (mostly nonprofit)
- mil – Military
- net – Network organizations

Unicode. The 16-bit character encoding used by Java for the char and java.lang.String data types.

user database. A database that contains user information, such as user names, passwords, groups, and roles.

wait mode. In the NonStop Kernel operating system, the mode in which the called procedure waits for the completion of an input/output (I/O) operation before returning a condition code to the caller. Compare [nowait mode](#).

Web Container. a Java runtime environment that manages the lifecycle of servlets and JSP.

Web clients. Programs that execute on IBM-compatible PC, Apple Macintosh, or Unix platforms, among others. They provide a graphic user interface (GUI) for access to documents and programs on the Web. A web browser is the most familiar example of a web client.

Web server. Web servers are programs that execute on a variety of server platforms. These include IBM-compatible servers, Apple Macintosh servers, Unix servers, and a large number of proprietary hosts. Web server functions can be divided into two parts. A file server part performs normal file server functions such as file transfer and buffering. A message switching facility allows messages from web clients to be forwarded to application programs.

WID keyfile. The file in which you maintain keys you generated using the keyadmin command with the -websafegen argument. These are the keys you use to generate certificates for hardware encryption. Compare [key database file](#).

World Wide Web (WWW) protocols. The WWW protocols were first defined by the CERN project in Switzerland and were later extended by a number of groups, most notably by the National Center for SuperComputing Applications (NCSA) at the University of Illinois. These WWW protocols were originally developed to improve communications over the Internet by providing the ability to access and display web-client

hardware-independent documents that not only contained ASCII text but that also contained pictures, graphics, and voice and video elements. In addition to accessing documents, the WWW protocols can also be used to provide document searching facilities and also interaction with user-written or vendor-provided servers.

WWW. See [World Wide Web \(WWW\) protocols](#).

XML. Short for Extensible Markup Language, a specification developed by the W3C. XML is a pared-down version of SGML, designed especially for Web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations.

zero downtime. Continuous availability of service that NSJSP offers without any downtime.

Index

A

- accessDeclaredMembers permission [8-39](#)
- access,failure value [8-41](#)
- Admin Server Class [3-2](#)
- Admin Web Application
 - administering user definitions [4-79](#)
 - groups [4-81](#)
 - roles [4-82](#)
 - users [4-80](#)
 - architecture of [4-57](#)
 - features of [4-59](#)
 - login and security of [4-59](#)
- Admin Web Application operations
 - administering Host [4-67](#)
 - administering Realm [4-71](#)
 - administering resources [4-74](#)
 - data sources [4-74](#)
 - environment entry [4-76](#)
 - mail session [4-75](#)
 - resource link [4-79](#)
 - user database [4-77](#)
 - administering server [4-62](#)
 - administering valve [4-72](#)
- Apache Axis2 [1-3](#)
- Apache Tomcat [1-2](#), [1-13](#)
- archiveDirectory [5-7](#)
- Attributes of Server element [3-40](#)

B

- BANK_CATALOG [3-19](#)
- base64 [8-4](#)
- base64 encoding format [8-3](#)

C

- Catalina [1-13](#)
- catalina.base [3-22](#)
- catalina.home [3-22](#)

- CATALINA_HOME environment variables [8-35](#)
- certificates [8-1](#)
- CLASSPATH [3-17](#)
- CLIENT-CERT [8-7](#)
- codeBase entry [8-35](#)
- CombinedRealm [8-8](#)
- Commons Logging [5-30](#)
- comparing management applications [4-92](#)
- components
 - in NSJSP servlet container [A-1](#)
- com.tandem.servlet.CONTEXT_PREFIXES [3-26](#)
- com.tandem.servlet.nsjsp [3-23](#)
- Considerations [8-1](#)
- context definition [7-20](#)
- context initialization parameters [4-25](#)
- context.xml [7-14](#)
- Coyote [1-13](#)
- Creating an NSJSP installation [2-16/2-18](#)

D

- data theft [8-1](#)
- datePattern [5-7](#)
- Debugging Using Eclipse Platform [6-2](#)
- Debugging Using Java Debugger Tool [6-1](#)
- Default context [7-19](#)
- deploying applications
 - at startup [4-90](#)
 - on a running NSJSP server [4-91](#)
- deploying web applications from desktop [4-55](#)
- deploying web applications from server [4-53](#)
- deployment descriptor [3-3](#)
- DiscardFileMapHistory [3-24](#)

E

- EnableJMXProxyServlet [3-24](#)

