

# Open Source Java Frameworks on NonStop User's Guide (Spring 3.0.2, Hibernate 3.5.1, MyFaces 2.0.2, Axis2/Java 1.5.2)



HP Part Number: 641664-002

Published: February 2012

Edition: J06.03 and subsequent J-series RVUs and H06.06 and subsequent H-series RVUs.

© Copyright 2011 Hewlett-Packard Development Company, L.P.

#### Legal Notice

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

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

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

Open Software Foundation, OSF, the OSF logo, OSF/1, OSF/Motif, and Motif are trademarks of the Open Software Foundation, Inc. OSF MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE OSF MATERIAL PROVIDED HEREIN, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. OSF shall not be liable for errors contained herein or for incidental consequential damages in connection with the furnishing, performance, or use of this material.

JBoss and Hibernate are registered trademarks and servicemarks of Red Hat, Inc.

Spring Framework is an open source project and is a trademark of Interface 21.

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

© 1987, 1988, 1989 Carnegie-Mellon University. © 1989, 1990, 1991 Digital Equipment Corporation.

© 1985, 1988, 1989, 1990 Encore Computer Corporation. © 1988 Free Software Foundation, Inc. © 1987, 1988, 1989, 1990, 1991

Hewlett-Packard Company. © 1985, 1987, 1988, 1989, 1990, 1991, 1992 International Business Machines Corporation. © 1988, 1989

Massachusetts Institute of Technology. © 1988, 1989, 1990 Mentat Inc. © 1988 Microsoft Corporation. © 1987, 1988, 1989, 1990, 1991,

1992 SecureWare, Inc. © 1990, 1991 Siemens Nixdorf Informationssysteme AG. © 1986, 1989, 1996, 1997 Sun Microsystems, Inc. © 1989,

1990, 1991 Transarc Corporation. OSF software and documentation are based in part on the Fourth Berkeley Software Distribution under license

from The Regents of the University of California. OSF acknowledges the following individuals and institutions for their role in its development: Kenneth

C.R.C. Arnold, Gregory S. Couch, Conrad C. Huang, Ed James, Symmetric Computer Systems, Robert Elz. © 1980, 1981, 1982, 1983, 1985,

1986, 1987, 1988, 1989 Regents of the University of California.

# Contents

<b>About This Document.....</b>	<b>8</b>
Supported Release Version Updates (RVUs).....	8
Intended Audience.....	8
New and Changed Information in This Edition.....	8
Document Organization.....	8
Notation Conventions.....	11
General Syntax Notation.....	11
Notation for Messages.....	13
Notation for Management Programming Interfaces.....	14
General Syntax Notation.....	14
Related Information.....	16
Publishing History.....	16
HP Encourages Your Comments.....	16
<b>1 Introduction.....</b>	<b>17</b>
Prerequisites.....	18
NonStop System.....	18
Windows System.....	23
<b>I Spring Framework.....</b>	<b>30</b>
<b>2 Spring Overview.....</b>	<b>33</b>
Spring Projects.....	33
Spring Framework.....	33
Spring Web Flow.....	34
Spring Applications on NonStop.....	34
<b>3 Installing the Spring Framework.....</b>	<b>36</b>
Prerequisites.....	36
NonStop System.....	36
Windows System.....	36
Installing Spring Framework Libraries on NonStop.....	36
Downloading the Spring Distribution on Windows.....	37
Downloading Spring Dependency JAR Files on Windows.....	38
Downloading Sample Spring Applications on Windows.....	38
Copying Spring Runtime Libraries from Windows to NonStop.....	39
Deploying and Running Sample Spring Applications on NonStop.....	39
PetClinic.....	40
JPetStore.....	46
<b>4 Configuring Spring Applications on NonStop Systems.....</b>	<b>52</b>
NonStop Platform Configurations.....	52
Determining the Application Parameters.....	52
Determining the Maximum Capacity of NSJSP Instance.....	52
Configuring iTP WebServer for Spring Applications.....	53
Configuring NSJSP for Spring Applications.....	57
Spring Framework Configurations.....	62
Configuring JDBC Driver for SQL/MX Database.....	63
Configuring Database Transaction Management.....	66
Connection Pooling.....	72
Module File Caching Configurations.....	73
Configuring NonStop SQL/MX DataSource for MFC.....	73
Modifying the Spring Application.....	74

<b>5 Getting Started with Spring.....</b>	<b>75</b>
Prerequisites.....	75
NonStop System.....	75
Windows System.....	75
Overview of EmplInfo.....	75
Developing EmplInfo on Windows using the Eclipse Galileo IDE.....	76
Deploying EmplInfo on NonStop.....	129
Running EmplInfo on NonStop.....	133
<b>A EmplInfo Database Script.....</b>	<b>134</b>
<b>B Customizing Sample Applications.....</b>	<b>135</b>
Customizing PetClinic.....	135
Added Directories.....	135
Added File.....	135
Modified Files.....	136
Customizing JPetStore.....	139
Added File.....	139
Modified Files.....	139
<b>C JDBC Configuration.....</b>	<b>144</b>
<b>D Installing Spring Web Flow.....</b>	<b>146</b>
Downloading Spring Web Flow Distribution on Windows.....	146
Copying Spring Web Flow Runtime Libraries from Windows to NonStop.....	146
<b>II Hibernate Framework.....</b>	<b>148</b>
<b>6 Hibernate Overview.....</b>	<b>151</b>
Hibernate Projects.....	151
Hibernate Annotations.....	151
Hibernate EntityManager.....	151
Hibernate Applications on NonStop.....	152
<b>7 Installing the Hibernate Framework.....</b>	<b>153</b>
Prerequisites.....	153
NonStop system.....	153
Windows system.....	153
Installing Hibernate Framework Libraries on NonStop.....	153
Downloading the Hibernate Distribution on Windows.....	154
Downloading Hibernate Dependency JAR Files using Maven.....	155
Including the Hibernate Dialect for the SQL/MX database to the Hibernate Distribution.....	157
Copying the Hibernate Distribution from Windows to NonStop.....	157
Deploying and Running Sample Hibernate Applications on NonStop.....	158
Caveat Emptor.....	159
The EventManager Web Application .....	163
<b>8 Configuring Hibernate Applications on NonStop Systems.....</b>	<b>170</b>
NonStop Platform Configurations.....	170
Determining the Application Parameters.....	170
Determining the Maximum Capacity of NSJSP Instance.....	170
Configuring iTP WebServer for Hibernate Applications.....	171
Configuring NSJSP for Hibernate Applications.....	175
Hibernate Framework Configurations for NonStop Systems.....	180
JDBC Driver for SQL/MX Database.....	181
Database Transaction Management.....	188
Connection Pooling.....	189
Module File Caching Configurations.....	190
Configuring NonStop SQL/MX DataSource for MFC.....	190

Modifying the Hibernate Application.....	191
Enabling Browse Access.....	191
Hibernate Applications.....	191
Spring and Hibernate Applications.....	192
<b>9 Getting Started with Hibernate.....</b>	<b>196</b>
Prerequisites.....	196
NonStop System.....	196
Windows System.....	196
Overview of EmployeeInfo.....	196
Developing EmployeeInfo on Windows using the Eclipse Galileo IDE.....	197
Setting Up the NonStop Environment.....	218
Running EmployeeInfo on NonStop.....	220
<b>E Customizing Sample Applications.....</b>	<b>221</b>
Customizing Caveat Emptor.....	221
Added Files.....	221
Modified Files.....	223
Customizing EventManager.....	223
Added Directories.....	223
Added Files.....	224
Modified File.....	225
<b>F JDBC Configuration.....</b>	<b>227</b>
<b>G Hibernate Environment Setup Script.....</b>	<b>228</b>
<b>III MyFaces Framework.....</b>	<b>229</b>
<b>10 MyFaces Overview.....</b>	<b>231</b>
MyFaces Projects.....	231
MyFaces Trinidad.....	231
MyFaces Tomahawk.....	231
MyFaces Application on NonStop.....	232
<b>11 Installing MyFaces Framework.....</b>	<b>233</b>
Prerequisites.....	233
NonStop System.....	233
Windows System.....	233
Downloading MyFaces Framework Libraries on Windows.....	233
Deploying and Running Sample MyFaces Application on NonStop.....	234
myfaces-components.....	234
<b>12 Configuring MyFaces Applications on NonStop Systems.....</b>	<b>240</b>
NonStop Platform Configurations.....	240
Determining the Application Parameters.....	240
Determining the Maximum Capacity of NSJSP Instance.....	240
Configuring iTP WebServer for MyFaces Applications.....	241
Configuring NSJSP for MyFaces Applications.....	245
<b>13 Getting Started with MyFaces.....</b>	<b>252</b>
Prerequisites.....	252
NonStop System.....	252
Windows System.....	252
Overview of SkinSelector.....	252
Developing SkinSelector on Windows using the Eclipse Galileo IDE.....	252
Deploying SkinSelector on NonStop.....	278
Running SkinSelector on NonStop.....	282

<b>H</b>	<b>Installing MyFaces Trinidad Framework Libraries on NonStop.....</b>	<b>285</b>
	Downloading MyFaces Trinidad Distribution.....	285
	Copying MyFaces Trinidad Runtime Libraries from Windows to NonStop.....	285
<b>I</b>	<b>Installing MyFaces Tomahawk Framework Libraries on NonStop.....</b>	<b>286</b>
	Downloading MyFaces Tomahawk Distribution on Windows.....	286
	Copying MyFaces Tomahawk Runtime Libraries from Windows to NonStop.....	286
<b>IV</b>	<b>Axis2/Java Framework.....</b>	<b>287</b>
	<b>14 Axis2/Java Overview.....</b>	<b>289</b>
	Apache Axis2/Java Project.....	289
	Axis2/Java Framework.....	289
	Axis2/Java Modules.....	289
	Axis2/Java Applications on NonStop.....	289
	<b>15 Installing Axis2/Java Framework.....</b>	<b>292</b>
	Prerequisites.....	292
	NonStop System.....	292
	Windows System.....	292
	Installing Axis2/Java on NonStop.....	292
	Downloading the Axis2/Java Distribution on Windows.....	292
	Building the Axis2/Java Web Archive using Standard Binary Distribution.....	293
	Deploying Axis2/Java as a Web Application on NonStop.....	294
	Running Axis2/Java on NonStop.....	296
	Installing the Sandesha2 Module.....	297
	Downloading the Sandesha2 Distribution on Windows.....	297
	Modifying the Configuration File on Windows.....	298
	Copying the Sandesha2 Binaries to the Axis2/Java Directory.....	298
	Modifying the Build File on Windows.....	298
	Building the Axis2/Java Web Archive with Sandesha2.....	299
	Deploying the Axis2/Java Web Archive with Sandesha2.....	299
	Installing the Rampart Module.....	299
	Downloading the Axis2/Java Distribution on Windows.....	299
	Copying the Rampart Binaries to Axis2/Java.....	300
	Building the Axis2/Java Web Archive with Rampart.....	300
	Deploying the Axis2/Java Web Archive with Rampart.....	300
	Deploying and Running Axis2/Java Sample Applications on NonStop.....	300
	FaultHandling.....	301
	MTOM.....	303
	<b>16 Configuring Axis2/Java Applications on NonStop Systems.....</b>	<b>307</b>
	NonStop Platform Configurations.....	307
	Determining the Application Parameters.....	307
	Determining the Maximum Capacity of an NSJSP Instance.....	307
	Configuring iTP WebServer for Axis2/Java Applications.....	308
	Configuring NSJSP for Axis2/Java Applications.....	312
	Axis2/Java Framework Configurations for NonStop Systems.....	317
	Global Configuration.....	317
	Service Configuration.....	318
	Module Configuration.....	318
	<b>17 Getting Started with Axis2/Java.....</b>	<b>319</b>
	Prerequisites.....	319
	NonStop System.....	319
	Windows System.....	319
	Overview of TemperatureConverter.....	319
	Code-First approach.....	319

Contract-First approach.....	336
<b>V Integrating Frameworks.....</b>	<b>346</b>
<b>  18 Using Spring Transaction Manager.....</b>	<b>348</b>
Why Transaction Management is required.....	348
Spring Transaction Management.....	348
Declarative Transaction Management.....	348
Programmatic Transaction Management.....	348
Example of Using Spring Transaction Manager.....	348
Modifying EmplInfo on Windows using Eclipse Galileo IDE.....	349
Deploying EmplInfo on NonStop.....	362
Running EmplInfo on NonStop.....	366
<b>  19 Integrating Hibernate into Spring.....</b>	<b>367</b>
Why Integrate Hibernate into Spring.....	367
Example of Integrating Hibernate into Spring.....	367
Modifying EmplInfo on Windows using Eclipse Galileo IDE.....	368
Deploying EmplInfo on NonStop.....	379
Running EmplInfo on NonStop.....	382
<b>  20 Integrating JPA with Hibernate into Spring.....</b>	<b>383</b>
Why Integrate JPA with Hibernate in Spring.....	383
Example of Integrating JPA with Hibernate into Spring.....	383
Modifying EmplInfo on Windows using Eclipse Galileo IDE.....	384
Deploying EmplInfo on NonStop.....	394
Running EmplInfo on NonStop.....	397
<b>  21 Integrating Axis2/Java into Spring.....</b>	<b>398</b>
Why Integrate Axis2/Java into Spring.....	398
Example of Integrating Axis2/Java with Spring.....	398
Modifying EmplInfo on Windows using Eclipse Galileo IDE.....	399
Deploying EmplInfo on NonStop.....	404
Running EmplInfo Web Service on NonStop.....	408
Developing EmplInfoClient on Windows.....	408
Running EmplInfoClient on Windows.....	416
<b>  22 Integrating MyFaces into Spring.....</b>	<b>421</b>
Why Integrate MyFaces into Spring.....	421
Example of Integrating MyFaces into Spring.....	421
Modifying EmplInfo on Windows using Eclipse Galileo IDE.....	422
Deploying EmplInfo on NonStop.....	444
Running EmplInfo on NonStop.....	448
<b>  23 Frequently Asked Questions.....</b>	<b>458</b>
<b>Glossary.....</b>	<b>461</b>
<b>Index.....</b>	<b>464</b>

# About This Document

This manual provides information on how Spring, Hibernate, MyFaces, and Axis2/Java frameworks can be used to develop applications that can run on the HP NonStop™ platform. The frameworks are described with respect to their installation, configuration, and use on NonStop systems. For detailed information about a framework, see the links to online and public resources provided in the relevant sections.

It contains instructions and guidelines on the following:

- Installing and configuring the frameworks on NonStop
- Customizing sample applications so that they can be deployed under NSJSP
- Developing sample web applications using the frameworks
- Integrating Spring with Hibernate, MyFaces, and Axis2/Java frameworks

## **NOTE:**

- This manual does not explain the frameworks in detail and aims at providing information to get you familiarized with these frameworks. The frameworks have not been modified or customized for use on NonStop. For more information on these frameworks, links to online resources are provided in relevant sections.
- This manual includes code snippets and code for sample applications for all frameworks. HP recommends that you copy the code to a text editor so that unwanted special and ASCII characters are not copied along with the code you want to use to build or test your applications.

## Supported Release Version Updates (RVUs)

This manual supports J06.03 and all subsequent J-series RVUs, and H06.06 and all subsequent H-series RVUs, until otherwise indicated by its replacement publications.

## Intended Audience

This manual is intended for application programmers who want to use Java-based web applications developed using Spring, Hibernate, MyFaces, or Axis2/Java. To use this product, the reader must be familiar with Spring, Hibernate, MyFaces, Axis2/Java, iTP WebServer, NSJSP, and the NonStop SQL/MX database.

## New and Changed Information in This Edition

- Added the “Enabling Browse Access” (page 191) section.
- Updated a note in “Downloading Hibernate Dependency JAR Files using Maven” (page 155).

## Document Organization

This manual is organized as follows:

### Chapter 1: Introduction

This chapter provides an overview of the advantages of Java frameworks on NonStop systems. It also lists the

	prerequisite software and hardware required to install and configure Hibernate and Spring on NonStop systems.
<b>Part I: Spring Framework</b>	<p><b>Part I mainly contains instructions and guidelines on the following:</b></p> <ul style="list-style-type: none"> <li>• <b>Installing and configuring Spring on NonStop</b></li> <li>• <b>Developing sample web applications using Spring</b></li> <li>• <b>Customizing Spring sample applications so that they can be deployed under NSJSP</b></li> </ul>
Chapter 2: Spring Overview	This chapter provides an overview of the following: <ul style="list-style-type: none"> <li>• Spring projects certified for use on NonStop</li> <li>• Deployment environment for a web application that is written using Spring and accesses data from the SQL/MX database.</li> </ul>
Chapter 3: Installing the Spring Framework	This chapter helps you to install Spring framework libraries and enables you to deploy and run sample Spring applications on your NonStop system.
Chapter 4: Configuring Spring Applications on NonStop Systems	This chapter provides information about configuring a Spring web application on NonStop systems.
Chapter 5: Getting Started with Spring	This chapter helps you to develop a web application on your Windows system using the Spring framework, and deploy and run the web application on your NonStop system.
Appendix A: EmplInfo Database Script	This appendix contains the EmplInfo database script.
Appendix B: Customizing Sample Applications	This appendix details the customizations made in the Spring sample applications: PetClinic and JPetStore.
Appendix C: JDBC Configuration	This appendix describes the consolidated JDBC Type 2 driver configuration and the JDBC Type 4 driver configuration in the applicationContext.xml and jdbc.properties files.
Appendix D: Installing Spring Web Flow	This appendix describes the steps required to install Spring Web Flow on your NonStop system.
<b>Part II: Hibernate Framework</b>	<p><b>Part II mainly contains instructions and guidelines on the following:</b></p> <ul style="list-style-type: none"> <li>• <b>Installing and configuring Hibernate on NonStop</b></li> <li>• <b>Developing sample Java applications using Hibernate</b></li> <li>• <b>Customizing Hibernate sample applications so that they can be deployed under NSJSP</b></li> </ul>
Chapter 6: Hibernate Overview	This chapter provides an overview of the following: <ul style="list-style-type: none"> <li>• Hibernate projects certified for use on NonStop</li> <li>• Message flow in an application that uses Hibernate on NonStop</li> </ul>
Chapter 7: Installing the Hibernate Framework	This chapter helps you to install Hibernate framework libraries, and enables you to deploy and run sample Hibernate applications on your NonStop system.
Chapter 8: Configuring Hibernate Applications on NonStop Systems	This chapter provides information about configuring applications that use Hibernate on NonStop systems.
Chapter 9: Getting Started with Hibernate	This chapter helps you to develop a Java application on your Windows system using Hibernate, and set up and run the Java application on your NonStop system.

Appendix E: Customizing Sample Applications	This appendix details the customizations made in the Hibernate sample applications: Caveat Emptor and EventManager.
Appendix F: JDBC Configuration	This appendix describes the consolidated JDBC Type 2 driver configuration and the JDBC Type 4 driver configuration in the <code>hibernate.cfg.xml</code> and <code>hibernate.properties</code> files.
Appendix G: Hibernate Environment Setup Script	This appendix contains the <code>ei_setenv</code> script.
<b>Part III: MyFaces Framework</b>	<b>Part III describes the procedure to customize the MyFaces framework to work on a NonStop system.</b>
Chapter 10: MyFaces Overview	This chapter provides an overview of the MyFaces framework, its architecture, key features, advantages, and message flow.
Chapter 11: Installing MyFaces Framework	This chapter helps you to install the MyFaces application on your NonStop system.
Chapter 12: Configuring MyFaces Applications on NonStop Systems	This chapter helps you to configure the MyFaces application on your NonStop system.
Chapter 13: Getting Started with MyFaces	This chapter helps you to develop a web application on your Windows system using MyFaces, and deploy and run the web application on your NonStop system.
Appendix H: Installing MyFaces Trinidad Framework Libraries on NonStop	This appendix describes the steps required to install MyFaces Trinidad Framework Libraries on your NonStop system.
Appendix I: Installing MyFaces Tomahawk Framework Libraries on NonStop	This appendix describes the steps required to install MyFaces Tomahawk Framework Libraries on your NonStop system.
<b>Part IV: Axis2/Java Framework</b>	<b>Part IV describes the procedure to customize the Axis2/Java framework to work on a NonStop system.</b>
Chapter 14: Axis2/Java Overview	This chapter provides an overview of the Axis2/Java framework, its architecture, key features, advantages, and message flow.
Chapter 15: Installing Axis2/Java Framework	This chapter helps you to install the Axis2/Java framework and enables you to develop, set up, deploy, and run the web application and its client on your NonStop system.
Chapter 16: Configuring Axis2/Java Applications on NonStop Systems	This chapter helps you to configure the Axis2/Java application on your NonStop system.
Chapter 17: Getting Started with Axis2/Java	This chapter helps you to develop a web application on your Windows system using Axis2/Java, and deploy and run the web application on your NonStop system.
<b>Part V: Integrating Frameworks</b>	<b>Part III helps you to use the Spring Transaction Manager and integrate Spring with Hibernate.</b>
Chapter 18: Using Spring Transaction Manager	This chapter helps you to use the Spring Transaction Manager.
Chapter 19: Integrating Hibernate into Spring	This chapter helps you to integrate Hibernate into the Spring framework.
Chapter 20: Integrating JPA with Hibernate into Spring	This chapter helps you to integrate JPA with Hibernate into the Spring framework.
<b>Chapter 23: Frequently Asked Questions</b>	<b>This chapter provides answers to some frequently asked questions.</b>

# Notation Conventions

## 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

### *Italic Letters*

Italic letters, regardless of font, indicate variable items that you supply. Items not enclosed in brackets are required. For example:

*file-name*

### Computer Type

Computer type letters indicate:

- C and Open System Services (OSS) keywords, commands, and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:  
Use the cextdecs.h header file.
- Text displayed by the computer. For example:

Last Logon: 14 May 2006, 08:02:23

- A listing of computer code. For example

```
if (listen(sock, 1) < 0)
{
    perror("Listen Error");
    exit(-1);
}
```

### 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.

### [ ] Brackets

Brackets enclose optional syntax items. For example:

TERM [\i{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 , error ) ;	!i !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 , filename2:length ) ; !i:i !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 , [ filename:maxlen ] ) ; !i !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.

### *Italic Text*

Italic text indicates 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

## 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:

SELECT

### *Italic Letters*

Italic letters, regardless of font, 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 case-sensitive keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:  
myfile.sh

## 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.

## [ ] Brackets

Brackets enclose optional syntax items. For example:

DATETIME [*start-field* TO] *end-field*

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:

DROP SCHEMA *schema* [CASCADE]  
[RESTRICT]

DROP SCHEMA *schema* [ CASCADE | RESTRICT ]

## { } Braces

Braces enclose required syntax items. For example:

FROM { *grantee*[, *grantee*] ... }

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:

INTERVAL { *start-field* TO *end-field* }  
{ *single-field* }

INTERVAL { *start-field* TO *end-field* | *single-field* }

## | Vertical Line

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

{*expression* | NULL}

## ... 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:

ATTRIBUTE[S] *attribute* [, *attribute*] ...  
, *sql-expression* } ...

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

*expression-n*...

## Punctuation

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

DAY (*datetime-expression*)

@*script-file*

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:

```
"{ " module-name [, module-name] . . . " }"
```

#### 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:

```
DAY (datetime-expression)
```

```
DAY(datetime-expression)
```

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:

```
myfile.sh
```

#### 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:

```
match-value [NOT] LIKE pattern
```

```
[ESCAPE esc-char-expression]
```

## Related Information

For a better understanding of the frameworks, see the following websites:

- <http://www.springsource.org>
- <http://www.hibernate.org>
- <http://myfaces.apache.org/>
- <http://axis.apache.org/axis2/java/core/>

## Publishing History

Part Number	Product Version	Publication Date
641664-001	N.A.	February 2011
641664-002	N.A.	February 2012

## 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](mailto: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

Java-based Web application development has been inundated by frameworks of every kind. There is a framework available for almost every application development requirement.

The effort for developing an application using the framework approach is significantly less as compared to developing an application afresh. The efficiency of Java-based Web development can be increased by using the appropriate framework.

OpenSource Java Frameworks on NonStop systems offer a Java environment on HP NonStop™ systems that enables the development of Enterprise Java applications using standards-based, open source technologies that are architecturally compatible with NonStop systems.

HP has certified Spring, Hibernate, Apache MyFaces, and Apache Axis2/Java frameworks for use on NonStop systems as these frameworks are stable frameworks for application development.

Following are some of the advantages provided by these frameworks:

- Spring - provides a lightweight framework to implement the business logic.  
Spring has a layered architecture and also provides an abstraction layer to integrate with other frameworks, such as Hibernate. It also provides lightweight remoting support
- Hibernate - is the ORM tool for incorporating database access.  
Hibernate scales well in any environment and is highly extensible and customizable. It also supports lazy initialization, fetching strategies, and optimistic locking with automatic versioning and time stamping.
- MyFaces - is a JSF implementation used to design the user interface for an application.  
MyFaces is a component-oriented, event-based framework, which also supports validation of user inputs. It also provides useful features, such as support for Tiles, use of converters, and so on.
- Axis2/Java - is a web services engine, SOAP, and WSDL engine.  
Axis2/Java is a high speed, flexible framework to develop web services. It provides features such as hot deployment, scalability, WSDL support, and so on.  
Axis2/Java can plug into servlet engines as a server and provides extensive support for the Web Service Description Language (WSDL). It also includes tools that can be used to create Java classes from the WSDL and back.

The frameworks are not restricted to these roles and can be customized for specific development requirements. The following versions of the frameworks are currently certified for use on NonStop:

- Spring 2.5 and Spring 3.0.2
- Hibernate 3.2.6 and Hibernate 3.5.1
- MyFaces 1.2.5 and MyFaces 2.0.2
- Axis2/Java 1.4.1 and Axis2/Java 1.5.2

This manual describes how to download, install, configure, and use the Spring 3.0.2, Hibernate 3.5.1, MyFaces 2.0.2, and Axis2/Java 1.5.2 frameworks on a NonStop system.

**NOTE:** The frameworks have not been modified or customized in any manner for use on NonStop systems. The steps to download the frameworks, and build sample applications, are explained using the Microsoft Windows platform only.

## Advantages of Java Frameworks on NonStop

NonStop provides a highly scalable servlet/JSP container to host the web applications developed using the frameworks. Because the applications are deployed on NSJSP, the Java frameworks can leverage the following advantages of NonStop:

- **Scalability**

Scalability in NSJSP refers to its ability to increase its capacity to process a large number of requests simultaneously, by adding resources, such as additional processes and logical processors, to a system. Because applications developed using the frameworks are deployed as a servlet under NSJSP, the applications are able to leverage features of NSJSP. This means that when the application is heavily loaded and all the static servers are busy, a dynamic server is created to cater to the excess load. Thus, scalability also promotes need-based allocation of the system resources.

- **Availability**

High-availability guarantees continuous availability of services even during a failure of some system resources, such as processors. The Java frameworks, deployed under NSJSP, use NonStop TS/MP to ensure high availability and scalability, which enables you to run, as a server class, several instances of the same process. If any of the application server processes fail, TS/MP restarts the server process and maintains a minimum number of server application processes running on the system.

- **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.

## Prerequisites

Before getting started, make sure that you have the requisite software installed on the NonStop and Windows system.

## NonStop System

The following software must be installed on the NonStop system:

---

**NOTE:** The subsequent topics provide quick reference information on installing and configuring the required resources. For detailed information on these products, see the respective manuals available on [Business Support Center](#).

---

- [NonStop iTP WebServer](#)

- is an HTTPD web server on NonStop.

- [NSJSP](#)

- is a scalable Java Servlet™ and JavaServer Pages™ (JSP) container based on the Tomcat container from Apache Software Foundation.

- [NonStop SQL/MX](#)

- is a relational database on NonStop.

- [JDBC Type 2 Driver for NonStop SQL/MX](#)
  - enables Java applications to use HP NonStop SQL/MX to access NonStop SQL databases.
- [NonStop Server for Java \(NSJ\)](#)
  - is a Java environment that supports compact, concurrent, dynamic, and portable programs for NonStop systems.

## NonStop iTP WebServer

- You must have either of the following versions of iTP WebServer installed on the NonStop system:
  - non-secure: version T8996H02 or later
  - secure: version T8997H02 or later

To verify the version:

1. Go to the `/usr/tandem/webserver/<iTP WebServer version>/bin` directory on OSS.

where,

`<iTP WebServer version>` is the version of the iTP WebServer release installed (for example, `T8996H02_15JUL10_BASE_H200_01`).

For example:

```
OSS> cd /usr/tandem/webserver/T8996H02_15JUL10_BASE_H200_01/bin
```

2. Use the VPROC utility to verify the version of the `httpd` object:

```
OSS> vproc httpd
```

- For the non-secure iTP WebServer version, the display should appear as:

```
/usr/tandem/webserver/T8996H02_30APR10_ADO_H215_01/bin/httpd
  Binder timestamp: 27APR2010 19:06:56
  Version procedure: T8432H01_05MAY2006_CCPLMAIN
  Version procedure: T8996H02_30APR10_ADO_H215_01
  TNS/E Native Mode: runnable file
```

- For the secure iTP WebServer version, the display should appear as:

```
/usr/tandem/webserver/T8997H02_10MAY10_AD_P_H216_01/bin
  Binder timestamp: 07MAY2010 17:49:16
  Version procedure: T8997H02_10MAY10_AD_P_H216_01
  Version procedure: T8432H01_05MAY2006_CCPLMAIN
  TNS/E Native Mode: runnable file
```

### NOTE:

- If you have more than one non-secure version of iTP WebServer available on the NonStop system, select iTP WebServer version T8996H02 or later.
- If you have more than one secure version of iTP WebServer available on the NonStop system, select iTP WebServer version T8997H02 or later.

For information on installing and setting up iTP WebServer (secure and non-secure) on a NonStop system, see the *iTP Secure WebServer System Administrator's Guide*.

- If you have iTP WebServer deployed on the NonStop system, you should know:
  - The location of the iTP WebServer deployment directory. This directory is `<iTP WebServer Deployment Directory>`.
  - IP Address and Port number on which your iTP WebServer is running.

If you do not have iTP WebServer deployed on the NonStop system, create your own iTP WebServer deployment:

1. Go to the /usr/tandem/webserver/ directory on OSS:

```
OSS> cd /usr/tandem/webserver
```

2. Select the iTP WebServer version you want to deploy.

3. Go to the version-specific iTP WebServer installation directory on OSS:

```
OSS> cd <iTP WebServer version>
```

where,

<iTP WebServer version> is the version of the iTP WebServer you want to deploy.

For example:

```
OSS> cd T8997H02_09FEB09_ADE_H028_01
```

where,

T8997H02\_09FEB09\_ADE\_H028\_01 is the OSS directory where iTP WebServer is installed.

4. Run the setup script to create your iTP WebServer deployment:

```
OSS> ./setup <iTP WebServer Deployment Directory>
```

where,

<iTP WebServer Deployment Directory> is the OSS directory where you want to deploy your iTP WebServer instance.

For example:

```
OSS> ./setup/home/sash_usr/webserver
```

The setup script guides you through the deployment of iTP WebServer.

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

## NSJSP

- You must have NSJSP version **T1222H60** or later installed on the NonStop system.

To verify the version:

1. Go to the /usr/tandem/nsjsp/<NSJSP version>/lib directory on OSS.

where,

<NSJSP version> is the version procedure of the NSJSP release installed (for example, T1222H60\_30NOV2010\_AAO\_V610\_4).

For example:

```
OSS> cd /usr/tandem/nsjsp/T1222H60_30NOV2010_AAO_V610_4/lib
```

2. Use the VPROC utility to verify the version of the libT1222.so object:

```
OSS> vproc libT1222.so
```

The display should appear as:

```
/usr/tandem/nsjsp/T1222H60_30NOV2010_AAO_V610_4/lib
  Binder timestamp: 22MAY2008 08:02:14
  Version procedure: T1222H60_30NOV2010_AAO_V610_4
  TNS/E Native Mode: runnable file
```

---

**NOTE:** If you have more than one version of NSJSP available on the NonStop system, select NSJSP version T1222H60 or later.

For information on installing and setting up NSJSP on a NonStop system, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

- NSJSP must be deployed on the NonStop system.
  - If you have NSJSP deployed on the NonStop system, you should know:
    - Location of the iTP WebServer deployment directory under which NSJSP is setup. This location will be referred to as *<NSJSP Deployment Directory>*.
    - IP Address and Port Number of the iTP WebServer deployment under which NSJSP is setup.
  - If you do not have NSJSP deployed on the NonStop system, create your own NSJSP deployment:
    1. Go to the /usr/tandem/nsjsp/ directory on OSS:  
OSS> cd /usr/tandem/nsjsp
    2. Select the NSJSP version you want to deploy.
    3. Go to the version-specific NSJSP installation directory on OSS:  
OSS> cd <NSJSP version>  
where,  
<NSJSP version> is the version-specific installation directory of NSJSP you want to deploy.
- For example:  
OSS> cd T1222H60\_30NOV2010\_AAO\_V610\_4  
where,  
T1222H60\_30NOV2010\_AAO\_V610\_4 is the OSS directory where NSJSP is installed.
- 4. Run the setup script to create your NSJSP deployment:  
OSS> ./setup  
The setup script guides you through the installation of NSJSP. For more information on creating the NSJSP deployment, see the *NonStop Servlets for JavaServerPages (NSJSP) 6.1 System Administrator's Guide*.

## NonStop SQL/MX

You must have SQL/MX version **T1050H23** or later installed on the NonStop system.

To verify the version:

1. Go to the <SQL/MX Installation Directory>/bin directory on OSS.

For example:

OSS> cd /usr/tandem/sqlmx/bin

where,

/usr/tandem/sqlmx/ is the installation directory.

2. Use the VPROC utility to verify the version of the mxci object:

```
OSS> vproc mxci
```

The display should appear as:

```
/usr/tandem/sqlmx/bin/mxci
    Binder timestamp: 08DEC2009 19:36:07
    Version procedure: T8432H02_01FEB2007_CCPLMAIN
    Version procedure: T6520H02_01MAY2005_TFDS_SH_H02
    Version procedure: T1050H23_06JAN2010_ALB_238_1208
    Version procedure: T6520H02_01MAY2005_TFDSAPI_14JUN05_H02
    Version procedure: T1054H23_06JAN2010_ALB_238_1208
    Version procedure: S0018H23_06JAN2010_ALB_CFL_1202
    TNS/E Native Mode: runnable file
```

For information on installing SQL/MX on a NonStop system, see the *SQL/MX Installation and Management Guide*.

## JDBC Type 2 Driver for NonStop SQL/MX

You must have JDBC Type 2 driver **T1275H50** or later installed on the NonStop system.

To verify the version:

1. Go to the <*JDBC T2 Installation Directory*>/lib directory on OSS.

For example:

```
OSS> cd /usr/tandem/jdbcMx/T1275H50/lib
```

where,

*/usr/tandem/jdbcMx/T1275H50* is the installation directory.

2. Use the VPROC utility to verify the version of the libjdbcMx.so object:

```
OSS> vproc libjdbcMx.so
```

The display should appear as:

```
/usr/tandem/jdbcMx/T1275H50/lib/libjdbcMx.so
    Binder timestamp: 07MAR2008 10:09:04
    Version procedure: T1275H50_08MAR2008_JDBCMXA_AF_0307
    TNS/E Native Mode: runnable file
```

For information on installing JDBC Type 2 driver on a NonStop system, see the *JDBC Driver for SQL/MX Programmer's Reference Manual*.

## NonStop Server for Java (NSJ)

You must have NSJ version **T2766H60** or later installed on the NonStop system.

To verify the version:

1. Go to the <*NSJ Installation Directory*>/bin directory on OSS.

For example:

```
OSS> cd /usr/tandem/nssjava/jdk160_h60/bin
```

where,

*/usr/tandem/nssjava/jdk160\_h60/* is the NSJ installation directory.

2. Use the VPROC utility to verify the version of the Java object:

```
OSS> vproc java
```

The display should appear as:

```
/usr/tandem/nssjava/jdk160_h60/bin/java
```

```
    Binder timestamp: 30MAR2010 03:04:11
    Version procedure: T2766H60_22FEB2010_jdk60_AB_P_30Mar2010
    Version procedure: T8432H02_21MAY2008_CCPLMAIN
    TNS/E Native Mode: runnable file
```

For information on installing NSJ on a NonStop system, see the *NonStop Server for Java 6.1 Programmer's Reference Manual*.

## Windows System

The following software must be installed on the Windows™ system:

- **Java Development Kit (JDK)**
  - is a Java development tool kit, which includes the Java Runtime Environment (JRE) that enables you to run Java applications and applets.
- **Ant**
  - is a Java library and command-line tool used to build Java projects. This tool is required to build Axis2/Java applications.

For more information, see <http://ant.apache.org/>.
- **Maven**
  - is a software project management and comprehension tool from Apache Software Foundation. This tool is used to manage a project's build, reporting, and documentation using a central repository.

For more information, see <http://maven.apache.org/>.
- **JDBC Type 4 Driver for NonStop SQL/MX**
  - enables Java applications to use HP NonStop SQL/MX to access NonStop SQL databases.
- **SVN**
  - is an open source version control system provided by Apache Subversion™ software project and is used to download the required software.

For more information, see <http://subversion.apache.org/>.
- **Eclipse Galileo IDE**
  - is an Integrated Development Environment (IDE) that provides comprehensive facilities for developing applications.

For more information, see <http://www.eclipse.org/>.

### Java Development Kit (JDK)

You must have JDK version **1.5** or later installed on the Windows system.

If you already have JDK installed, check its version using the following command:

```
command prompt> java -version  
java version "1.6.0_07"  
Java(TM) SE Runtime Environment (build 1.6.0_07-b06)  
Java HotSpot(TM) Client VM (build 10.0-b23, mixed mode, sharing)
```

**NOTE:** The above output is for JDK 1.6.

If the `java -version` command returns the following error, set the `PATH` and `JAVA_HOME` variables:

'java' is not recognized as an internal or external command, operable program or batch file.

To set the variables:

1. Right-click **My Computer** on your desktop and select **Properties**.  
The System Properties window appears.
2. Click the **Advanced** tab.

3. Click **Environment Variables**.  
A list of environment variables appears.
4. If the PATH and JAVA\_HOME variables are listed in the **User Variables** dialog box, verify that:
  - The PATH variable includes the bin directory of <Java Installation Directory>.
  - The JAVA\_HOME variable is set to <Java Installation Directory>.

If the PATH and JAVA\_HOME variables exist, but do not contain the respective directories, do the following:

1. Add <Java Installation Directory>\bin to PATH, separated by a semicolon, and <Java Installation Directory> to JAVA\_HOME.  
For example: Add C:\Program Files\Java\jdk1.6.0\_03\bin to the PATH variable and C:\Program Files\Java\jdk1.6.0\_03 to the JAVA\_HOME variable.
2. Click **OK**.
3. Click **Apply Changes**.

If the PATH and JAVA\_HOME variables do not exist, follow these steps:

1. Click **New** in the **User Variables** dialog box.
2. Enter the variable name as **PATH** and variable value as <Java Installation Directory>\bin.
3. Enter the variable name as **JAVA\_HOME** and variable value as <Java Installation Directory>.  
For example: Enter C:\Program Files\Java\jdk1.6.0\_03\bin as the value of PATH and C:\Program Files\Java\jdk1.6.0\_03 as the value of JAVA\_HOME.
4. Click **OK**.
5. Click **Apply Changes**.

If you do not have JDK installed on your Windows system, download it from <http://java.sun.com/javase/downloads/index.jsp> and set the PATH and JAVA\_HOME variables as described above.

---

**NOTE:** All applications have been verified using JDK version 1.6.

---

## Ant

You must have Ant version **1.7.0** or later installed on the Windows system.

If you already have Ant installed, check its version using the command:

```
command prompt> ant -version  
Apache Ant version 1.7.0 RC1 compiled on November 5 2006
```

The version of Ant must be **1.7.0** or later.

If the ant -version command returns the following error, set the PATH and ANT\_HOME variables:

```
'ant' is not recognized as an internal or external command,  
operable program or batch file.
```

To set the variables:

1. Right-click **My Computer** on your desktop and select **Properties**.

The System Properties window appears.

2. Click the **Advanced** tab.
3. Click **Environment Variables**.

A list of environment variables appears.

4. If the PATH and ANT\_HOME variables are listed in the **User Variables** dialog box, verify that:
  - The PATH variable includes the bin directory of <Ant Installation Directory>
  - The ANT\_HOME variable is set to <Ant Installation Directory>

If the PATH and ANT\_HOME variables exist but do not contain the respective directories, do the following:

1. Add <Ant Installation Directory>\bin to PATH, separated by semicolon, and <Ant Installation Directory> to ANT\_HOME.  
For example: Add C:\ant1.7.0RC1\bin to PATH and C:\ant1.7.0RC1 to ANT\_HOME.
2. Click **OK**.
3. Click **Apply Changes**.

If the PATH and ANT\_HOME variables do not exist, follow these steps:

1. Click **New** on the **User Variables** dialog box.
2. Enter the variable name as **PATH** and variable value as <Ant Installation Directory>\bin.
3. Enter the variable name as **ANT\_HOME** and variable value as <Ant Installation Directory>.   
For example: Enter C:\ant1.7.0RC1\bin as the value of PATH and C:\ant1.7.0RC1 as the value of ANT\_HOME.
4. Click **OK**.
5. Click **Apply Changes**.

If you do not have Ant 1.7.0 installed on your Windows system, download the apache-ant-1.7.0-bin.zip file from <http://archive.apache.org/dist/ant/binaries> and set the PATH and ANT\_HOME variables as described above.

## Maven

You must have Maven version **2.2.1** installed on the Windows system.

If you already have Maven installed, check its version using the command:

command prompt> mvn -v

```
Apache Maven 2.2.1 (r801777; 2009-08-07 00:46:01+0530)
Java version: 1.6.0_03
Java home: C:\Program Files\Java\jdk1.6.0_03\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows xp" version: "5.1" arch: "x86" Family: "windows"
```

If the mvn -v command returns the following error, set the PATH and MAVEN\_HOME variables:

```
'mvn' is not recognized as an internal or external command,
operable program or batch file.
```

To set the variables:

1. Right-click **My Computer** on your desktop and select **Properties**.  
The System Properties window appears.
2. Click the **Advanced** tab.
3. Click **Environment Variables**.  
A list of environment variables appears.
4. If the PATH and MAVEN\_HOME variables are listed in the **User Variables** dialog box, verify that:
  - The PATH variable includes the bin directory of < Maven Installation Directory >
  - The MAVEN\_HOME variable is set to < Maven Installation Directory >

If the PATH and MAVEN\_HOME variables exist but do not contain the respective directories, do the following:

1. Add < Maven Installation Directory>\bin to PATH, separated by semicolon, and < Maven Installation Directory> to MAVEN\_HOME.

For example: Add C:\apache-maven-2.2.1\bin to PATH and C:\apache-maven-2.2.1 to MAVEN\_HOME.

2. Click **OK**.
3. Click **Apply Changes**.

If the PATH and MAVEN\_HOME variables do not exist, follow these steps:

1. Click **New** on the **User Variables** dialog box.
2. Enter the variable name as **PATH** and variable value as < Maven Installation Directory>\bin.
3. Enter the variable name as **MAVEN\_HOME** and variable value as < Maven Installation Directory>.

For example: Enter C:\apache-maven-2.2.1\bin as the value of PATH and C:\apache-maven-2.2.1 as the value of MAVEN\_HOME.

4. Click **OK**.
5. Click **Apply Changes**.

If you do not have Maven 2.2.1 installed on your Windows system, download the Maven 2.2.1 (zip) file from <http://maven.apache.org/download.html> and set the PATH and MAVEN\_HOME variables as described above.

---

#### **NOTE:**

- If you are using a proxy server to access Internet, you need to edit the following proxy information in <MAVEN\_HOME>\conf\settings.xml:

```
<proxies>
  <proxy>
    <active>true</active>
    <protocol>http</protocol>
    <host>proxy.somewhere.com</host>
    <port>8080</port>
    <username>proxyuser</username>
    <password>proxypassword</password>
    <nonProxyHosts>www.google.com|*.somewhere.com</nonProxyHosts>
  </proxy>
</proxies>
```

- If the firewall restricts access to the websites that host Maven repositories, you might get Connection Refused or Connection Time Out error messages.
  - If you want to use the Eclipse Galileo IDE, you can download the Maven plugin for Eclipse from <http://m2eclipse.sonatype.org/installing-m2eclipse.html>.
- 

## JDBC Type 4 Driver for NonStop SQL/MX

You must have JDBC Type 4 driver version **T1249V11** or later installed on the Windows system.

If you already have JDBC Type 4 driver installed, check its version by following these instructions:

1. Go to the <JDBC T4 Installation Directory>\lib directory on the Windows system.  
 command prompt> cd <JDBC T4 Installation Directory>\lib  
 For example:  
 command prompt> cd C:\JDBCT4\lib
2. Verify that the version of JDBC Type 4 driver is T1249V11 or later:  
 command prompt> java -jar t4sqlmx.jar  
 Version procedure: **T1249\_V11\_15MAY05\_HP\_NONSTOP (TM) \_JDBCT4\_2005\_04\_28**

For information on installing JDBC Type 4 driver on a Windows system, see the *HP NonStop JDBC Type 4 Driver Programmer's Reference Manual*.

## SVN

---

**NOTE:** SVN is not required if you download the customized samples (T0874AAB) from SCOUT. It is required only when you download the Spring sample applications from the Spring repository.

You must have SVN version **1.5.4** or later installed on the Windows system.

If you already have SVN installed, check its version using the command:

```
command prompt> svn --version
svn, version 1.5.4 (r33841)
      compiled Oct 24 2008, 16:12:55
```

```
Copyright (C) 2000-2008 CollabNet.
Subversion is open source software, see http://subversion.tigris.org/
This product includes software developed by CollabNet (http://www.Collab.Net/) .
```

The following repository access (RA) modules are available:

- \* **ra\_neon** : Module for accessing a repository via WebDAV protocol using Neon.
  - handles 'http' scheme
  - handles 'https' scheme
- \* **ra\_svn** : Module for accessing a repository using the svn network protocol.
  - with Cyrus SASL authentication
  - handles 'svn' scheme
- \* **ra\_local** : Module for accessing a repository on local disk.
  - handles 'file' scheme
- \* **ra\_serf** : Module for accessing a repository via WebDAV protocol using serf.
  - handles 'http' scheme
  - handles 'https' scheme

If the svn --version command returns the following error, set the PATH and ACE\_HOME variables:

'svn' is not recognized as an internal or external command, operable program or batch file.

To set the variables:

1. Right-click **My Computer** on your desktop and select **Properties**.  
 The System Properties window appears.
2. Click the **Advanced** tab.
3. Click **Environment Variables**.  
 A list of environment variables appears.
4. If the PATH and ACE\_HOME variables are listed in the **User Variables** dialog box, verify that:
  - The PATH variable includes the bin directory of <SVN Installation Directory>
  - The ACE\_HOME variable is set to <SVN Installation Directory>

If the PATH and ACE\_HOME variables exist but do not contain the respective directories, do the following:

1. Add <SVN Installation Directory>\bin to PATH, separated by semicolon, and <SVN Installation Directory> to ACE\_HOME.

For example: Add C:\svn-win32-1.5.4\bin to PATH and C:\svn-win32-1.5.4 to ACE\_HOME.

2. Click **OK**.
3. Click **Apply Changes**.

If the PATH and ACE\_HOME variables do not exist, follow these steps:

1. Click **New** on the **User Variables** dialog box.
2. Enter the variable name as **PATH** and variable value as <SVN Installation Directory>\bin.
3. Enter the variable name as **ACE\_HOME** and variable value as <SVN Installation Directory>.

For example: Enter C:\svn-win32-1.5.4\bin as the value of PATH and C:\svn-win32-1.5.4 as the value of ACE\_HOME.

4. Click **OK**.
5. Click **Apply Changes**.

To install SVN 1.5.4:

1. Go to <http://subversion.tigris.org/servlets/ProjectDocumentList?folderID=11150&expandFolder=11150&folderID=91>.

A list of SVN download packages appears.

2. Download and extract **svn-win32-1.5.4.zip** to a location on the Windows system.

---

**NOTE:** This directory will be referred as <ACE\_Home>.

3. Create the svn root directory (for example, c:\svn).
4. Create a repository folder in the svn root directory:

command prompt > svnadmin create c:\svn\repository

The repository folder gets created under the svn folder.

5. In the repository folder, uncomment the following lines in the conf/svnserve.conf file.  
Before the change:

```
#anon-access = none  
#auth-access = write  
#password-db = passwd
```

After the change:

```
anon-access = none  
auth-access = write  
password-db = passwd
```

6. Add users to the conf/passwd file.

---

**NOTE:** You can uncomment the default harry and sally users and add new users.

Before the change:

```
harry = harryssecret  
sally = sallyssecret
```

After the change:

```
Testuser1 = testpassword1  
Testuser2 = testpassword2
```

7. Install SVN as a Windows service:

```
command prompt> sc create svnserver binpath= "c:\svn\bin\svnserve.exe --service -r c:\svn\repository"  
command prompt> displayname= "Subversion" depend= Tcpip start= auto
```

8. Start the SVN server:

```
command prompt> net start svnserver
```

SVN 1.5.4 is now installed and running on the Windows system.

---

**NOTE:** If want to use the Eclipse Galileo IDE, download the SVN plugin for Eclipse from <http://www.eclipse.org/subversive/downloads.php>.

---

## Eclipse Galileo IDE

You must have Eclipse Galileo IDE version 3.3.1.1 or a later installed on the Windows system.

If you do not have the Eclipse Galileo IDE installed on your Windows system, download it from:  
<http://archive.eclipse.org/eclipse/downloads/drops/R-3.3.1.1-200710231652/winPlatform.php#EclipseSDK>

Click the `eclipse-SDK-3.3.1.1-win32.zip` file from the Download column. You can also download it directly from <http://archive.eclipse.org/eclipse/downloads/drops/R-3.3.1.1-200710231652/download.php?dropFile=eclipse-SDK-3.3.1.1-win32.zip>.

---

# Part I Spring Framework

Part I includes the following chapters:

- [“Spring Overview” \(page 33\)](#)

This chapter provides an overview of the following:

- Spring projects certified for use on NonStop
- Deployment environment for a web application that is written using Spring and accesses data from the SQL/MX database.

- [“Installing the Spring Framework” \(page 36\)](#)

This chapter helps you to install Spring framework libraries and enables you to deploy and run sample Spring applications on your NonStop system.

- [“Configuring Spring Applications on NonStop Systems” \(page 52\)](#)

This chapter provides information about configuring a Spring web application on NonStop systems.

- [“Getting Started with Spring” \(page 75\)](#)

This chapter helps you to develop a web application on your Windows system using the Spring framework, and deploy and run the web application on your NonStop system.

- [“EmplInfo Database Script” \(page 134\)](#)

This appendix contains the EmplInfo database script.

- [“Customizing Sample Applications” \(page 135\)](#)

This appendix details the customizations made in the Spring sample applications, namely, PetClinic and JPetStore.

- [“JDBC Configuration” \(page 144\)](#)

This appendix describes the consolidated JDBC Type 2 driver configuration and the JDBC Type 4 driver configuration in the `applicationContext.xml` and `jdbc.properties` files.

- [“Installing Spring Web Flow” \(page 146\)](#)

This appendix describes the steps required to install the Spring Web Flow on a NonStop system.

---

# Contents

<b>2 Spring Overview.....</b>	<b>33</b>
Spring Projects.....	33
Spring Framework.....	33
Spring Web Flow.....	34
Spring Applications on NonStop.....	34
<b>3 Installing the Spring Framework.....</b>	<b>36</b>
Prerequisites.....	36
NonStop System.....	36
Windows System.....	36
Installing Spring Framework Libraries on NonStop.....	36
Downloading the Spring Distribution on Windows.....	37
Downloading Spring Dependency JAR Files on Windows.....	38
Downloading Sample Spring Applications on Windows.....	38
Copying Spring Runtime Libraries from Windows to NonStop.....	39
Deploying and Running Sample Spring Applications on NonStop.....	39
PetClinic.....	40
JPetStore.....	46
<b>4 Configuring Spring Applications on NonStop Systems.....</b>	<b>52</b>
NonStop Platform Configurations.....	52
Determining the Application Parameters.....	52
Determining the Maximum Capacity of NSJSP Instance.....	52
Configuring iTP WebServer for Spring Applications.....	53
Configuring NSJSP for Spring Applications.....	57
Spring Framework Configurations.....	62
Configuring JDBC Driver for SQL/MX Database.....	63
Configuring Database Transaction Management.....	66
Connection Pooling.....	72
Module File Caching Configurations.....	73
Configuring NonStop SQL/MX DataSource for MFC.....	73
Modifying the Spring Application.....	74
<b>5 Getting Started with Spring.....</b>	<b>75</b>
Prerequisites.....	75
NonStop System.....	75
Windows System.....	75
Overview of EmplInfo.....	75
Developing EmplInfo on Windows using the Eclipse Galileo IDE.....	76
Deploying EmplInfo on NonStop.....	129
Running EmplInfo on NonStop.....	133
<b>A EmplInfo Database Script.....</b>	<b>134</b>
<b>B Customizing Sample Applications.....</b>	<b>135</b>
Customizing PetClinic.....	135
Added Directories.....	135
Added File.....	135
Modified Files.....	136
Customizing JPeteStore.....	139
Added File.....	139
Modified Files.....	139

C JDBC Configuration.....	144
D Installing Spring Web Flow.....	146
Downloading Spring Web Flow Distribution on Windows.....	146
Copying Spring Web Flow Runtime Libraries from Windows to NonStop.....	146

## 2 Spring Overview

Spring is an open source framework that can be used to develop enterprise applications. Its features and functions are packaged in different modules, which facilitate Java application development. It includes a lightweight container that provides a centralized, automated configuration and wiring of your application objects. Spring includes the following:

- A flexible MVC web application framework, built on core Spring functionality. This framework is highly configurable using strategy interfaces, and accommodates multiple view technologies, such as JSF.
- Integration with other frameworks, such as Hibernate, with lots of IoC convenience features, addressing many typical Hibernate integration issues.
- A JDBC abstraction layer that supports exception hierarchy, simplifies error handling, and reduces the effort in generating code.
- Support for application testability and scalability by allowing software components to be first developed and tested in isolation, then scaled up for deployment in any environment.

The NonStop system provides a platform comprising the Java Virtual Machine (JVM) and a servlet container — NSJSP. On NonStop systems, Java applications (created using Spring libraries) can be run/deployed as:

- Standalone applications on the JVM
- Web applications under NSJSP

**NOTE:** Throughout this Spring refers to Spring version 3.0.2.

## Spring Projects

The Spring community maintains several projects such as Spring Web Flow, Spring Framework, Spring Web Services, and Spring Security. Among these projects, only the Spring Framework and the Spring Web Flow is presently certified for use on the NonStop platform.

For information on how to use the Spring Framework to develop a web application on NonStop platform, see “[Getting Started with Spring](#)” (page 75).

## Spring Framework

The Spring framework is a layered development platform for developing Java applications in different environments. Because the framework provides abstraction layers and modules, you can use the Spring modules selectively based on your requirement.

The basic Spring framework consists of the following:

- Core Container
- Data Access/Integration layer (For examples on the Data Access/Integration layer, see “[Integrating Frameworks](#)” (page 346).)
- Web layer (For examples on the Web layer, see “[Getting Started with Spring](#)” (page 75).)
- AOP and Instrumentation module (For examples on the AOP and Instrumentation layer, see “[Using Spring Transaction Manager](#)” (page 348).)
- Test module

For more information about Spring Frameworks, see <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html>.

## Spring Web Flow

At times, web applications need to execute the same sequence of steps in different contexts. The Spring Web Flow can be used to implement such repetitive steps or flows because it integrates with the Spring Web MVC platform to provide a flow definition language.

For information about downloading and installing the Spring Web Flow, see [Appendix D: Installing Spring Web Flow](#).

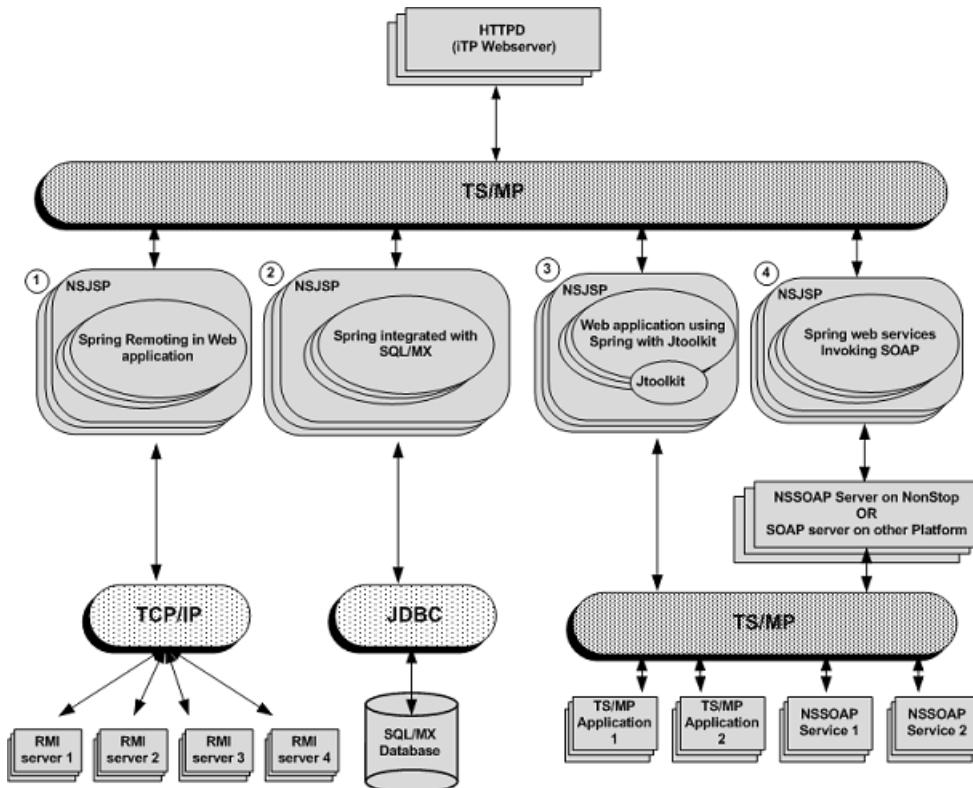
## Spring Applications on NonStop

Applications created using Spring, when deployed on NonStop, are able to leverage features provided by the NonStop platform. For instance, when you create a web application using Spring and deploy it on NSJSP, the web application inherits all the benefits of NSJSP, such as fault tolerance, scalability, hot deployment, and so on. In a similar manner you can leverage all features of the NonStop platform by developing web applications using the Spring framework.

Figure 1 shows how a Spring application can interface with other NonStop products.

**NOTE:** In Figure 1, the Spring applications are deployed on NSJSP. The same holds true when running the Spring applications as a standalone application too.

**Figure 1 Spring Web Applications**



The combinations displayed in Figure 1 are as follows:

1. Spring web applications accessing RMI services (using Spring Remoting).

The Spring framework includes integration classes for remoting support using different technologies. One such technology is Remote Method Invocation (RMI). Using Spring and RMI, you can transparently expose your services through the RMI infrastructure. You must use NonStop TCP/IPv6 or NonStop IP-CIP as the TCP/IP transport provider because the default RMI implementation uses TCP/IP as its underlying transport layer.

There can be multiple RMI server instances listening on the same port. When an RMI client invokes an RMI server, the TCP/IP provider distributes the requests across the RMI servers.

instances listening on the port in a round-robin manner. If the RMI servers are distributed across all the processors it leads to RMI calls getting distributed across all the processors.

---

**NOTE:** NonStop TCPIpv6 limits one process per processor to listen on any given port, while IP-CIP does not have any such restriction.

---

**2.** Spring applications accessing SQL/MX database.

Spring web applications can be configured to access a SQL/MX database through the JDBC drivers (JDBC Type 2 or JDBC type 4 drivers).

---

**NOTE:** Hibernate has been certified on HP NonStop platform for use with SQL/MX database. For information about using Hibernate on NonStop, see "[Hibernate Framework](#)" (page 148).

---

**3.** Spring web applications accessing TS/MP applications.

The Spring framework can be integrated with the JToolkit library to expose TS/MP applications as web applications. This feature is especially useful to expose legacy TS/MP applications as web applications without any major changes to the application.

**4.** Spring applications accessing SOAP services.

Spring web applications can be configured to access SOAP services. These applications can be developed to invoke SOAP services hosted on a NonStop platform or any other platform.

There are other combinations of Spring and related technologies that can be used to develop your web applications. The subsequent chapters discuss the installation and configuration details of Spring that are required for web application development.

# 3 Installing the Spring Framework

This chapter describes the procedure to install Spring framework libraries on a Nonstop system. It also describes the steps required to deploy and run sample Spring applications, which help you to verify if you have successfully installed Spring on the NonStop system.

The following tasks are described in this chapter:

1. “[Installing Spring Framework Libraries on NonStop](#)” (page 36)
2. “[Deploying and Running Sample Spring Applications on NonStop](#)” (page 39)

## Prerequisites

Before getting started, make sure that you have the following software installed on the NonStop and Windows system:

### NonStop System

The following software must be installed on the NonStop system:

- NonStop iTP WebServer version T8996H02 or later
- NSJSP version T1222H60 or later
- NonStop SQL/MX version T1050H23 or later
- JDBC Type 2 driver version T1275H50 or later, or JDBC Type 4 driver version T1249V11 or later
- NSJ version T2766H60 or later

### Windows System

The following software must be installed on the Windows system:

- JDK version 1.5 or later
- Maven version 2.2.1
- JDBC Type 4 driver version T1249V11 or later
- SVN version 1.5.4 or later

---

**NOTE:** For more information about installing the software required on NonStop and Windows system, see “[Prerequisites](#)” (page 18).

---

## Installing Spring Framework Libraries on NonStop

Installing Spring framework libraries on a NonStop system requires the following actions:

1. “[Downloading the Spring Distribution on Windows](#)” (page 37)
2. “[Downloading Spring Dependency JAR Files on Windows](#)” (page 38)

3. "Downloading Sample Spring Applications on Windows" (page 38)
  4. "Copying Spring Runtime Libraries from Windows to NonStop" (page 39)
- 

**NOTE:** Throughout the chapter, references are made to the following directories:

- *<Spring Home>*: The directory on the Windows system where the Spring distribution files are extracted.
  - *<NonStop Spring Home>*: The OSS directory on the NonStop system where the Spring runtime JAR files are located.
  - *<My SASH Home>*: The directory on the Windows system where the contents of the SAMPLES.zip file (distributed as a part of the NS Samples for Java Frameworks - T0874 and available for download in Scout for NonStop Servers) is extracted.
  - *<Spring Dependency Home>*: The directory on the NonStop system where the Spring dependency JAR files are downloaded and extracted.
  - *<NSJSP Deployment Directory>*: The location of the iTP WebServer deployment directory (on the NonStop system) in which NSJSP is set up.
- 

## Downloading the Spring Distribution on Windows

To download the Spring distribution on the Windows system, complete the following steps:

1. Go to <http://www.springsource.com/download/community?project=Spring%20Framework>.  
The Spring Community Downloads web page appears.
  2. Do one of the following:
    1. Register yourself.
    2. Click **I'd rather not fill in the form. Just take me to the download page.**  
The Spring frameworks available for download appear.
  3. Click **3.0.2.RELEASE**.  
The links to the following files appear:
    - spring-framework-3.0.2.RELEASE-dependencies.zip
    - spring-framework-3.0.2.RELEASE-with-docs.zip
    - spring-framework-3.0.2.RELEASE.zip
  4. Download the `spring-framework 3.0.2.RELEASE.zip` file.
- 

**NOTE:**

- The `spring-framework 3.0.2.RELEASE.zip` file does not include the Spring sample applications, the Spring framework libraries, and dependency JAR files that are required to build the sample applications.
5. Extract the `spring-framework 3.0.2.RELEASE.zip` file into a directory on the Windows system.

This directory will be referred as *<Spring Home>*. The *<Spring Home>* includes the following sub-directories:

\dist

includes the Spring JAR files.

\projects

includes the source of Spring modules, such as AOP, aspects, and so on.

\docs

includes the Spring reference documentation in PDF and HTML format.

\src  
includes the source of the Spring framework.

## Downloading Spring Dependency JAR Files on Windows

To download the Spring dependency JAR files on the Windows system, complete the following steps:

1. Go to <http://www.springsource.com/download/community?project=Spring%20Framework>.  
The Spring Community Downloads web page appears.
2. Do one of the following:
  1. Register yourself.
  2. Click **I'd rather not fill in the form. Just take me to the download page.**  
The Spring frameworks available for download appear.
3. Click **3.0.2.RELEASE**.  
The links to the following files appear:
  - spring-framework-3.0.2.RELEASE-dependencies.zip
  - spring-framework-3.0.2.RELEASE-with-docs.zip
  - spring-framework-3.0.2.RELEASE.zip
4. Download the `spring-framework-3.0.2.RELEASE-dependencies.zip` file.
5. Extract the `spring-framework-3.0.2.RELEASE-dependencies.zip` file into a directory on the Windows system.  
This directory will be referred as *<Spring Dependency Home>*.

## Downloading Sample Spring Applications on Windows

The Spring 3.0.2.RELEASE does not include sample Spring applications. You need to download Spring samples (PetClinic and JPetStore) from the Spring repository.

To download Spring samples, you must either have SVN or Eclipse plug-ins installed on the system.

---

**NOTE:** This section describes the steps to download Spring samples using SVN.

To use the Eclipse Galileo IDE, download the SVN plugin for Eclipse from <http://www.eclipse.org/subversive/downloads.php>.

---

If you have SVN installed, you can checkout the PetClinic and JPetStore sample applications from the Spring repository to a location on the Windows system, as follows:

- To checkout the PetClinic sample application:

```
command prompt> svn checkout  
https://src.springframework.org/svn/spring-samples/petclinic/trunk/  
c:\svn\repository
```

The PetClinic sample application is downloaded to C:\svn\repository.

- To checkout the JPetStore sample application:

```
command prompt> svn checkout  
https://src.springframework.org/svn/spring-samples/jpetstore/  
c:\svn\repository
```

The JPetStore sample application is downloaded to C:\svn\repository.

---

**NOTE:** The sample Spring applications require modifications to make them compatible with NonStop systems. The modifications made to the PetClinic and JPetStore sample applications are discussed in “Customizing Sample Applications” (page 135).

## Copying Spring Runtime Libraries from Windows to NonStop

To copy the Spring runtime libraries from a Windows system to a NonStop system, complete the following steps:

1. Go to <Spring Home> and create JAR files of <Spring Home>\dist and <Spring Dependency Home> directories on the Windows system:

```
command prompt> cd <Spring Home>
command prompt> jar -cvf spring_dist.jar dist
command prompt> cd <Spring Dependency Home>
command prompt> jar -cvf spring_dependency.jar *
```

For example:

```
command prompt> cd C:\spring-framework-3.0.2.RELEASE
command prompt> jar -cvf spring_dist.jar dist
command prompt> cd C:\spring-framework-3.0.2.RELEASE\dependencies
command prompt> jar -cvf spring_dependency.jar *
```

2. Create the <NonStop Spring Home> directory in the OSS environment on the NonStop system:

```
OSS> mkdir -p <NonStop Spring Home>
```

For example, create a directory structure /usr/tandem/sash/spring\_3.0.2.

```
OSS> mkdir -p /usr/tandem/sash/spring_3.0.2
```

3. Transfer the spring\_dist.jar file from <Spring Home> to <NonStop Spring Home> and extract it:

```
OSS> cd <NonStop Spring Home>
OSS> jar -xvf spring_dist.jar
```

4. Create the <NonStop Spring Dependency Home> directory in the OSS environment on the NonStop system:

```
OSS> mkdir -p <NonStop Spring Dependency Home>
```

For example, create /usr/tandem/sash/spring\_3.0.2.-dependencies directory structure:

```
OSS> mkdir -p /usr/tandem/sash/spring_3.0.2.-dependencies
```

5. Copy the spring\_dependency.jar file from <Spring Dependency Home> to <NonStop Spring Dependency Home> and extract it:

```
OSS> cd <NonStop Spring Dependency Home>
OSS> jar -xvf spring_dependency.jar
```

After extraction, <NonStop Spring Home> must have the <NonStop Spring Home>/dist sub-directory and <NonStop Spring Home> must have the dependency JAR files.

---

**NOTE:** The sub-directories on the NonStop system must contain the same list of files as those in the <Spring Home>\dist and <Spring Dependency Home> directories on the Windows system.

The Spring runtime libraries are installed on the NonStop system. You can use these libraries to develop and run Spring applications on a NonStop system.

## Deploying and Running Sample Spring Applications on NonStop

The sample Spring applications are included in the SAMPLES.zip file and require modifications to make them compatible with NonStop systems.

---

**NOTE:** The SAMPLES.zip file is distributed as a part of the NS Samples for Java Frameworks - T0874.

SAMPLES.zip is present in the T0874AAB.BIN file in Scout for NonStop Servers. For information on how to install the T0874AAB.BIN file from Scout, see [https://h20453.www2.hp.com/scout/download\\_help.htm](https://h20453.www2.hp.com/scout/download_help.htm).

Before you deploy the sample applications, complete the following steps:

1. Download the SAMPLES.zip file from Scout for NonStop servers.
2. Add the .zip extension to it.  
The file is renamed as SAMPLES.zip.
3. Extract the SAMPLES.zip file to a location on the Windows system.  
The NS-Samples-for-Java-Frameworks folder appears.

---

**NOTE:** The absolute path of the NS-Samples-for-Java-Frameworks folder is referred as <My SASH Home>.

---

This section describes the steps to build, set up, deploy, and run the PetClinic and JPetStore sample applications on NonStop systems.

---

**NOTE:** The modifications made to the PetClinic and JPetStore sample applications are discussed in "Customizing Sample Applications" (page 135).

---

## PetClinic

The PetClinic sample application is an information system that is accessible using a web browser. The intended users of the application are employees of the clinic, who, in the course of their work, need to view and manage information regarding veterinarians, clients, and their pets.

This section describes the following steps for the PetClinic sample application.

- "Building PetClinic on Windows" (page 40)
- "Setting up PetClinic Database on NonStop" (page 42)
- "Deploying PetClinic on NonStop" (page 43)
- "Running PetClinic on NonStop" (page 45)

### Building PetClinic on Windows

To build PetClinic on the Windows system, complete the following steps:

1. Go to the <My SASH Home>\spring\samples\petclinic directory on the Windows system. Among other files, this directory must include the following sub-directories:
  - \src
    - includes the customized source for PetClinic.
  - \etc
    - includes the Hibernate dialect file (hibernate3sqlmx.jar) for the SQL/MX database.
2. Configure the JDBC driver settings for the NonStop SQL/MX database.
  1. Go to the <My SASH Home>\spring\samples\petclinic\src\main\resources directory on the Windows system.

2. Modify the `jdbc.properties` file to update the JDBC configuration. You can use either the JDBC Type 2 driver or the JDBC Type 4 driver. The SQL/MX settings for each of them is as follows:

- To use the JDBC Type 2 driver, uncomment the SQL/MX settings for the JDBC Type 2 driver in the `jdbc.properties` file, so that the SQL/MX settings for JDBC Type 2 driver appear as:

```
# Properties that control the population of schema and data for a new data source  
# SQL/MX Settings for JDBC Type 2 Driver  
jdbc.driverClassName=com.tandem.sqlmx.SQLMXDriver  
jdbc.url=jdbc:sqlmx:///  
jdbc.username=  
jdbc.password=  
jdbc.catalog=petcliniccat  
jdbc.schema=petclinicsch  
hibernate.dialect=org.hibernate.dialect.SqlmxDialect
```

---

**NOTE:** Because the JDBC Type 2 driver is located on the NonStop system, you need not mention the username and password in the `jdbc.username` and `jdbc.password` fields.

- To use the JDBC Type 4 driver, uncomment the SQL/MX settings for the JDBC Type 4 driver in the `jdbc.properties` file, and enter the values of the JDBC URL (NonStop system IP Address and port number of the JDBC data source), NonStop system username, and password.

The SQL/MX settings for the JDBC Type 4 driver in the `jdbc.properties` file appear as:

```
-----  
# Properties that control the population of schema and data for a new data source  
# SQL/MX Settings for JDBC Type 4 Driver  
jdbc.driverClassName= com.tandem.t4jdbc.SQLMXDriver  
jdbc.url= jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.>  
jdbc.username=<HP NonStop Username>  
jdbc.password=<HP NonStop Password>  
jdbc.catalog=petcliniccat  
jdbc.schema=petclinicsch  
hibernate.dialect=org.hibernate.dialect.SqlmxDialect
```

3. Build the PetClinic web application archive (WAR) file (`petclinic.war`).

1. Go to the `<My SASH Home>\spring\samples\petclinic` directory:

command prompt> cd `<My SASH Home>\spring\samples\petclinic`

2. Build the PetClinic web archive using JDBC Type 2 driver or JDBC Type 4 driver.

- To use the JDBC Type 2 driver:

1. Enter the following command:

command prompt> mvn clean package

The application WAR file (`petclinic.war`) is created in the `<My SASH Home>\spring\samples\petclinic\target` directory.

2. Create the `WEB-INF\lib` folder in `<My SASH Home>\spring\samples\petclinic\target` directory.
3. Copy the SQL/MX hibernate dialect JAR file (`hibernate3sqlmx.jar`) to `<My SASH Home>\spring\samples\petclinic\target\WEB-INF\lib` directory.

---

**NOTE:** The Hibernate dialect file maps the Hibernate queries (HQL) to the SQL/MX specific queries. The Hibernate dialect file is included in the `<My SASH Home>\spring\samples\petclinic\etc` directory located in the `SAMPLES.zip` file. For information on Hibernate, see "[Installing the Hibernate Framework](#)" (page 153).

4. Go to <My SASH Home>\spring\samples\petclinic\target and run the following commands:

```
command prompt> cd <My SASH Home>\spring\samples\petclinic\target  
command prompt> jar uf petclinic.war WEB-INF\lib\hibernate3sqlmx.jar
```

The hibernate3sqlmx.jar is copied to <My SASH Home>\spring\samples\petclinic\target\petclinic.war\WEB-INF\lib.

- To use JDBC Type 4 driver:

1. Enter the following command:

```
command prompt> mvn clean package
```

The application WAR file (petclinic.war) is created in the <My SASH Home>\spring\samples\petclinic\target directory.

2. Create the WEB-INF\lib folder in the <My SASH Home>\spring\samples\petclinic\target directory.
3. Copy hibernate3sqlmx.jar and t4sqlmx.jar to the <My SASH Home>\spring\samples\petclinic\target\WEB-INF\lib directory.
4. Go to <My SASH Home>\spring\samples\petclinic\target and run the following commands:

```
command prompt> cd <My SASH Home>\spring\samples\petclinic\target  
command prompt> jar uf petclinic.war WEB-INF\lib\hibernate3sqlmx.jar WEB-INF\lib\t4sqlmx.jar
```

The hibernate3sqlmx.jar and t4sqlmx.jar files are copied to <My SASH Home>\spring\samples\petclinic\target\petclinic.war\WEB-INF\lib.

## Setting up PetClinic Database on NonStop

To set up the PetClinic database on your NonStop system, complete the following steps:

1. Edit the <My SASH Home>\spring\samples\petclinic\src\main\resources\db\mxci\petclinic\_tables\_script.sql file to include the following:

- <\$datavol>: specify the Guardian volume for the primary partition of the table. For example: \$data01
- <user>: specify the owner of the schema. For example: super.sashusr
- <node.\$vol>: specify the location of the metadata tables for the catalog. For example: \NSK01.\$data01
- <subvol reference>: specify the designated subvolume name for the schema. For example: SASH1

---

**NOTE:** The subvolume name is always prefixed by ZSD and must have eight characters (including ZSD).

---

2. Create a directory in OSS to place the database script files, using the command:

```
OSS> mkdir -p <NonStop SASH Home>/spring/samples/petclinic/dbconfig
```

For example:

```
OSS> mkdir -p /home/sash_usr/sash/spring/samples/petclinic/dbconfig
```

---

**NOTE:** <NonStop SASH Home> can be any working directory. It is suggested that you create spring/samples/petclinic/dbconfig directory structure in <NonStop SASH Home>.

---

3. Copy the following scripts from the <My SASH Home>\spring\samples\petclinic\src\main\resources\db\mxci\Windows directory to the <NonStop SASH Home>/spring/samples/petclinic/dbconfig OSS directory:
    - petclinic\_tables\_script.sql
    - petclinic\_dataloader\_script.sql
- 

**NOTE:**

- The petclinic\_tables\_script.sql script creates the SQL/MX database catalog, schema, and tables for PetClinic.
  - The petclinic\_dataloader\_script.sql script loads the database with sample data.
- 

4. Go to the OSS directory where the petclinic\_tables\_script.sql and petclinic\_dataloader\_script.sql files are copied using the command:

```
OSS> cd <NonStop SASH Home>/spring/samples/petclinic/dbconfig  
For example:
```

```
OSS> cd /home/sash_usr/sash/spring/samples/petclinic/dbconfig
```

5. Create the PetClinic database catalog, schema, and tables, using the SQL/MX command:  
mxci>> obey petclinic\_tables\_script.sql;
- 

**NOTE:** By default, the petclinic\_tables\_script.sql file creates the database catalog name as petcliniccat and schema name as petclinicsch. If these names conflict with any of the existing catalog and schema names, modify the database catalog and schema names in the petclinic\_tables\_script.sql script file.

If you modify the database catalog and schema names, edit the jdbc.properties file present in the <My SASH Home>\spring\samples\petclinic\src\main\resources directory, present in the SAMPLES.zip file.

---

6. Load the sample data in the PetClinic sample application database tables using the SQL/MX command:

```
mxci>> obey petclinic_dataloader_script.sql;
```

The PetClinic database is setup on your NonStop system.

## Deploying PetClinic on NonStop

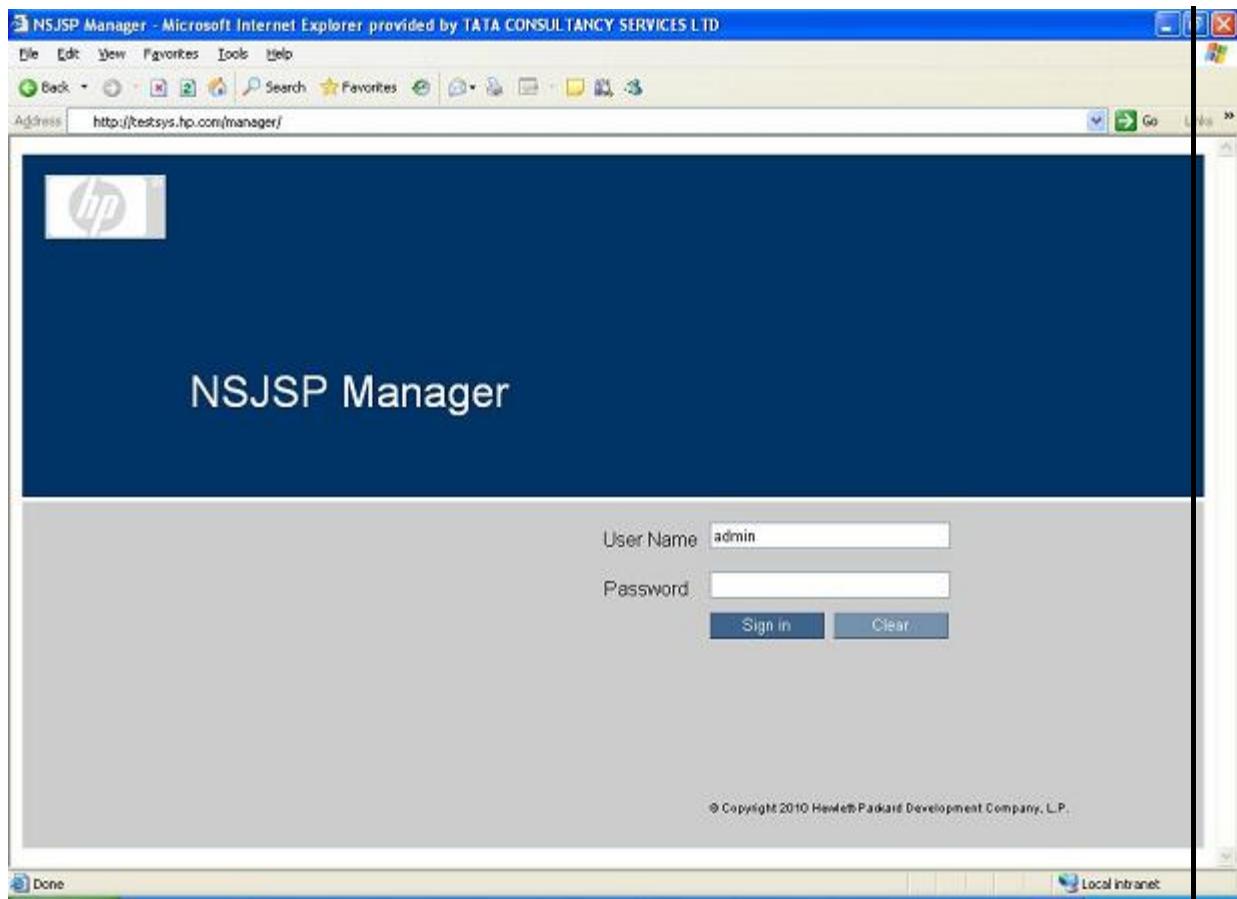
To deploy PetClinic on the NonStop system, complete the following steps:

1. Go to [http://<IP Address of the iTP WebServer>:<port#>/<manager\\_directory>](http://<IP Address of the iTP WebServer>:<port#>/<manager_directory>).

The **NSJSP Manager Login** screen appears.

Figure 2 shows the **NSJSP Manager Login** screen.

**Figure 2 NSJSP Manager Login Screen**

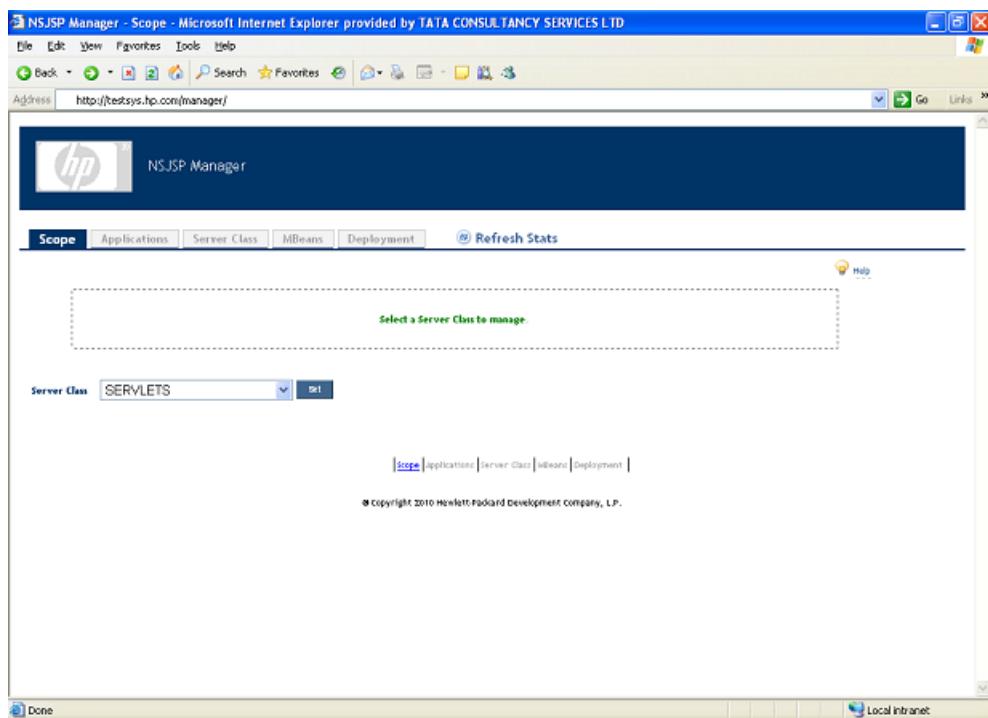


2. Enter the **User name:** and **Password:** and click **OK**.

The **NSJSP Manager** screen appears.

Figure 3 shows the **NSJSP Manager** screen.

**Figure 3 NSJSP Manager Screen**

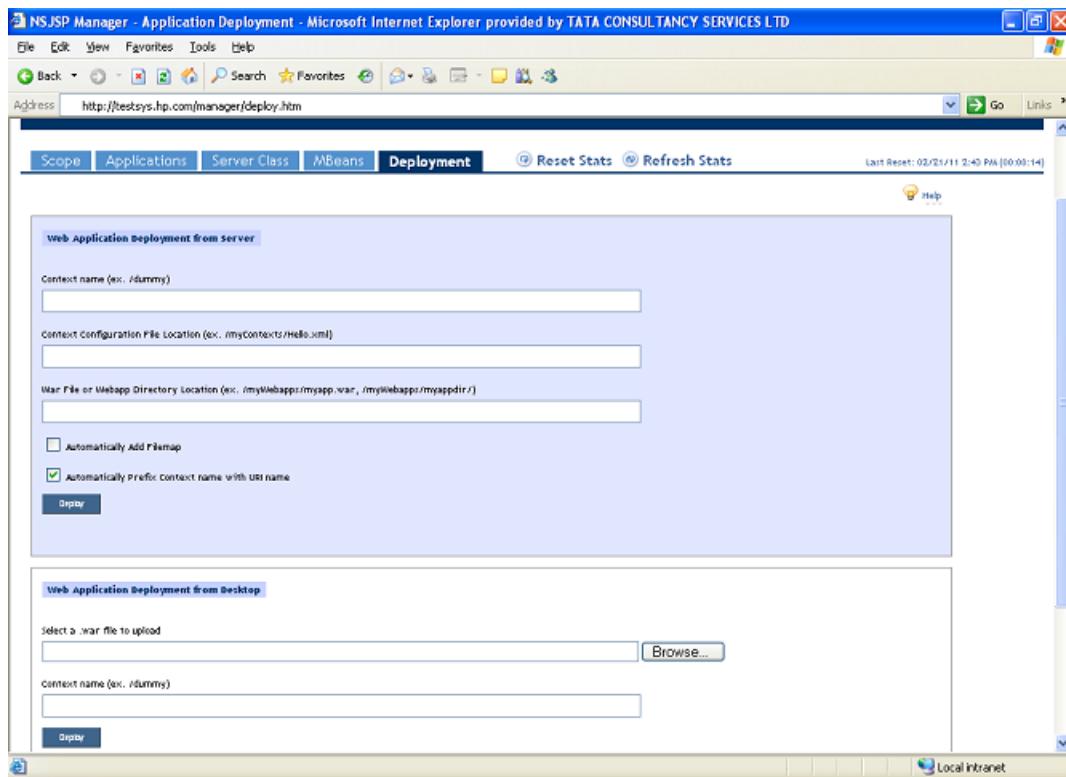


3. Select **SERVLETS** for the **Server Class** and click **Set**.

The **Host (in server.xml)** tab is enabled and populated with **localhost**. Click **Set**, which enables the **Deployment** tab on the **NSJSP Manager** screen.

Figure 4 shows the **Deployment** tab of **NSJSP Manager**.

**Figure 4 NSJSP Manager Screen - Deployment tab**



4. In the **Deployment** tab, complete the following steps in the **Web Application Deployment from Desktop** section:

1. In the **Select a .war file to upload** field, click **Browse...** and locate the `petclinic.war` file on the Windows system.
2. **(Optional)** In the **Context name** field, enter a name for the application context.
3. Click **Deploy**.

PetClinic is deployed on NSJSP and is listed under **Applications** tab of the **NSJSP Manager** screen.

**NOTE:** HP recommends that you use the NSJSP manager for deployment. Deployment using the FTP or any other mechanism is not recommended. For more information on using the NSJSP manager, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

## Running PetClinic on NonStop

To run PetClinic on the NonStop system, click `/<servlet directory>/petclinic` path under **Applications** on the **NSJSP Web Application Manager** screen.

You can now perform the following actions using PetClinic:

- View a list of veterinarians and their specialties
- View and update information of a pet owner
- Add a new pet owner to the system
- View and update information of a pet

- Add a new pet to the system
- View information about the pet's visit to the clinic

## JPetStore

The intended users of the JPetStore sample application are Visitors and Shoppers. A Visitor is anyone who visits the site. A Shopper is an authenticated visitor who is signed in to the site. Visitors can browse the site for items of their interest. Shoppers can purchase items of their interest.

This section describes the following steps for the JPetStore sample application.

- “Building JPetStore on Windows” (page 46)
- “Setting up JPetStore Database on NonStop” (page 47)
- “Deploying JPetStore on NonStop” (page 48)
- “Running JPetStore on NonStop” (page 51)

### Building JPetStore on Windows

To build JPetStore on the Windows system, complete the following steps:

1. Go to the `<My SASH Home>\spring\samples\jpetstore` directory on the Windows system. Among other files, this directory includes the following sub-directories:

`\db`

includes the database tables creation scripts and dataload scripts.

`\src`

includes the customized source for JPetStore.

2. Configure the JDBC driver settings for the NonStop SQL/MX database.

1. Go to the `<My SASH Home>\spring\samples\jpetstore\src\main\webapp\WEB-INF` directory on the Windows system.

2. Modify the `jdbc.properties` file to update the JDBC configuration. You can use either the JDBC Type 2 driver or the JDBC Type 4 driver by setting the SQL/MX settings as follows:

- To use the JDBC Type 2 driver, uncomment the SQL/MX settings for the JDBC Type 2 driver in the `jdbc.properties` file, so that the SQL/MX settings for the JDBC Type 2 driver appear as:

```
#-----
# SQL/MX Settings for JDBC Type 2 Driver
jdbc.driverClassName=com.tandem.sqlmx.SQLMXDriver
jdbc.url=jdbc:sqlmx://
jdbc.username=
jdbc.password=
jdbc.catalog=jpetstorecat
jdbc.schema=jpetstoresch
```

---

**NOTE:** Because the JDBC Type 2 driver is located on the NonStop system, you need not mention the `username` and `password` in the `jdbc.username` and `jdbc.password` fields.

- To use the JDBC Type 4 driver, uncomment the SQL/MX settings for JDBC Type 4 driver in the `jdbc.properties` file, and enter the values of the JDBC URL (NonStop system IP Address and port number of the JDBC data source), NonStop system `username`, and `password`. The SQL/MX settings for the JDBC Type 4 driver in the `jdbc.properties` file appear as:

```
#-----
# SQL/MX Settings for JDBC Type 4 Driver
jdbc.driverClassName= com.tandem.t4jdbc.SQLMXDriver
jdbc.url= jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.>
```

```
jdbc.username=<HP NonStop Username>
jdbc.password=<HP NonStop Password>
jdbc.catalog=jpetstorecat
jdbc.schema=jpetstoresch
```

---

**NOTE:** <HP NonStop Username> must have access to the schema created for JPetstore.

---

3. Build the JPetStore WAR file

(org.springframework.samples.jpetstore-1.0.0-SNAPSHOT.war).