F

FileHandler [5-10](#)
Filemap [3-33](#)
Filemap Directive [3-29](#)
filemaps.config [3-33](#)
formatter [5-2](#)
form-based authentication
 accessing a web resource [8-6](#)

G

Generic servlet.config [3-6](#)

H

handler [5-2](#)
Hibernate [1-3](#)
High-availability [1-11](#)
HTTP/1.0 specification [8-3](#)

I

Installation directories [2-20](#)
Installation-Specific servlet.config [3-7](#)
Installing NSJSP
 iTP Secure WebServer environment
 for [2-14](#)
 prerequisites for [2-1](#)
Introduction [1-1](#)
iTP Secure Webserver
 configuration files required by [2-26](#)
iTP Secure WebServer operations [4-85](#)
iTP_catalina.policy file [8-35](#)
I/O Info [4-38](#)

J

JAAS_CONFIG_FILE [3-13](#)
Java
 security manager [8-35](#)
Java Authentication and Authorization
Service (JAAS) [8-8](#)
Java Database Connectivity (JDBC) [8-8](#)

Java Management Extensions (JMX) [A-1](#)
Java Naming and Directory Interface
(JNDI) [8-8](#)
Java Servlets [1-2](#)
JavaServer Pages [1-2](#)
javax.management.builder.initial [3-21](#)
java.compiler [3-20](#)
java.io.tmpdir [3-21](#)
java.policy file [8-35](#)
java.security.debug property [8-41](#)
java.util.logging [5-1](#)
java.util.logging.config.file [3-21](#)
java.util.logging.manager [3-20](#)
JAVA_HOME [3-18](#)
JAVA_HOME environment variable [8-35](#)
jdbc.config [3-35](#)
JDBC/MX T2 driver [2-2](#)
JDBC/MX T4 driver [2-2](#)
JREHOME [3-18](#)
JULI [5-5](#)
JVCP [3-15](#)
JVM_POLICY_FILE [3-12](#)
j_password [8-6](#)
j_security_check [8-6](#)
j_username [8-6](#)

L

LINKDEPTH [3-28](#)
Load balancing [1-11](#)
log files [5-27](#)
Log interface [5-29](#)
Log Manager [5-3](#)
log rollover [5-6](#)
 archiveDirectory [5-7](#)
 datePattern [5-7](#)
 maxFileSize [5-6](#)
LogFactory [5-28](#)
Logger element [7-23](#)
Loggers [5-1](#)
logging architecture [5-1](#)
logging.properties [5-20](#)

login form

 action attribute [8-6](#)

 HTML fields [8-6](#)

M

Manager Web Application [4-85](#)

managing web application [4-26](#)

 reloading [4-28](#)

 starting [4-27](#)

 stopping [4-26](#)

 undeploying [4-29](#)

maxFileSize [5-6](#)

MAXLINKS [3-28](#)

MAXSERVERS [3-28](#), [3-32](#)

maxWaitTimeSecs [3-26](#)

MBeans

 domain [A-2](#)

 in NSJSP container [A-10](#)

 object name [A-2](#)

 parameters in the NSJSP MBeans
 page [4-45](#)

Multiple NSJSP installations [2-26](#)

MyFaces [1-3](#)

N

NonStop SQL

 database [3-84](#)

NSJSP [1-2](#)

 ServerClass [3-3](#)

NSJSP Formatter [5-5](#)

NSJSP installation

 directory structure of [2-20](#)

NSJSP Log Handler [5-5](#)

NSJSP Manager

 architecture of [4-104](#)

 features of [4-2](#), [4-6](#)

 installtion of

[2-21/2-22](#)

 removing installation directory [2-25](#)

server class of

 including multiple Hosts [4-100](#)

supporting multiple NSJSP

installations [4-101](#)

NSJSP Manager Application

 comparing MBean attribute values [4-46](#)

 logging in to [4-4](#)

 modifying MBean attribute values [4-50](#)

 selecting server class and Host [4-5](#)

server class operations

 freeze [4-42](#)

 start [4-41](#)

 stop [4-39](#)

 thaw [4-43](#)

viewing application details for each
NSJSP process [4-14](#)

viewing application summary [4-10](#)

viewing configuration parameters [4-34](#)

viewing context descriptor details [4-21](#)

viewing deployment descriptor
details [4-22](#)

viewing filter details [4-24](#)

viewing HTTP method statistics [4-20](#)

viewing initialization parameters [4-25](#)

viewing MBeans [4-44](#)

viewing NSJSP connector
statistics [4-33](#)

viewing NSJSP process details [4-32](#)

viewing server class information [4-30](#)

viewing server class statistics [4-36](#)

viewing servlet mappings [4-23](#)

viewing sessions for an
application [4-16](#)

viewing URI statistics for an
application [4-18](#)

viewing web application statistics [4-8](#)