1. Go to the <My SASH Home>\spring\samples\jpetstore directory:

command prompt> cd <My SASH Home>\spring\samples\jpetstore

2. Build the JPetStore web archive using the JDBC Type 2 driver or JDBC Type 4 driver.

- To use the JDBC Type 2 driver:

command prompt> mvn clean package

The application WAR file

(org.springframework.samples.jpetstore-1.0.0-SNAPSHOT.war) is created in the <My SASH Home>\spring\samples\jpetstore\target directory.

- To use the JDBC Type 4 driver:

1. Enter the following command:

command prompt> mvn clean package

The application WAR file

(org.springframework.samples.jpetstore-1.0.0-SNAPSHOT.war) is created in the <My SASH Home>\spring\samples\jpetstore\target directory.

2. Create the WEB-INF\lib folder in the <My SASH Home>\spring\samples\jpetstore\target directory.

3. Copy the JDBC Type 4 driver file (t4sqlmx.jar) to <My SASH Home>\spring\samples\jpetstore\target\WEB-INF\lib.

4. Go to <My SASH Home>\spring\samples\jpetstore\target and run the following command:

```
Command Prompt> cd <My SASH Home>\spring\samples\jpetstore\target
Command Prompt> jar uf org.springframework.samples.jpetstore-1.0.0-SNAPSHOT.war WEB-INF
\lib\t4sqlmx.jar
```

The t4sqlmx.jar is copied to <My SASH

Home>\spring\samples\jpetstore\target\

org.springframework.samples.jpetstore-1.0.0-SNAPSHOT.war\WEB-INF\lib.

## Setting up JPetStore Database on NonStop

To set up the JPetStore database on the NonStop system, complete the following steps:

1. Edit the <My SASH

Home>\spring\samples\jpetstore\db\mxci\jpetstore\_tables\_script.sql file to add the following:

- <\$datavol>: specify the Guardian volume for the primary partition of the table. For example: \$data01.
- <user>: specify the owner of the schema. For example: super.sashusr.

- <node.\$vol>: specify the location of the metadata tables for the catalog. For example: \NSK01.\$data01.
- <subvol reference>: specify the designated subvolume name for the schema. For example: SASH2 .

---

**NOTE:** The subvolume name is always prefixed by ZSD and must have eight characters (including ZSD).

---

2. Create a directory in OSS to place the database script files:

```
OSS> mkdir -p <NonStop SASH Home>/spring/samples/jpetstore/dbconfig
```

For example:

```
OSS> mkdir -p /home/sash_usr/sash/spring/samples/jpetstore/dbconfig
```

---

**NOTE:** <NonStop SASH Home> can be any working directory. It is suggested that you create the spring/samples/jpetstore/dbconfig directory structure in <NonStop SASH Home>.

---

3. Copy the following scripts from the <My SASH Home>\spring\samples\jpetstore\db Windows directory to the <NonStop SASH Home>/spring/samples/jpetstore/dbconfig OSS directory:

- jpetstore\_tables\_script.sql
  - jpetstore\_dataload\_script.sql
- 

**NOTE:**

- The jpetstore\_tables\_script.sql script creates the SQL/MX database catalog, schema, and tables for JPetStore.
  - The jpetstore\_dataload\_script.sql script loads the database with sample data.
- 

4. Go to the OSS directory, where the jpetstore\_tables\_script.sql and jpetstore\_dataload\_script.sql files are copied:

```
OSS> cd <NonStop SASH Home>/spring/samples/jpetstore/dbconfig
```

For example:

```
OSS> cd /home/sash_usr/sash/spring/samples/jpetstore/dbconfig
```

5. Create the JPetStore database catalog, schema, and tables:

```
mxci>> obey jpetstore_tables_script.sql;
```

---

**NOTE:** By default, the jpetstore\_tables\_script.sql file creates the database catalog name as jpetstorecat and schema name as jpetstoresch. If these names conflict with the existing catalog and schema names, modify the database catalog and schema names in the jpetstore\_tables\_script.sql script file.

If you modify the database catalog and schema names, the new names must be updated in the jdbc.properties file located in the <My SASH Home>\spring\samples\jpetstore\src\main\webapp\WEB-INF directory, included in the SAMPLES.zip file.

---

6. Load the sample data in the JPetStore database tables:

```
mxci>> obey jpetstore_dataload_script.sql;
```

The JPetStore database is set up on the NonStop system.

## Deploying JPetStore on NonStop

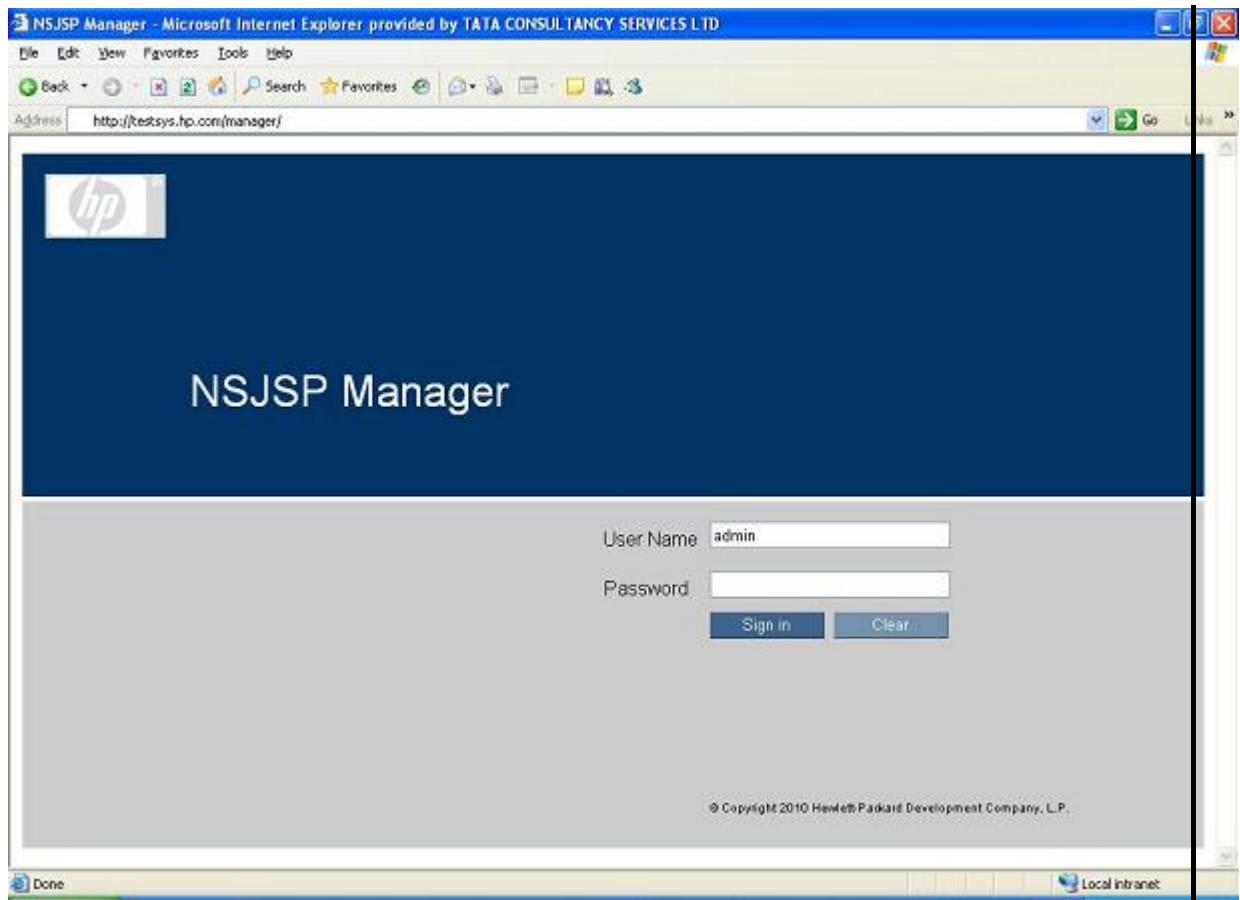
To deploy JPetStore on the NonStop system, complete the following steps:

1. Go to [http://<IP Address of the iTP WebServer>:<port#>/<manager\\_directory>](http://<IP Address of the iTP WebServer>:<port#>/<manager_directory>).

The **NSJSP Manager Login** screen appears.

[Figure 5](#) shows the **NSJSP Manager Login** screen.

[Figure 5 NSJSP Manager Login Screen](#)

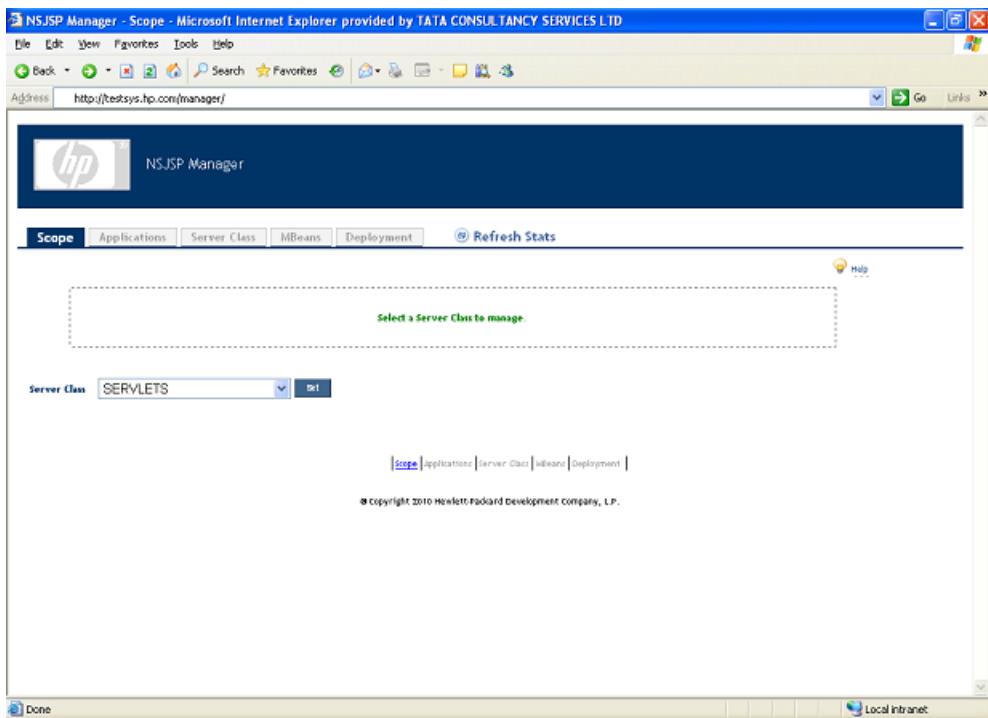


2. Enter the **User name:** and **Password:** and click **OK**.

The **NSJSP Manager** screen appears.

[Figure 6](#) shows the **NSJSP Manager** screen.

**Figure 6 NSJSP Manager Screen**

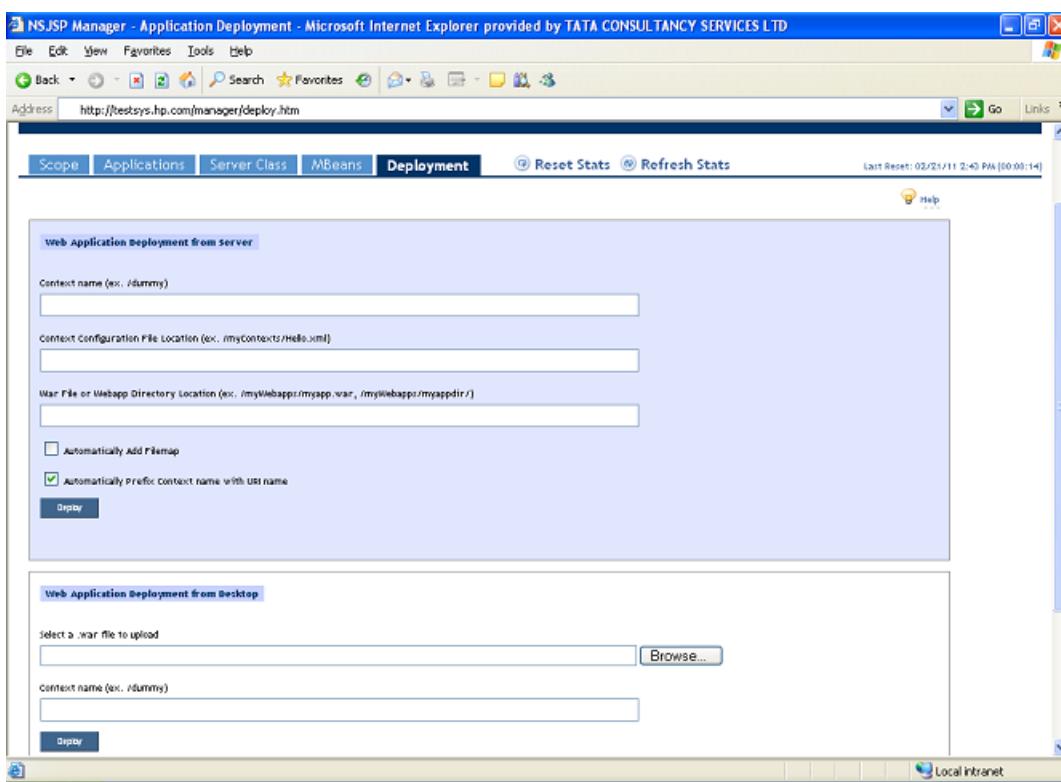


3. Select **SERVLETS** for the **Server Class** and click **Set**.

The **Host (in server.xml)** tab is enabled and populated with **localhost**. Click **Set**, which enables the **Deployment** tab on the **NSJSP Manager** screen.

Figure 4 shows the **Deployment** tab of **NSJSP Manager**.

**Figure 7 NSJSP Manager Screen - Deployment tab**



4. In the **Deployment** tab, complete the following steps in the **Web Application Deployment from Desktop** section:
  1. In the **Select a .war file to upload** field, click **Browse...** and locate the `org.springframework.samples.jpetstore-1.0.0-SNAPSHOT.war` file on the Windows system.
  2. **(Optional)** In the **Context name** field, enter a name for the application context.
  3. Click **Deploy**.

JPetStore is deployed on NSJSP and is listed under **Applications** tab of the **NSJSP Manager** screen.

**NOTE:** HP recommends that you use the NSJSP manager for deployment. Deployment using the FTP or any other mechanism is not recommended. For more information on using the NSJSP manager, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

## Running JPetStore on NonStop

To run JPetStore on the NonStop system, click the `/<servlet directory>/org.springframework.samples.jpetstore-1.0.0-SNAPSHOT` path under **Applications** in the NSJSP Web Application Manager screen.

As a visitor, you can now perform the following activities using JPetStore:

- Select a Product Category labeled with words or icons, view the item detail, or access the MenuBar features.
- Search by entering a keyword or phrase to locate items.
- Add an item to a Cart, either from a summary list or from a detail view.
- Collect items for later purchase.
- Increase or decrease the quantity for any item in a Cart, up to available inventory, or remove the item from the Cart by changing the quantity to zero.
- Complete an online form to create an Account.

As a shopper, you can perform the following activities including those performed by a visitor using JPetStore:

- Enter a username and password to access your account.
- Store shipping information and preferences for your use at future sessions.
- Purchase the items collected in a Cart.
- Provide payment information.

# 4 Configuring Spring Applications on NonStop Systems

This chapter provides information about configuring Spring applications on NonStop systems and includes the following sections:

- “NonStop Platform Configurations” (page 52)
- “Spring Framework Configurations” (page 62)
- “Module File Caching Configurations” (page 73)

## NonStop Platform Configurations

On a NonStop system, an application developed using the Spring framework is deployed as a web application under the NSJSP container. Thus, it inherits all the NSJSP properties and configurations, and runs like any other NSJSP web application running on a NonStop system. This section discusses the following tasks:

- “Determining the Application Parameters” (page 52)
- “Determining the Maximum Capacity of NSJSP Instance” (page 52)
- “Configuring iTP WebServer for Spring Applications” (page 53)
- “Configuring NSJSP for Spring Applications” (page 57)

### Determining the Application Parameters

You need to determine the following parameters of your application:

- Average Response Time
- Average Load
- Maximum Load

For a unit response time, the number of active requests to the iTP WebServer is a maximum of its static capacity during the average load conditions. During peak load conditions, the number of active requests to the iTP WebServer is a maximum of its dynamic capacity.

For example,

If the average response time of your Spring application is 1 second and the iTP WebServer is configured to handle 100 requests statically and 50 requests dynamically, then during average load conditions, the number of active requests to the iTP WebServer can be a maximum of 100. During peak load conditions, 50 more requests can be served dynamically and hence, the active requests to the iTP WebServer can be a maximum of 150. Further requests are queued up and remain inactive until started by httpd.

The other parameters, Average Load and Maximum Load, help you decide the configuration of the static and dynamic httpd processes of the iTP WebServer.

### Determining the Maximum Capacity of NSJSP Instance

Before configuring NSJSP parameters, you must determine the capacity of a single instance of NSJSP. To determine the maximum load for a single instance of NSJSP, it is important to first configure the relevant TS/MP and server parameters of NSJSP (of the single instance) to their maximum limit in the following way:

1. Set the value of Numstatic to 1. This limits the number of static instances of NSJSP to 1.
2. Set the value of Maxservers to 1. This limits the number of NSJSP instances that can be started to 1. This implies that TS/MP cannot start more than one instance of NSJSP.

3. Set the value of `Maxlinks` to 250. This is the maximum number of links to the server process (NSJSP process). This means that the server process must be capable of processing 250 requests simultaneously.
4. Set the value of `TANDEM_RECEIVE_DEPTH` to 250 (because the values of `Maxlinks` and `TANDEM_RECEIVE_DEPTH` should be equal). A value of 250 means that the NSJSP process is able to read a maximum of 250 messages simultaneously from its `$RECEIVE` file.
5. Configure the `Executor` element in the `<NSJSP Deployment Directory>/conf/server.xml` file on OSS. The `Executor` is used by the Connector as a thread pool. Set `maxThreads = 300`, `minSpareThreads = 10` and `maxIdleTime = 30000`. Using an `Executor` helps monitor the number of active threads at any given time. A value of 300 for `maxThreads` ensures that you have enough threads to process all the incoming requests (a maximum of 250) simultaneously. A value of 30000 for `maxIdleTime` ensures that if a thread is idle for more than 30 seconds, that thread will be stopped.

After configuring the parameters, you might want to use a tool that can simulate the HTTP clients and can handle the HTTP cookies. The tool reveals the number of HTTP clients that a single instance of NSJSP can handle and indicates the number of simultaneous HTTP requests that NSJSP is capable of handling.

---

**NOTE:** There are a number of HTTP client simulators available, for example, Apache JMeter, HP LoadRunner, and Radview Webload. These tools provide a good interface to monitor the test results. You can use any of these tools to determine the maximum handling capacity of each NSJSP instance.

---

To arrive at the required numbers, complete the following steps:

1. Run the test tool with a single instance of HTTP client simulator.

---

**NOTE:** The test tool must be capable of displaying the response time of the HTTP requests.

---

2. Monitor the response time of the HTTP requests and allow your application to attain a steady state (in terms of response time per HTTP request).
3. At steady state, check if the response times are within the Service Level Agreement (SLA).
4. If the response time is within the SLA, increase the number of HTTP client instances and repeat step 2 onwards.
5. If the response time is beyond the acceptable SLA, stop the tests and determine the number of HTTP clients and the number of HTTP requests that NSJSP instance could process simultaneously.

While the test tool can indicate the number of HTTP clients that were being processed simultaneously, the number of simultaneous HTTP requests can be arrived at using different means. Following are some of the calculation strategies:

- The number of HTTP requests recorded by the testing tool and idea of the number of requests, which required processing by NSJSP, provides the total number of HTTP requests processed by NSJSP.
- The total number of HTTP requests processed by NSJSP, together with the test duration, indicates the average number of simultaneous HTTP requests handled by NSJSP.

## Configuring iTP WebServer for Spring Applications

The `httpd` process of the iTP WebServer is responsible for managing the load and forwarding the requests to their corresponding application servers. The context of this section is limited to the configurations related to `httpd` processes only.

Configuring the `httpd` process involves the following tasks:

- “Configuring `httpd` Processes to Handle Maximum Load” (page 54)
- “Limiting the Maximum Number of Incoming Requests” (page 56)

## Configuring httpd Processes to Handle Maximum Load

The httpd processes are configured using the Server configuration directive in the *<iTP WebServer Deployment Directory>/conf/httpd.config* file on OSS. The Server directive controls the creation of the PATHMON environment, where the webserver runs.

A typical default configuration of the httpd process of the iTP WebServer appears as follows:

```
Server $root/bin/httpd {
    eval $DefaultServerAttributes
    CWD [pwd]
    Arglist -server [HTTPD_CONFIG_FILE]
    Env TANDEM_RECEIVE_DEPTH=50
    Priority 170
    Numstatic 5
    Maxservers 50
    MapDefine =TCPIP^RESOLVER^NAME /G/system/ztcpip/resconf
    MapDefine =TCPIP^PROCESS^NAME $transport
}
```

The number of httpd processes are governed by the following TS/MP attributes:

- Numstatic: This specifies the number of static servers running under PATHMON. The static processes run on the system, irrespective of the load.
- Maxservers: This specifies the maximum number of server processes that can run under PATHMON.
- TANDEM\_RECEIVE\_DEPTH: This specifies the capacity of each of the configured httpd processes.
- (Maxservers - Numstatic): The difference denotes the number of dynamic servers. A dynamic server is a need-based server. When the iTP WebServer is under heavy load and all the static httpd processes are busy, a dynamic server process, httpd, is created to support the excess load. These servers are dynamically created by TS/MP and are terminated once the process is complete.

The capacity of the iTP WebServer environment can be summarized as:

- The static capacity of iTP WebServer is [Numstatic X TANDEM\_RECEIVE\_DEPTH].
- The dynamic capacity is [(Maxservers - Numstatic) X TANDEM\_RECEIVE\_DEPTH] requests.
- The total capacity of iTP WebServer is [Maxservers X TANDEM\_RECEIVE\_DEPTH].

Example 1:

Assume that the httpd process of iTP WebServer has the following configurations:

```
Server $root/bin/httpd {
    eval $DefaultServerAttributes
    CWD [pwd]
    Arglist -server [HTTPD_CONFIG_FILE]
    Env TANDEM_RECEIVE_DEPTH=100
    Priority 170
    Numstatic 5
    Maxservers 50
    MapDefine =TCPIP^RESOLVER^NAME /G/system/ztcpip/resconf
    MapDefine =TCPIP^PROCESS^NAME $transport
}
```

When you start the iTP WebServer, five static httpd processes will be started, governed by the value of Numstatic. Because the value of TANDEM\_RECEIVE\_DEPTH is set to 100,

each of the five static processes can handle 100 requests. In this example, the capacity of the iTP WebServer environment can be summarized as follows:

- The static capacity of iTP WebServer is [Numstatic X TANDEM\_RECEIVE\_DEPTH]= 500.
- The dynamic capacity is [(Maxservers - Numstatic) X TANDEM\_RECEIVE\_DEPTH] = 4500.
- The total capacity of iTP WebServer is [Maxservers X TANDEM\_RECEIVE\_DEPTH] = 5000.

Using this configuration, the iTP WebServer can handle 500 simultaneous requests statically. As soon as it receives the 501st request, a dynamic httpd process is created, which functions as a normal httpd process. The capacity of each of the dynamic httpd processes is the value of TANDEM\_RECEIVE\_DEPTH. When the load decreases, the dynamic httpd processes goes down.

For more information on the Server configuration directive of iTP WebServer, see the *iTP Secure WebServer System Administrator's Guide*.

### Guidelines for Configuring the Server Directive

Before configuring the various attributes of the Server directive, you must determine the following parameters:

- Processors for httpd processes
- Application parameters: Average Load, Peak Load, and Average Response Time

### Processors for httpd Processes

When you start the iTP WebServer, the static httpd processes equal to the value of Numstatics are started on your NonStop system. By default, all the configured processors are taken into account. If the value of Numstatic is set to n, and the number of configured processors on the system are equal to n, one static httpd process is started on each of the processors. However, if the value of Numstatic is less than the number of configured processors, one static httpd process starts on some of the processors, while the remaining processors will not have any static httpd process.

The following example explains how you can determine the processors on which you intend to start the httpd processes and configure the processors directive.

Example:

Assume that you have a 16-processor system and you want to start httpd processes on processors 0, 1, 2, 3, 4, 5. For this, the processors configuration attribute must be configured as processors 0 1 2 3 4 5.

In this scenario:

- If you set the value of Numstatic equal to 6, each of the processors 0, 1, 2, 3, 4, 5 will have one static httpd process.
- If you set the value of Numstatic equal to a multiple of 6, each of the processors 0, 1, 2, 3, 4, 5 will have (Numstatic/6) static httpd processes.
- If you set the value of Numstatic equal to 5, any 5 of the processors 0, 1, 2, 3, 4, 5 will have one static httpd process, and remaining processor will not have any httpd process.
- If you set the value of Numstatic equal to 9, any three of the processors 0, 1, 2, 3, 4, 5 will have two static httpd process and each of the remaining three processors will have one httpd process.

After determining the values, you can use any the following approaches to make necessary configurations for the TS/MP parameters of the `httpd` processes:

- Using the default value of `TANDEM_RECEIVE_DEPTH`
- Modifying the value of `TANDEM_RECEIVE_DEPTH`

#### Using the default value of `TANDEM_RECEIVE_DEPTH`

With `TANDEM_RECEIVE_DEPTH` set to the default value of 50, the static processes can be configured on the basis of Average Load and dynamic processes can be configured on the basis of Peak Load on your system. You may use the following approaches while deciding the values of `Numstatic` and `Maxservers`:

- The value of `Numstatic` must be set to  $(\text{Average Load})/50$
- The value of `Maxservers` must be set to  $(\text{Peak Load})/50$

For example:

If the Average Load and Peak Load on your Spring application turn out to values 1000 and 1500 requests and you limit `TANDEM_RECEIVE_DEPTH` to its default value of 50, the `Numstatic` must be set to 20 and `Maxservers` must be set to 30.

---

**NOTE:** It is advisable to set the value of `Numstatic` to, at least, the number of configured processors.

---

#### Modifying the value of `TANDEM_RECEIVE_DEPTH`

If the number of `httpd` processes configured on the basis of default value of `TANDEM_RECEIVE_DEPTH` does not fulfill your application requirement, you can change the value of `TANDEM_RECEIVE_DEPTH` and calculate the values of `Numstatic` and `Maxservers` accordingly. However, you must consider the following before increasing the value of `TANDEM_RECEIVE_DEPTH`:

- It is recommended to keep the web server running with 80 percent of its capacity. To achieve this, set the value of `TANDEM_RECEIVE_DEPTH` accordingly.  
For example, if your calculation suggests that `TANDEM_RECEIVE_DEPTH` of 80 is required, you must set it to a value of 100.
- The maximum value of `TANDEM_RECEIVE_DEPTH` is 255.
- Do not reduce `TANDEM_RECEIVE_DEPTH` to a value less than 50.

### Limiting the Maximum Number of Incoming Requests

Because of constraints on available system resources, you might want to limit the number of requests to your iTP WebServer. If you want to configure your iTP WebServer to handle only a certain number of requests at an instance, use the `MaxConnections` configuration directive to specify the maximum number of connections that can be served by iTP WebServer at any given instance. The iTP WebServer serves the number of requests equal to the multiple of `Numstatic` greater than or equal to `MaxConnections` count.

#### Syntax:

```
MaxConnections -count <integer value> -replytype <customized/RST>
```

For example:

```
MaxConnections -count 101 -replytype RST
```

Consider the scenario of Example 1 above, where `Numstatic` is 5 with the following `MaxConnections` configuration:

```
MaxConnections -count 101 -replytype customized
```

In this case, the iTP WebServer serves 105 requests (higher multiple of Numstatic nearest to the count value). Here, the 106th request displays the following error message:

Maximum connections reached: The server reached its maximum configured capacity.  
with HTTP response code:

200 OK

To customize the error message, create a new message ID error-maximum-connection. This customized message is displayed if the message configuration directive is used in the <iTP WebServer Deployment Directory>/conf/httpd.config file with the new message ID. For more information on the MaxConnections configuration directive and creating a customized error message using the Message configuration directive, see the *iTP Secure WebServer System Administrator's Guide*.

---

**NOTE:** To use the MaxConnections configuration directive, your iTP WebServer must be configured for static environment only, that is, the values of Numstatic and MaxServers of the httpd process must be equal.

---

## Configuring NSJSP for Spring Applications

When a Spring application runs on a NonStop system, it runs as an instance of NSJSP. Therefore, before configuring the NSJSP environment, it is important to determine the load each instance of NSJSP is expected to handle.

This section describes the following configuration aspects:

- “Configuring SessionBasedLoadBalancing” (page 57)
- “Configuring Connector Threads” (page 58)
- “Configuring TS/MP Specific Parameters” (page 59)
- “Configuring Java Runtime Arguments” (page 61)

### Configuring SessionBasedLoadBalancing

The NSJSP Container maintains sessions in the form of serialized Java objects. Each session object is identified by a unique identifier called the session-ID. The session-ID is sent to the HTTP client either as a cookie or in the form of URL-rewriting. The name of the cookie is JSESSIONID and when the user application creates a session, NSJSP generates a cookie the JSESSIONID as the name and session-ID as the cookie value. The session objects can either be kept in the process memory or persisted in a persistent store (for example, a database table). When a session object is kept in a process, it is available only for the process that created it. If it is kept in a persistent store, it is available for any process under the NSJSP environment. When the SessionBasedLoadBalancing feature is enabled, all requests related to a particular session are routed to the process that has the session object in its memory.

To enable the SessionBasedLoadBalancing feature, you must modify the configurations of the servlet.ssc object in the <iTP WebServer Deployment Directory>/conf/servlet.config file on OSS. The servlet.ssc object is configured under the Server directive. The SessionBasedLoadBalancing feature is governed by the -DSaveSessionOnCreation and -DSessionBasedLoadBalancing arguments in the Arglist of the Server directive.

- -DSaveSessionOnCreation

Enables or disables saving the sessions in a persistent store during their creation time.

Syntax:

-DSaveSessionOnCreation= [ true | false ]

---

**NOTE:** The default value of `-DSaveSessionOnCreation` is false.

---

- `-DSessionBasedLoadBalancing`

Enables or disables SessionBasedLoadBalancing.

Syntax:

```
-DSessionBasedLoadBalancing=[ true | false ]
```

---

**NOTE:** The default value of `-DSessionBasedLoadBalancing` is set to true.

---

For example, if you want to enable the SessionBasedLoadBalancing feature for your Spring application and save the application sessions in a persistent store, set the Arglist as:

```
Server $server_objectcode {  
    ...  
    ...  
    ...  
    ...  
  
    Arglist -DSessionBasedLoadBalancing=true \  
-DSaveSessionOnCreation=true \  
    ...  
    ...  
    ...  
    ...  
}
```

where,

`server_objectcode` is mapped to the `servlet.ssc` object in the `<iTP WebServer Deployment Directory>/bin` directory on OSS.

While setting these arguments, consider the following:

1. If both the `SaveSessionOnCreation` and `SessionBasedLoadBalancing` options are set to false, and if a Persistent Manager is configured with a persistent store, 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.
2. Enabling or disabling the `SessionBasedLoadBalancing` feature depends on the application requirement. Your Spring application might encounter any of the following scenarios:
  - The application depends heavily on the state stored in session objects. Therefore, session objects cannot be lost and the application cannot recover from loss of state.
  - Application response is of prime importance and the application can recover from a loss of state.
  - The application is expected to handle largely varying loads and having a large number of static NSJSP instances is not an option.
  - The session objects must be valid for large durations (like an entire day).
  - The session objects must be available whenever the application restarts.

For more information on the `SessionBasedLoadBalancing` and `SessionBasedLoadBalancing` configuration directives, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

## Configuring Connector Threads

The Connector is responsible for creating and maintaining the threads that are used for message processing. A thread is allocated for each incoming message; therefore, the number of messages

that can be processed simultaneously can be controlled by limiting the number of threads that NSJSP Connector can spawn.

---

**NOTE:** A single connector can service multiple instances of the container. The connector must be configured such that it can spawn threads sufficient for all the container instances. In terms of configuration, the Host element in the server.xml configuration file represents a container that is serviced by the connector.

---

The NSJSP Connector thread pool can be connected using the following ways:

- ["Configuring the Connector Element" \(page 59\)](#)
- ["Configuring the Executor Element" \(page 59\)](#)

### Configuring the Connector Element

Configure the maxThreads attribute of the Connector element in the <NSJSP Deployment Directory>/conf/server.xml file on OSS. The Connector element is the child element of the Service element.

For example:

The following snippet shows the configuration in the <NSJSP Deployment Directory>/conf/server.xml file on OSS, if you want to configure 75 simultaneous connector threads:

```
...
<Service name="NSJSP">
    <Connector protocol="HTTP/1.1"
               connectionTimeout="0"
               acceptCount="25"
               maxThreads="75" />
...
</Service>
```

### Configuring the Executor Element

You can configure the Executor element as a child element of the Service element in the <NSJSP Deployment Directory>/conf/server.xml file on OSS. The following snippet of the server.xml configuration file shows the configuration of the Executor element and how a Connector element can be configured to use an Executor element:

```
...
<Service name="NSJSP">
    <Executor name="SpringExec"
              className="org.apache.catalina.core.StandardThreadExecutor"
              namePrefix="SpringAPP"
              maxThreads="75"
              minSpareThreads="20"
              maxIdleTime="10000" />
    <Connector executor="SpringExec" />
...
...
```

---

**NOTE:** All executors that are configured in NSJSP must implement the Java interface org.apache.catalina.Executor. NSJSP provides a standard implementation of this interface in the class org.apache.catalina.core.StandardThreadExecutor.

---

## Configuring TS/MP Specific Parameters

This section describes the following TS/MP specific configuration parameters:

1. Numstatic – Determines the number of static NSJSP processes.
2. Maxservers – Determines the total number of NSJSP processes.
3. TANDEM\_RECEIVE\_DEPTH – Determines the handling capacity of each instance of NSJSP.
4. Maxlinks – Determines the maximum number of TS/MP links. The number of simultaneous messages that can be delivered to an instance of NSJSP is determined by Maxlinks. NSJSP reads these many number of messages from the \$RECEIVE file simultaneously.
5. Number of Connector threads.

The configurations for Numstatic, Maxservers, TANDEM\_RECEIVE\_DEPTH, and Maxlinks can be set under the Server directive in the *<iTP WebServer Deployment Directory>/conf/servlet.config* file on OSS. Before configuring NSJSP, it is important that you determine the maximum load that each instance of NSJSP is expected to handle. The manner in which these configuration directives drive the NSJSP environment depends on whether the SessionBasedLoadBalancing feature is turned ON or OFF.

- “[NSJSP Configured with SessionBasedLoadBalancing Turned OFF](#)” (page 60)
- “[NSJSP Configured with SessionBasedLoadBalancing Turned ON](#)” (page 61)

### [NSJSP Configured with SessionBasedLoadBalancing Turned OFF](#)

- The value of Numstatic must be  $[(\text{Average Load}) / (\text{Max load one instance of NSJSP can handle})]$ .
- Maxservers must be  $[(\text{Peak Load}) / (\text{Max load one instance of NSJSP can handle})]$ .
- The value of Maxlinks must be set to the peak load that one instance of NSJSP is expected to handle.
- TANDEM\_RECEIVE\_DEPTH must be twice the value of Maxlinks. Normally, to handle  $n$  number of messages, you need  $n$  number of threads. In extreme cases, all the  $n$  requests may timeout. While the same number of threads processing the timeout requests could still be busy, you need the same  $n$  number of threads to process the new incoming requests. Therefore, the number of threads that must be configured is  $(n + n = 2n)$ .
- The maximum number of connector threads, maxThreads must be equal to the value of TANDEM\_RECEIVE\_DEPTH.

For example:

The parameters Maxlinks, TANDEM\_RECEIVE\_DEPTH, and the connector threads are closely linked to each other. These parameters are explained using examples. The following examples assume that the `httpd` processes are configured to serve 100 simultaneous HTTP requests, and all 100 HTTP requests are serviced by the Spring application deployed in NSJSP (that is, there is no static content in the application). Also, the peak load that one single instance of NSJSP can handle is 25.

Because the SessionBasedLoadBalancing feature is turned OFF, all messages between `HTTPPD` and `NSJSP` flow through TS/MP. The number of simultaneous messages that can be delivered to an instance of NSJSP is determined by the value of Maxlinks; set the value of Maxlinks to 25. NSJSP can now read 25 messages from its `$RECEIVE` queue simultaneously. In an extreme case, all the 25 messages time out; there must be enough space in the `$RECEIVE` queue to accommodate 25 more messages. Therefore, `$RECEIVE` must be opened with a Receive Depth (controlled by `TANDEM_RECEIVE_DEPTH`) of (25+25). Thus, you must set the value of `TANDEM_RECEIVE_DEPTH` to 50. The reason for setting `TANDEM_RECEIVE_DEPTH` to a value of 50 can be explained as:

To handle 25 messages, you need 25 threads. In an extreme case where all the 25 requests time out, you need another 25 threads to process the new incoming 25 requests because the threads

processing the timeout requests could still be busy. Therefore, the number of threads that must be configured is  $(25+25 = 50)$ .

### NSJSP Configured with SessionBasedLoadBalancing Turned ON

- The value of Numstatic must be  $[(\text{Peak Load}) / (\text{Max load one instance of NSJSP can handle})]$ .
- The value of Maxservers must be  $[(\text{Peak Load}) / (\text{Max load one instance of NSJSP can handle})]$ . This ensures that no dynamic process is created, so that the session object is not lost when there are many File System calls.
- The value of Maxlinks must be set to the peak load that one instance of NSJSP is expected to handle.
- Arriving at a definite value for TANDEM\_RECEIVE\_DEPTH is difficult when NSJSP is configured with SessionBasedLoadBalancing turned ON. Deciding an optimum value for TANDEM\_RECEIVE\_DEPTH requires a good understanding of the application. The following example provides an insight into making this decision.
- The maximum number of connector threads, maxThreads must be equal to the value of TANDEM\_RECEIVE\_DEPTH.

For example:

In this example, it is evident how a single instance of NSJSP, although having Maxlinks set to 25, can serve all the 100 requests.

Because SessionBasedLoadBalancing is turned ON, all messages between HTTPD and NSJSP flow through TS/MP and file system calls. The number of simultaneous messages that can be delivered to an instance of NSJSP is determined by the value of Maxlinks; set the value of Maxlinks to 25.

With Maxlinks set to 25, a single instance of NSJSP can handle 25 concurrent requests through TS/MP, all being the first requests of web dialogs. The first call on a web dialog is always delivered to NSJSP through TS/MP and there will not be any file system call. After the 25 requests are serviced, subsequent requests on those 25 web dialogs are delivered to NSJSP through file system I/O operations.

At this stage, all 25 links to the NSJSP instance are free to process more incoming requests delivered through TS/MP. Therefore, the NSJSP instance should now be able to handle 25 more concurrent requests all being the first requests of web dialogs. After these 25 requests are processed, the subsequent requests on these new connections arrive at NSJSP through the file system I/O. At this stage, NSJSP could be handling up to 50 concurrent requests and all these requests are being delivered through file system calls from httpd, not through TS/MP. Hence, the 25 links are free to process new web dialogs. Therefore, a single instance of NSJSP can serve all the 100 requests.

At this point, one instance of NSJSP could be processing requests from all the possible 100 connections to the httpd processes. This means that there could be a scenario where all the 100 requests time out. Therefore, there must be enough space in the \$RECEIVE file to handle  $(100+100)$  messages. That would mean TANDEM\_RECEIVE\_DEPTH should have a value of 200. The reason for setting TANDEM\_RECEIVE\_DEPTH to a value of 200 can be explained as:

To handle 100 requests, you need 100 threads. In an extreme case of all the 100 requests timing out, you need another 100 threads to process the new incoming 100 requests because the threads processing the timeout requests could still be busy. Therefore, the number of threads that need to be configured is  $(100+100 = 200)$ .

### Configuring Java Runtime Arguments

The configurations for Java runtime arguments can be written in the *<ITP WebServer Deployment Directory>/conf/servlet.config* file on OSS. The `sscaux` object is configured under the `Server` directive. Java runtime arguments are populated in the Arglist.

Some of the important Java runtime arguments that you must consider during the deployment of your Spring applications are:

- “`-Xmx`” (page 62)
- “`-Xss`” (page 62)
- “`-Xnklassgc`” (page 62)

There are other Java runtime arguments supported by NSJSP. For more information, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

#### `-Xmx`

Sets the maximum size of the memory allocation pool, which is the garbage collected heap.

Syntax:

```
-Xmx maximum-heap-size [ k | m ]
```

where,

#### **maximum-heap-size**

is the maximum size of the memory allocated for the garbage collected. It must be greater than or equal to 1000 bytes.

#### **k**

sets the value of `maximum-heap-size` to be read in kilobytes.

#### **m**

sets the value of `maximum-heap-size` to be read in megabytes.

#### `-Xss`

Sets the maximum stack size that can be used by a Java thread.

Syntax:

```
-Xmx maximum-stack-size
```

where,

#### **maximum-stack-size**

is the maximum size of the stack trace in kilobytes.

#### `-Xnklassgc`

Is an optional argument to stop the Java class garbage collection.

Syntax:

```
-Xnklassgc
```

By default, the Java runtime reclaims space for unused Java classes. Including this optional argument may prevent any memory-leak problems.

## Spring Framework Configurations

This section discusses the following configurations for your Spring applications:

- “Configuring JDBC Driver for SQL/MX Database” (page 63)
- “Configuring Database Transaction Management” (page 66)
- “Connection Pooling” (page 72)

JDBC configurations are carried out in `applicationContext.xml` and `jdbc.properties` files. All the configurations for Transaction Management and Connection Pooling are configured in `applicationContext.xml`.

## Configuring JDBC Driver for SQL/MX Database

The JDBC driver to be used when a Spring application (using Spring DAO) connects to the SQL/MX database needs to be specified in the `applicationContext.xml` and `jdbc.properties` file provided by the Spring framework.

This section discusses the following:

- “Configuring JDBC Type 2 Driver for SQL/MX Database” (page 63)
- “Configuring JDBC Type 4 Driver for SQL/MX Database” (page 65)

### Configuring JDBC Type 2 Driver for SQL/MX Database

To configure JDBC Type 2 Driver for connecting a Spring application with SQL/MX database, complete the following configurations:

1. “Configurations in the `jdbc.properties` File” (page 63)
2. “Configurations in the `applicationContext.xml` File” (page 63)

---

**NOTE:** Install JDBC Type 2 driver version T1275H50 by following the steps discussed in the “Installing the Spring Framework” (page 36) chapter.

#### Configurations in the `jdbc.properties` File

For making necessary JDBC configurations, complete the following activities:

- “Defining JDBC Type 2 Driver Class for SQL/MX Database” (page 63)
- “Defining the Connection URL” (page 63)
- “Establishing the Connection” (page 63)

#### Defining JDBC Type 2 Driver Class for SQL/MX Database

Specify the JDBC Type 2 driver class `com.tandem.sqlmx.SQLMXDriver` for the SQL/MX database in the `jdbc.properties` file:

```
jdbc.driverClassName=com.tandem.sqlmx.SQLMXDriver
```

#### Defining the Connection URL

After you have set the JDBC Type 2 driver, enter the location of the SQL/MX server. For URLs referring to SQL/MX database, use the `jdbc:` protocol embedded within the URL. To define the connection URL, specify the connection URL as `jdbc:sqlmx://` for the SQL/MX database in the `jdbc.properties` file:

```
jdbc.properties file:  
jdbc.url=jdbc:sqlmx://
```

#### Establishing the Connection

Specify the username and password with no values in the `jdbc.properties` file:

```
jdbc.username=  
jdbc.password=
```

#### Configurations in the `applicationContext.xml` File

For making necessary JDBC configurations, complete the following activities:

1. “Defining the placeholder for JDBC Type 2 Driver Class” (page 64)
2. “Defining the Placeholder for Connection URL” (page 64)
3. “Defining the Placeholder for Establishing the Connection” (page 64)
4. “Wiring of JDBC Properties” (page 64)

## Defining the placeholder for JDBC Type 2 Driver Class

Modify the `applicationContext.xml` file to define a placeholder for the JDBC Type 2 driver class name `com.tandem.sqlmx.SQLMXDriver` from the `jdbc.properties` file as shown below.

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClassName">
        <value>${jdbc.driver}</value>
    </property>
```

**NOTE:** It is recommended to use `com.mchange.v2.c3p0.ComboPooledDataSource` as the datasource class because it provides various parameters for connection pooling. In case you do not use connection pooling, you can either use `org.apache.commons.dbcp.BasicDataSource` provided by Apache DBCP or `org.springframework.jdbc.datasource.DriverManagerDataSource` provided by the Spring framework.

## Defining the Placeholder for Connection URL

Modify the `applicationContext.xml` file to define a placeholder for the connection URL from the `jdbc.properties` file as shown below.

```
<property name="url">
    <value>${jdbc.url}</value>
</property>
```

## Defining the Placeholder for Establishing the Connection

To establish the database connection, modify the `applicationContext.xml` file to define a placeholder for the username and password from the `jdbc.properties` file as shown below.

```
<property name="username">
    <value>${jdbc.user}</value>
</property>
<property name="password">
    <value>${jdbc.password}</value>
</property>
```

## Wiring of JDBC Properties

After all the database properties are set, wire the `jdbc.properties` file in the `applicationContext.xml` file so that the actual values are available during runtime. The `PropertyPlaceholderConfigurer` class provided by the Spring framework substitutes all the database properties specified in the `applicationContext.xml` file with the values specified in the `jdbc.properties` file during runtime.

Modify the `applicationContext.xml` file for wiring the `jdbc.properties` file as shown below:

```
<bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
        <list>
            <value>classpath:jdbc.properties</value>
        </list>
    </property>
</bean>
</beans>
```

**NOTE:** For information on the complete configuration snippet of the `applicationContext.xml` file and the `jdbc.properties` file for JDBC Type 2 driver, see “[JDBC Configuration](#)” (page 144).

## Configuring JDBC Type 4 Driver for SQL/MX Database

To configure JDBC Type 4 driver for connecting a Spring application with SQL/MX database, complete the following configurations:

1. “Configurations in the `jdbc.properties` File” (page 65)
2. “Configurations in the `applicationContext.xml` File” (page 65)

---

**NOTE:** Verify and install JDBC Type 4 driver version T1249V11 by following the steps discussed in the “Installing the Spring Framework” (page 36) chapter.

### Configurations in the `jdbc.properties` File

For making necessary JDBC configurations, you need to create the `jdbc.properties` file and complete the following activities:

1. “Defining JDBC Type 4 Driver Class for SQL/MX Database” (page 65)
2. “Defining the Connection URL” (page 65)
3. “Establishing the Connection” (page 65)

#### Defining JDBC Type 4 Driver Class for SQL/MX Database

To set the JDBC Type 4 driver, specify the JDBC Type 4 driver class `com.tandem.t4jdbc.SQLMXDriver` for the SQL/MX database in the `jdbc.properties` file:

```
jdbc.driverClassName=com.tandem.t4jdbc.SQLMXDriver
```

#### Defining the Connection URL

After you have set the JDBC Type 4 driver, specify the location of the SQL/MX server. URLs referring to SQL/MX use the `jdbc:` protocol and have the server host and port number embedded within the URL. To define the connection URL:

- Specify the connection URL as `jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.>` for the SQL/MX database in the `jdbc.properties` file as:

```
jdbc.url=jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.>
```

#### Establishing the Connection

To establish connection, specify the username and password of your NonStop system in the `jdbc.properties` file:

```
jdbc.username=<HP NonStop Username>
jdbc.password=<HP NonStop Password>
```

### Configurations in the `applicationContext.xml` File

For making necessary JDBC configurations, you need to create the `applicationContext.xml` file and complete the following activities:

1. “Defining the Placeholder for JDBC Type 4 Driver Class” (page 65)
2. “Defining the Placeholder for Connection URL” (page 66)
3. “Defining the Placeholder for Establishing the Connection” (page 66)
4. “Wiring of JDBC Properties” (page 66)

#### Defining the Placeholder for JDBC Type 4 Driver Class

Modify the `applicationContext.xml` file to define a placeholder for the JDBC Type 4 driver class name `com.tandem.t4jdbc.SQLMXDriver` from the `jdbc.properties` file as shown below.

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClassName">
```

```
<value>${jdbc.driver}</value>
</property>
```

**NOTE:** It is recommended that you use `com.mchange.v2.c3p0.ComboPooledDataSource` as the datasource class because it provides various parameters for connection pooling. Other available datasource classes are `org.apache.commons.dbcp.BasicDataSource` provided by Apache DBCP and `org.springframework.jdbc.datasource.DriverManagerDataSource` provided by Spring framework.

### Defining the Placeholder for Connection URL

Modify the `applicationContext.xml` file to define a placeholder for the connection URL from the `jdbc.properties` file as shown below.

```
<property name="url">
    <value>${jdbc.url}</value>
</property>
```

### Defining the Placeholder for Establishing the Connection

Modify the `applicationContext.xml` file to define a placeholder for the username and password from the `jdbc.properties` file as shown below.

```
<property name="username">
    <value>${jdbc.user}</value>
</property>
<property name="password">
    <value>${jdbc.password}</value>
</property>
```

### Wiring of JDBC Properties

After the database properties are set, wire the `jdbc.properties` file in the `applicationContext.xml` file so that the actual values are available during runtime. The `PropertyPlaceholderConfigurer` class provided by the Spring framework substitutes the database properties specified in the `applicationContext.xml` file with the values specified in the `jdbc.properties` file during runtime.

Modify the `applicationContext.xml` file for wiring the `jdbc.properties` file as shown below

```
<bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
<property name="locations">
<list>
<value>classpath:jdbc.properties</value>
</list>
</property>
</bean>
</beans>
```

**NOTE:** For information on the complete configuration snippet of the `applicationContext.xml` file and the `jdbc.properties` file, see “[JDBC Configuration](#)” (page 144).

## Configuring Database Transaction Management

Spring framework provides the following ways for configuring transaction management:

- **Declarative Transaction Management:** In this case, the transaction action is defined in an XML configuration file for each method (or class).
- **Programmatic Transaction Management:** In this case, the transaction action is hardcoded in the code. This is similar to the JDBC transaction.

## Declarative Transaction Management

This section describes the following steps to configure transactions in Spring applications using declarative approach:

- “Defining the Transactional Object” (page 67)
  - “Defining the Transactional Advice” (page 68)
  - “Defining the Transactional Execution Point” (page 68)
  - “Defining the Transaction Datasource” (page 69)
  - “Defining the PlatformTransactionManager” (page 69)
  - “Defining the Transaction Rollback (Optional)” (page 70)
- 

**NOTE:** In a typical Spring application, this configuration must be done in the `applicationContext.xml` file.

---

### Defining the Transactional Object

To define a service object to be used as a transactional object, add the following line in the `applicationContext.xml` file:

```
<bean id="<bean name>" class="<package name>.<service name>" />
```

For example:

Consider the following service interface, and its implementation:

```
// the service interface that we want to make transactional  
package a.b.service;
```

```
public interface MyService {  
    MyObject getMyObject(String myObjectName);  
    MyObject getMyObject(String myObjectName, String myObjectDesc);  
    void insertMyObject(MyObject myobject);  
    void updateMyObject(MyObject myobject);  
}
```

```
// an implementation of the above interface  
package a.b.service;
```

```
public class DefaultMyService implements MyService {  
    public MyObject getMyObject(String myObjectName) {  
        throw new UnsupportedOperationException();  
    }  
    public MyObject getMyObject(String myObjectName, String myObjectDesc) {  
        throw new UnsupportedOperationException();  
    }  
    public void insertMyObject(MyObject myobject) {  
        throw new UnsupportedOperationException();  
    }  
    public void updateMyObject(MyObject myobject) {  
        throw new UnsupportedOperationException();  
    }  
}
```

To make the `DefaultMyService` service object as transactional, the configuration in the `applicationContext.xml` file is:

```
<!-- this is the service object that we want to make transactional -->  
<bean id="myService" class="a.b.service.DefaultMyService"/>
```

## Defining the Transactional Advice

The next step is to add the transactional advice to the transactional object. The transaction semantics that must be applied are encapsulated in the `<tx:advice/>` definition.

If you plan to define the transaction advice with the business rule as follows:

"All methods starting with 'get' are to execute in the read-only transaction mode and rest of the methods are to execute with the default transaction mode",

the `<tx:advice/>` definition is configured as:

```
<!-- the transactional advice -->
<tx:advice id=<transactional advice id> transaction-manager=<transaction manager id>>
<!-- the transactional semantics... -->
<tx:attributes>
<!-- all methods starting with 'get' are read-only -->
<tx:method name="get*" read-only="true"/>
<!-- other methods use the default transaction settings -->
<tx:method name="*"/>
</tx:attributes>
</tx:advice>
```

The `transaction-manager` attribute of the `<tx:advice/>` tag is set to the name of the `PlatformTransactionManager` bean that will drive the transactions.

For example:

Assume that the first two methods of the `MyService` interface (`getMyObject(String)` and `getMyObject(String, String)`) must execute in the context of a transaction with read-only semantics, and that the other methods (`insertMyObject(MyObject)` and `updateMyObject(MyObject)`) must execute in the context of a transaction with read-write Semantics. The configuration is:

```
<!-- the transactional advice -->
<tx:advice id="txAdvice" transaction-manager="txManager">
<!-- the transactional semantics... -->
<tx:attributes>
<!-- all methods starting with 'get' are read-only -->
<tx:method name="get*" read-only="true"/>
<!-- other methods use the default transaction settings -->
<tx:method name="*"/>
</tx:attributes>
</tx:advice>
```

## Defining the Transactional Execution Point

To ensure that the transactional advice defined in the above step is executed at the appropriate point in the program, you must define the transactional execution point. This is done using the `<aop:config/>` definition in the `applicationContext.xml` file.

Define a pointcut that matches the execution of any operation defined in the Transactional Service interface. Associate the pointcut with the advisor. The result indicates that at the execution of a Transactional Service operation, the advice you have defined will be run.

```
<!-- ensure that the above transactional advice runs for any execution
of an operation defined by the Transactional service interface -->
<aop:config>
<aop:pointcut id=<transactional operation id> expression=<execution expression>/>
<aop:advisor advice-ref=<transactional advice id> pointcut-ref=<transactional operation id>/>
</aop:config>
```

For example:

To run the transactional advice for any execution of an operation defined in the `MyService` interface, define the transactional execution point as:

```
<aop:config>
<aop:pointcut id="myServiceOperation" expression="execution(* x.y.service.MyService.*(..))"/>
<aop:advisor advice-ref="txAdvice" pointcut-ref="myServiceOperation"/>
```

```
</aop:config>
```

In this case, the `<aop:config/>` definition ensures that the transactional advice defined by the `txAdvice` bean actually executes at the appropriate points in the program. Define a pointcut that matches the execution of any operation defined in the `MyService` interface (`myServiceOperation`). Associate the pointcut with the `txAdvice` using an advisor. The result indicates that at the execution of a `myServiceOperation`, the advice defined by `txAdvice` will be run.

## Defining the Transaction Datasource

This section describes the steps to define the datasource that will be used by the transaction. For more information on datasource configuration, see ["Configuring JDBC Driver for SQL/MX Database" \(page 63\)](#).

Add the following lines in the `applicationContext.xml` file to create the datasource:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName">
            <value>${jdbc.driver}</value>
        </property>
        <property name="url">
            <value>${jdbc.url}</value>
        </property>
        <property name="username">
            <value>${jdbc.user}</value>
        </property>
        <property name="password">
            <value>${jdbc.password}</value>
        </property>
    </bean>

    <bean id="propertyConfigurer"
        class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations">
            <list>
                <value>classpath:jdbc.properties</value>
            </list>
        </property>
    </bean>
</beans>
```

---

**NOTE:** Here `org.apache.commons.dbcp.BasicDataSource` is used as the datasource.

## Defining the PlatformTransactionManager

The `PlatformTransactionManager` is an interface; it is not tied to a lookup strategy such as JNDI, thereby making it abstract while working with JTA.

The `PlatformTransactionManager` interface implementations normally require knowledge of the environment in which they work, such as JDBC, or JTA or Hibernate and so on.

For JDBC, the implementation of `PlatformTransactionManager` is `org.springframework.jdbc.datasource.DataSourceTransactionManager`.

```
<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
</beans>
```

## Defining the Transaction Rollback (Optional)

This section describes the steps to control the rollback of transactions in a simple declarative fashion.

The recommended way to indicate to the Spring framework transaction infrastructure that a transaction work must be rolled back is to throw an Exception from the code that is currently executing in the context of a transaction. The Spring framework transaction infrastructure code catches any unhandled Exception as it bubbles up the call stack, and marks the transaction for rollback.

You can configure the Exception type to mark a transaction for rollback. The following example is a snippet of XML configuration that demonstrates how to configure rollback for a checked, application-specific Exception type.

```
<tx:advice id="txAdvice" transaction-manager="txManager">
<tx:attributes>
<tx:method name="get*" read-only="true" rollback-for="NoProductInStockException"/>
<tx:method name="*"/>
</tx:attributes>
</tx:advice>
```

It is also possible to specify 'no rollback rules', for those times when you do not want a transaction to be marked for rollback when an exception is thrown. In the example below, the Spring framework transaction infrastructure commits the attendant transaction even in the face of an unhandled InstrumentNotFoundException.

```
<tx:advice id="txAdvice">
<tx:attributes>
<tx:method name="updateStock" no-rollback-for="InstrumentNotFoundException"/>
<tx:method name="*"/>
</tx:attributes>
</tx:advice>
```

When the Spring framework transaction infrastructure has caught an exception and is consulting any configured rollback rules to determine whether or not to mark the transaction for rollback, the strongest matching rule wins. Therefore, in the case of the following configuration, any exception other than an InstrumentNotFoundException would result in the attendant transaction being marked for rollback.

```
<tx:advice id="txAdvice">
<tx:attributes>
<tx:method name="*" rollback-for="Throwable" no-rollback-for="InstrumentNotFoundException"/>
</tx:attributes>
</tx:advice>
```

This completes the basic transaction handling in Spring using the declarative approach.

## Programmatic Transaction Management

The Spring framework provides two means of programmatic transaction management:

- “Using the TransactionTemplate” (page 70)
- “Using a PlatformTransactionManager” (page 71)

### Using the TransactionTemplate

The TransactionTemplate handles the transaction lifecycle and possible exceptions so that the calling code does not explicitly handle transactions.

The application code that must execute in a transactional context, and that uses TransactionTemplate explicitly, can be achieved by completing the following steps:

1. Write a TransactionCallback implementation that contains all the codes you need to execute in the context of a transaction.
2. Pass an instance of your custom TransactionCallback to the execute( . . . ) method exposed on the TransactionTemplate.

```
public class SimpleService implements Service {
// single TransactionTemplate shared amongst all methods in this instance
private final TransactionTemplate transactionTemplate;
```

```

// use constructor-injection to supply the PlatformTransactionManager
public SimpleService(PlatformTransactionManager transactionManager) {
    Assert.notNull(transactionManager, "The 'transactionManager' argument must not be null.");
    this.transactionTemplate = new TransactionTemplate(transactionManager);
}
public Object someServiceMethod() {
    return transactionTemplate.execute(new TransactionCallback() {
        // the code in this method executes in a transactional context
        public Object doInTransaction(TransactionStatus status) {
            updateOperation1();
            return resultOfUpdateOperation2();
        }
    });
}

```

If there is no return value, use the convenient `TransactionCallbackWithoutResult` class via an anonymous class:

```

transactionTemplate.execute(new TransactionCallbackWithoutResult() {
    protected void doInTransactionWithoutResult(TransactionStatus status) {
        updateOperation1();
        updateOperation2();
    }
});

```

Code within the callback can roll the transaction back by calling the `setRollbackOnly()` method on the supplied `TransactionStatus` object.

```

transactionTemplate.execute(new TransactionCallbackWithoutResult() {
    protected void doInTransactionWithoutResult(TransactionStatus status) {
        try {
            updateOperation1();
            updateOperation2();
        } catch (SomeBusinessException ex) {
            status.setRollbackOnly();
        }
    }
});

```

## Using a PlatformTransactionManager

You can perform transaction management using

`org.springframework.transaction.PlatformTransactionManager` by completing the following steps:-

1. Pass the implementation of the `PlatformTransactionManager` to the bean.
2. Use the `TransactionDefinition` and `TransactionStatus` objects to initiate transactions, rollback, and commit.

```

DefaultTransactionDefinition def = new DefaultTransactionDefinition();
// explicitly setting the transaction name is something that can only be done programmatically
def.setName("SomeTxName");
def.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);
TransactionStatus status = txManager.getTransaction(def);
try {
    // execute your business logic here
}
catch (MyException ex) {
    txManager.rollback(status);
    throw ex;
}
txManager.commit(status);

```

## Recommendation

1. Programmatic transaction management is usually a good idea only if you have a small number of transactional operations. For example, if you have a web application that requires transactions only for certain update operations, you may not want to set up transactional proxies using Spring or any other technology. In this case, using the `TransactionTemplate`

might be a good approach. Being able to set the transaction name explicitly is also something that can only be done using the programmatic approach to transaction management.

2. If your application has numerous transactional operations, declarative transaction management is usually worthwhile. It keeps transaction management out of business logic, and is not difficult to configure. When using the Spring framework, rather than EJB CMT, the configuration cost of declarative transaction management is greatly reduced.

## Connection Pooling

Connection Pooling is a cache of database connections maintained in the database memory so that the connections can be reused when the database receives future requests for data. Every time a database connection needs to be established, a request is made to pool or any object that holds all the connections. Once that particular database activity is completed, the connection is returned to the pool.

Many vendors provide their own customized Datasource Interfaces. However, we recommend the use of Apache C3P0 as the Datasource Interface to be used with Spring applications. Spring framework distribution contains the necessary files to use Apache C3P0 as the Datasource Interface for connection pooling. In this section, we will discuss the steps to configure connection pooling in your application using Apache C3P0.

### Connection Pooling using Apache C3P0

C3P0 is an easy-to-use library for augmenting traditional (DriverManager-based) JDBC drivers with JNDI-bindable DataSources, including DataSources that implement Connection and Statement Pooling, as described by the jdbc3 spec and jdbc2 std extension.

To configure connection pooling using Apache C3P0, complete the following steps:-

1. Add the c3p0-0.9.1.2.jar in your application CLASSPATH.

---

**NOTE:** c3p0-0.9.1.2.jar is available along with the Spring distribution package at *<Spring Home>/lib/c3p0* directory.

---

2. Add the following basic configurations in applicationContext.xml of your Spring application to configure Apache C3P0 connection pooling:

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
```

---

**NOTE:** C3P0 is tunable and offers a number of configuration parameters.

For more information on C3P0 configuration, see <http://sourceforge.net/projects/c3p0>.

For the complete list of parameters, see [http://www.mchange.com/projects/c3p0/index.html#appendix\\_a](http://www.mchange.com/projects/c3p0/index.html#appendix_a)

---

## Recommendations

Connection Pooling mechanism should be used when your application:

- Cannot handle the overhead of obtaining and releasing connections frequently.
- Requires JTA transactions
- Does not manage the specifics of creating a connection, such as the database name, username, or password.

## Tuning Connection Pool

Performance improvements can be made by correctly tuning the parameters on the connection pool. This section details each of the properties found on the Connection Pooling and how they can be tuned for optimal performance.

### **Minimum Pool Size**

The correct minimum value for the pool can be determined by examining the applications that are using the pool. If it is determined, for example, that at least four connections are needed at any point in time, the minimum number of connections should be set to 4 to ensure that all requests can be fulfilled without connection wait timeout exceptions. At off-peak times, the pool shrinks back to this minimum number of connections. A good rule of thumb is to keep this number as small as possible to avoid holding connections unnecessarily open.

### **Maximum Pool Size**

The best practice is to ensure that only one connection is required on a thread at any time. This avoids possible deadlocks when the pool is at maximum capacity and no connections are left to fulfill a connection request. Therefore, with one connection per thread, the maximum pool size can be set to the maximum number of threads.

When using servlets, this can be determined by looking at the MaxConnections property in the Servlet Engine. If multiple connections are required on a thread, the maximum pool size value can be determined using the following formula:

$$T * (C - 1) + 1$$

Where T is the number of threads, and C is the number of connections.

### **checkoutTimeout**

It is important to know roughly how long the average application will use a connection in the pool. For example, if all applications using a connection pool are known to only hold a connection for an average time of 5 seconds, with a maximum of 10 seconds, it may be useful to have a Timeout value of 10 or 15 seconds.

### **maxIdleTime**

This is a very useful parameter if you are using a resource strapped machine. If you have set your maximum and minimum connection pool sizes properly, you may want to lower the Idle Timeout so that when usage of the pool is low, there are no connections sitting open that are not doing anything. Be very careful in how low to set this parameter because setting it to low will introduce the cost of creating connections to more applications when a transformation from light load to heavy load begins.

## **Module File Caching Configurations**

The Module File Caching (MFC) feature shares the prepared statement plans among the NonStop SQL/MX database connections and helps in reducing resource consumption. MFC is disabled by default; therefore, it must be configured to enable it.

To use the MFC feature, you must perform the following activities:

- “Configuring NonStop SQL/MX DataSource for MFC” (page 73)
- “Modifying the Spring Application” (page 74)

---

**NOTE:** JDBC Type 4 driver offers MFC feature.

---

## **Configuring NonStop SQL/MX DataSource for MFC**

To configure MFC on a NonStop system, complete the following steps:

1. Enter the SQL/MX Connectivity Service (MXCS) subsystem using the SQL/MX Conversational Interface (MXCI) subsystem:

```
mxci>> mode mxcs;
```

2. Configure a datasource for the user specified association service (Mxoas service):

```
mxci>> add ds $mcbs."MFC_datasource";
```

---

**NOTE:** You can configure any number of servers based on your requirement. If you do not specify the number of servers, the default value (maxserver=5, initserver=1, and idleserver=1) is assumed.

---

3. Stop the datasource:

```
mxci>> stop ds MFC_datasource, reason 'test';
```

---

**NOTE:** This step must be performed only if you have a configured datasource in the started status.

---

4. To enable the MFC feature, add the statement\_module\_caching environment variable to the datasource:

```
mxci>> add evar $mcbs."MFC_datasource".statement_module_caching, type set,  
        value 'true';
```

---

**NOTE:** MFC can have only two values: true or false.

---

5. Add the compiled\_module\_location environment variable to the datasource:

```
mxci>> add evar $mfcc."MFC_datasource".compiled_module_location, type set,  
        value 'default';
```

---

**NOTE:** The compiled\_module\_location can be set to a value other than default, such as, /usr/sash. However, you must ensure that the directory is created before specifying it.

---

```
mxci>> add evar $mfcc."MFC_datasource".compiled_module_location, type set,  
        value '/usr/sash';
```

6. Start the datasource:

```
>> start ds $mfcc."MFC_datasource";
```

For more details, see the *JDBC Type 4 Driver 2.0 Programmer's Reference*.

## Modifying the Spring Application

---

**NOTE:** If the default datasource is configured for MFC, do not modify the Spring application.

---

To use the MFC feature, modify the Spring application by using the following steps:

1. Modify the applicationContext.xml file as follows:

- To set serverDataSource to the datasource name on which MFC is configured, add the following under connectionProperties tag.

```
<prop key="serverDataSource">MFC_datasource</prop>
```

# 5 Getting Started with Spring

This chapter explains how to develop a web application using Spring. It describes the steps to build a basic employee management system on the Windows system and deploy it on the NonStop system.

## Prerequisites

Before getting started, make sure that you have the following software installed on the NonStop and Windows system:

### NonStop System

The following software must be installed on the NonStop system:

- NonStop iTP WebServer version T8996H02 or later
- NSJSP version T1222H60 or later
- NonStop SQL/MX version T1050H23 or later
- JDBC Type 2 driver version T1275H50 or later, or JDBC Type 4 driver version T1249V11 or later
- NSJ version T2766H60 or later

### Windows System

The following software must be installed on the Windows system:

- JDK version 1.5 or later
- JDBC Type 4 driver version T1249V11 or later
- Eclipse Galileo IDE version 3.3.1.1 or a later

---

**NOTE:** For more information about installing the software required on NonStop and Windows system, see “[Prerequisites](#)” (page 18).

## Overview of EmplInfo

EmplInfo is a basic employee management system developed using Spring. This application enables you to perform the following tasks:

- Add an employee record and the employee details, such as Employee ID, First Name, Last Name, Age, and email ID.
- Search an employee record using Employee ID.
- Delete an employee record using Employee ID.

In this application:

- The presentation layer is developed using the Spring-MVC framework.
- The business services layer is developed using the Spring framework.
- The persistence services are developed using Spring JDBC API.
- The employee data is stored in the NonStop SQL/MX database.

This section describes the steps required to develop, deploy, and run the EmplInfo application.

- “[Developing EmplInfo on Windows using the Eclipse Galileo IDE](#)” (page 76)
- “[Deploying EmplInfo on NonStop](#)” (page 129)
- “[Running EmplInfo on NonStop](#)” (page 133)

## Developing EmplInfo on Windows using the Eclipse Galileo IDE

### NOTE:

- It is not mandatory for you to use the Eclipse Galileo IDE. You can use an IDE that supports Java.
- The screen captures in this section are based on Eclipse Galileo IDE version 3.3.1.1. The screen captures might look different if you use a different version of Eclipse Galileo.

The following activities are required to develop the EmplInfo application using the Eclipse Galileo IDE:

1. "Developing a Basic EmplInfo Application" (page 77)
2. "Developing and Configuring Views and the Controller" (page 95)
3. "Developing Business Logic and Providing the Web Interface" (page 102)
4. "Integrating the Web-tier of EmplInfo with NonStop SQL/MX Database" (page 106)
5. "Enabling EmplInfo to Delete and Retrieve Employee Details" (page 116)

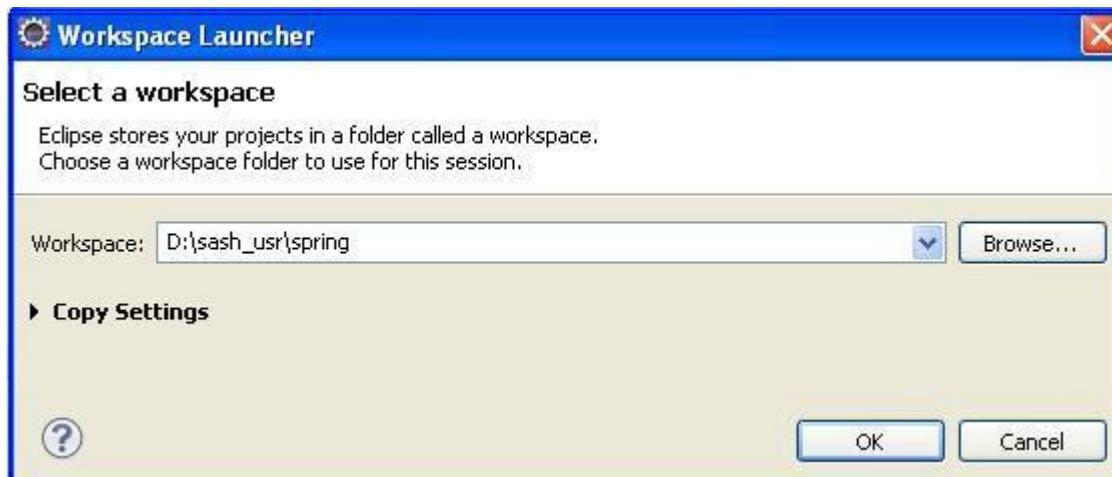
### Creating the Eclipse Workspace

To create a new workspace using the Eclipse Galileo IDE, complete the following steps:

1. To open the Eclipse workspace, double-click the `eclipse.exe` file in *<Eclipse IDE Installation Directory>*.

The Workspace Launcher dialog box appears. By default, the workspace is set to the existing workspace, if already created.

**Figure 8 Workspace Launcher Dialog Box**



2. Click **OK**.

To create a new workspace, click **Browse** and select the folder you want to use as the workspace.

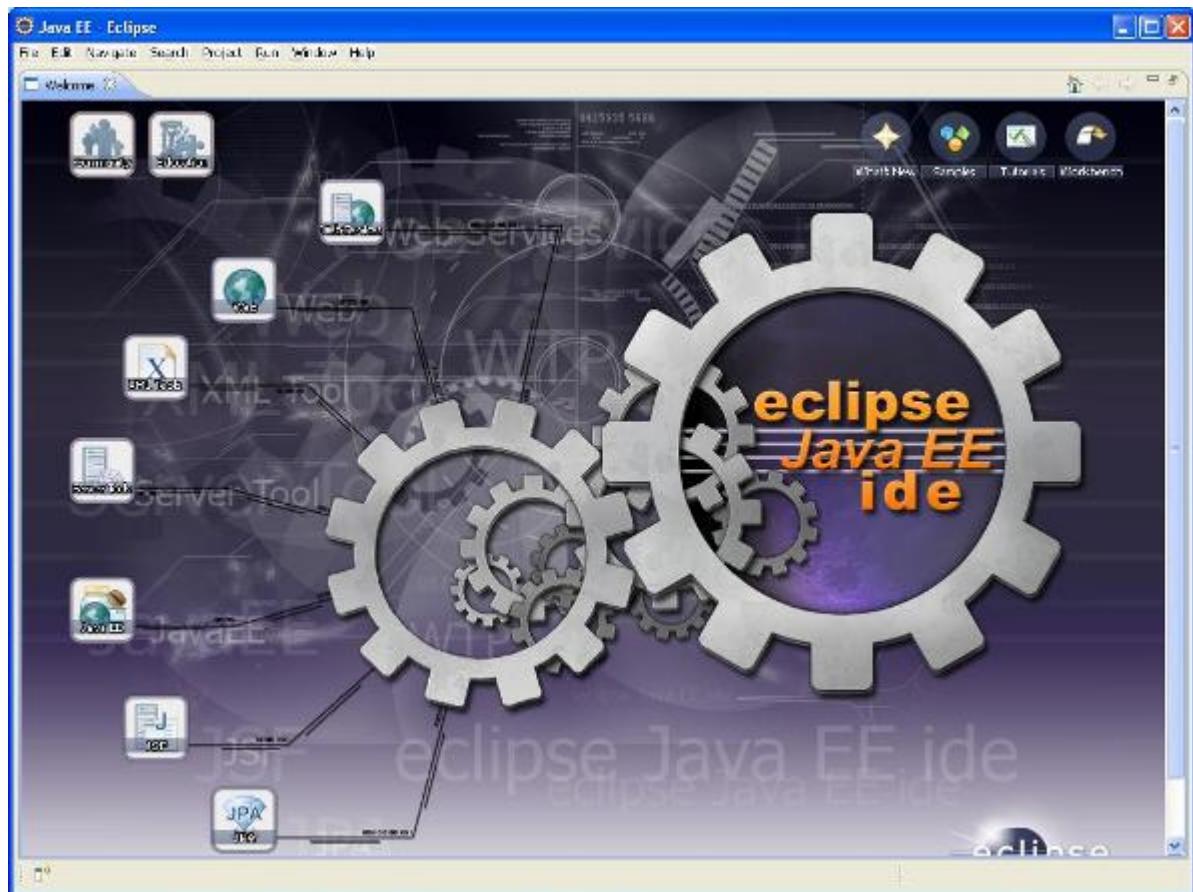
The Eclipse SDK Welcome screen appears.

---

**NOTE:** D:\sash\_usr\spring is the sample workspace used to develop the EmplInfo application.

---

**Figure 9 Eclipse SDK Welcome Screen**



Close the welcome screen. The workspace is created.

---

**NOTE:** The source code for the EmplInfo application is in the SAMPLES.zip file.

---

## Developing a Basic EmplInfo Application

Developing the EmplInfo application using Eclipse Galileo IDE involves the following activities:

1. "Creating a Dynamic Web Project" (page 78)
2. "Creating the index.jsp File" (page 80)
3. "Modifying the web.xml File" (page 84)
4. "Modifying the web.xml File" (page 84)
5. "Creating the EmpInfo-servlet.xml File" (page 86)
6. "Adding Dependency JAR Files in the Project Library Path" (page 89)
7. "Creating the Controller for EmplInfo" (page 91)
8. "Creating the View" (page 94)

## Creating a Dynamic Web Project

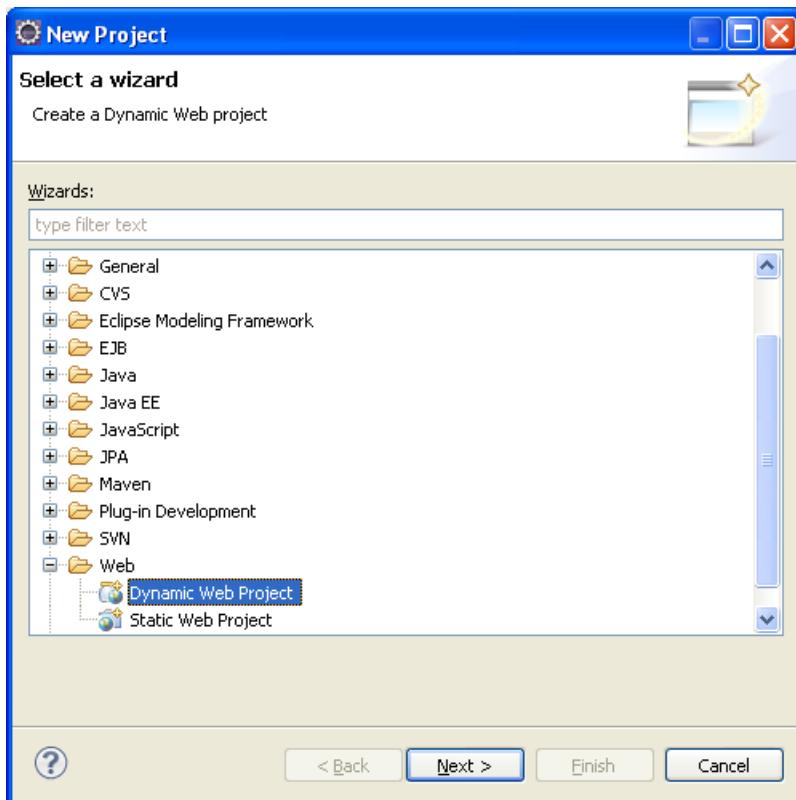
To create a new Eclipse project for the EmpInfo application, complete the following steps:

1. Click **File > New > Project**.

The New Project dialog box appears.

2. From the list of folders, select **Web > Dynamic Web Project** and click **Next**.

**Figure 10 New Project Wizard Dialog Box**

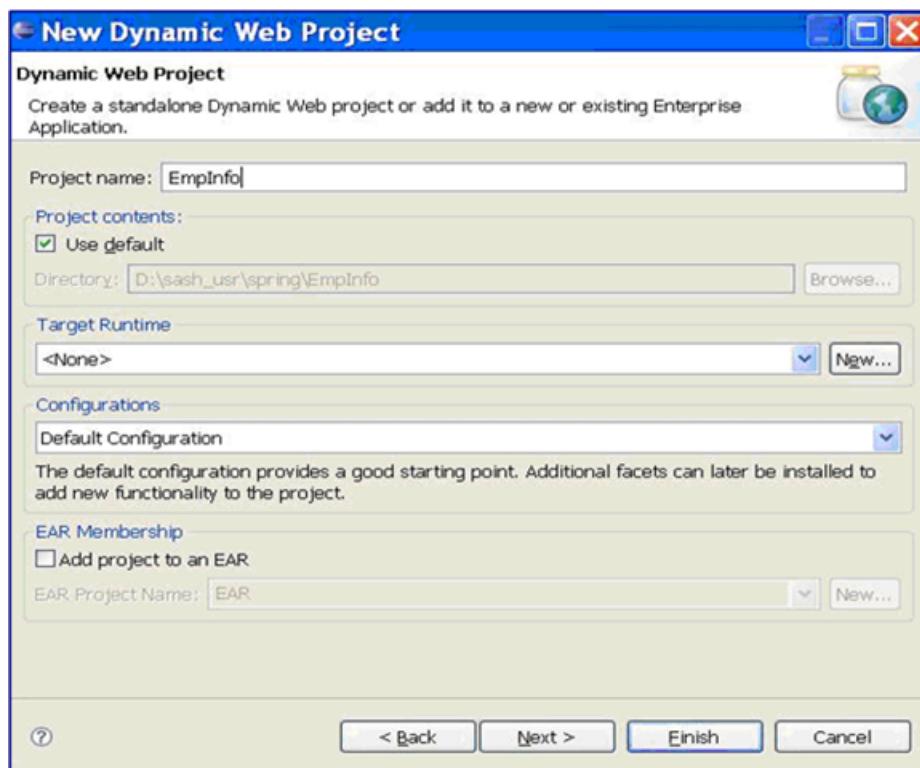


The New Dynamic Web Project dialog box appears.

3. In the **Project name** field, type **EmpInfo** and click **Finish**.

**NOTE:** The other fields in the New Dynamic Web Project dialog box are selected by default.

**Figure 11 New Dynamic Web Project Dialog Box**



When prompted, change the perspective to Java EE and click **Yes** to open the new perspective.

**NOTE:** The perspective should be changed to Java EE because the Dynamic Web Project is associated with the Java EE perspective.

The Project Structure appears.

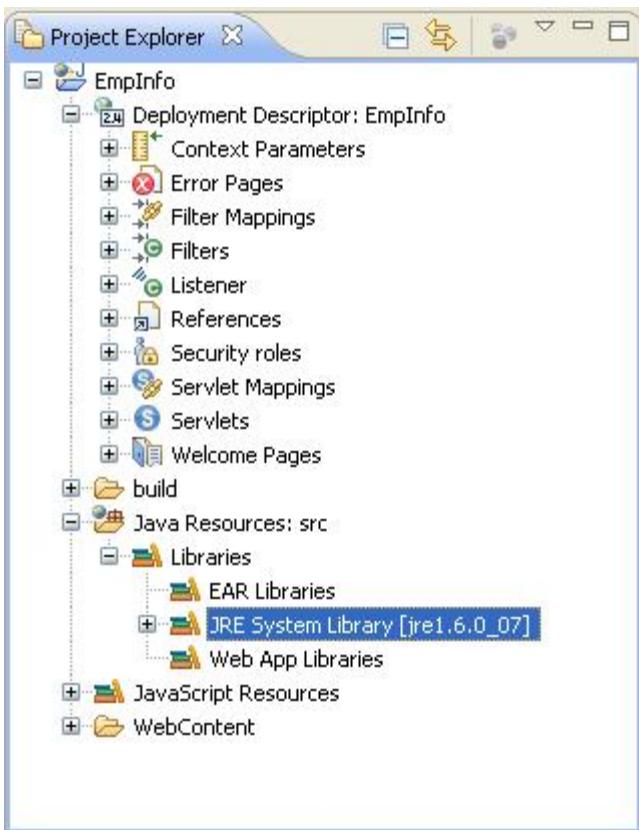
4. In the Project Structure, confirm that the JRE System Library is set to JRE version 1.6 or later, as shown in [Figure 12](#).

If the JRE version is not set to 1.6, right-click **JRE System Library** to select **JRE 1.6**.

**NOTE:** The JRE version used in this example is 1.6.

The EmpInfo project with the project directory structure is created.

**Figure 12 Project Explorer View**



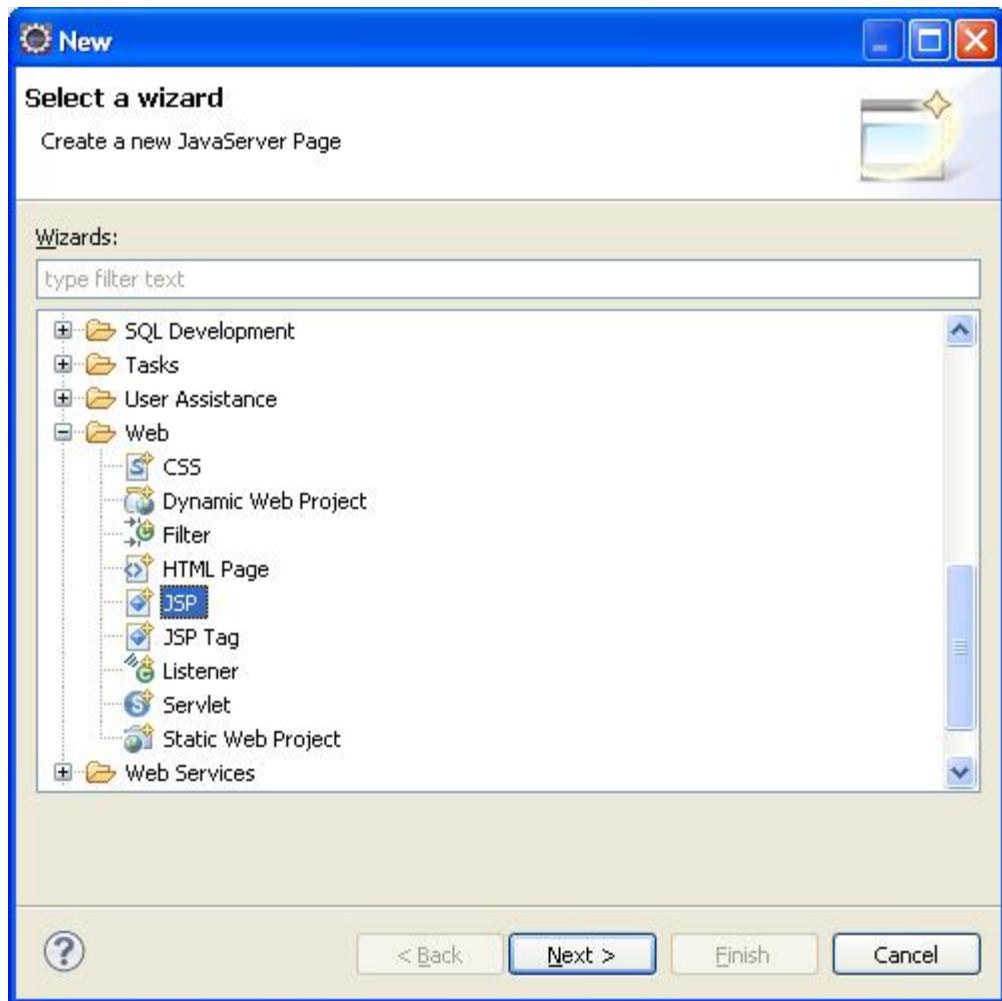
### Creating the index.jsp File

Create the `index.jsp` file in the `EmpInfo/WebContent` directory. This JSP page is the entry point `EmplInfo`.

To create the `index.jsp` file, complete the following steps:

1. On the Project Explorer frame, right-click `EmplInfo` and select **New > Other**.  
The New File dialog box appears.
2. From the list of folders, select **Web > JSP** and click **Next**.

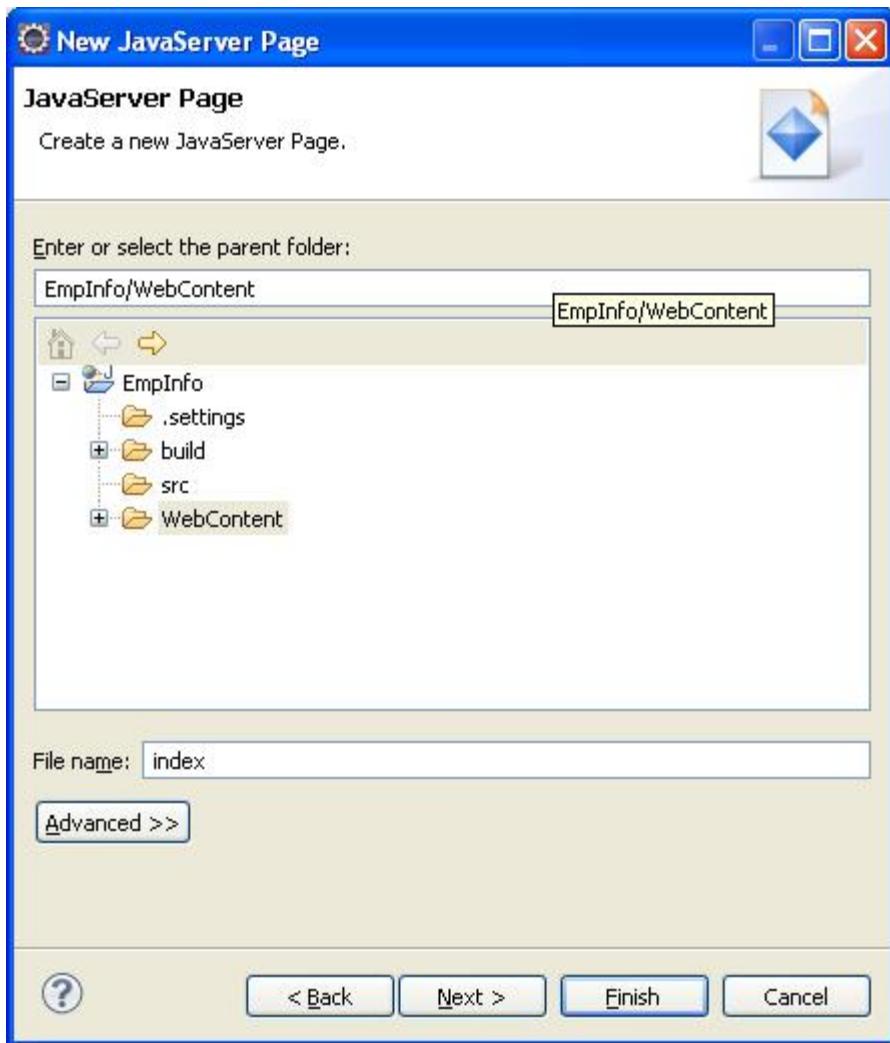
**Figure 13 New File Dialog Box**



The New JavaServer Page dialog box appears.

3. In the **File name** field, type **index** and ensure that the parent folder is set to **EmpInfo/WebContent**. Click **Next**.