NSJSP Manager Application Interface

attributes

 in the Applications page [4-10](#)

 in the Filters page [4-25](#)

in the HTTP Method Statistics page [4-20](#)
 in the NSJSP Connector Stats page [4-34](#)
 in the NSJSP Information page [4-31](#)
 in the Process View page [4-16](#)
 in the Server Class Processes page [4-32](#)
 in the Server Class Statistics page [4-38](#)
 in the Servlet Mappings page [4-23](#)
 in the URI Statistics page [4-19](#)
 in the Web Application Deployment from Desktop page [4-56](#)
 in the Web Application Deployment from Server page [4-54](#)
 using the Application Summary page [4-12](#)
 using the In-Memory Sessions page [4-18](#)

operations

using the Application Summary page [4-11](#)
 using the Applications page [4-9](#)
 using the Compare - NSJSP MBean page [4-49](#)
 using the In-Memory Sessions page [4-17](#)
 using the Process View page [4-15](#)

parameters

in the Compare - NSJSP MBean page [4-50](#)
 in the Modify MBean page [4-52](#)
 in the NSJSP MBeans page [4-45](#)

NSJSP Manager Web Application

security [8-41](#)

NSJSP Servlet Container [3-1](#)

nsjspadmin.config [3-30](#)

NSJSPLockoutRealm [8-8](#)

NSJSPLogHandler [5-11](#)

NSJSP_CONFIG_FILE [3-19](#)

NSJSP_DLL_PATH [3-17](#)

NSJSP_HOME [3-12](#)

NSJSP_JAAS_CONFIG [3-14](#)

NSJSP_SECMGR [3-13](#)

NSJSP_SECMGR_POLICY [3-13](#)

NUMSTATIC [3-28](#), [3-32](#)

O

online-upgrade [1-11](#)

OSS

command [3-85](#)

P

package.definition

property [8-40](#)

permissions, assigning additional to web applications [8-35](#)

persistent session data [3-85](#)

public key certificate [8-2](#)

Public Key Certificate (PKC) [8-7](#)

Q

Queue Info [4-37](#)

R

Region Directive [3-28](#)

Removing NSJSP installation directory [2-25](#)

Running the IPSetup program [2-2/2-13](#)

Running the setup script [2-14/2-15](#)

S

SaveSessionOnCreation [3-25](#)

Scalability [1-11](#)

secure channel [8-2](#)

Secure Sockets Layer [8-2](#)

security considerations [8-1](#)

security exception [8-39](#)

server class configuration parameters [4-36](#)

[server.xml 3-37](#)
[server_objectcode 3-11](#)
[Servlet server class 3-1](#)
[SERVLET_BANK 3-16](#)
[Session 3-2](#)
[Session object 3-2](#)
[SessionBasedCookieExpiry 3-22](#)
[SessionBasedLoadBalancing 3-22](#)
[setup script 1-4](#)
[signedBy entry 8-35](#)
[Spring 1-3](#)
[SSL 8-2, 8-3](#)
[StaticContentFilter 7-13](#)
[STDERR 5-27](#)
[Stderr 3-28](#)
[STDOUT 5-27](#)
[Stdout 3-28](#)

T

[TANDEM_FILEMAPS_CONFIG 3-18](#)
[TANDEM_HTTPD_SC_NAME 3-32](#)
[TANDEM_RECEIVE_DEPTH 3-19](#)
[TANDEM_SERVLET_SC_NAME 3-32](#)
[TANDEM_SERVLET_SC_PATH 3-32](#)
[TS/MP Pathway domain 4-102](#)
[TS/MP Server Class operations](#)
 [start 4-89](#)
 [stop 4-87](#)

U

[Updating NSJSP installation 2-24](#)
[User Application-Specific Filemap Definitions 3-34](#)
[user credential 8-3](#)

V

[Verifying NSJSP Manager installation 2-23](#)

W

[web.xml 7-13](#)

X

[Xms 3-20](#)
[Xmx 3-19](#)
[Xss 3-20](#)
[X.509 certificate 8-2](#)

Special Characters

[\\${catalina.home} property 8-35](#)
[\\${java.home} property 8-35](#)
[-Djava.security.manager option 8-35](#)
[_RLD_LIB_PATH 3-18](#)