**Figure 14 New JavaServer Page Dialog Box**

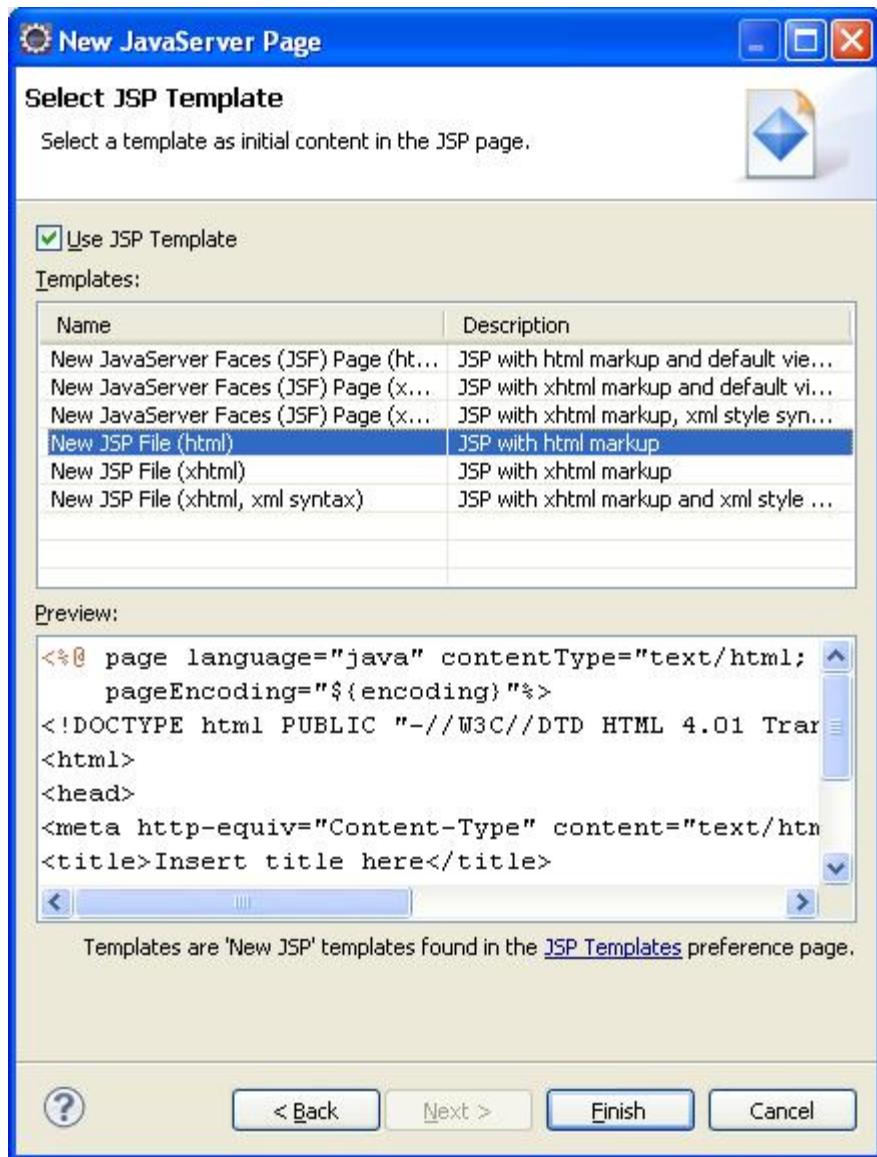


The New JavaServer Page: JSP Template dialog box appears.

4. From the Select JSP Template dialog box, select **New JSP File (html)** and click **Finish**.

**NOTE:** The **Use JSP Template** check box is selected by default.

**Figure 15 New JavaServer Page: JSP Template Dialog Box**



The template for the index.jsp file is generated.

### Modifying the index.jsp File

To modify the index.jsp file, complete the following step:

- Modify the EmpInfo/WebContent/index.jsp file to add a header item.  
Add **<Spring Getting Started with EmpInfo Application>** as the title tag, **<Welcome to the EmpInfo Application>** as the header item and **<This is EmpInfo Test Page>** within the body tag of the EmpInfo/WebContent/index.jsp file as shown below:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Getting Started with EmpInfo Application</title>
</head>
<body>
<h2 align="center">Welcome to the EmpInfo Application</h2>
<br>
<p>This is EmpInfo Test Page.</p>
</body>
</html>

```

At this point, you can either deploy and verify the EmpInfo application you have developed so far on the NonStop system, or you can proceed with the steps for modifying the web.xml file.

---

**NOTE:** The code of the EmpInfo application developed so far is located in <My SASH Home>\spring\getting-started\EmpInfo-InParts\Part-1

---

To verify the EmpInfo application, complete the following steps:

1. Deploy the EmpInfo application using the steps described in “[Deploying EmpInfo on NonStop](#)” (page 129).
2. Verify the EmpInfo application by accessing the following URL:  
[http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/EmpInfo](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/EmpInfo)  
The Welcome Screen of the EmpInfo application appears.

**Figure 16 EmpInfo Welcome Screen**



### Modifying the web.xml File

To modify the web.xml file, you must set the dispatcher servlet and its mapping using the following steps:

1. Double-click the EmpInfo/WebContent/WEB-INF/web.xml file in the Project Explorer frame to open it.

---

**NOTE:** By default, XML files open in the XML Editor. The XML Editor has two views: Design and Source view. Select the Source view.

---

2. Open the EmpInfo/WebContent/WEB-INF/web.xml file.

The template for the default web.xml file generated during project creation is:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>
    EmpInfo</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

3. Set the index.jsp file as <welcome-file> by deleting other files from <welcome-file-list>.
4. Set the class for the DispatcherServlet of the EmpInfo application in the web.xml file as shown below:

```
<servlet>
  <servlet-name>EmpInfo</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

5. Specify the URL pattern as \*.htm in the <servlet-mapping> tag as shown below:

```
<servlet-mapping>
  <servlet-name>EmpInfo</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

---

**NOTE:** This servlet definition maps to the application patterns. Any URL with the .htm extension is routed to the EmpInfo servlet (the DispatcherServlet).

---

After modification, the web.xml file appears as:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>
    EmpInfo</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>EmpInfo</servlet-name>
```

```

<servlet-class>
    org.springframework.web.servlet.DispatcherServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>EmpInfo</servlet-name>
    <url-pattern>*.htm</url-pattern>
</servlet-mapping>
</web-app>

```

### Creating the EmpInfo-servlet.xml File

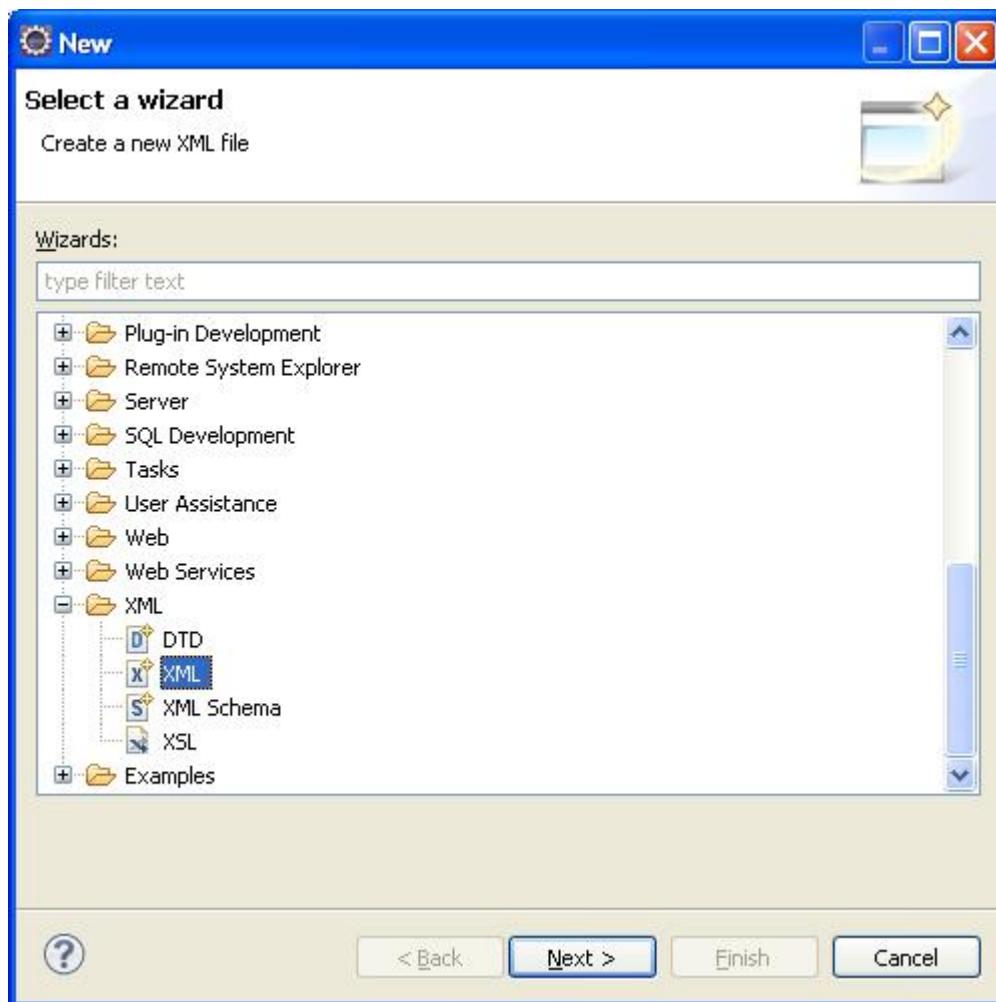
Create a WebApplicationContext file, which contains references to all the web-related components. The name of this file is determined by the value of the `<servlet-name>` element in the `web.xml` file, with `-servlet` appended to it (hence, `EmpInfo-servlet.xml`). This is the standard naming convention used with the Spring Web MVC framework.

The `EmpInfo-servlet.xml` file contains the bean definitions (plain Java objects) used by the `DispatcherServlet`.

To create the `EmpInfo-servlet.xml` file in the `EmpInfo/WebContent/WEB-INF` directory, complete the following steps:

1. On the Project Explorer frame, right-click **EmpInfo** and select **New > Other**.  
The New File Wizard dialog box appears.
2. From the list of folders, select **XML > XML** and click **Next**.

**Figure 17** New File Wizard Dialog Box



The New XML File dialog box appears.

3. In the **File name** field, type **EmpInfo-servlet.xml** and ensure that the parent folder is set to **EmpInfo/WebContent/WEB-INF**. Click **Next**.

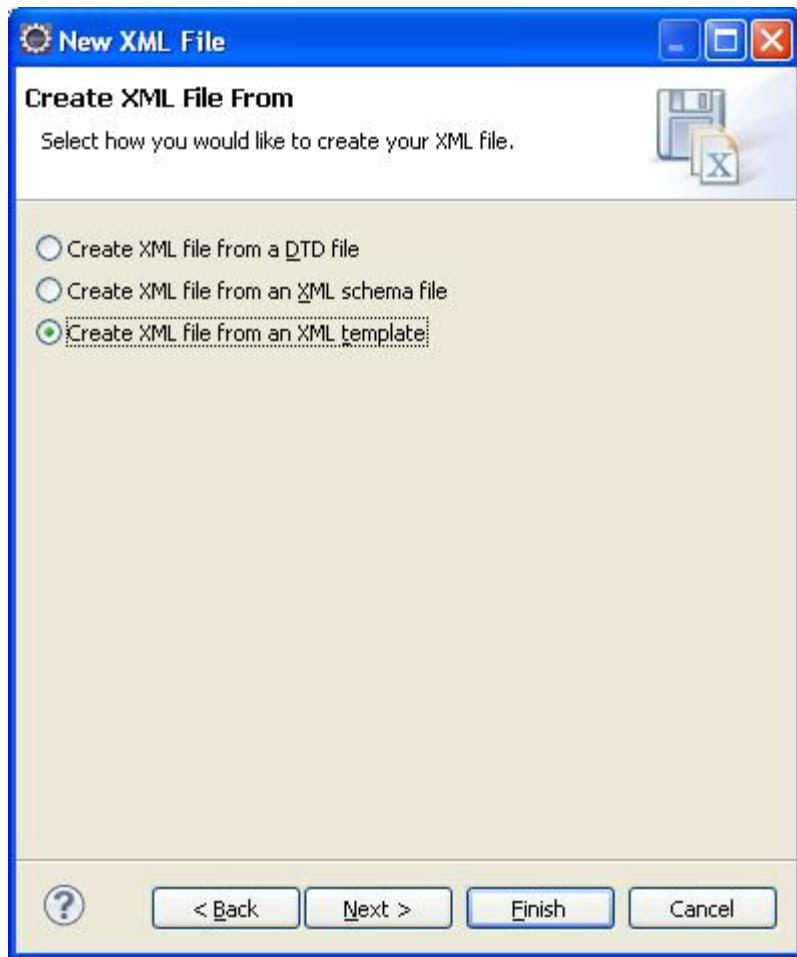
**Figure 18 New XML File Dialog Box**



The New XML File: Options dialog box with XML template options appears.

4. Select **Create XML file from an XML template** and click **Finish**.

**Figure 19 New XML File: Options Dialog Box**



The `EmpInfo-servlet.xml` file is created in the `EmpInfo/WebContent/WEB-INF` directory.

5. Modify the `EmpInfo-servlet.xml` file by adding a bean entry named `/insert.htm` and `com.hp.empinfo.web.EmployeeController` as its class.

**NOTE:** The `com.hp.empinfo.web.EmployeeController` class provides controller for the `EmplInfo` application to service a request according to its corresponding URL mapping in `/insert.htm`. The Spring Web MVC framework uses an interface class called `HandlerMapping` to define the mapping between a request URL and the handler object, which handles the request. The controller ensures that the URL mapping the `EmplInfo` application is `com.hp.empinfo.web.EmployeeController`, instead of `DispatcherServlet`.

After modification, the `EmpInfo-servlet.xml` file appears as:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-2.5.xsd
                           http://www.springframework.org/schema/jee"
```

```

http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

<bean name="/insert.htm" class="com.hp.empinfo.web.EmployeeController">
</bean>
</beans>

```

### Adding Dependency JAR Files in the Project Library Path

Because the EmplInfo application uses the Spring framework, it requires the following dependency JAR files.

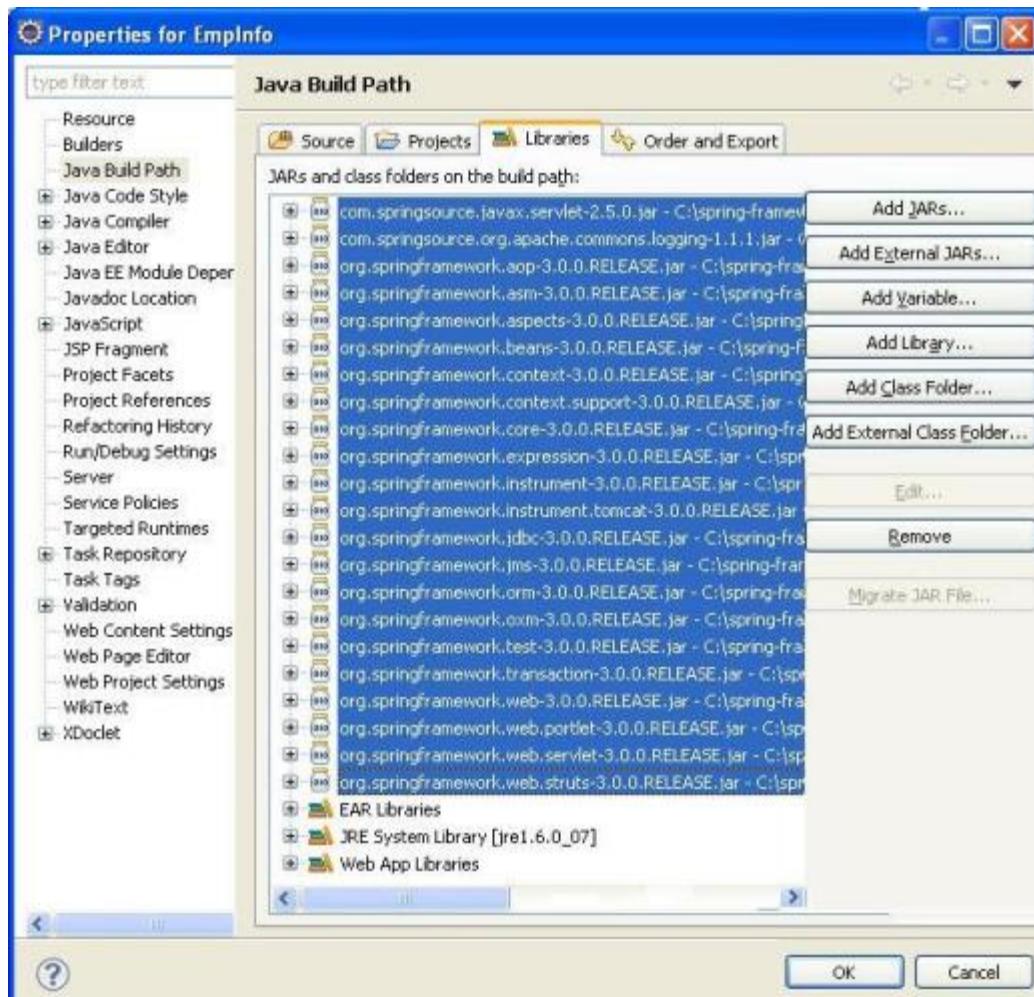
**Table 1 Spring Dependency JAR Files**

Dependency JAR Files	Source Location
com.springsource.org.apache.commons.logging-1.1.1.jar	< <i>Spring Dependency Home</i> >/org.apache.commons/com.springsource.org.apache.commons.logging/1.1.1
com.springsource.javax.servlet-2.5.0.jar	< <i>Spring Dependency Home</i> >/javax.servlet/com.springsource.javax.servlet/2.5.0
*.jar	< <i>Spring Home</i> >/dist

To add the dependency JAR files in the EmplInfo project library path, complete the following steps:

1. On the Project Explorer frame, right-click **EmplInfo** and select **Properties**.  
The Java Build Path dialog box appears.
2. From the type filter text, select **Java Build Path**.
3. Click the **Libraries** tab.
4. Click **Add External JARs** to add each of the above-mentioned dependency JAR files and then, browse to specify the location of each of the dependency JAR files.

**Figure 20 Java Build Path Dialog Box**



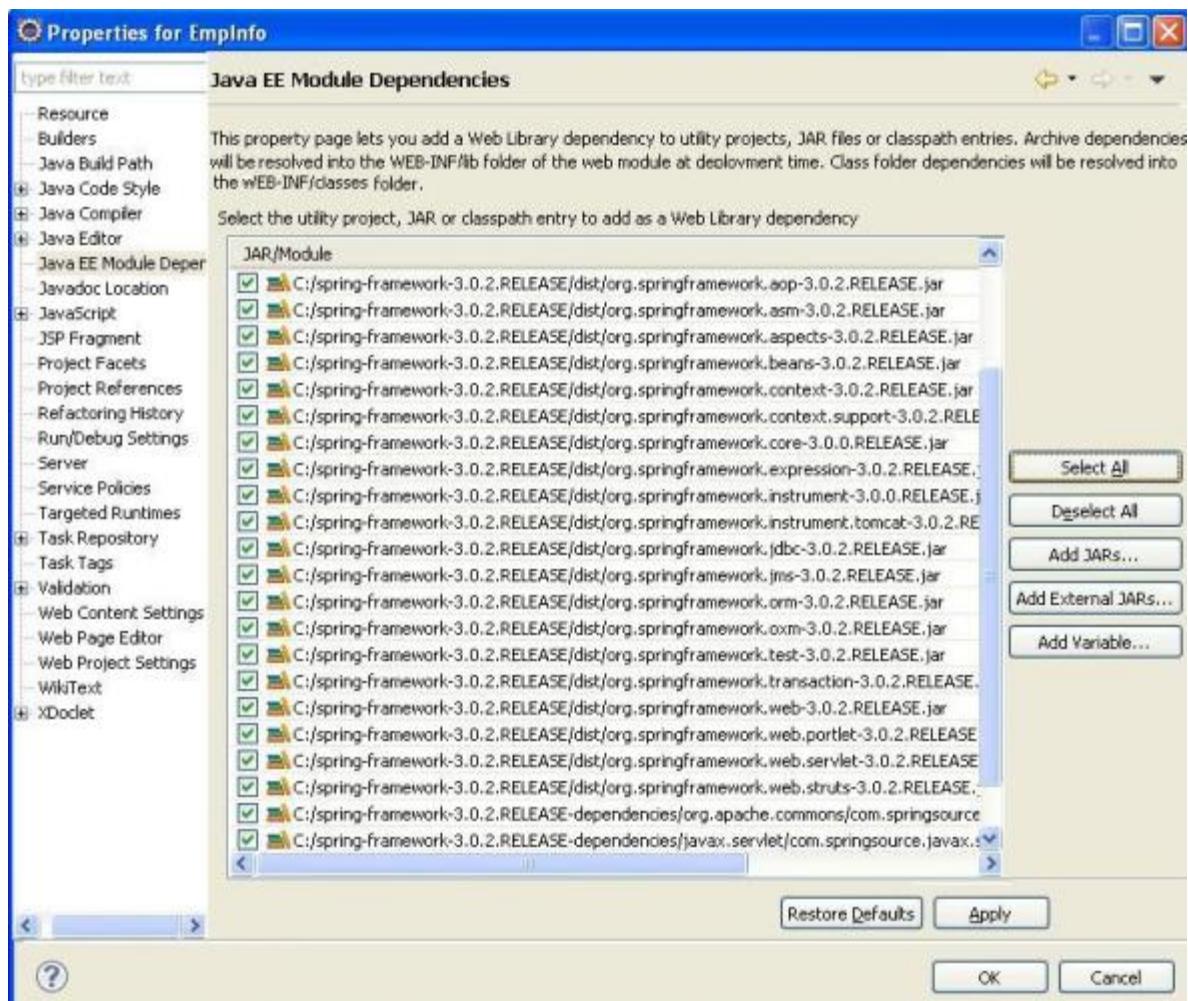
5. Click **OK**.

The dependency JAR files are added in the EmplInfo project library path.

To resolve the J2EE module dependency on these JAR files, complete the following steps on the Eclipse Galileo IDE:

1. On the Project Explorer frame, right-click **EmplInfo** and select **Properties**.  
The Project Properties dialog box appears.
2. From the type filter text, select **J2EE Module dependencies**.
3. Select all the dependency JAR files.

**Figure 21 J2EE Module Dependencies Dialog Box**



4. Click **OK**.

The J2EE module dependency on the JAR files is resolved.

### Creating the Controller for EmplInfo

In the Spring Web MVC framework, the controller handles the request and returns a **ModelAndView**.

Create the Controller to handle the navigation logic and to interact with the service tier of the business logic.

To create the Controller, complete the following steps:

1. “Create the Package for the Controller Class” (page 91)
2. “Create the Controller Class” (page 92)

### Create the Package for the Controller Class

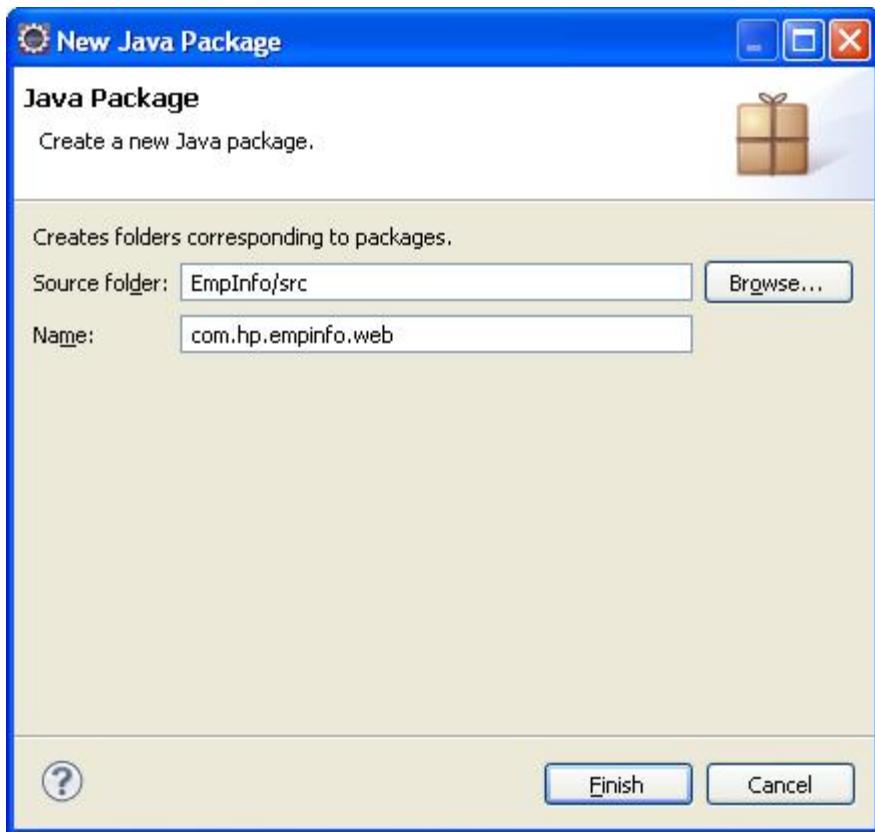
To create a package `com.hp.empinfo.webpackage`, complete the following steps:

1. On the Project Explorer frame, right-click **EmplInfo** and select **New >Package**.

The New Java Package dialog box appears.

2. In the **Name** field, type `com.hp.empinfo.web` and ensure that the **Source folder** is set to `EmplInfo/src`.

**Figure 22 New Java Package Dialog Box**



3. Click **Finish**.

The `com.hp.empinfo.web` package is created.

#### Create the Controller Class

To create the Controller class, complete the following steps:

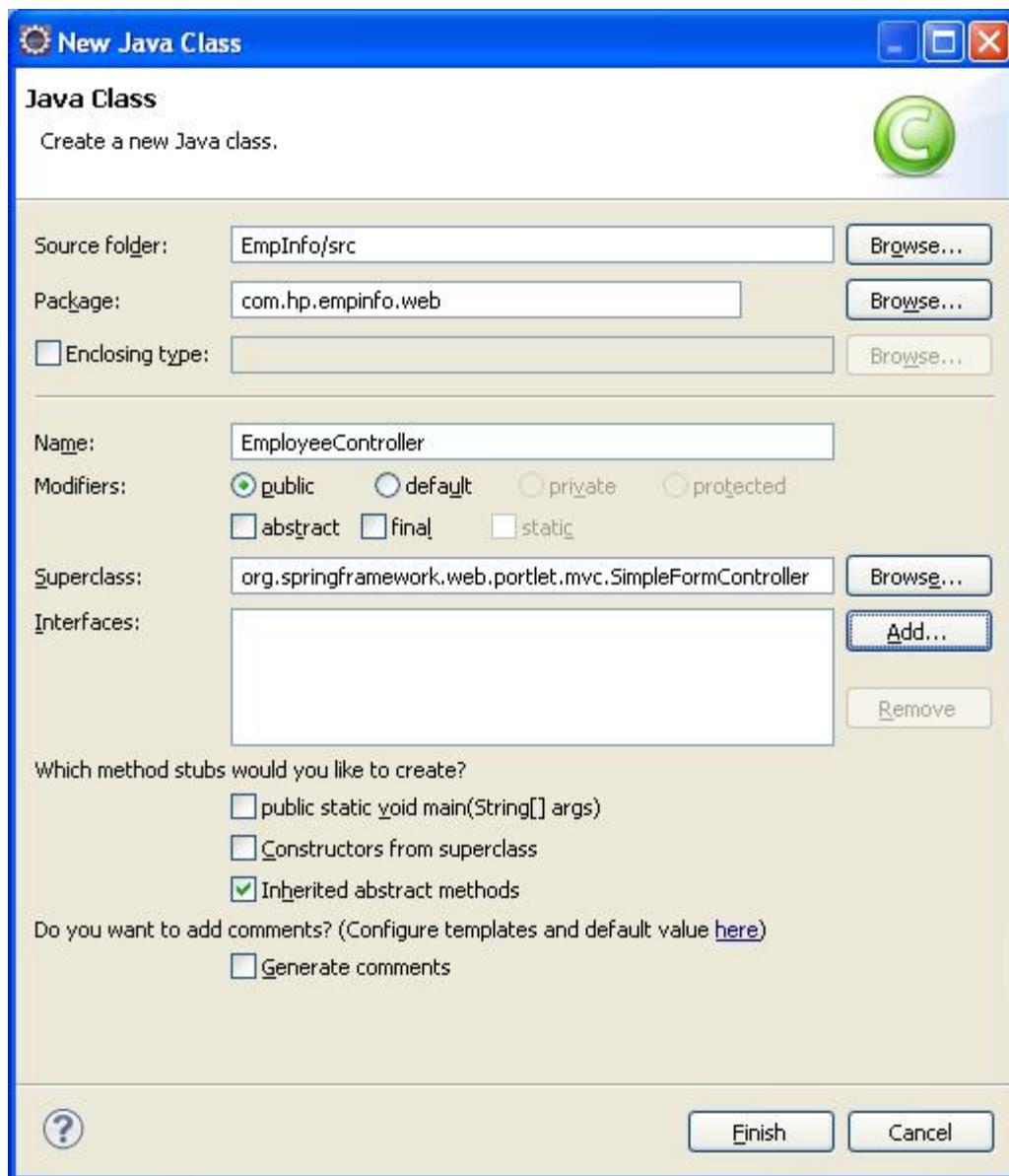
1. On the Project Explorer frame, right-click **EmplInfo** and select **New >Class**.
2. In the **Package** field, type `com.hp.empinfo.web`. In the **Name** field, type `EmployeeController`.
3. In the **Superclass** field, type `org.springframework.web.servlet.mvc.SimpleFormController` and click **Finish**.

---

**NOTE:** The Controller class `EmployeeController` extends the `SimpleFormController` class in the `org.springframework.web.servlet.mvc` package provided by the Spring framework.

---

**Figure 23 New Java Class Dialog Box**



The EmployeeController.java class file is created.

4. Modify the EmployeeController.java class file to create a basic controller.

After modification, the EmployeeController.java file appears as:

```
package com.hp.empinfo.web;
import org.springframework.web.servlet.mvc.SimpleFormController;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import java.io.IOException;

public class EmployeeController extends SimpleFormController{

    protected final Log logger = LogFactory.getLog(getClass());
    public ModelAndView handleRequest(HttpServletRequest request,
    HttpServletResponse response)
        throws ServletException, IOException {
```

```

        logger.info("Returning hello view");
        return new ModelAndView("insert.jsp");
    }
}

```

Your basic controller implementation is ready. This controller is expanded further to add more functionality to the EmplInfo application.

## [Creating the View](#)

After completing the controller implementation for EmplInfo, complete the following steps to create the first view of the EmplInfo application:

1. Create a new JSP page `insert.jsp` in `EmpInfo/WebContent` by completing the steps explained in [“Creating the index.jsp File” \(page 80\)](#).
2. Add `<Spring Getting Started with EmpInfo Application>` as the title tag, `<Welcome to the Enter Employee Details Page>` as the header item and `<Enter Employee Details>` within the body tag of `EmpInfo/WebContent/insert.jsp`, as shown below:

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Getting Started with EmpInfo Application</title>
</head>
<body>
    <h2 align="center"> Welcome to the EmpInfo Application </h2>
    <br>
    <p><B><font color = "003399">Enter Employee Details</font></B></p>
</body>
</html>

```

At this point, you can either deploy and verify the EmplInfo application you have developed so far on the NonStop system, or you can proceed to the [“Developing and Configuring Views and the Controller” \(page 95\)](#) section.

---

**NOTE:** The code of the EmplInfo application developed so far is located in `<My SASH Home>\spring\getting-started\EmpInfo-InParts\Part-2`

---

To verify the EmplInfo application, complete the following steps:

1. Deploy the EmplInfo application using the steps described in [“Deploying EmplInfo on NonStop” \(page 129\)](#).
2. Verify the EmplInfo application by accessing the following URL:  
`http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/EmplInfo/insert.htm`

The Welcome screen of the EmplInfo application appears.

**Figure 24 EmplInfo: Employee Details Screen**



## Developing and Configuring Views and the Controller

This section describes the process to develop and configure Views and the Controller.

Views are provided by JSP pages and the controller has the intelligence to recognize the business logic based on the input provided in the JSP pages.

This section describes the following activities:

1. "Configuring the JSP Standard Tag Library" (page 95)
2. "Adding JSTL and JSP Related Dependency JARs" (page 98)
3. "Enhancing the Controller" (page 98)
4. "Decoupling View from the Controller" (page 100)

### Configuring the JSP Standard Tag Library

Configuring the JSP Standard Tag Library (JSTL) involves the following activities:

1. "Creating a folder to store JSPs" (page 95)
2. "Creating the `include.jsp` File" (page 96)
3. "Modifying the `index.jsp` File" (page 97)
4. "Modifying the `insert.jsp` File" (page 97)

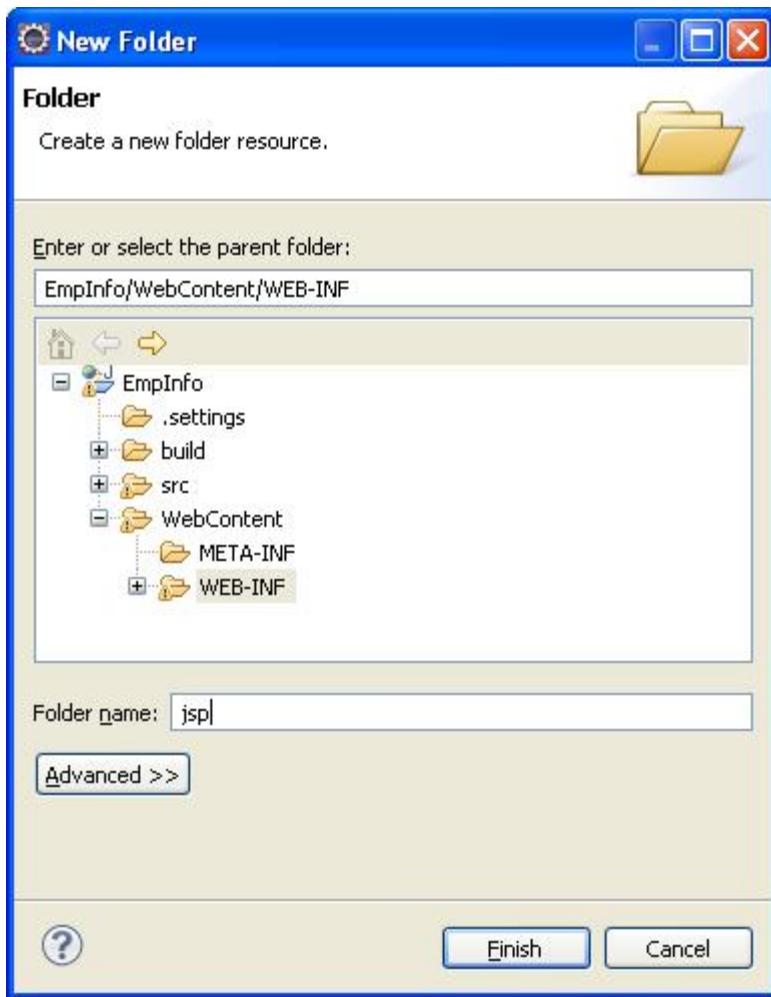
To configure the JSTL, complete the following steps:

#### Creating a folder to store JSPs

To create a new folder named JSP, complete the following steps:

1. On the Project Explorer frame, right-click **EmplInfo** and select **New > Folder**.  
A New Folder dialog box appears.
2. In the **Folder name** field, type `jsp` and select **EmplInfo/WebContent/WEB-INF** as the parent folder.

**Figure 25 New Folder Dialog Box**



3. Click **Finish**.

The **jsp** folder is created.

#### **[Creating the include.jsp File](#)**

To create the `include.jsp` file, complete the following steps:

1. Create the `include.jsp` file in the `EmpInfo/WebContent/WEB-INF/jsp` directory as described in "[Creating the index.jsp File](#)" (page 80).

2. Add the tag extensions (taglibs) to include the .jsp file so that the include.jsp file appears as:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ page session="false"%>

<%@ taglib prefix="c"
uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt"
uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="form"
uri="http://www.springframework.org/tags/form" %>
```

---

**NOTE:** The created header file must be included in every JSP page. This ensures that the same definitions are included in all JSPs. You must add all the JSPs in a subdirectory named jsp in the WEB-INF directory. This allows views to be accessible via the controller because these pages cannot be accessed via a URL.

---

### Modifying the index.jsp File

To modify the index.jsp file, complete the following steps:

1. Include the include.jsp file:

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
```

2. Provide a link to the insert.jsp page as follows:

```
<h4><a href=<c:url value="insert.htm"/>>Insert Employee</a></h4>
```

3. Delete the following line from the <body> tag.

```
<p>This is EmpInfo Test Page.</p>
```

After modification, the index.jsp file appears as:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Getting Started with EmpInfo Application</title>
</head>
<body>
    <h2 align="center"> Welcome to the EmpInfo Application </h2>
    <br>
    <h4><a href=<c:url value="insert.htm"/>>Insert Employee</a></h4>
    <br>
</body>
</html>
```

### Modifying the insert.jsp File

To modify the insert.jsp file, complete the following steps:

1. Move the insert.jsp file to the EmpInfo/WebContent/WEB-INF/jsp directory by dragging and dropping the file on the Eclipse window.

2. Include the include.jsp file in insert.jsp:

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
```

3. Add the <c:out> tag to display the current date and time retrieved from the model passed to the view, which will be rendered using the JSTL.

For example:

```
<h3 align="center"><c:out value="${now}" /></h3>
```

After modification, the insert.jsp file appears as:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Getting Started with EmpInfo Application</title>
</head>
<body>
    <h2 align="center"> We
    lcome to the EmpInfo Application </h2>
    <br>
    <h3 align="center"><c:out value="${now}" /></h3>
    <br>
    <p><B><font color = "003399">Enter Employee Details</font></B></p>
</body>
</html>
```

## Adding JSTL and JSP Related Dependency JARs

Add the following JSTL and JSP dependency JAR files to the EmpInfo project in Eclipse:

**Table 2 JSTL and JSP Dependency JAR Files**

Dependency JAR Files	Source Location
com.springsource.javax.servlet.jsp.jstl-1.1.2.jar	< <i>Spring Dependency Home</i> >/javax.servlet/com.springsource.javax.servlet.jsp.jstl/1.1.2
com.springsource.org.apache.taglibs.standard-1.1.2.jar	< <i>Spring Dependency Home</i> >/org.apache.taglibs/com.springsource.org.apache.taglibs.standard/1.1.2

To add the dependency JAR files in the project library path and to resolve the J2EE module dependency on these JARs, follow the instructions described in “[Adding Dependency JAR Files in the Project Library Path](#)” (page 89).

## Enhancing the Controller

To add more functionality to the controller class, complete the following steps:

1. Update the EmployeeController.java by setting the resource reference of the view to its new location EmpInfo/WebContent/WEB-INF/jsp/insert.jsp. Set the key/value pair for the current date and time value in the model with the key identifier: “now” and the string value: ‘now’ as shown below:

Before modification:

```
logger.info("Returning hello view");
return new ModelAndView("insert.jsp");
```

After modification:

```
String now = (new Date()).toString();
logger.info("Returning view with " + now);
return new ModelAndView("WEB-INF/jsp/insert.jsp", "now", now);
```

2. Add the following import statement:

```
import java.util.Date;
```

After modification, the controller file EmployeeController.java appears as:

```
package com.hp.empinfo.web;
import org.springframework.web.servlet.mvc.SimpleFormController;
```

```

import org.springframework.web.servlet.ModelAndView;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import java.io.IOException;
import java.util.Date;
public class EmployeeController extends SimpleFormController {
protected final Log logger = LogFactory.getLog(getClass());
public ModelAndView handleRequest(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
String now = (new Date()).toString();
logger.info("Returning view with " + now);
return new ModelAndView("WEB-INF/jsp/insert.jsp", "now", now);
}
}

```

At this point, you can either deploy and verify the EmplInfo application you have developed so far on the NonStop system, or you can continue with the steps explained in “[Decoupling View from the Controller](#)” (page 100).

---

**NOTE:** The code of the EmplInfo application developed so far is located in <My SASH Home>\spring\getting-started\EmpInfo-InParts\Part-3

---

To verify the EmplInfo application developed so far, complete the following steps:

1. Deploy the EmplInfo application using the steps specified in “[Deploying the EmplInfo WAR File in NSJSP on NonStop](#)” (page 131).
2. Verify the EmplInfo application by accessing the URL:

[http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/EmplInfo](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/EmplInfo)

The EmplInfo: Insert Employee screen appears.

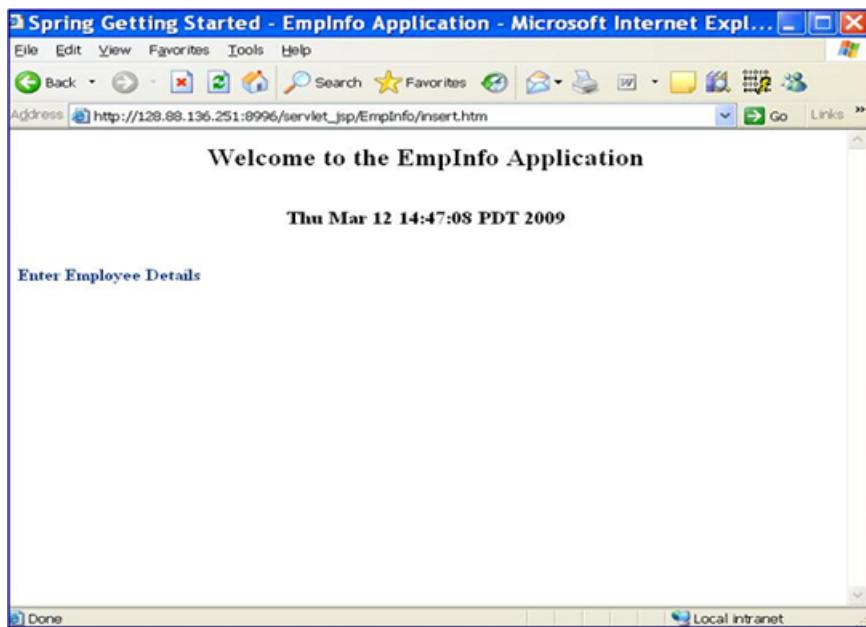
**Figure 26 EmplInfo: Insert Employee Screen**



3. Click **Insert Employee**.

The EmplInfo: Employee Details screen appears.

**Figure 27 EmplInfo: Employee Details Screen**



### Decoupling View from the Controller

So far, the controller is specifying full path of the view, thereby creating unnecessary dependency between the controller and the view. Ideally, the view must be mapped using a logical name, so that the view can be switched without changing the controller.

To decouple the view from the controller, complete the following steps:

1. Modify the `EmplInfo-servlet.xml` file in the `EmplInfo/WebContent/WEB-INF` directory to add the following bean to declare a new `ViewResolver` entry.

```
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="viewClass"
    value="org.springframework.web.servlet.view.JstlView"></property>
<property name="prefix"
    value="/WEB-INF/jsp/"></property>
<property name="suffix" value=".jsp"></property>
</bean>
```

After modification, the `EmplInfo-servlet.xml` file appears as:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<bean name="/insert.htm"
    class="com.hp.empinfo.web.EmployeeController"/>
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="viewClass"
    value="org.springframework.web.servlet.view.JstlView"></property>
<property name="prefix"
    value="/WEB-INF/jsp/"></property>
<property name="suffix"
    value=".jsp"></property>
</bean>
</beans>
```

2. Modify the `EmployeeController` class `EmplInfo/com/hp/empinfo/web/EmployeeController.java` to decouple the controller from view using `ViewResolver`, as shown below:

Before modification:

```
return new ModelAndView("WEB-INF/jsp/insert.jsp", "now", now);
```

After modification:

```
return new ModelAndView("insert", "now", now);
```

After modification, the EmployeeController.java file appears as:

```
package com.hp.empinfo.web;

import org.springframework.web.servlet.mvc.SimpleFormController;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import java.io.IOException;
import java.util.Date;

public class EmployeeController extends SimpleFormController {

protected final Log logger = LogFactory.getLog(getClass());
public ModelAndView handleRequest(HttpServletRequest request,
    HttpServletResponse response)
throws ServletException, IOException {
String now = (new Date()).toString();
logger.info("Returning hello view with " + now);
return new ModelAndView("insert", "now", now);
}
}
```

At this point, you can either deploy and verify the EmplInfo application you have developed so far on the NonStop system, or you can continue with the steps described in “[Developing Business Logic and Providing the Web Interface](#)” (page 102).

---

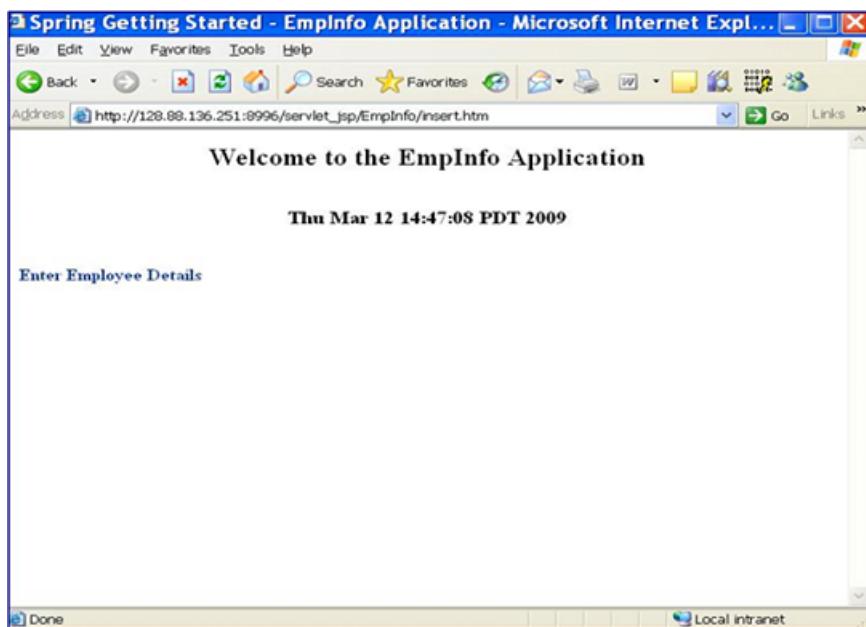
**NOTE:** The code of the EmplInfo application developed so far is located in <My SASH Home>\spring\getting-started\EmpInfo-InParts\Part-4

To verify the EmplInfo application developed so far, complete the following steps:

1. Deploy the EmplInfo application using the steps described in “[Deploying the EmplInfo WAR File in NSJSP on NonStop](#)” (page 131).
2. Verify the EmplInfo application by accessing the following URL:  
[http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/EmplInfo](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/EmplInfo)
3. Click **Insert Employee**.

The EmplInfo: Employee Details screen appears.

**Figure 28 EmplInfo: Employee Details Screen**



## Developing Business Logic and Providing the Web Interface

This section describes the following tasks:

1. "Creating a Simple POJO" (page 102)
2. "Creating Data Access Object Implementation for JDBC" (page 103)
3. "Creating and Modifying Supporting Views to Display Business Data" (page 103)
4. "Adding Dependency JAR Files" (page 105)

### Creating a Simple POJO

To create a simple POJO, complete the following steps:

1. Create a com.hp.empinfo.domain package and an Employee class in the com.hp.empinfo.domain package, as described in "[Creating the Controller for EmplInfo \(page 91\)](#)".
2. Modify the Employee.java class file to add some properties and their getter and setter methods.

After modification, the Employee.java file must appear as:

```
package com.hp.empinfo.domain;
```

```
public class Employee implements java.io.Serializable {  
  
    private int empid;;  
    private String firstname;  
    private String lastname;  
    private int age;  
    private String email;  
  
    public int getEmpid() {  
        return empid;  
    }  
    public void setEmpid(int empid) {  
        this.empid = empid;  
    }  
    public String getFirstname() {  
        return firstname;
```

```

    }
    public void setFirstname (String firstname) {
        this. firstname= firstname;
    }
    public String getLastname () {
        return lastname;
    }
    public void setLastname (String lastname) {
        this. lastname= lastname;
    }
    public int getAge () {
        return age;
    }
    public void setAge (int age) {
        this. age= age;
    }
    public String getEmail () {
        return email;
    }
    public void setEmail (String email) {
        this. email= email;
    }
}

```

## Creating Data Access Object Implementation for JDBC

To run the Data Access Object (DAO) implementation for JDBC, complete the following steps:

1. Create a com.hp.empinfo.service package and an EmployeeDao class under the com.hp.empinfo.service package, as described in [“Creating the Controller for EmplInfo” \(page 91\)](#). This will contain the logic for handling Employee Database Transactions.
2. Modify the EmployeeDao.java class file to add the function for inserting employee details. After modification, the EmployeeDao.java file must appear as:

```

package com.hp.empinfo.service;

import java.sql.SQLException;
import org.springframework.jdbc.core.support.JdbcDaoSupport;
public class EmployeeDao extends JdbcDaoSupport{

    public void insertDetail(int empid, String firstname,
        String lastname, int age, String email)
        throws SQLException
    {
        getJdbcTemplate().update("insert into employee values(?, ?, ?, ?, ?, ?)",
            new Object[] { empid,firstname,lastname,age,email });
    }
}

```

## Creating and Modifying Supporting Views to Display Business Data

To create and modify supporting views to display business data, create the JSP files to create the view by completing the following steps:

1. Create the insertresult.jsp file in the Empinfo/WebContent/Web-INF/jsp directory as described in [“Creating the index.jsp File” \(page 80\)](#).
2. Modify the insertresult.jsp file to provide a link to the insert.jsp page.

After modification, the insertresult.jsp file must appear as:

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

```

```

"http://www.w3.org/TR/html4/loose.dtd">
<%@ include file="/WEB-INF/jsp/include.jsp"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Getting Started with EmpInfo Application</title>
</head>
<body>
    <h2 align="center"> Insert Operation Successfully Executed </h2>
    <br>
    ${model.add}
    <br>

    <br>
    <br>
    <h4><a href=<c:url value="insert.htm"/>>Insert Employee</a></h4>
</body>
</html>

```

3. Modify the insert.jsp file in the EmpInfo/WebContent/Web-INF/jsp/insert.jsp directory to receive all the input data required to add a new employee record to the database, by completing the following steps:

1. Delete the following <c:out> tag from the <body> tag.

```
<h3 align="center"><c:out value="${now}" /></h3>
```

2. Add the following lines of code in the body tag to receive all the inputs required to add a new employee record.

```

<form:form method="post" commandName="emp">
    <table width="95%" bgcolor="#f8f8ff" border="0" cellspacing="0" cellpadding="5">
        <tr>
            <td align="left" width="20%">Employee Id :</td>
            <td><form:input path="empid"/> </td>
        </tr>
        <tr>
            <td align="left" width="20%">First Name :</td>
            <td> <form:input path="firstname"/> </td>
        </tr>
        <tr>
            <td align="left" width="20%">Last Name :</td>
            <td> <form:input path="lastname"/> </td>
        </tr>
        <tr>
            <td align="left" width="20%">Age :</td>
            <td> <form:input path="age"/> </td>
        </tr>
        <tr>
            <td align="left" width="20%">Email :</td>
            <td> <form:input path="email"/> </td>
        </tr>
    </table>
    <br>
    <input type="submit" align="left" value="Insert">
</form:form>

```

After modification, the insert.jsp appears as:

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
    <head>
        <title> Spring Getting Started with EmpInfo Application </title>
    </head>
    <body>
        <br>
        <p><B><font color = "003399">Enter Employee Details</font></B></p>
        <form:form method="post" commandName="emp">
            <table width="95%" bgcolor="#f8f8ff" border="0" cellspacing="0" cellpadding="5">
                <tr>
                    <td align="left" width="20%">Employee Id :</td>
                    <td><form:input path="empid"/> </td>
                </tr>
                <tr>
                    <td align="left" width="20%">First Name :</td>

```

```

<td> <form:input path="firstname"/> </td>
</tr>
<tr>
    <td align="left" width="20%">Last Name :</td>
    <td> <form:input path="lastname"/> </td>
</tr>
<tr>
    <td align="left" width="20%">Age :</td>
    <td> <form:input path="age"/> </td>
</tr>
<tr>
    <td align="left" width="20%">Email :</td>
    <td> <form:input path="email"/> </td>
</tr>
</table>
<br>
<input type="submit" align="left" value="Insert">
</form:>
</body>
</html>

```

### Adding Dependency JAR Files

Add the following JAR file required for XML parsing:

**Table 3 XML Parser Dependency JAR File**

Dependency JAR Files	Source Location
com.springsource.org.dom4j-1.6.1.ja	< <i>Spring Dependency Home</i> >/org.dom4j/com.springsource.org.dom4j/1.6.1

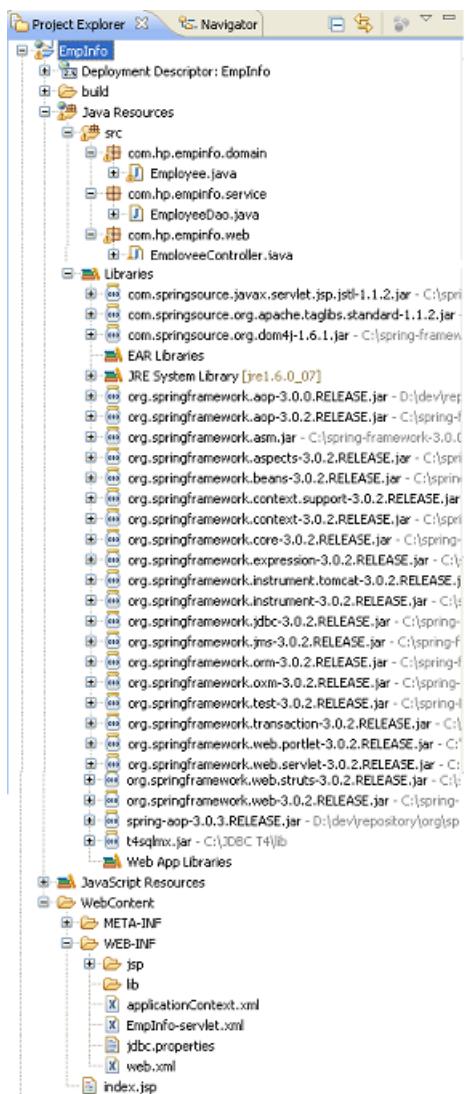
To add the dependency JAR file in the project library path and to resolve the J2EE module dependency on it, follow the instructions described in “[Adding Dependency JAR Files in the Project Library Path](#)” (page 89).

---

**NOTE:** The code of the EmplInfo application developed so far is located in <*My SASH Home*>\spring\getting-started\EmpInfo-InParts\Part-5

The project directory structure for the EmplInfo application must appear as follows:

**Figure 29 Project Explorer View**



## Integrating the Web-tier of EmpInfo with NonStop SQL/MX Database

This section describes the following tasks:

1. “[Creating the applicationContext.xml File](#)” (page 106)
2. “[Creating the jdbc.properties File](#)” (page 107)
3. “[Modifying the EmpInfo-servlet.xml File](#)” (page 109)
4. “[Improving the Controller](#)” (page 110)
5. “[Modifying the web.xml File](#)” (page 111)
6. “[Adding Dependency JAR Files](#)” (page 112)
7. “[Creating Database Catalog, Schema, and Tables on NonStop](#)” (page 113)

### Creating the applicationContext.xml File

To create the applicationContext.xml file, complete the following steps:

1. Create the applicationContext.xml file in the EmpInfo/WebContent/WEB-INF directory, as described in “[Creating the EmpInfo-servlet.xml File](#)” (page 86).
2. Create a new XML file in the EmpInfo/WebContent/WEB-INF directory and type the name of the XML file as **applicationContext**.
3. Modify the applicationContext.xml file to add a bean definition of the EmployeeDao class. After modification, the applicationContext.xml file must appear as:

```

<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource"
    >
        <property name="driverClassName">
            <value>${jdbc.driver}</value>
        </property>
        <property name="url">
            <value>${jdbc.url}</value>
        </property>
        <property name="username">
            <value>${jdbc.user}</value>
        </property>
        <property name="password">
            <value>${jdbc.password}</value>
        </property>
        <property name="connectionProperties">
            <props>
                <prop key="catalog">
                    ${jdbc.catalog}
                </prop>
                <prop key="schema">
                    ${jdbc.schema}
                </prop>
            </props>
        </property>
        </bean>
        <bean id="empdao" class="com.hp.empinfo.service.EmployeeDao">
            <property name="dataSource">
                <ref bean="dataSource" />
            </property>
        </bean>
        <bean id="propertyConfigurer"
            class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
            <property name="locations" value="/WEB-INF/jdbc.properties"/>
        </bean>
    </beans>

```

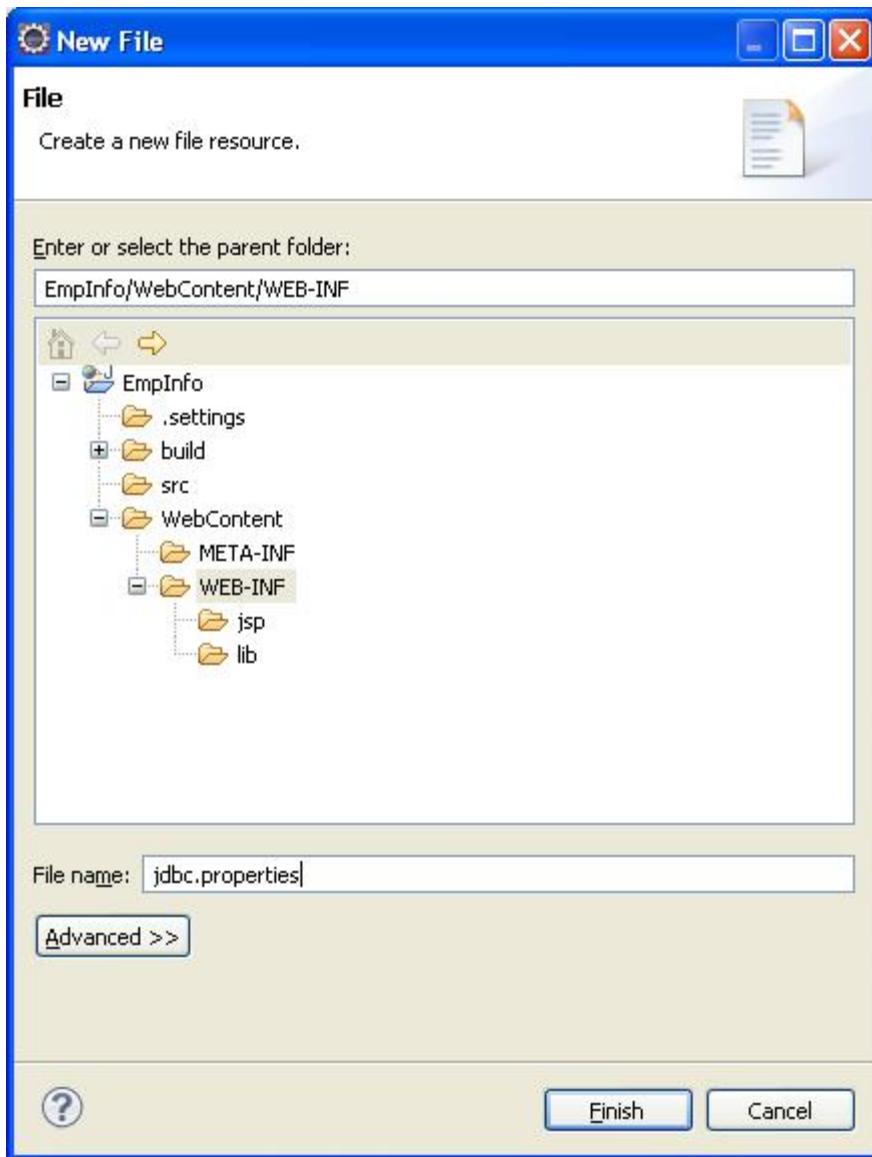
### Creating the `jdbct.properties` File

The JdbcDaoSupport feature of Spring framework is used for connecting to the NonStop SQL/MX database and for carrying out requested database transactions. To fulfill this requirement, the `jdbct.properties` file must be created to include all the database related information.

To create the `jdbct.properties` file, complete the following steps:

1. On the Project Explorer frame, right-click **EmplInfo** and select **New > File**.  
The New File dialog box appears.
2. In the **File name** field, type `jdbct.properties` and ensure that the parent folder is set to the `EmplInfo/WebContent/WEB-INF` directory. Click **Finish**.

**Figure 30 New File Dialog Box**



The `jdbc.properties` file is created.

3. Modify the `jdbc.properties` file based on the JDBC driver type you plan to use:

- If you plan to use the JDBC Type 2 driver, the SQL/MX settings for JDBC Type 2 driver in the `jdbc.properties` file must appear as:

```
#-----  
# SQL/MX Settings for JDBC Type 2 Driver  
  
jdbc.url=jdbc:sqlmx://  
jdbc.driver=com.tandem.sqlmx.SQLMXDriver  
jdbc.user=  
jdbc.password=  
jdbc.catalog=empinfocat  
jdbc.schema=empinfosch
```

---

**NOTE:** Because JDBC Type 2 driver resides on the NonStop system, you need not type the username and password in the `jdbc.username` and `jdbc.password` fields.

- If you plan to use the JDBC Type 4 driver, the SQL/MX settings for JDBC Type 4 driver in the `jdbc.properties` file must appear as:

```
#-----  
# SQL/MX Settings for JDBC Type 4 Driver  
  
jdbc.driver=com.tandem.t4jdbc.SQLMXDriver  
jdbc.url=jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.>  
jdbc.user=<HP NonStop Username>  
jdbc.password=<HP NonStop Password>  
jdbc.catalog=empinfocat  
jdbc.schema=empinfosch
```

---

**NOTE:**

- To use the JDBC Type 4 driver, type the JDBC URL (NonStop system IP Address and Port Number of the JDBC data source), NonStop system username, and password.
- The name of the database catalog used in the example is `empinfocat` and the schema name is `empinfosch`. If these database catalog and schema names conflict with any of the existing catalog and schema names on the NonStop system, modify the database catalog and schema names in the `jdbc.properties` file.

---

### Modifying the `EmpInfo-servlet.xml` File

Modify the `EmpInfo-servlet.xml` file in the `EmpInfo/WebContent/WEB-INF` directory to define the new form and controller as shown below:

Before modification:

```
<bean name="/insert.htm" class="com.hp.empinfo.web.EmployeeController"/>
```

After modification:

```
<bean name="/insert.htm" class="com.hp.empinfo.web.EmployeeController">  
<property name="formView">  
    <value>insert</value>  
</property>  
<property name="commandClass">  
    <value>com.hp.empinfo.domain.Employee</value>  
</property>  
<property name="commandName">  
    <value>emp</value>  
</property>  
</bean>
```

After modification, the `EmpInfo-servlet.xml` file appears as:

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
       xmlns:aop="http://www.springframework.org/schema/aop"  
       xmlns:p="http://www.springframework.org/schema/p"  
       xmlns:context="http://www.springframework.org/schema/context"
```

```

xmlns:jee="http://www.springframework.org/schema/jee"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.5.xsd
    http://www.springframework.org/schema/jee
    http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
<bean name="/insert.htm" class="com.hp.empinfo.web.EmployeeController">
<property name="formView">
    <value>insert</value>
</property>
<property name="commandClass">
    <value>com.hp.empinfo.domain.Employee</value>
</property>
<property name="commandName">
    <value>emp</value>
</property>
</bean>
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="viewClass" value="org.springframework.web.servlet.view.JstlView"></property>
<property name="prefix" value="/WEB-INF/jsp/"></property>
<property name="suffix" value=".jsp"></property>
</bean>
</beans>

```

## Improving the Controller

Modify the EmployeeController.java file to perform the employee database transactions.

To modify the EmployeeController.java file, complete the following steps:

1. Add a reference to the EmployeeDao.java in the EmployeeController.java controller class as shown below:

```
private EmployeeDao empdao;
```

2. Replace the onSubmit method with the handleRequest method.

This is done to:

1. Recognize the inputs from the View
2. Pass the recognized inputs to the Model
3. Pass the output back to the View

Before modification:

```
public ModelAndView handleRequest(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {

    String now = (new Date()).toString();
    logger.info("Returning view with " + now);
    return new ModelAndView("insert", "now", now);
}
```

After modification:

```
public ModelAndView onSubmit(Object command) throws ServletException,
    SQLException {
    WebApplicationContext wac = WebApplicationContextUtils
        .getRequiredWebApplicationContext(getApplicationContext());

    empdao = (EmployeeDao) wac.getBean("empdao");
    int empid = ((Employee) command).getEmpid();
    String firstname = ((Employee) command).getFirstname();
    String lastname = ((Employee) command).getLastname();
    int age = ((Employee) command).getAge();
    String email = ((Employee) command).getEmail();
    empdao.insertDetail(empid, firstname, lastname, age, email);

    Map<String, String> model = new HashMap<String, String>();
    model
```

```

    .put("add",
        "Transaction Complete - One Employee Added to Employee Database");
    return new ModelAndView("insertresult", "model", model);
}

```

3. Add the following import statements to include the classes used in the controller.

```

import com.hp.empinfo.domain.Employee;
import com.hp.empinfo.service.EmployeeDao;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.support.WebApplicationContextUtils;
import java.sql.SQLException;
import java.util.HashMap;
import java.util.Map;

```

After incorporating the above modifications, the EmployeeController.java controller class must appear as:

```

package com.hp.empinfo.web;

import org.springframework.web.servlet.mvc.SimpleFormController;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.ServletException;
import java.util.Date;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import java.io.IOException;

import com.hp.empinfo.domain.Employee;
import com.hp.empinfo.service.EmployeeDao;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.support.WebApplicationContextUtils;
import java.sql.SQLException;
import java.util.HashMap;
import java.util.Map;

public class EmployeeController extends SimpleFormController {

    private EmployeeDao empdao;

    public ModelAndView onSubmit(Object command) throws ServletException,
        SQLException {
        WebApplicationContext wac = WebApplicationContextUtils
            .getRequiredWebApplicationContext(getServletContext());

        empdao = (EmployeeDao) wac.getBean("empdao");
        int empid = ((Employee) command).getEmpid();
        String firstname = ((Employee) command).getFirstname();
        String lastname = ((Employee) command).getLastname();
        int age = ((Employee) command).getAge();
        String email = ((Employee) command).getEmail();
        empdao.insertDetail(empid, firstname, lastname, age, email);

        Map<String, String> model = new HashMap<String, String>();
        model
            .put("add",
                "Transaction Complete - One Employee Added to Employee Database");
        return new ModelAndView("insertresult", "model", model);
    }
}

```

## Modifying the web.xml File

To modify the web.xml in the EmpInfo/WebContent/ directory, complete the following steps:

1. Add ContextLoaderListener to load the applicationContext.xml file under the <web-app> tag:

```

<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>

```

2. Increase the startup load to 2 after adding the ContextLoaderListener to make DispatcherServlet as the secondary servlet class. The ContextLoaderListener servlet class becomes the primary servlet class.

After modification, the web.xml file must appear as:

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>EmpInfo</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>EmpInfo</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>EmpInfo</servlet-name>
    <url-pattern>*.htm</url-pattern>
  </servlet-mapping>
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml</param-value>
  </context-param>
</web-app>

```

## Adding Dependency JAR Files

Add the following dependency JAR files to the EmpInfo project library.

**Table 4 Jakarta Commons Dependency JAR Files**

Dependency JAR Files	Source Location
com.springsource.org.apache.commons.collections-3.2.1.jar	< <i>Spring Dependency Home</i> >/org.apache.commons/com.springsource.org.apache.commons.collections/3.2.1
com.springsource.javax.servlet-2.5.0.jar	< <i>Spring Dependency Home</i> >/org.apache.commons/com.springsource.org.apache.commons.lang/2.1.0

If you plan to use the JDBC Type 4 driver, you must add the JAR file of the JDBC Type 4 driver to the EmplInfo project library.

**Table 5 JDBC Type 4 Dependency JAR File**

Dependency JAR File	Source Location
t4sqlmx.jar	<JDBC T4 Installation Directory>\lib

To add the dependency JAR files in the project library path and to resolve the J2EE module dependency on these JAR files, follow the instructions described in ["Adding Dependency JAR Files in the Project Library Path" \(page 89\)](#).

### Creating Database Catalog, Schema, and Tables on NonStop

This section describes the steps to create the database catalog, schema, and tables on the NonStop system.

**NOTE:** The database catalog, schema, and table creation script `empinfo_tables_script.sql` is included in the `SAMPLES.zip` file.

For information on the `empinfo_tables_script.sql`, see ["EmplInfo Database Script" \(page 134\)](#).

To create a database catalog, schema, and table on NonStop, complete the following steps:

1. Copy the `empinfo_tables_script.sql` script from the `<My SASH Home>\spring\getting-started\dbconfig` Windows directory to the `<NonStop SASH Home>/spring/my_samples/empinfo/dbconfig` OSS directory.
2. Go to the OSS directory, where the `empinfo_tables_script.sql` script is copied, using the OSS command:  
`OSS> cd <NonStop SASH Home>/spring/my_samples/empinfo/dbconfig`  
For example:  
`OSS> cd /home/sash_usr/sash/spring/my_samples/empinfo/dbconfig`
3. Create the EmplInfo application database catalog, schema, and tables using the SQL/MX command:  
`mxci>> obey empinfo_tables_script.sql;`

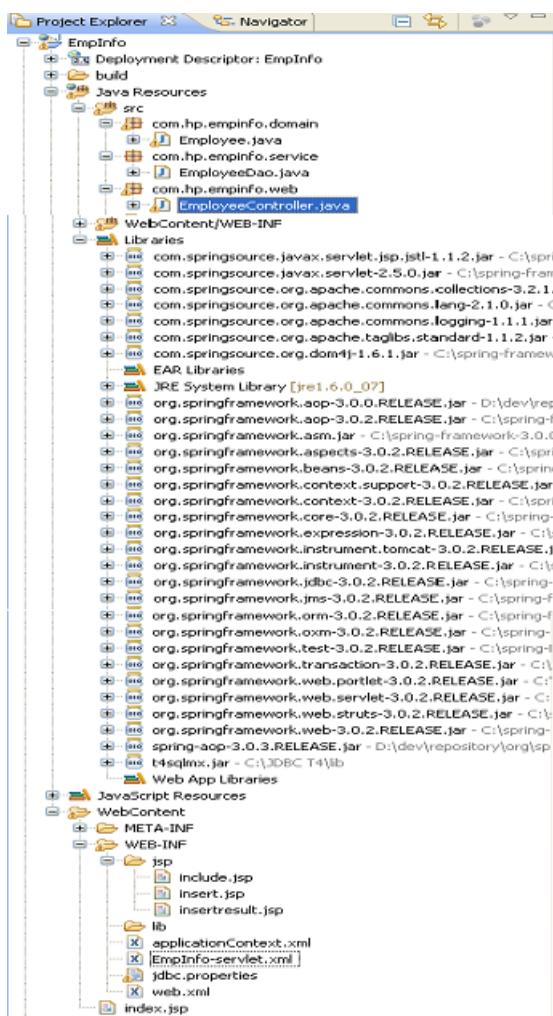
**NOTE:** By default, the `empinfo_tables_script.sql` script creates the database catalog name as `empinfocat` and the schema name as `empinfosch`. If these database catalog and schema names conflict with any of the existing catalog and schema names on the NonStop system, modify the database catalog and schema in the `empinfo_tables_script.sql` script file.

If you modify the database catalog and schema names, the new database catalog names must be updated in the `EmplInfo/WebContent/WEB-INF/jdbc.properties` file.

For information on the `empinfo_tables_script.sql`, see the [EmplInfo Database Script](#).

The project directory structure for the EmplInfo application must appear as follows:

**Figure 31 Project Explorer View**



At this point, you can either deploy and verify the EmplInfo application you have developed so far on the NonStop system, or you can proceed to the “[Enabling EmplInfo to Delete and Retrieve Employee Details](#)” (page 116) section.

**NOTE:** The code for the EmplInfo application developed so far is located in `<My SASH Home>\spring\getting-started\EmpInfo-InParts\Part-6`

To verify the EmplInfo application developed so far, complete the following steps:

1. Deploy the EmplInfo application using the steps described in “[Deploying EmplInfo on NonStop](#)” (page 129).
2. Verify the EmplInfo application by accessing the following URL:

[http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/EmplInfo/insert.htm](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/EmplInfo/insert.htm)

You can now access the application at [http://<IP Address of the Webserver>:<port#>/<servlet\\_directory>/Emplinfo](http://<IP Address of the Webserver>:<port#>/<servlet_directory>/Emplinfo)

**Figure 32 EmplInfo: Insert Employee Screen**



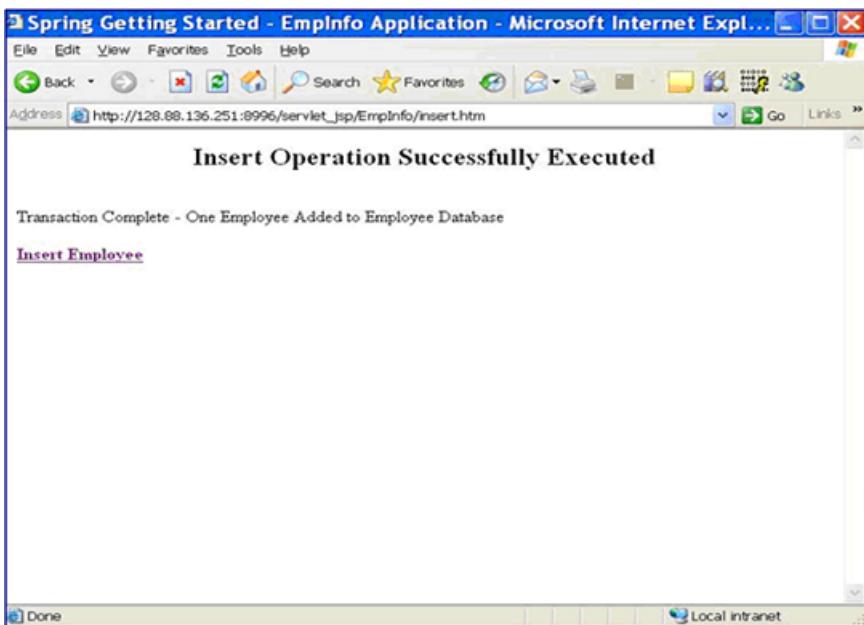
3. On the EmplInfo: Insert Employee screen, click **Insert Employee**.  
The EmplInfo: Employee Details screen appears.
4. Type the employee details and click **Insert**.

**Figure 33 EmplInfo: Employee Details Screen**

A screenshot of a Microsoft Internet Explorer window titled "Fetch Employee Details - Microsoft Internet Explorer". The address bar shows "http://128.88.136.198:1984/servlet\_jsp/EmpInfo/insert.htm". The main content area displays the heading "Enter Employee Details" followed by a form with five input fields: "Employee Id" (value 0), "First Name", "Last Name", "Age" (value 0), and "Email". Below the form is a blue "Insert" button.

The **Insert Operation Successfully Executed** message is displayed.

**Figure 34 EmplInfo: Message Screen**



## Enabling EmplInfo to Delete and Retrieve Employee Details

This section describes the following activities:

1. "Creating JSPs to Delete and Retrieve Employee Details" (page 116)
2. "Creating the RowMapper Class" (page 118)
3. "Modifying the EmployeeDao.java Class File" (page 119)
4. "Modifying the Employee.java File" (page 119)
5. "Modifying the EmployeeController.java File" (page 120)
6. "Modifying the EmpInfo-servlet.xml File" (page 121)
7. "Modifying the index.jsp File" (page 122)
8. "Implementing the RMI Service in EmplInfo" (page 123)

### Creating JSPs to Delete and Retrieve Employee Details

To delete or retrieve any employee details in the EmplInfo application, you must create the following files:

- retrieveordelete.jsp
- retrieveresult.jsp
- deleteresult.jsp

To create these JSP files, complete the following steps:

1. Create the retrieveordelete.jsp file in the EmpInfo/WebContent/WEB-INF/jsp directory, as described in "Creating the index.jsp File" (page 80).
2. Modify the retrieveordelete.jsp file, to retrieve an Employee ID from the JSP page, as follows:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
    <head>
        <title> Spring Getting Started with EmplInfo Application </title>
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
```

```

</head>
<body>
<h2 align="center">Welcome to the EmpInfo Application</h2>
<br>
<h2>Enter Employee Id</h2>
<form:form method="post" commandName="emprord">
<table width="95%" bgcolor="#f8f8ff" border="0" cellspacing="0" cellpadding="5">
<tr>
<td align="left" width="20%>Employee Id :</td>
<td><form:input path="empid"/> </td>
</tr>
<tr>
<td align="left" width="20%>Action to Perform (Retrieve/Delete) :</td>
<td><form:input path="rord"/> </td>
</tr>
</table>
<br>
<input type="submit" align="left" value="Retrieve/Delete">
<h4><a href=<c:url value="/" />> Home page </a></h4>
</form:form>
</body>
</html>

```

3. Create the retrieveresult.jsp file in the EmpInfo/WebContent/WEB-INF/jsp directory, as described in "[Creating the index.jsp File](#)" (page 80).
4. Modify the retrieveresult.jsp file to display the details of an employee record on the JSP page.

After modification, the retrieveresult.jsp file appears as:

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
    <head>
        <title> Spring Getting Started with EmpInfo Application </title>
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
    </head>
    <body>

        <table width="95%" bgcolor="#f8f8ff" border="0"
        cellspacing="0" cellpadding="5">
            <tr>
                <td align="left" width="20%>Employee Id :</td>
            <td><c:out value="${model.empid}"/> </td>
            </tr>
            <tr>
                <td align="left" width="20%>Employee First Name : </td><td><c:out value="${model.empfn}"/> </td>
            </tr>
            <tr>
                <td align="left" width="20%>Employee Last Name :</td> <td><c:out value="${model.empln}"/> </td>
            </tr>
            <tr>
                <td align="left" width="20%>Employee Age : </td><td><c:out value="${model.empage}"/> </td>
            </tr>
            <tr>
                <td align="left" width="20%>Employee Email Id :</td><td> <c:out value="${model.empemail}"/> </td>
            </tr>
        </table>

        <a href=">Back</a>

```

```
</body>
</html>
```

5. Create the deleteresult.jsp file in the EmpInfo/WebContent/WEB-INF/jsp directory, as described in “[Creating the index.jsp File](#)” (page 80).
6. Modify the deleteresult.jsp file to remove the details of an employee record from the JSP page.

After modification, the deleteresult.jsp file appears as:

```
<%@ include file="/WEB-INF/jsp/include.jsp"%>
<%@ taglib prefix="form"
uri="http://www.springframework.org/tags/form"%>

<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1">
<title> Spring Getting Started with EmpInfo Application </title>

</head>
<body text="#00008B" >
<h2>Employee Deleted</h2>
<br>
${model.del}
<br>
<a href=<c:url value="retrieveordelete.htm"/>>Back </a>
</body>
</html>
```

## [Creating the RowMapper Class](#)

The RowMapper class is used to map the retrieved ResultSet to the POJO class.

To create the RowMapper class, complete the following steps:

1. Create a class EmployeeRowMapper in the com.hp.empinfo.domain package, as described in “[Creating the Controller for EmpInfo](#)” (page 91).
2. Modify the EmployeeRowMapper.java file to map ResultSet to the Employee.java class.

After modification, the EmployeeRowMapper.java file appears as:

```
package com.hp.empinfo.domain;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

public class EmployeeRowMapper implements RowMapper {

    public Object mapRow(ResultSet rs, int rowcount) throws SQLException {
        Employee employee=new Employee();
        employee.setEmpid(rs.getInt(1));
        employee.setFirstname(rs.getString(2));
        employee.setLastname(rs.getString(3));
        employee.setAge(rs.getInt(4));
        employee.setEmail(rs.getString(5));
        return employee;
    }
}
```

## Modifying the EmployeeDao.java Class File

To modify the EmployeeDao.java in com.hp.empinfo.service package, complete the following steps:

1. Add two new methods getDetail and deleteEmployee in the EmployeeDao.java class file to add the **retrieve** and **delete** functionality in the application.
2. Add two import statements that are being used in the modified EmployeeDao.java file:

```
import com.hp.empinfo.domain.Employee;
import com.hp.empinfo.domain.EmployeeRowMapper;
```

After modification, the EmployeeDao.java file appears as:

```
package com.hp.empinfo.service;

import java.sql.SQLException;
import org.springframework.jdbc.core.support.JdbcDaoSupport;
import com.hp.empinfo.domain.Employee;
import com.hp.empinfo.domain.EmployeeRowMapper;

public class EmployeeDao extends JdbcDaoSupport {
    public Employee getDetail(int empid) throws SQLException {
        Employee employee;
        employee = (Employee) getJdbcTemplate().queryForObject(
            "select * from employee where emp_id = ?",
            new Object[] { empid }, new EmployeeRowMapper());
        return employee;
    }
    public void insertDetail(int empid, String firstname, String lastname,
        int age, String email) throws SQLException {
        getJdbcTemplate().update("insert into employee values(?, ?, ?, ?, ?)",
            new Object[] { empid, firstname, lastname, age, email });
    }
    public String deleteEmployee(int empid) {
        getJdbcTemplate().update("delete from employee where emp_id= ?",
            new Object[] { empid });
        return "Employee deleted";
    }
}
```

## Modifying the Employee.java File

You must modify the Employee.java file in the com.hp.empinfo.domain package to add a property record, which can be used to distinguish between the retrieve and delete request.

Modify the POJO to add a new property rord and its getter and setters.

After modification, the Employee.java file appears as:

```
package com.hp.empinfo.domain;
```

```
public class Employee implements java.io.Serializable {

    private int empid;
    private String firstname;
    private String lastname;
    private int age;
    private String email;
    private String rord;

    public String getRord() {
        return rord;
    }
    public void setRord(String rord) {
        this.rord = rord;
    }
}
```

```

    }
    public int getEmpid() {
        return empid;
    }
    public void setEmpid(int empid) {
        this.empid = empid;
    }
    public String getFirstname() {
        return firstname;
    }
    public void setFirstname (String firstname) {
        this.firstname= firstname;
    }
    public String getLastname () {
        return lastname;
    }
    public void setLastname (String lastname) {
        this.lastname= lastname;
    }
    public int getAge () {
        return age;
    }
    public void setAge (int age) {
        this. age= age;
    }
    public String getEmail () {
        return email;
    }
    public void setEmail (String email) {
        this. email= email;
    }
}

```

### Modifying the EmployeeController.java File

The EmployeeController.java file in the com.hp.empinfo.web package is modified to map the retrieve or delete request to the underlying database operations.

After modification, the EmployeeController.java file appears as:

```

package com.hp.empinfo.web;

import org.springframework.web.servlet.mvc.SimpleFormController;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.ServletException;
import com.hp.empinfo.domain.Employee;
import com.hp.empinfo.service.EmployeeDao;
import java.sql.SQLException;
import java.util.HashMap;
import java.util.Map;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.support.WebApplicationContextUtils;

public class EmployeeController extends SimpleFormController {

    private EmployeeDao empdao;

    public ModelAndView onSubmit(Object command) throws ServletException,
        SQLException {

        WebApplicationContext wac = WebApplicationContextUtils.
            getRequiredWebApplicationContext(getServletContext ());

        empdao = (EmployeeDao) wac.getBean("empdao");
        int empid = ((Employee) command).getEmpid();
    }
}

```

```

String firstname = ((Employee) command).getFirstname();
String lastname = ((Employee) command).getLastname();
int age = ((Employee) command).getAge();
String email = ((Employee) command).getEmail();
String rord = ((Employee) command).getRord();

if (rord!=null && rord.equalsIgnoreCase("Retrieve")) {

    Employee emp1 = empdao.getDetail(empid);

    Map<String, String> model = new HashMap<String, String>();
    model.put("empid", "" + emp1.getEmpid());
    model.put("empfn", emp1.getFirstname());
    model.put("empln", emp1.getLastname());
    model.put("empage", "" + emp1.getAge());
    model.put("empemail", emp1.getEmail());

    return new ModelAndView("retrieveresult", "model", model);
}

if (rord!=null && rord.equalsIgnoreCase("Delete")) {

    String str = empdao.deleteEmployee(empid);

    Map<String, String> model = new HashMap<String, String>();
    model.put("del", str);
    return new ModelAndView("deleteresult", "model", model);
}

else {
    empdao.insertDetail(empid, firstname, lastname, age, email);

    Map<String, String> model = new HashMap<String, String>();
    model
        .put("add",
            "Transaction Complete - One Employee Added to Employee Database");
    return new ModelAndView("insertresult", "model", model);
}
}
}

```

### Modifying the EmpInfo-servlet.xml File

Modify the EmpInfo-servlet.xml file in the EmpInfo/WebContent/WEB-INF directory to map the command name and its corresponding command class for retrieveordelete.jsp.

After modification, the EmpInfo-servlet.xml file appears as:

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-2.5.xsd
        http://www.springframework.org/schema/jee"

```

```

http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
    http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

<bean name="/insert.htm"
class="com.hp.empinfo.web.EmployeeController">
<property name="formView">
    <value>insert</value>
</property>
<property name="commandClass">
    <value>com.hp.empinfo.domain.Employee</value>
</property>
<property name="commandName">
    <value>emp</value>
</property>
</bean>

<bean name="/retrieveordelete.htm"
class="com.hp.empinfo.web.EmployeeController">
<property name="formView">
    <value>retrieveordelete</value>
</property>
<property name="commandClass">
    <value>com.hp.empinfo.domain.Employee</value>
</property>
<property name="commandName">
    <value>emprord</value>
</property>
</bean>
<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"></property>
    <property name="prefix" value="/WEB-INF/jsp/"></property>
    <property name="suffix" value=".jsp"></property>
</bean>

</beans>

```

## Modifying the index.jsp File

Modify the index.jsp file in the EmpInfo/WebContent directory to provide the links to retrieve or delete a page as shown:

```

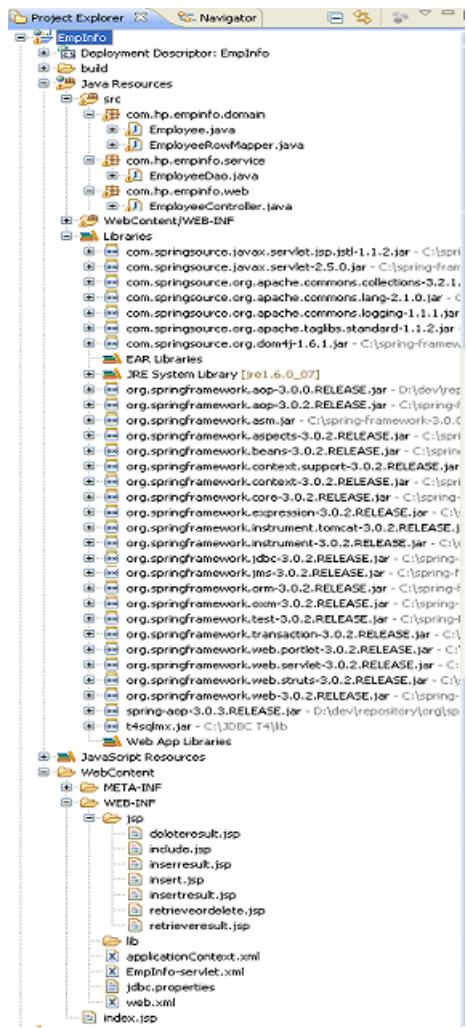
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Getting Started with EmpInfo Application</title>
</head>
<body>
    <h2 align="center"> Welcome to the EmpInfo Application </h2>
    <br>
<h4><a href=<c:url value="insert.htm"/>>Insert Employee</a></h4>

<h4><a href=<c:url value="retrieveordelete.htm"/>> Retrieve or Delete</a></h4>
<br>
</body>
</html>

```

Figure 35 shows what the EmpInfo application directory structure must look like after creating the requisite files on the system.

**Figure 35 Project Explorer View**



## Implementing the RMI Service in EmplInfo

To implement the RMI service in the EmplInfo, complete the following steps:

1. Creating RMI Service Interface.
2. Creating Implementation Class of RMI service.
3. Modify applicationContext.xml file.
4. Creating Test class to Test the RMI service.
5. Creating the clientContext.xml file
6. Creating the client.properties file
7. Adding Dependency JARs

### Creating RMI Service Interface

To create the RMI service interface complete the following steps:

1. Create a com.hp.empinfo.service package and an EmployeeInfo class under the com.hp.empinfo.service package, as described in "[Creating the Controller for EmplInfo \(page 91\)](#)".

2. Modify the EmployeeInfo.java file as follows:

```
package com.hp.empinfo.service;

import com.hp.empinfo.domain.Employee;

public interface EmployeeInfo {
    public Employee getEmployee(int empId);
}.
```

This creates the RMI service interface which you must implement as described in the subsequent section.

### **Creating Implementation Class of RMI service Interface**

To create an implementation class of RMI service, complete the following steps:

1. Create a com.hp.empinfo.service package and an EmployeeInfoImpl class under the com.hp.empinfo.service package, as described in “[Creating the Controller for EmplInfo](#)” (page 91).
2. Modify the EmployeeInfoImpl.java class as follows:

```
package com.hp.empinfo.service;

import java.sql.SQLException;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.hp.empinfo.domain.Employee;

public class EmployeeInfoImpl implements EmployeeInfo {
    private EmployeeDao empdaoBean;

    public EmployeeDao getEmpdaoBean() {
        return empdaoBean;
    }
    public void setEmpdaoBean(EmployeeDao empdaoBean) {
        this.empdaoBean = empdaoBean;
    }
    public static String [] configFileNames = new String [] {"applicationContext.xml","clientContext.xml"};
    private static ApplicationContext applicationContext = null;

    public Employee getEmployee(int empId) {
        Employee detail = null;

        applicationContext = new ClassPathXmlApplicationContext(configFileNames);
        EmployeeDao empDao = (EmployeeDao)applicationContext.getBean("empdao");

        try {
            detail = empDao.getDetail(empId);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return detail;
    }
}
```

### **Modifying the applicationContext.xml file**

You must modify the applicationContext.xml file to add the Spring RMI bean. To achieve this, add the following code in the applicationContext.xml file:

```
<bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location"><value>classpath:jdbc.properties</value></property>
</bean>

<bean id="empinfo" class="com.hp.empinfo.service.EmployeeInfoImpl">
    <property name="empdaoBean" ref="empdao"/>
</bean>

<bean id="emp-rmi" class="org.springframework.remoting.rmi.RmiServiceExporter">
    <property name="service" ref="empinfo"/>

```

```

<property name="serviceInterface" value="com.hp.empinfo.service.EmployeeInfo"/>
<property name="serviceName" value="empDetail"/>
<property name="registryPort" value="1099"/>
</bean>

```

### **Creating Test class to Test the RMI service**

To create the test class that can be used to test the RMI Service, complete the following steps:

1. Create a com.hp.empinfo.domain package and an RmiClientTest class under the com.hp.empinfo.domain package, as described in “[Creating the Controller for EmplInfo](#)” (page 91).
2. Modify the RmiClientTest.java file to invoke the RMI service by adding the following code:

```

package com.hp.empinfo.domain;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.hp.empinfo.service.EmployeeInfo;

public class RmiClientTest {

    public static String [] configFileNames = new String [] {"applicationContext.xml","clientContext.xml"};
    private static ApplicationContext applicationContext = null;

    public static void main(String[] args) {
        applicationContext = new ClassPathXmlApplicationContext(configFileNames);
        EmployeeInfo empInfo = (EmployeeInfo)applicationContext.getBean("rmiProxy");
        Employee employee = empInfo.getEmployee(2);

        System.out.println("Employee First Name : "+employee.getFirstname());
        System.out.println("Employee Last Name : "+employee.getLastname());
        System.out.println("Employee Email : "+employee.getEmail());
        System.out.println("Employee Id : "+employee.getEmpid());

    }
}

```

### **Creating the clientContext.xml file**

To create the clientContext.xml file, complete the following steps:

1. Create the clientContext.xml file in the EmpInfo/WebContent/WEB-INF/ directory, as described in “[Creating the EmpInfo-servlet.xml File](#)” (page 86).
2. Modify the clientContext.xml file to add the RMI client bean. After modification, the clientContext.xml file must appear as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>

    <!-- Resolves ${...} placeholders from client.properties -->
    <bean id="propertyConfigurer"
        class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location"><value>classpath:client.properties</value></property>
    </bean>

    <bean id="rmiProxy" class="org.springframework.remoting.rmi.RmiProxyFactoryBean">
        <property name="serviceUrl">
            <value>rmi://${serverName}:${rmiPort}/empDetail</value>
        </property>
        <property name="serviceInterface">
            <value>com.hp.empinfo.service.EmployeeInfo</value>
        </property>
    </bean>
</beans>

```

### **Creating the client.properties file**

To create the client.properties file, complete the following steps:

1. Create the client.properties file in the EmpInfo/WebContent/WEB-INF/ directory, as described in “[Creating the jdbc.properties File](#)” (page 107).

2. Modify the `client.properties` file to add the NonStop System IP address (where your RMI service is running) and the RMI port number.

```
serverName= <HP NonStop System IP Address>
rmiPort=<port number>
```

### Adding Dependency JARs

Add the following dependency jars to the EmplInfo project library:

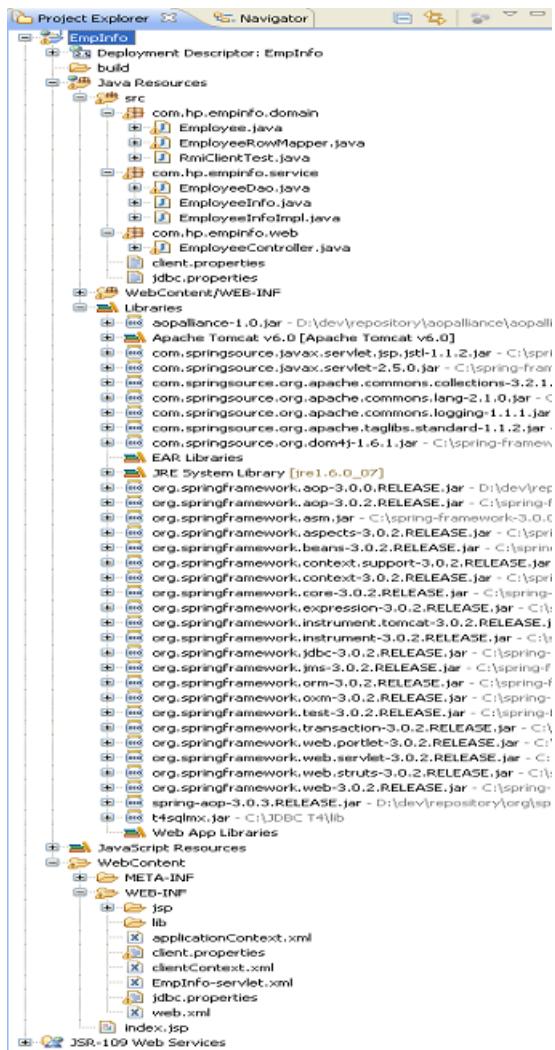
**Table 6 Aopalliance Dependency JAR Files**

Dependency JAR Files	Source Location
aopalliance-1.0.jar	<Spring Dependency Home>\org.aopalliance\com.springsource.org.aopalliance\1.0.0

To add the dependency JAR files in the project library path and to resolve the J2EE module dependency on these JAR files, follow the instructions described in “[Adding Dependency JAR Files in the Project Library Path](#)” (page 89).

Figure 36 shows what the EmplInfo application directory structure must look like after creating the requisite files on the system.

**Figure 36 Project Explorer View**



The EmplInfo application is complete now. You can now deploy and verify the EmplInfo application on a NonStop system, by following the steps described in “[Deploying EmplInfo on NonStop](#)” (page 129).

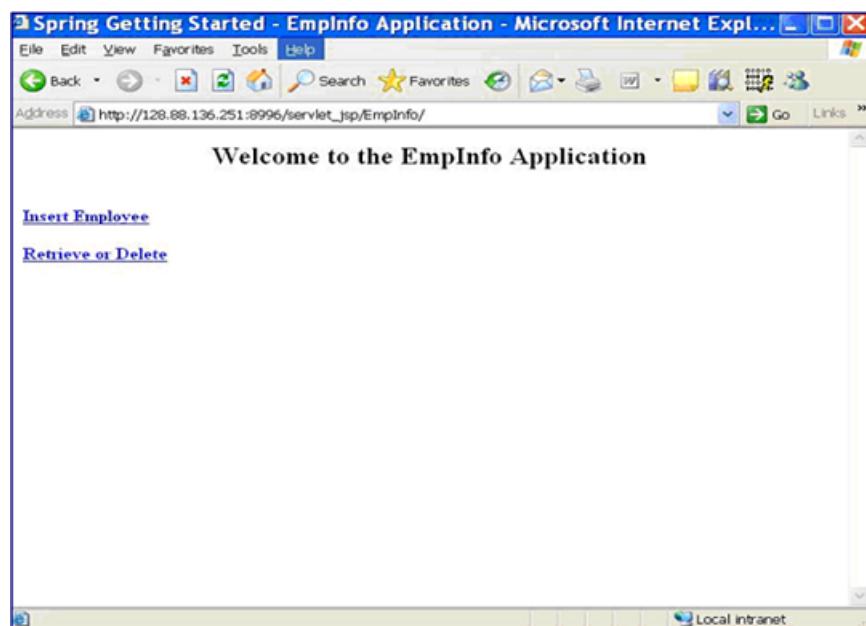
**NOTE:** The complete code of the EmplInfo application is located in <My SASH Home>\spring\getting-started\EmpInfo .

To verify if the application is working, complete the following steps:

1. Start the iTP WebServer and access the index page:

[http://<IP Address of the Webserver>:<port#>/<servlet\\_directory>/EmplInfo](http://<IP Address of the Webserver>:<port#>/<servlet_directory>/EmplInfo)

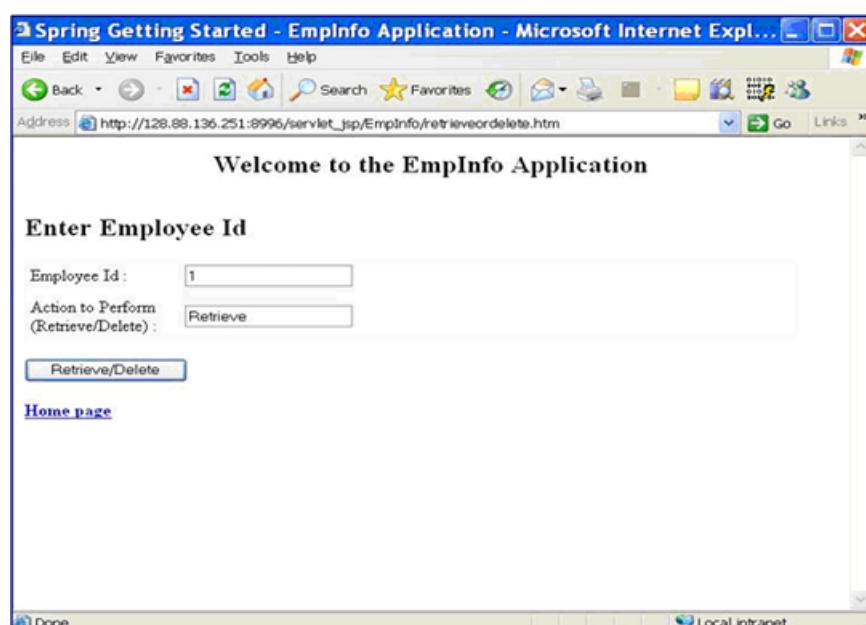
**Figure 37 EmplInfo: Retrieve or Delete Employee Details**



2. Click **Retrieve or Delete** to access the following URL:

[http://<IP Address of the Webserver>:<port#>/<servlet\\_directory>/EmplInfo/retrieveordelete.htm](http://<IP Address of the Webserver>:<port#>/<servlet_directory>/EmplInfo/retrieveordelete.htm)

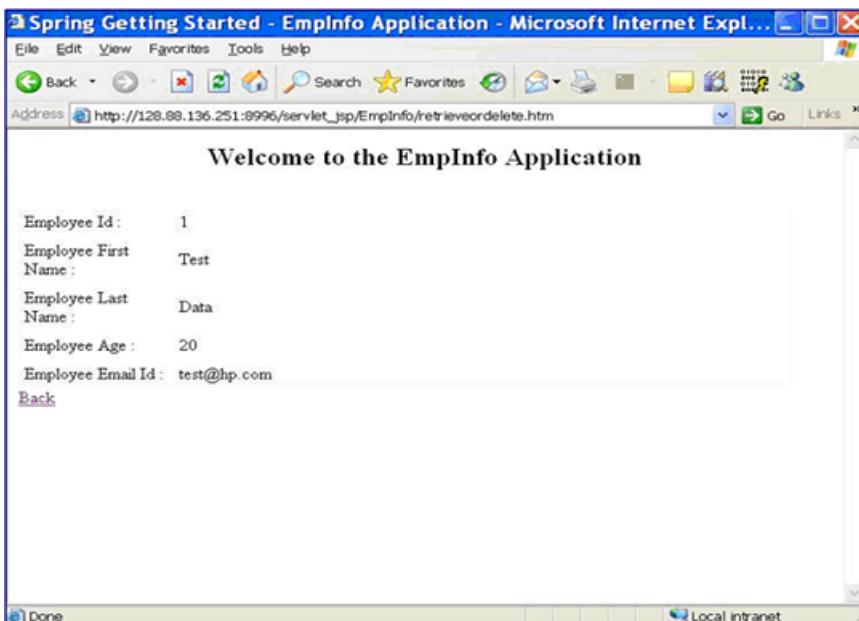
**Figure 38 EmplInfo: Retrieve Screen**



3. To retrieve an employee record, in the **Action to Perform** field, type **Retrieve**. Click **Retrieve/Delete**.

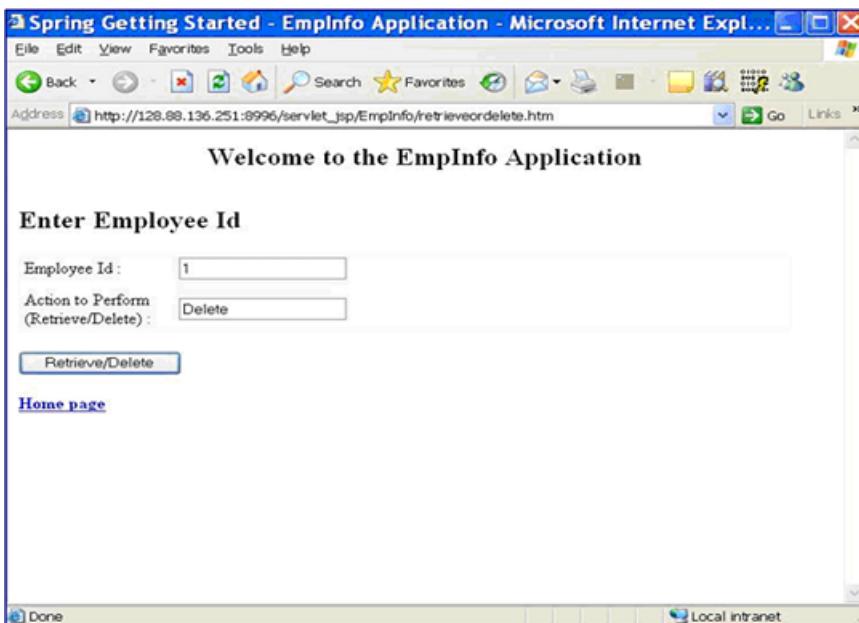
The employee record details are displayed.

**Figure 39 EmplInfo: Employee Details Screen**



4. Click **Back** to return to the EmplInfo: Delete screen.

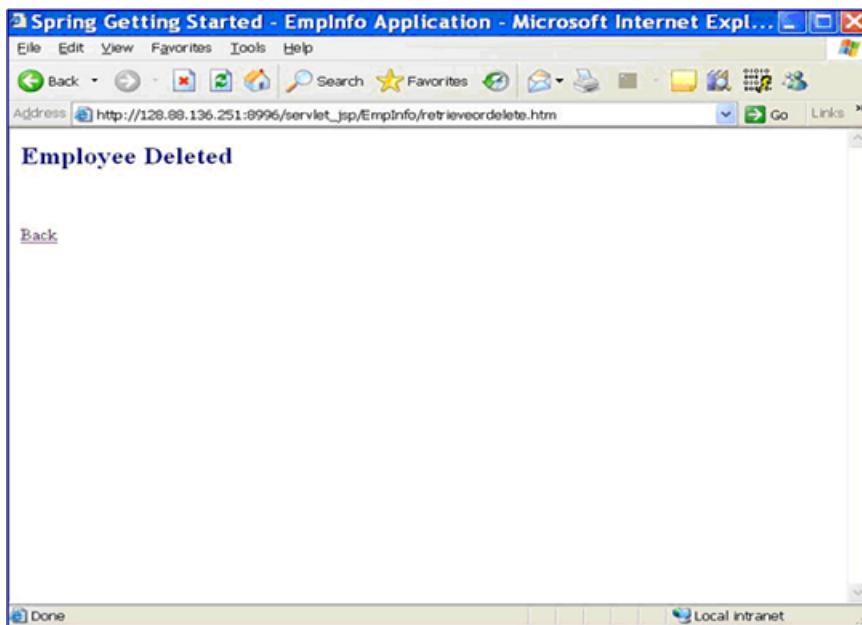
**Figure 40 EmplInfo: Delete Screen**



5. To delete an employee record, in the **Action to Perform** field, type **Delete**. Click **Retrieve/Delete**.

The employee record is deleted.

**Figure 41 EmplInfo: Deleted Screen**



## Deploying EmplInfo on NonStop

This section describes the following tasks:

1. "Creating the Application WAR File on Windows" (page 129)
2. "Deploying the EmplInfo WAR File in NSJSP on NonStop" (page 131)

### Creating the Application WAR File on Windows

A WAR file is essential to deploy the web application. It contains property and configuration files, Java class file, and JSPs of the web application.

To create the application WAR file, complete the following steps:

1. On the Project Explorer frame, right-click **EmplInfo** and select **Export > Export**.  
The Export dialog box appears.
2. From the list of folders, select **Web > WAR file** and click **Next**.

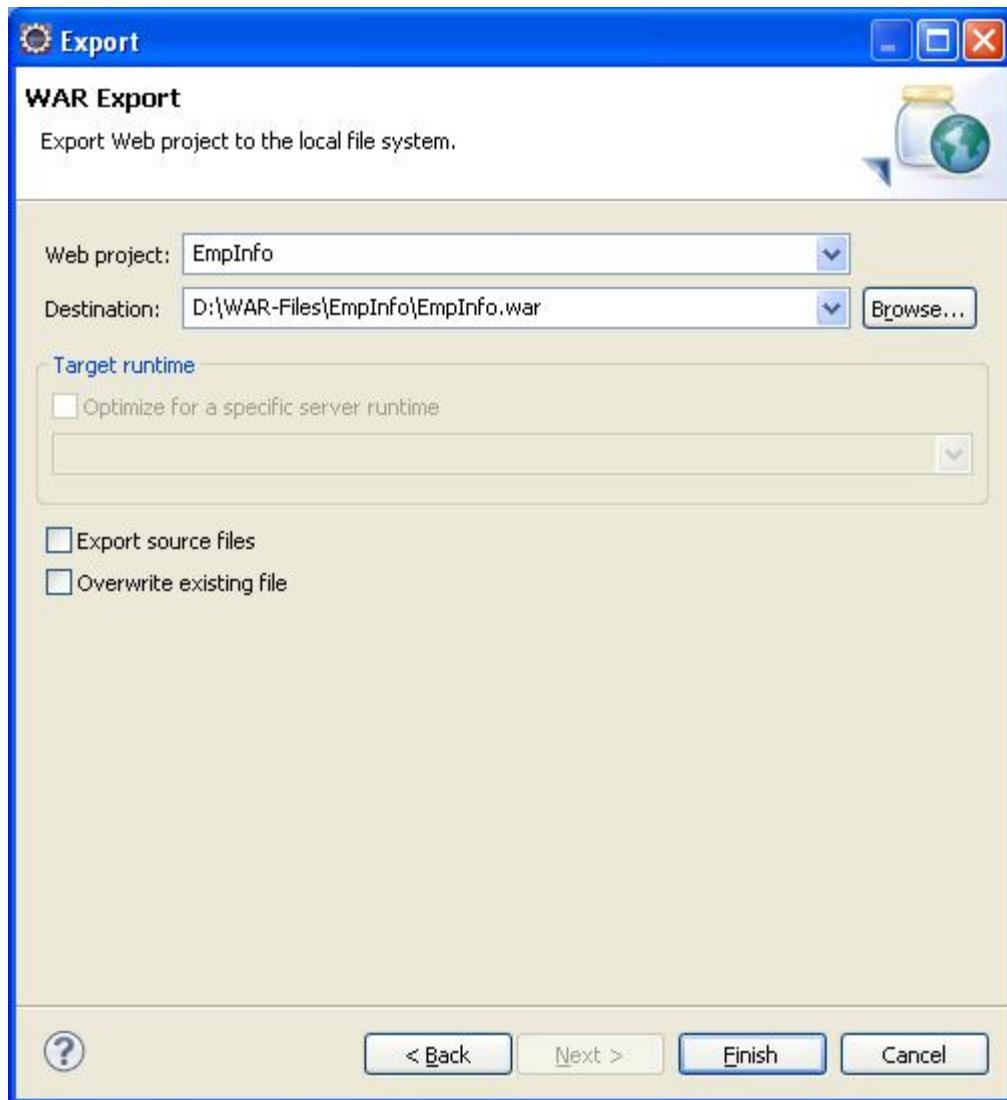
**Figure 42 Export Dialog Box**



The WAR Export dialog box appears.

3. In the **Web project** field, type `EmpInfo` and browse to the destination where you want to save the WAR file.

**Figure 43 WAR Export Dialog Box**



4. Click **Finish**. The WAR file is created.

**NOTE:** If you have specified an existing name for the WAR file, the **Finish** button is disabled. In this case, change the name of the WAR file. If you want use the existing name, select the **Overwrite existing file** check box.

## Deploying the EmplInfo WAR File in NSJSP on NonStop

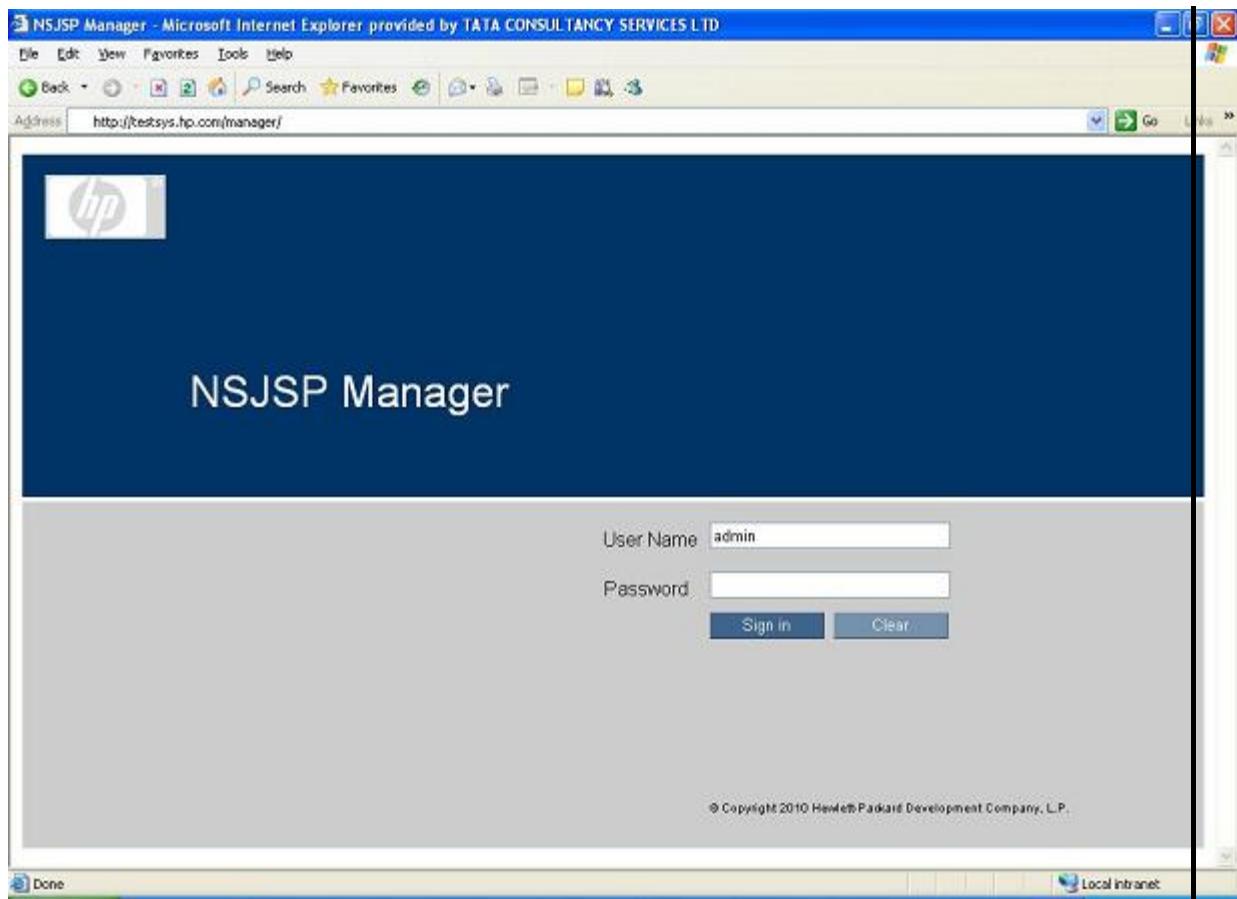
To deploy the EmplInfo WAR file on the NonStop system, complete the following steps:

1. Go to [http://<IP Address of the iTP WebServer>:<port#>/<manager\\_directory>](http://<IP Address of the iTP WebServer>:<port#>/<manager_directory>).

The **NSJSP Manager Login** screen appears.

Figure 44 shows the **NSJSP Manager Login** screen.

**Figure 44 NSJSP Manager Login Screen**

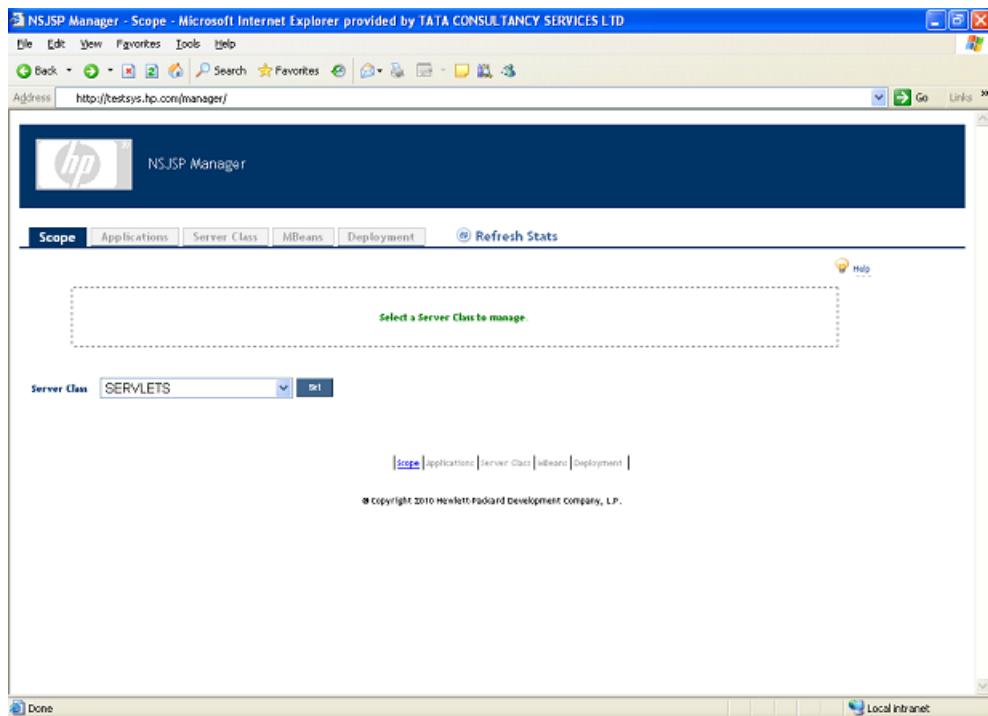


2. Enter the **User name:** and **Password:** and click **OK**.

The **NSJSP Manager** screen appears.

Figure 45 shows the **NSJSP Manager** screen.

**Figure 45 NSJSP Manager Screen**

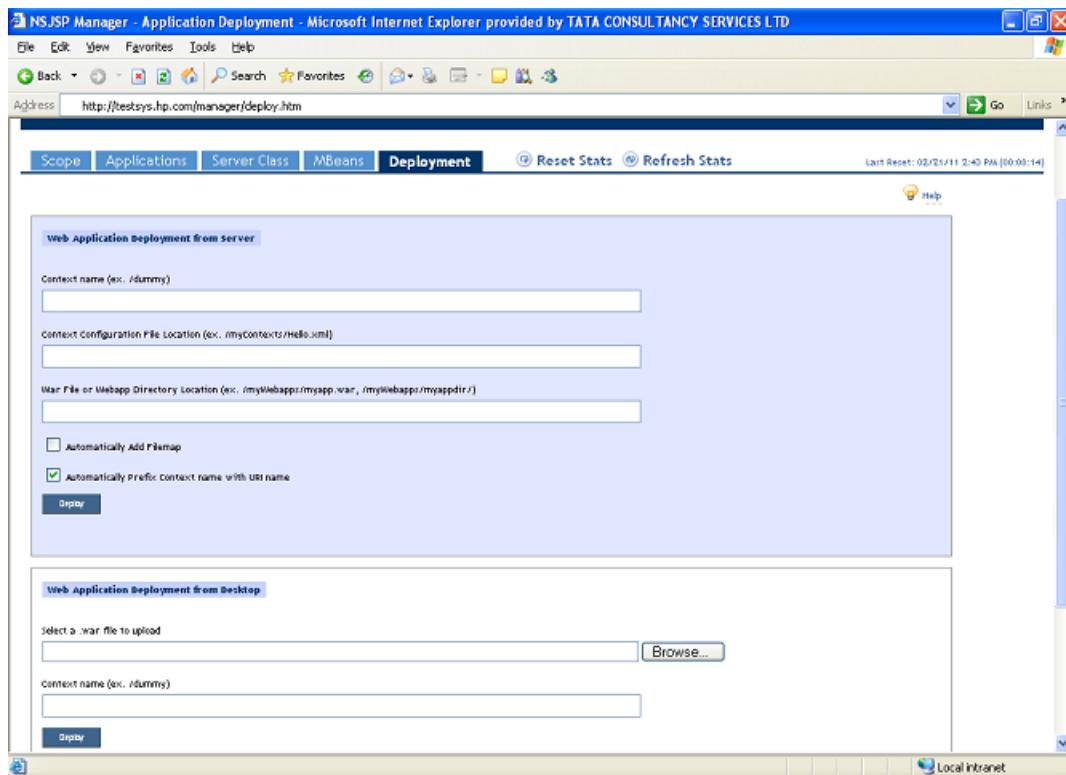


3. Select **SERVLETS** for the **Server Class** and click **Set**.

The **Host (in server.xml)** tab is enabled and populated with **localhost**. Click **Set**, which enables the **Deployment** tab on the **NSJSP Manager** screen.

Figure 46 shows the **Deployment** tab of **NSJSP Manager**.

**Figure 46 NSJSP Manager Screen - Deployment tab**



4. In the **Deployment** tab, complete the following steps in the **Web Application Deployment from Desktop** section:

1. In the **Select a .war file to upload** field, click **Browse...** and locate the `EmplInfo.war` file on the Windows system.
2. **(Optional)** In the **Context name** field, enter a name for the application context .
3. Click **Deploy**.

`EmplInfo` is deployed on NSJSP and is listed under **Applications** tab of the **NSJSP Manager** screen.

**NOTE:** HP recommends that you use the NSJSP manager for deployment. Deployment using the FTP or any other mechanism is not recommended. For more information on using the NSJSP manager, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

## Running EmplInfo on NonStop

To run `EmplInfo` on the NonStop system, click the `/<servlet directory>/EmplInfo` path under **Applications** in the NSJSP Web Application Manager screen.

You can now add, search, and delete employee details.

# A EmplInfo Database Script

The `emplinfo_tables_script.sql` script file is used to create the SQL/MX database for the EmplInfo application. The content of this file is as follows:

```
Log EmpInfo.log;
drop schema empinfocat.empinfosch cascade;
drop catalog empinfocat;

create catalog empinfocat LOCATION <node.$vol>;
create schema empinfocat.empinfosch AUTHORIZATION "<user>" LOCATION ZSD<subvol reference>;

set schema empinfocat.empinfosch;

create table employee(
emp_id int not null,
first_name varchar(30),
last_name varchar(30),
age int,
email varchar(50),
PRIMARY KEY(emp_id)
);
```

# B Customizing Sample Applications

This appendix provides information about how PetClinic and JPetStore sample applications were customized to run on NonStop systems.

You can download the Spring sample applications from <http://src.springframework.org/svn/spring-samples>.

## Customizing PetClinic

To customize the PetClinic sample application to run on NonStop systems, two directories and one file were added, and four files were modified.

### Added Directories:

- /mxci
- /etc

### Added File:

- SetDatabase.java

### Modified Files:

- web.xml
- petclinic.hbm.xml
- applicationContext-datasource.xml
- jdbc.properties

## Added Directories

### /mxci

(*<My SASH Home>\spring\samples\petclinic\src\main\resources\db\mxci*)

This directory contains the database script that is required to set up the PetClinic database on SQL/MX.

### /etc

(*<My SASH Home>\spring\samples\petclinic\etc*)

This directory is included in the SAMPLES.zip file and contains the hibernate3sqlmx.jar file.

## Added File

### SetDatabase.java

(*<My SASH Home>\spring\samples\petclinic\src\main\java\org\springframework\samples\petclinic\datasource\SetDatabase.java*)

The SetDatabase.java file was created, in the org.springframework.samples.petclinic.datasource package, for adding customized connection properties to the JDBC driver.

The SetDatabase.java class file appeared as:

```
package org.springframework.samples.petclinic.datasource;

import org.apache.commons.dbcp.BasicDataSource;

public class SetDataSource extends BasicDataSource{
    private String catalog;
    private String schema;
    public void setCatalog(String catalog) {
```

```

        this.catalog=catalog;
        addConnectionProperty("catalog",this.catalog);
    }

    public void setSchema(String schema) {
        this.schema=schema;
        addConnectionProperty("schema",this.schema);
    }
}

```

## Modified Files

The following files were modified to customize PetClinic.

### `web.xml`

(*<My SASH Home>\spring\samples\petclinic\src\main\webapp\WEB-INF\web.xml*)

This is the deployment descriptor configuration file for PetClinic. This file was modified to enable Hibernate because PetClinic fails during ADD operations when used with JDBC.

The Add operation requires that the primary key ID values be generated for all the tables to be auto-incremented. This feature of auto-incrementing is not supported by the SQL/MX database and therefore the `web.xml` file was modified to enable Hibernate.

### Changes to the `web.xml` file

By default, the PetClinic sample application distributed with the Spring source is configured to use the HSQLDB database via JDBC.

The following modifications were made to connect the SQL/MX database using Hibernate:

- The `<param-value>` tag with value `/WEB-INF/applicationContext-jdbc.xml` was commented.
- The `<param-value>` tag with value `/WEB-INF/applicationContext-hibernate.xml` was uncommented.

Before the change:

```

<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext-jdbc.xml</param-value>
    <!--
    <param-value>/WEB-INF/applicationContext-hibernate.xml</param-value>
    <param-value>/WEB-INF/applicationContext-jpa.xml</param-value>
    -->

    <!--
    To use the JPA variant above,
    you will need to enable Spring load-time weaving in your
    server environment.
    See PetClinic's readme and/or Spring's JPA documentation for
    information on how to do this.
    -->
</context-param>

```

After the change:

```

<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext-hibernate.xml</param-value>
    <!--
    <param-value>/WEB-INF/applicationContext-jdbc.xml</param-value>
    <param-value>/WEB-INF/applicationContext-jpa.xml</param-value>
    -->

    <!--

```

```

To use the JPA variant above,
you will need to enable Spring load-time weaving in your
server environment.
See PetClinic's readme and/or Spring's JPA documentation for
information on how to do this.
-->
</context-param>
```

## `petclinic.hbm.xml`

(*My SASH Home*)\spring\samples\petclinic\src\main\resources\petclinic.hbm.xml)

This Hibernate mapping file is used to map database tables with the corresponding Java objects. This file was modified to use the Hibernate Generator class as increment instead of identity, because identity is not supported by the SQL/MX database. During the Add operation, the Hibernate Generator class increment selects the max ID value for each table, increments it by 1, and then adds the new record with this incremented ID value.

### Changes to the `petclinic.hbm.xml` file

The changes to the `petclinic.hbm.xml` are as follows:

The value identity was replaced with increment for all `<generator class="identity"/>` tags in the (*My SASH Home*)\spring\samples\petclinic\src\main\resources\petclinic.hbm.xml file.

Before the change:

```
<id name="id" column="id">
  <generator class="identity"/>
</id>
```

After the change:

```
<id name="id" column="id">
  <generator class="increment"/>
</id>
```

## `applicationContext-dataSource.xml`

(*My SASH Home*)\spring\samples\petclinic\src\main\webapp\WEB-INF\spring\applicationContext-dataSource.xml)

This file is used for configuring the Hibernate settings for PetClinic and was modified to add the PetClinic database catalog and schema.

### Changes to the `applicationContext-dataSource.xml` file

The `dataSource` bean was modified to add the PetClinic database catalog and schema property.

Before the change:

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
  p:driverClassName="${jdbc.driverClassName}" p:url="${jdbc.url}"
  p:username="${jdbc.username}"
  p:password="${jdbc.password}"/>
</bean>
```

After the change:

```
<bean id="dataSource" class="org.springframework.samples.petclinic.datasource.SetDataSource">
  <property name="driverClassName" value="${jdbc.driverClassName}"/>
  <property name="url" value="${jdbc.url}"/>
  <property name="username" value="${jdbc.username}"/>
  <property name="password" value="${jdbc.password}"/>
  <property name="catalog" value="${jdbc.catalog}"/>
  <property name="schema" value="${jdbc.schema}"/>
</bean>>
```

## [jdbc.properties](#)

```
(<My SASH Home>\spring\samples\petclinic\src\main\resources\  
jdbc.properties)
```

This configuration file is used for connecting to the SQL/MX database, and was modified to add the SQL/MX JDBC properties to override the default HSQL JDBC properties used by PetClinic.

### [Changes to the jdbc.properties file](#)

The following modifications were made to set the JDBC properties for the SQL/MX database:

- The HSQL Settings section that includes the JDBC properties and the HSQL dialect file name was commented.
- The SQL/MX Settings section that includes the JDBC Type 2 and Type 4 drivers was added.

Before the change:

```
#-----  
# HSQL Settings  
  
jdbc.driverClassName=org.hsqldb.jdbcDriver  
jdbc.url=jdbc:hsqldb:hsq://localhost:9001  
jdbc.username=sa  
jdbc.password=  
  
# Property that determines which Hibernate dialect to use  
# (only applied with "applicationContext-hibernate.xml")  
hibernate.dialect=org.hibernate.dialect.HSQLDialect  
  
# Property that determines which database to use with an AbstractJpaVendorAdapter#  
jpa.database=HSQL
```

After the change:

```
#-----  
# HSQL Settings  
#jdbc.driverClassName=org.hsqldb.jdbcDriver  
#jdbc.url=jdbc:hsqldb:hsq://localhost:9001  
#jdbc.username=sa  
  
#jdbc.password=  
# Property that determines which Hibernate dialect to use  
# (only applied with "applicationContext-hibernate.xml")  
#hibernate.dialect=org.hibernate.dialect.HSQLDialect  
# Property that determines which database to use with an AbstractJpaVendorAdapter  
# jpa.database=HSQL  
...  
....  
....  
...  
#-----  
# SQL/MX Settings for JDBC Type 2 Driver  
#jdbc.driverClassName=com.tandem.sqlmx.SQLMXDriver  
#jdbc.url=jdbc:sqlmx://  
#jdbc.username=  
#jdbc.password=  
#jdbc.catalog=petcliniccat  
#jdbc.schema=petclinicsch  
#hibernate.dialect=org.hibernate.dialect.SqlmxDialect  
  
#-----  
# SQL/MX Settings for JDBC Type 4 Driver  
#jdbc.driverClassName=com.tandem.t4jdbc.SQLMXDriver  
#jdbc.url=jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.>  
#jdbc.username=<HP NonStop Username>  
#jdbc.password=<HP NonStop Password>  
#jdbc.catalog=petcliniccat  
#jdbc.schema=petclinicsch  
#hibernate.dialect=org.hibernate.dialect.SqlmxDialect  
  
#For JDBC Type 4 Driver:  
#  
#<HP NonStop System IP Address> - This is the IP address of your NonStop system  
#<Port No.> - This is the Port Number of JDBC Data Source  
#<HP NonStop Username> - This is the HP NonStop system UserName  
#<HP NonStop Password> - This is the HP NonStop system Password
```

## Customizing JPetStore

To customize the JPetStore sample application to run on NonStop systems, one file was added and five files were modified.

### Added File:

- `SetDatabase.java`

### Modified Files:

- `Item.xml`
- `dataAccessContext-local.xml`
- `jdbc.properties`
- `Order.xml`
- `Pom.xml`

## Added File

The following file was added to customize JPetStore:

### `SetDatabase.java`

(*<My SASH Home>\spring\samples\jpetstore\src\main\java\org\springframework\samples\jpetstore\property\SetDatabase.java*)

The `SetDatabase.java` file was created, in the `org.springframework.samples.jpetstore.property` package, for adding customized connection properties to the JDBC driver.

The `SetDatabase.java` class file appeared as:

```
package org.springframework.samples.jpetstore.property;
import org.apache.commons.dbcp.BasicDataSource;
public class SetDatabase extends BasicDataSource{
    private String catalog;
    private String schema;
    public void setCatalog(String catalog) {
        this.catalog=catalog;
        addConnectionProperty("catalog",this.catalog);
    }

    public void setSchema(String schema) {
        this.schema=schema;
        addConnectionProperty("schema",this.schema);
    }
}
```

## Modified Files

The following files were modified to customize JPetStore:

### `Item.xml`

(*<My SASH Home>\spring\samples\jpetstore\src\main\java\org\springframework\samples\jpetstore\dao\ibatis\Item.xml*)

This is the Ibatis mapping file used to automate the mapping between database tables and Java objects. The mappings are decoupled from the application logic by packaging the SQL statements in XML configuration files.

In Ibatis, the term 'as value' is used to select a single primitive value from the database. Because 'value' is a keyword in the SQL/MX database, 'as value' could not be used and hence `Item.xml` was modified.

## Changes to the Item.xml file

Before the change:

```
<select id="getInventoryQuantity" resultclass="java.lang.integer">
    select qty as value from inventory where itemid = #value#
</select>
```

After the change:

```
<resultMap id="result-for-quantity" class="java.lang.Integer">
    <result property="qty" column="qty" />
</resultMap>
<select id="getInventoryQuantity" resultMap="result-for-quantity">
    select qty from inventory where itemid = #value#
</select>
```

## dataAccessContext-local.xml

(*<My SASH Home>\spring\samples\jpetstore\src\main\webapp\WEB-INF\dataAccessContext-local.xml*)

The dataAccessContext-local.xml file contains all the configuration metadata for JPetStore. The bean with the dataSource ID in the dataAccessContext-local.xml file was modified to use the customized SetDatabase class instead of the BasicDataSource class.

## Changes to the dataAccessContext-local.xml file

Before the change:

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="${jdbc.driverClassName}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
</bean>
```

After the change:

```
<bean id="dataSource" class="org.springframework.samples.jpetstore.property.SetDatabase">
    <property name="driverClassName" value="${jdbc.driverClassName}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
    <property name="catalog" value="${jdbc.catalog}" />
    <property name="schema" value="${jdbc.schema}" />
</bean>
```

## jdbc.properties

(*<My SASH Home>\spring\samples\jpetstore\src\main\webapp\WEB-INF\jdbc.properties*)

This is the configuration file for connecting to the SQL/MX database. This file was modified to add the SQL/MX JDBC properties instead of the default HSQL JDBC properties used by JPetStore.

## Changes to the jdbc.properties file

The following modifications were made to set the JDBC properties for the SQL/MX database:

- The HSQL Settings section that includes the JDBC properties and the HSQL dialect file name was commented.
- The SQL/MX Settings section that includes the JDBC Type 2 and Type 4 drivers was added.

Before the change:

```
# Properties file with JDBC-related settings.
# Applied by PropertyPlaceholderConfigurer from "dataAccessContext-local.xml".
# Targeted at system administrators, to avoid touching the context XML files.

jdbc.driverClassName=org.hsqldb.jdbcDriver
jdbc.url=jdbc:hsqldb:hsq1://localhost:9002
```

```

jdbc.username=sa
jdbc.password=

After the change:

# Properties file with JDBC-related settings.
# Applied by PropertyPlaceholderConfigurer from "dataAccessContext-local.xml".
# Targeted at system administrators, to avoid touching the context XML files.

#jdbc.driverClassName=org.hsqldb.jdbcDriver
#jdbc.url=jdbc:hsqldb:hsq://localhost:9002
#jdbc.username=sa
#jdbc.password=

-----
# SQL/MX Settings for JDBC Type 2 Driver
#jdbc.driverClassName=com.tandem.sqlmx.SQLMXDriver
#jdbc.url=jdbc:sqlmx://
#jdbc.username=
#jdbc.password=
#jdbc.catalog=jpetstorecat
#jdbc.schema=jpetstoresch

-----
# SQL/MX Settings for JDBC Type 4 Driver
#jdbc.driverClassName=com.tandem.t4jdbc.SQLMXDriver
#jdbc.url=jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.>
#jdbc.username=<HP NonStop Username>
#jdbc.password=<HP NonStop Password>
#jdbc.catalog=jpetstorecat
#jdbc.schema=jpetstoresch

#For JDBC Type 4 Driver:
#
#<HP NonStop System IP Address> - This is the IP address of your NonStop system
#<Port No.> - This is the Port Number of JDBC Data Source
#<HP NonStop Username> - This is the HP NonStop system UserName
#<HP NonStop Password> - This is the HP NonStop system Password

```

## [Order.xml](#)

(*<My SASH Home>\spring\samples\jpetstore\src\main\java\org\springframework\samples\jpetstore\dao\ibatis\maps\Order.xml*)

Order.xml is the ibatis mapping file that maps the Order.java class to the order table. In SQL/MX database, 'timestamp' is a keyword; therefore, the insert query for orderstatus in the order.xml file was modified.

### [Changes to the Order.xml file](#)

**Before the change:**

```

<insert id="insertOrderStatus">
    insert into orderstatus (ordered , linenum , timestamp , status)
    values (#orderId#, #orderId#, #orderDate#, #status#)
</insert>

```

**After the change:**

```

<insert id="insertOrderStatus">
    insert into orderstatus (orderid, linenum, "timestamp", status)
    values (#orderId#, #orderId#, #orderDate#, #status#)
</insert>

```

## [Pom.xml](#)

(*<My SASH Home>\spring\samples\jpetstore\Pom.xml*)

Pom.xml is a Maven file and is used to build the JPetStore application.

## Changes to the Pom.xml file

The following modifications were made to the POM.xml file:

- The Spring-version property in the properties section of the pom.xml file was modified to 3.0.3.

Before the change:

```
<properties>
    <spring.version>3.0.0.BUILD-SNAPSHOT</spring.version>
    <tiles.version>2.2.0</tiles.version>
    <hsqldb.version>1.8.0.7</hsqldb.version>
    .
    .
    .

</properties>
```

After the change:

```
<properties>
    <spring.version>3.0.3.RELEASE</spring.version>
    <tiles.version>2.2.0</tiles.version>
    <hsqldb.version>1.8.0.7</hsqldb.version>
    .
    .
    .

</properties>
```

- The slf4j logging APIs were added.

```
<!-- Logging -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${org.slf4j-version}</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <version>${org.slf4j-version}</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>${org.slf4j-version}</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.16</version>
    <scope>runtime</scope>
</dependency>
```

- The following properties were added for the slf4j logging APIs.

Before the change:

```
<properties>
    <spring.version>3.0.0.BUILD-SNAPSHOT</spring.version>
    <tiles.version>2.2.0</tiles.version>
    <hsqldb.version>1.8.0.7</hsqldb.version>
    <commons-fileupload.version>1.2.1</commons-fileupload.version>
    <struts.version>1.2.9</struts.version>
    <xml-rpc.version>1.1.0</xml-rpc.version>
    <dbcp.version>1.2.2.osgi</dbcp.version>
    <junit.version>4.6</junit.version>
    <commons-io.version>1.3.2</commons-io.version>
    <urlrewrite.version>3.1.0</urlrewrite.version>
```

```
<spring-js.version>2.0.7.RELEASE</spring-js.version>
<ibatis.version>2.3.4.726</ibatis.version>
<caucho.version>3.2.1</caucho.version>
<axis.version>1.4.0</axis.version>
<wsdl.version>1.6.1</wsdl.version>
<jstl.version>1.2</jstl.version>
<aspectj.version>1.6.5</aspectj.version>
<servlet-api.version>2.5</servlet-api.version>
<jsp.version>2.1</jsp.version>

</properties>
```

After the change:

```
<properties>
    <spring.version>3.0.3.RELEASE</spring.version>
    <tiles.version>2.2.0</tiles.version>
    <hsqldb.version>1.8.0.7</hsqldb.version>
    <commons-fileupload.version>1.2.1</commons-fileupload.version>
    <struts.version>1.2.9</struts.version>
    <xml-rpc.version>1.1.0</xml-rpc.version>
    <dbcp.version>1.2.2.osgi</dbcp.version>
    <junit.version>4.6</junit.version>
    <commons-io.version>1.3.2</commons-io.version>
    <urlrewrite.version>3.1.0</urlrewrite.version>
    <spring-js.version>2.0.7.RELEASE</spring-js.version>
    <ibatis.version>2.3.4.726</ibatis.version>
    <caucho.version>3.2.1</caucho.version>
    <axis.version>1.4.0</axis.version>
    <wsdl.version>1.6.1</wsdl.version>
    <jstl.version>1.2</jstl.version>
    <aspectj.version>1.6.5</aspectj.version>
    <servlet-api.version>2.5</servlet-api.version>
    <jsp.version>2.1</jsp.version>
    <org.slf4j-version>1.5.8</org.slf4j-version>
</properties>
```

# C JDBC Configuration

This appendix describes the consolidated JDBC Type 2 driver configuration and the JDBC Type 4 driver configuration in the `applicationContext.xml` and `jdbc.properties` files.

- JDBC Type 2 driver configurations

The consolidated JDBC Type 2 driver configuration in `applicationContext.xml` is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName">
            <value>${jdbc.driver}</value>
        </property>
        <property name="url">
            <value>${jdbc.url}</value>
        </property>
        <property name="username">
            <value>${jdbc.user}</value>
        </property>
        <property name="password">
            <value>${jdbc.password}</value>
        </property>
    </bean>
    <bean id="propertyConfigurer"
        class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations">
            <list>
                <value>classpath:jdbc.properties</value>
            </list>
        </property>
    </bean>
</beans>
```

The SQL/MX settings for JDBC Type 2 driver in `jdbc.properties` appear as follows:

```
#-----
# SQL/MX Settings for JDBC Type 2 Driver

jdbc.url= jdbc:sqlmx://
jdbc.driver= com.tandem.sqlmx.SQLMXDriver
jdbc.user =
jdbc.password=
```

- JDBC Type 4 driver configurations

The consolidated JDBC Type 4 driver configuration in `applicationContext.xml` is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName">
            <value>${jdbc.driver}</value>
        </property>
        <property name="url">
            <value>${jdbc.url}</value>
        </property>
        <property name="username">
            <value>${jdbc.user}</value>
        </property>
        <property name="password">
            <value>${jdbc.password}</value>
        </property>
    </bean>
    <bean id="propertyConfigurer"
        class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations">
            <list>
                <value>classpath:jdbc.properties</value>
            </list>
        </property>
    </bean>
```

```
</bean>
</beans>
```

The SQL/MX settings for JDBC Type 4 driver in `jdbc.properties` appear as follows:

```
#-----# SQL/MX Settings for JDBC Type 4
Driver
jdbc.driverClassName=com.tandem.t4jdbc.SQLMXDriver
jdbc.url=jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.>
jdbc.username=<HP NonStop Username>
jdbc.password=<HP NonStop Password>
```

# D Installing Spring Web Flow

To use Spring Web Flow for developing your application, you need to install the Spring Web Flow distribution on the NonStop system. Installing the Spring Web Flow libraries on a NonStop system requires the following actions:

- “[Downloading Spring Web Flow Distribution on Windows](#)” (page 146)
- “[Copying Spring Web Flow Runtime Libraries from Windows to NonStop](#)” (page 146)

## Downloading Spring Web Flow Distribution on Windows

To download the Spring Web Flow distribution on the Windows system, complete the following steps:

1. Go to <http://www.springsource.com/products/spring-community-download>.  
The Spring Community Downloads web page appears.
2. Do one of the following:
  1. Register yourself.
  2. Click **I'd rather not fill in the form. Just take me to the download page.**The Spring Web Flow distributions available for download are displayed.
3. Download the `spring-webflow-2.0.7.RELEASE-with-dependencies.zip` file.
4. Extract the `spring-webflow-2.0.7.RELEASE-with-dependencies.zip` file to a directory on the Windows system.

### NOTE:

- Extracting this zip file creates the `spring-webflow-2.0.7.RELEASE` folder on the Windows system.
- The absolute path of the `spring-webflow-2.0.7.RELEASE` folder on the Windows system will be referred to as `<Spring Web Flow Home>`.

## Copying Spring Web Flow Runtime Libraries from Windows to NonStop

To copy the Spring Web Flow runtime libraries from the Windows system to the NonStop system, complete the following steps:

1. Go to `<Spring Web Flow Home>` and create JAR files of the `<Spring Web Flow Home>\dist` directory on the Windows system using these commands:

```
command prompt> cd <Spring Web Flow Home>
command prompt> jar -cvf springwebflow_dist.jar dist
```

For example:

```
command prompt> cd C:\spring-webflow-2.0.7.RELEASE
command prompt> jar -cvf springwebflow_dist.jar dist
```
2. Create a version-specific Spring Web Flow directory `<NonStop Spring Web Flow Home>` in the OSS environment on the NonStop system using the command:

```
OSS> mkdir -p <NonStop Spring Web Flow Home>
```

For example, create a directory structure `/usr/tandem/sash/spring-webflow-2.0.7:`

```
OSS> mkdir -p /usr/tandem/sash/spring-webflow-2.0.7
```
3. Copy the `springwebflow_dist.jar` file from `<Spring Web Flow Home>` to `<NonStop Spring Web Flow Home>` and extract it using these OSS commands:

```
OSS> cd <NonStop Spring web Flow Home>
OSS> jar -xvf springwebflow_dist.jar
```

After extraction, <*NonStop Spring Web Flow Home*> must have a sub-directory:  
<*NonStop Spring Web Flow Home*>/dist.

This completes the installation of the Spring Web Flow runtime libraries on a NonStop system. You can use these libraries when developing and running Spring Web Flow applications on a NonStop system, using the steps similar to those for the Spring framework.

---

## Part II Hibernate Framework

Part II includes the following chapters:

- [“Hibernate Overview” \(page 151\)](#)

This chapter provides an overview of the following:

- Hibernate projects certified for use on NonStop
- Message flow in an application that uses Hibernate on NonStop

- [“Installing the Hibernate Framework” \(page 153\)](#)

This chapter helps you to install Hibernate framework libraries, and enables you to deploy and run sample Hibernate applications on your NonStop system.

- [“Configuring Hibernate Applications on NonStop Systems” \(page 170\)](#)

This chapter provides information about configuring applications that use Hibernate on NonStop systems.

- [“Getting Started with Hibernate” \(page 196\)](#)

This chapter helps you to develop a Java application on your Windows system using Hibernate, and set up and run the Java application on your NonStop system.

- [“Customizing Sample Applications” \(page 221\)](#)

This appendix details the customizations made in the Hibernate sample applications: Caveat Emptor and EventManager.

- [“JDBC Configuration” \(page 227\)](#)

This appendix describes the consolidated JDBC Type 2 driver configuration and the JDBC Type 4 driver configuration in the `hibernate.cfg.xml` and `hibernate.properties` files.

- [“Hibernate Environment Setup Script” \(page 228\)](#)

This appendix provides the contents of the `ei_setenv` script file.

# Contents

<b>6 Hibernate Overview.....</b>	<b>151</b>
Hibernate Projects.....	151
Hibernate Annotations.....	151
Hibernate EntityManager.....	151
Hibernate Applications on NonStop.....	152
<b>7 Installing the Hibernate Framework.....</b>	<b>153</b>
Prerequisites.....	153
NonStop system.....	153
Windows system.....	153
Installing Hibernate Framework Libraries on NonStop.....	153
Downloading the Hibernate Distribution on Windows.....	154
Downloading Hibernate Dependency JAR Files using Maven.....	155
Including the Hibernate Dialect for the SQL/MX database to the Hibernate Distribution.....	157
Copying the Hibernate Distribution from Windows to NonStop.....	157
Deploying and Running Sample Hibernate Applications on NonStop.....	158
Caveat Emptor.....	159
The EventManager Web Application .....	163
<b>8 Configuring Hibernate Applications on NonStop Systems.....</b>	<b>170</b>
NonStop Platform Configurations.....	170
Determining the Application Parameters.....	170
Determining the Maximum Capacity of NSJSP Instance.....	170
Configuring iTP WebServer for Hibernate Applications.....	171
Configuring NSJSP for Hibernate Applications.....	175
Hibernate Framework Configurations for NonStop Systems.....	180
JDBC Driver for SQL/MX Database.....	181
Database Transaction Management.....	188
Connection Pooling.....	189
Module File Caching Configurations.....	190
Configuring NonStop SQL/MX DataSource for MFC.....	190
Modifying the Hibernate Application.....	191
Enabling Browse Access.....	191
Hibernate Applications.....	191
Spring and Hibernate Applications.....	192
<b>9 Getting Started with Hibernate.....</b>	<b>196</b>
Prerequisites.....	196
NonStop System.....	196
Windows System.....	196
Overview of EmployeeInfo.....	196
Developing EmployeeInfo on Windows using the Eclipse Galileo IDE.....	197
Setting Up the NonStop Environment.....	218
Running EmployeeInfo on NonStop.....	220
<b>E Customizing Sample Applications.....</b>	<b>221</b>
Customizing Caveat Emptor.....	221
Added Files.....	221
Modified Files.....	223
Customizing EventManager.....	223
Added Directories.....	223
Added Files.....	224
Modified File.....	225

F JDBC Configuration.....	227
G Hibernate Environment Setup Script.....	228

# 6 Hibernate Overview

Hibernate is an object-relational mapping (ORM) tool for Java environments that provides a framework for mapping a data representation from an object model to a relational data model with an SQL-based schema.

Hibernate provides configuration files to map Java classes with database tables, and to configure database connectivity properties, such as the database URL, database credentials, connection pooling configuration parameters, and so on. It also provides data query and retrieval facilities.

The NonStop system provides a platform comprising of JVM and a servlet container to run applications that use Hibernate.

---

**NOTE:** Any mention of Hibernate in this document implies association with Hibernate version 3.5.1. For information on Hibernate, see <http://www.hibernate.org>.

This chapter provides an overview of the following:

- Hibernate projects certified for use on NonStop
- Hibernate Applications on NonStop

## Hibernate Projects

In addition to the Hibernate core framework, the JBoss community maintains other projects, such as Hibernate EntityManager, Hibernate Annotations, Hibernate Shards, and so on. Among these projects, only Hibernate EntityManager and Hibernate Annotations are presently certified for use on the NonStop platform.

## Hibernate Annotations

In Hibernate, the application metadata is commonly provided to the Java Virtual Machine (JVM) through XML files. You can also use Annotations to provide the application metadata.

Annotations enable you to provide the metadata for the Object and Relational Table mapping by including the metadata information in the Plain Old Java Objects (POJO) file along with the code. This helps you to understand the table structure and POJO simultaneously during application development. This also reduces the complexity of the application because you need not maintain separate files for the metadata.

To leverage the above-mentioned features of Annotations in Hibernate, the Hibernate community provides the Hibernate Annotations package.

The Hibernate Annotations package includes:

- Standardized Java Persistence and EJB 3.0 (JSR 220) object/relational mapping annotations
- Hibernate-specific extension annotations for performance optimization and special mappings

---

**NOTE:** The Annotation feature is supported in Java 5.0 and later versions.

## Hibernate EntityManager

Java EE provides EJB3 specifications to standardize the basic APIs and the metadata required for any object/relational persistence mechanism. The Hibernate community provides the Hibernate EntityManager project, which implements the programming interfaces and lifecycle rules as defined by the EJB3 persistence specifications.

The Hibernate EntityManager implements the following:

- Standard Java Persistence management API
- Standard Java Persistence Query Language

- Standard Java Persistence object lifecycle rules
- Standard Java Persistence configuration and packaging

You can implement a complete EJB3 persistence solution using Hibernate Annotations and the Hibernate EntityManager with the Hibernate package.

---

**NOTE:** The EJB3 specification is supported on Java 5.0 and above versions.

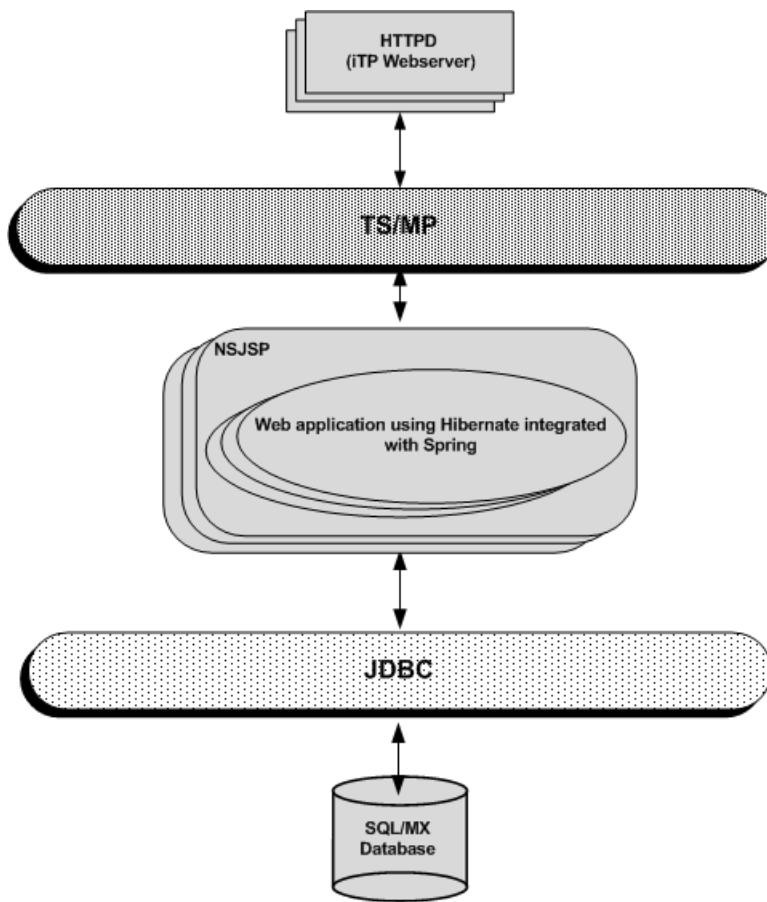
---

## Hibernate Applications on NonStop

Hibernate can be used in a standalone Java application or in a Java web application using servlets or EJB session beans.

This section describes the message flow when Hibernate is used in a Java web application that uses the NonStop SQL/MX database and is hosted on NSJSP.

**Figure 47** Hibernate Application on NonStop



A Hibernate message flows through all the components in the following sequence:

1. An HTTP request passes through the iTP WebServer and then to the NSJSP web container.
2. The Servlet application, which is deployed under the NSJSP web container, processes the HTTP request and passes it to the Application layer.
3. The Application layer passes the request to the persistence layer (that is, Hibernate). Hibernate establishes connection with the database, performs database operations (such as create, update, retrieve, and delete), and then returns persistent object to the application.

---

**NOTE:** Persistent objects are the Java classes, which are mapped with database tables.

---

The subsequent chapters discuss installing and configuring Hibernate to customize it for your web application development requirements.

# 7 Installing the Hibernate Framework

This chapter describes the procedure to install Hibernate framework libraries on a Nonstop system. It also describes the steps required to deploy and run sample Hibernate applications, which help you to verify if you have successfully installed Hibernate on the NonStop system.

The following tasks are described in this chapter:

1. “[Installing Hibernate Framework Libraries on NonStop](#)” (page 153)
2. “[Deploying and Running Sample Hibernate Applications on NonStop](#)” (page 158)

## Prerequisites

Before getting started, make sure that you have the following software installed on the NonStop and Windows system:

### NonStop system

The following software must be installed on the NonStop system:

- JDBC Type 2 driver version T1275H50 or later, or JDBC Type 4 driver version T1249V11 or later
- NonStop SQL/MX version T1050H23 or later
- NSJ version T2766H50 or later

### Windows system

The following software must be installed on the Windows system:

- JDK version 1.5
- JDBC Type 4 driver version T1249V11 or later
- Maven version 2.2.1

---

**NOTE:** For more information about installing the software required on NonStop and Windows system, see “[Prerequisites](#)” (page 18).

---

## Installing Hibernate Framework Libraries on NonStop

Installing Hibernate framework libraries on a NonStop system requires the following actions:

1. “[Downloading the Hibernate Distribution on Windows](#)” (page 154)
2. “[Downloading Hibernate Dependency JAR Files using Maven](#)” (page 155)
3. “[Including the Hibernate Dialect for the SQL/MX database to the Hibernate Distribution](#)” (page 157)
4. “[Copying the Hibernate Distribution from Windows to NonStop](#)” (page 157)

---

**NOTE:**

- Before you start installing the Hibernate framework, extract the contents of the SAMPLES.zip file (containing OpenSource Java Frameworks for NonStop) to a directory on the Windows system.
  - Throughout the chapter, references are made to the following directories:
    - *<Hibernate Home>*: The directory on the Windows system where the Hibernate distribution files are extracted.
    - *<NonStop Hibernate Home>*: The OSS directory on the NonStop system where the Hibernate runtime JAR files are located.
    - *<My SASH Home>*: The directory on the Windows system where the contents of the SAMPLES.zip file (distributed as a part of the NS Samples for Java Frameworks - T0874 and available for download in Scout for NonStop servers) is extracted.
- 

## Downloading the Hibernate Distribution on Windows

To download the Hibernate distribution on the Windows system, complete the following steps:

1. Go to <http://www.hibernate.org/downloads.html>.
  2. Click **The release bundles have been uploaded to SourceForge** on the Hibernate download web page.
  3. Click **View all files**.  
A web page from sourceforge.net , showing various Hibernate packages that can be downloaded, appears.
  4. Click **hibernate3**, and then click **3.5.1-Final**.  
The links to the following files appear:
    - hibernate-distribution-3.5.1-Final-dist.zip
    - hibernate-distribution-3.5.1-Final-dist.tar.gz
  5. Download the hibernate-distribution-3.5.1-Final-dist.zip file.
- 

**NOTE:**

- The hibernate-distribution-3.5.1-Final-dist.zip and hibernate-distribution-3.5.1-Final-dist.tar.gz files include the Hibernate sample application, the Hibernate framework libraries, Hibernate Annotations, and Hibernate Entity Manager sub-projects. They do not include third party libraries that are required to build the sample application.
  - The difference between hibernate-distribution-3.5.1-Final-dist.zip and hibernate-distribution-3.5.1-Final-dist.tar.gz files is in their compression format.
  - The sample applications, namely Caveat Emptor and Event Manager web application, discussed in this section have been verified using Hibernate version 3.5.1 only.
-

6. Extract the hibernate-distribution-3.5.1-Final-dist.zip file into a directory on the Windows system.

This directory will be referred as *<Hibernate Home>*. Among other directories and files, *<Hibernate Home>* includes the following sub-directories:

\project

includes the Hibernate source code.

\project\annotations

includes the source code of Hibernate Annotations.

\project\entitymanager

includes the source code of Hibernate EntityManager.

\lib

includes libraries required to build Hibernate applications.

\documentation

includes the Hibernate reference documentation in PDF and HTML format.

## Downloading Hibernate Dependency JAR Files using Maven

Because Hibernate 3.5.1 dependency JARs are placed in the Maven repository, you need to download the dependency JAR files using Maven.

To download the Hibernate dependency JAR files using Maven, complete the following steps:

1. Go to the *<Hibernate Home>\projects* directory on the Windows system.
2. Run the following command to build the Hibernate framework and create the Hibernate dependency JARs:

```
"command prompt> set MAVEN_OPTS=-Xmx512m  
command prompt> mvn -fae install"
```

The Hibernate framework, Hibernate sub-projects, and dependency JAR files are downloaded to C:\Documents and Settings\<User name>\.m2\repository where,

<User name> is the name of the user account being used on the Windows system.

---

**NOTE:**

- The C:\Documents and Settings\<User name>\.m2\repository location will be referred as <*Hibernate Repository Home*>.
- You can run Hibernate 3.5.1 with JDK 1.6. However, Hibernate 3.5.1 distribution cannot be complied with JDK 1.6 . Therefore, you need to compile Hibernate 3.5.1 with JDK1.5.
- Update <MAVEN\_HOME>\conf\settings.xml with the section below.

```
<?xml version="1.0"?>
<settings>
  <pluginGroups>
    <pluginGroup>org.jboss.maven.plugins</pluginGroup>
  </pluginGroups>
  <proxies>
    <!-- proxy
        | Specification for one proxy, to be used in connecting to the network.
        | -->
    <proxy>
      <id>proxyId</id>
      <active>true</active>
      <protocol>https</protocol>
      <username>proxyuser</username>
      <password>proxypassword</password>
      <host>proxy.somewhere.com</host>
      <port>8080</port>
      <nonProxyHosts>www.google.com|*.somewhere.com</nonProxyHosts>
    </proxy>
  </proxies>
  <profiles>
    <profile>
      <id>repos</id>
      <properties>
        <!-- Here we point to our local JDK 1.6 home -->
        <jdk16_home>C:\Program Files\Java\jdk1.6.0_21\</jdk16_home>
      </properties>
        <!-- Here we define the JBoss release and snapshot repos -->
      <repositories>
        <repository>
          <id>jboss</id>

          <url>https://repository.jboss.org/nexus/content/groups/public</url>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </repository>
      </repositories>
        <!-- Here we define the JBoss release and snapshot repos *for plugins* -->
      <pluginRepositories>
```

```

<pluginRepository>
  <id>jboss-plugins</id>

  <url>https://repository.jboss.org/nexus/content/groups/public</url>
    <releases>
      <enabled>true</enabled>
      <updatePolicy>never</updatePolicy>
    </releases>
    <snapshots>
      <enabled>true</enabled>
      <updatePolicy>never</updatePolicy>
    </snapshots>
  </pluginRepository>
</pluginRepositories>
</profile>
</profiles>
<!-- Here we add our profile defined in settings.xml to the active profiles --&gt;
&lt;activeProfiles&gt;
  &lt;activeProfile&gt;repos&lt;/activeProfile&gt;
&lt;/activeProfiles&gt;
&lt;/settings&gt;
</pre>


---



```

## Including the Hibernate Dialect for the SQL/MX database to the Hibernate Distribution

The Hibernate dialect for the SQL/MX database is required for running any Hibernate application on a NonStop system.

The DIALECT file (which includes hibernate3sqlmx.jar) is distributed as a part of the SQL/MX Hibernate Dialect - T0873 and can be downloaded from Scout for NonStop servers.

To obtain the hibernate3sqlmx.jar file, complete the following steps:

1. Download the DIALECT file from Scout for NonStop servers.
2. Add the .zip extension to it.
3. Extract the DIALECT.zip file to a location on the Windows system.  
The SQL/MX-Hibernate-Dialect folder containing hibernate3sqlmx.jar appears.
4. Place hibernate3sqlmx.jar in <Hibernate Home>\lib.

**NOTE:** If you have already extracted the SAMPLES.zip file from Scout for NonStop servers, you can find the hibernate3sqlmx.jar file at <My SASH Home>\hibernate\sqlmxdialect.

You can now transfer the Hibernate distribution to the NonStop system.

**NOTE:** hibernate3sqlmx.jar is the Hibernate dialect file for SQL/MX database that maps HQL to SQL/MX specific queries.

## Copying the Hibernate Distribution from Windows to NonStop

To copy Hibernate distribution from a Windows system to a NonStop system, complete the following steps:

1. Go to <Hibernate Home> and create a JAR file of the <Hibernate Home>\lib directory on the Windows system:

```

command prompt> cd <Hibernate Home>
command prompt> jar -cvf hibernate_lib.jar lib

```

For example:

```
command prompt> cd C:\hibernate-distribution-3.5.1-Final  
command prompt> jar -cvf hibernate_lib.jar lib
```

2. Create a Hibernate directory <*NonStop Hibernate Home*> in the OSS environment on the NonStop system:

```
OSS> mkdir -p <NonStop Hibernate Home>
```

For example, create a directory structure /usr/tandem/sash/  
hibernate-distribution-3.5.1-Final as follows:

```
OSS> mkdir -p /usr/tandem/sash/hibernate-distribution-3.5.1-Final
```

3. Copy the hibernate\_lib.jar and hibernate3.jar files from <*Hibernate Home*> to <*NonStop Hibernate Home*> and extract hibernate\_lib.jar:

```
OSS> cd <NonStop Hibernate Home>
```

```
OSS> jar -xvf hibernate_lib.jar
```

After extraction, <*NonStop Hibernate Home*> must include the <*NonStop Hibernate Home*>/lib sub-directory.

---

**NOTE:** The contents of the <*NonStop Hibernate Home*>/lib directory on the NonStop system must be exactly same as the <*Hibernate Home*>\lib directory on the Windows system.

---

4. Go to C:\Documents and Settings\<*User name*> and create a JAR file of the <*User name*>\.m2 directory on the Windows system:

```
command prompt> cd C:\Documents and Settings\<User name>  
command prompt> jar -cvf hibernate_Dependency.jar .m2
```

For example:

```
command prompt> cd C:\Documents and Settings\hpadmin  
command prompt> jar -cvf hibernate_Dependency.jar .m2
```

5. Transfer the hibernate\_Dependency.jar files from C:\Documents and Settings\<*User name*> to the OSS root directory.

6. Extract hibernate\_Dependency.jar:

```
OSS> cd <OSS root directory>  
OSS> jar -xvf hibernate_Dependency.jar
```

The Hibernate framework libraries are installed on the NonStop system. You can use these libraries to develop and run Hibernate applications on a NonStop system.

## Deploying and Running Sample Hibernate Applications on NonStop

The sample applications (Caveat Emptor and the Event Manager Web Application) are included in the SAMPLES.zip file in the Hibernate distribution, and require modifications to make them compatible with NonStop systems.

---

**NOTE:** The SAMPLES.zip file is distributed as a part of the NS Samples for Java Frameworks - T0874.

SAMPLES.zip is present in the T0874AAB.BIN file in Scout for NonStop Servers. For information on how to install the T0874AAB.BIN file from Scout, see [https://h20453.www2.hp.com/scout/download\\_help.htm](https://h20453.www2.hp.com/scout/download_help.htm).

Before you deploy the sample applications, complete the following steps:

1. Download the SAMPLES.zip file from Scout for NonStop servers.

2. Add the .zip extension to it.

The file is renamed as SAMPLES.zip.

3. Extract the SAMPLES.zip file to a location on the Windows system.

The NS-Samples-for-Java-Frameworks folder appears.

---

**NOTE:** The absolute path of the NS-Samples-for-Java-Frameworks folder is referred as <My SASH Home>.

---

This section describes the steps required to build, set up, deploy, and run the Caveat Emptor and Event Manager sample applications on NonStop systems.

The modifications to the sample applications are discussed in “[Customizing Sample Applications](#)” (page 221).

## Caveat Emptor

The Caveat Emptor sample application is an auction system. The intended users of the application are bidders and the sellers of various items.

This section describes the following steps for the Caveat Emptor sample application.

- “[Building Caveat Emptor on Windows](#)” (page 159)
- “[Setting up the Deployment Environment on NonStop](#)” (page 161)
- “[Deploying Caveat Emptor on NonStop](#)” (page 163)
- “[Running Caveat Emptor on NonStop](#)” (page 163)

### Building Caveat Emptor on Windows

To build Caveat Emptor on the Windows system, complete the following steps:

1. Go to the <My SASH Home>\hibernate\samples\eg directory on the Windows system. Among the other files, this directory must include the following sub-directories:

\setup

includes the environment setup file required to run Caveat Emptor.

\dbconfig

includes the catalog, schema, and table creation script.

\src

includes the Java source files and mapping files of Caveat Emptor.

2. Configure the JDBC driver settings for the NonStop SQL/MX database.

- a. Go to the <My SASH HOME>\hibernate\samples\eg\src\main\resources\ directory on the Windows system.
- b. Modify the hibernate.properties file to update the JDBC configuration. You can use either the JDBC Type 2 driver or the JDBC Type 4 driver. The SQL/MX settings for each of them are:
  - To use the JDBC Type 2 driver, uncomment the SQL/MX settings for the JDBC Type 2 driver in the hibernate.properties file so that the SQL/MX settings for JDBC Type 2 driver appear as:

```
#-----
# SQL/MX Settings for JDBC Type 2 Driver
hibernate.dialect org.hibernate.dialect.SqlmxDialect
hibernate.connection.driver_class com.tandem.sqlmx.SQLMXDriver
hibernate.connection.url jdbc:sqlmx://
hibernate.connection.username
hibernate.connection.password
hibernate.connection.catalog auctioncat
hibernate.connection.schema auctionsch
```

---

**NOTE:** Because JDBC Type 2 driver is located on the NonStop system, you need not mention the username and password in the hibernate.connection.username and hibernate.connection.password fields.

---

- To use the JDBC Type 4 driver, uncomment the SQL/MX settings for the JDBC Type 4 driver in the hibernate.properties file, and type the values of the JDBC URL (NonStop system IP Address and port number of the JDBC data source), NonStop system username, and password.

The SQL/MX settings for JDBC Type 4 driver in hibernate.properties appear as:

```
#-----
# SQL/MX Settings for JDBC Type 4 Driver
hibernate.dialect=org.hibernate.dialect.SqlmxDialect
hibernate.connection.driver_class com.tandem.t4jdbc.SQLMXDriver
hibernate.connection.url jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.>
hibernate.connection.username <HP NonStop Username>
hibernate.connection.password <HP NonStop Password>
hibernate.connection.catalog auctioncat
hibernate.connection.schema auctionsch
```

---

**NOTE:** The hibernate.properties file, located in <My SASH HOME>\hibernate\samples\eg\src\main\resources, has the database catalog name set to auctioncat and schema name set to auctionsch. If the database catalog name auctioncat and schema name auctionsch already exists, or if you want to use a different database catalog and schema name, modify the following:

- values for the hibernate.connection.catalog and hibernate.connection.schema property in the hibernate.properties file in <My SASH HOME>\hibernate\project\tutorials\eg\src\main\resources.
  - database catalog and schema names in the caveatemperor\_script.sql script file, as discussed in “[Setting up the Deployment Environment on NonStop](#) (page 161).
- 

### 3. Build Caveat Emptor (Caveatemperor.jar).

- a. Go to the <My SASH Home>\hibernate\samples\eg\ directory.
- b. Use Maven to build the Caveat Emptor JAR file.  
If you want to use the JDBC Type 2 driver, build the Caveat Emptor JAR using the command:  
command prompt> mvn clean package

After successful build of Caveat Emptor, a new directory named target is created in the <My SASH Home>\hibernate\samples\eg directory. The application JAR file (hibernate-tutorial-eg-3.5.0-CR-2.jar) is created in the target directory.

Caveat Emptor is built on the Windows system.

## Setting up the Deployment Environment on NonStop

Setting up the deployment environment on a NonStop system comprises the following actions:

1. “Setting up SQL/MX database on NonStop” (page 161)
2. “Setting up the Hibernate and JDBC environment on NonStop” (page 162)

### Setting up SQL/MX database on NonStop

To set up the SQL/MX database for Caveat Emptor on a NonStop system, complete the following steps:

1. Edit the <My SASH Home>\hibernate\samples\eg\dbconfig\caveatemptor\_script.sql file as follows:
  - <\$datavol>: specify the Guardian volume for the primary partition of the table. For example: \$data01.
  - <user>: specify the owner of the schema. For example: super.sashusr.
  - <node.\$vol>: specify the location of the metadata tables for the catalog. For example: \NSK01.\$data01.
  - <subvol reference>: specify the designated subvolume name for the schema. For example: SASH1.

---

**NOTE:** The subvolume name is always prefixed by ZSD and must have eight characters (including ZSD).

---

2. Create a directory in OSS to place the database script files:

OSS> mkdir -p <NonStop SASH Home>/hibernate/samples/eg/dbconfig

For example:

OSS> mkdir -p /home/sash\_usr/sash/hibernate/samples/eg/dbconfig

---

**NOTE:** <NonStop SASH Home> can be any working directory. It is suggested that you create the hibernate/samples/eg/dbconfig directory structure in <NonStop SASH Home>.

---

3. Copy the caveatemptor\_script.sql script from the <My SASH Home>\hibernate\samples\eg\dbconfig Windows directory to the <NonStop SASH Home>/hibernate/samples/eg/dbconfig OSS directory.
4. Go to the OSS directory where the caveatemptor\_script.sql script is copied:
 

OSS> cd <NonStop SASH Home>/hibernate/samples/eg/dbconfig

For example:

OSS> cd /home/sash\_usr/sash/hibernate/samples/eg/dbconfig
5. Create the Caveat Emptor database catalog, schema, and tables:

```
mxci>> obey caveatemptor_script.sql;
```

**NOTE:** By default, the `caveatemptor_script.sql` script file creates the database catalog `auctionccat` and schema `auctions`. If the database catalog name `auctionccat` and schema name `auctions` already exist, or if you want to use a different database catalog and schema name, modify the following:

- database catalog and schema names in the `caveatemptor_script.sql` script file.
- database catalog and schema names in the `hibernate.properties` file in the `<My SASH Home>\hibernate\samples\eg` directory of the `SAMPLES.zip` file.

## Setting up the Hibernate and JDBC environment on NonStop

To set the CLASSPATHs for the Hibernate and JDBC driver environment, complete the following steps:

1. Modify the `<My SASH Home>\hibernate\samples\eg\setup\setenv` script file:
  - a. Set the value of the `hibernatehome` variable to the `<NonStop Hibernate Home> OSS` directory.

For example, if `<NonStop Hibernate Home>` is `/usr/tandem/sash/hibernate-distribution-3.5.1-Final`, the `hibernatehome` variable must be set to `hibernatehome=/usr/tandem/sash/hibernate-distribution-3.5.1-Final`.
  - b. Set the value of the `mavenrepository` variable to the `<NonStop Hibernate Maven Repository> OSS` directory.

For example, if `<NonStop Hibernate Maven Repository>` is `/usr/tandem/sash/.m2`, the `mavenrepository` variable must be set to `mavenrepository =/usr/tandem/sash/.m2`.
  - c. Set the value of the JDBC driver locations.
    - To use the JDBC Type 2 driver, set the value of the `t2jdbc` variable to the `<JDBC T2 Installation Directory> OSS` directory.

For example, if `<JDBC T2 Installation Directory>` is `/usr/tandem/jdbcMx/current`, the `t2jdbc` variable must be set to `t2jdbc=/usr/tandem/jdbcMx/current`
    - To use the JDBC Type 4 driver, set the value of the `t4jdbc` variable to the `<NonStop SASH Home> OSS` directory.

For example, if `<NonStop SASH Home>` is `/home/sash_usr/sash`, the `t4jdbc` variable must be set to `t4jdbc=/home/sash_usr/sash`
2. Create a directory in OSS to place the script files:

```
OSS> mkdir -p <NonStop SASH Home>/hibernate/samples/eg/setup
```

For example:

```
OSS> mkdir -p /home/sash_usr/sash/hibernate/samples/eg/setup
```
3. Copy the `setenv` script file from `<My SASH Home>\hibernate\samples\eg\setup` to `<NonStop SASH Home>/hibernate/samples/eg/setup`.
4. Set the execute permissions of the `setenv` script file:

**NOTE:** It is suggested that you create `hibernate/samples/eg/setup` directory structure in `<NonStop SASH Home>`.

```
OSS> chmod +x setenv
```

**5. Run the setenv script file:**

- To use the JDBC Type 2 driver, type:

```
OSS> ./setenv T2
```

- To use the JDBC Type 4 driver, type:

```
OSS> ./setenv T4
```

## Deploying Caveat Emptor on NonStop

To deploy Caveat Emptor on a NonStop system, transfer the application JAR file (`caveatemptor.jar`) from the `<My SASH Home>\hibernate\eg\dist` Windows directory to the `<NonStop SASH Home>/hibernate/samples/eg` OSS directory.

## Running Caveat Emptor on NonStop

To run Caveat Emptor, complete the following steps:

**1. Go to `<NonStop SASH Home>/hibernate/samples/eg`:**

```
OSS> cd <NonStop SASH Home>/hibernate/samples/eg
```

For example:

```
OSS> cd /home/sash_usr/sash/hibernate/samples/eg
```

**2. Extract the `caveatemptor.jar` file:**

```
OSS> jar -xvf caveatemptor.jar
```

**3. Run the Caveat Emptor application:**

```
OSS> java org.hibernate.auction.Main
```

When `org.hibernate.auction.Main` is run, Caveat Emptor performs the following actions:

- Populates database tables with sample data
- Displays auction details
- Modifies user details
- Modifies item description
- Displays the auction done by a particular user
- Displays auction by its description

## The EventManager Web Application

The EventManager web application is an event management system. The `EventManagerServlet` lists all events stored in the database, and provides an HTML form to enter new events.

This section describes the following steps for the EventManager sample application.

- “Building EventManger on Windows” (page 163)
- “Setting up the SQL/MX database on NonStop” (page 166)
- “Deploying JPetStore on NonStop” (page 48)
- “Running EventManager on NonStop” (page 169)

## Building EventManger on Windows

To build EventManger on a Windows system, complete the following steps:

1. Go to the `<My SASH Home>\hibernate\samples\web` directory on the Windows system. Among other files, this directory must include the following sub-directories:

```

\dbcconfig
    includes the catalog, schema, and table creation script.

\src
    includes the Java source files and mapping files of EventManger.

\etc
    includes the Hibernate dialect file (hibernate3sqlmx.jar) for the SQL/MX database.

```

2. Configure the JDBC driver settings for the NonStop SQL/MX database.

1. Go to the `<My SASH HOME>\hibernate\samples\web\src\main\resources` directory on the Windows system.
2. Modify the `hibernate.properties` file to update the JDBC configuration. You can use either the JDBC Type 2 driver or the JDBC Type 4 driver. The SQL/MX settings for each of them is as follows:
  - To use the JDBC Type 2 driver, uncomment the SQL/MX settings for JDBC Type 2 driver in the `hibernate.properties` file so that the SQL/MX settings for the JDBC Type 2 driver appear as:

```

-----
# SQL/MX Settings for JDBC Type 2 Driver
hibernate.dialect org.hibernate.dialect.SqlmxDialect
hibernate.connection.driver_class com.tandem.sqlmx.SQLMXDriver
hibernate.connection.url jdbc:sqlmx://
hibernate.connection.username
hibernate.connection.password
hibernate.connection.catalog eventcat
hibernate.connection.schema eventsch

```

---

**NOTE:** Because JDBC Type 2 driver is located on the NonStop system, you need not mention the `username` and `password` in the `hibernate.connection.username` and `hibernate.connection.password` fields.

---

- To use the JDBC Type 4 driver, uncomment the SQL/MX settings for JDBC Type 4 driver in the `hibernate.properties` file and enter the values of JDBC URL (NonStop system IP Address and port number of the JDBC data source), NonStop system `username`, and `password`.

The SQL/MX settings for JDBC Type 4 driver in `hibernate.properties` appear as:

```

-----
# SQL/MX Settings for JDBC Type 4 Driver
hibernate.dialect org.hibernate.dialect.SqlmxDialect
hibernate.connection.driver_class com.tandem.t4jdbc.SQLMXDriver
hibernate.connection.url jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.>
hibernate.connection.username <HP NonStop Username>
hibernate.connection.password <HP NonStop Password>
hibernate.connection.catalog eventcat
hibernate.connection.schema eventsch

```

---

**NOTE:** The `hibernate.properties` file, located in `<My SASH HOME>\hibernate\samples\web\src\main\resources`, has the database catalog name set to `eventcat` and schema name set to `eventsch`. If the database catalog name `eventcat` and schema name `eventsch` already exists, or if you want to use a different database catalog and schema name, modify the:

- values for the `hibernate.connection.catalog` and `hibernate.connection.schema` property in the `hibernate.properties` file in `<My SASH HOME>\hibernate\samples\web\src\main\resources`.
  - database catalog and schema names in the `eventmanager_script.sql` script file, as discussed in “[Setting up the Deployment Environment on NonStop \(page 161\)](#)”.
- 

3. Build the EventManager web application archive WAR file (`hibernate-tutorial-web-3.5.0-CR-2.war`).
  1. Go to the `<My SASH Home>\hibernate\samples\web` directory:  
command prompt> `cd <My SASH Home>\hibernate\samples\web`
  2. Build the Event Manager web archive using JDBC Type 2 driver or JDBC Type 4 driver.
    - To use the JDBC Type 2 driver:
      1. Build the Event Manager web archive:  
command prompt> `mvn clean package`  
The `hibernate-tutorial-web-3.5.0-CR-2.war` file is created in the `<My SASH Home>\hibernate\samples\web\target` directory.
      2. Create the `WEB-INF\lib` folder in the `<My SASH Home>\hibernate\samples\web\target` directory.
      3. Copy the `hibernate3sqlmx.jar` file to the `<My SASH Home>\hibernate\samples\web\target\WEB-INF\lib` directory.
      4. Go to `<My SASH Home>\hibernate\samples\web\target` and run the following commands:  
command prompt> `cd <My SASH Home>\hibernate\samples\web\target`  
command prompt> `jar uf hibernate-tutorial-web-3.5.0-CR-2.war WEB-INF\lib\hibernate3sqlmx.jar`  
The `hibernate3sqlmx.jar` file is copied to `<My SASH Home>\hibernate\samples\web\target\hibernate-tutorial-web-3.5.0-CR-2.war\WEB-INF\lib`.
    - To use the JDBC Type 4 driver:
      1. Build the Event Manager web archive:  
command prompt> `mvn clean package`  
The `hibernate-tutorial-web-3.5.0-CR-2.war` file is created in the `<My SASH Home>\hibernate\samples\web\target` directory.
      2. Create the `WEB-INF\lib` folder in the `<My SASH Home>\hibernate\samples\web\target` directory.
      3. Copy `hibernate3sqlmx.jar` and `t4sqlmx.jar` to the `<My SASH Home>\hibernate\samples\web\target\WEB-INF\lib` directory.

4. Go to <My SASH Home>\hibernate\samples\web\target and run the following commands:

```
command prompt> cd <My SASH Home>\hibernate\samples\web\target  
command prompt> jar uf hibernate-tutorial-web-3.5.0-CR-2.war WEB-INF\  
lib\hibernate3sqlmx.jar WEB-INF\lib\t4sqlmx.jar
```

The hibernate3sqlmx.jar and t4sqlmx.jar files are copied to  
<My SASH Home>\hibernate\samples\web\target\  
hibernate-tutorial-web-3.5.0-CR-2.war\WEB-INF\lib.

## Setting up the SQL/MX database on NonStop

To set up the SQL/MX database for EventManager on a NonStop system, complete the following steps:

1. Edit the <My SASH Home>\hibernate\samples\web\dbconfig\evenmanager\_script.sql:
  - <\$datavol>: specify the Guardian volume for the primary partition of the table. For example: \$data01.
  - <user>: specify the owner of the schema. For example: super.sashusr.
  - <node.\$vol>: specify the location of the metadata tables for the catalog. For example: \NSK01.\$data01.
  - <subvol reference>: specify the designated subvolume name for the schema. For example: SASH1.

---

**NOTE:** The subvolume name is always prefixed by ZSD and must have eight characters (including ZSD).

---

2. Create a directory in OSS to place the database script files:

```
OSS> mkdir -p <NonStop SASH Home>/hibernate/samples/web/dbconfig
```

For example:

```
OSS> mkdir -p /home/sash_usr/sash/hibernate/samples/web/dbconfig
```

---

**NOTE:** <NonStop SASH Home> can be any working directory. It is suggested that you create the hibernate/samples/web/dbconfig directory structure in <NonStop SASH Home>.

---

3. Copy the evenmanager\_script.sql script from the <My SASH Home>\hibernate\samples\web\dbconfig Windows directory to the <NonStop SASH Home>/hibernate/samples/web/dbconfig OSS directory.
4. Go to the OSS directory where the evenmanager\_script.sql script is copied:

```
OSS> cd <NonStop SASH Home>/hibernate/samples/web/dbconfig
```

For example:

```
OSS> cd /home/sash_usr/sash/hibernate/samples/web/dbconfig
```

5. Create the EventManage database catalog, schema, and tables:

```
mxci>> obey evenmanager_script.sql;
```

---

**NOTE:** By default, the evenmanager\_script.sql script file creates the database catalog eventcat and schema eventsch. If the database catalog name eventcat and schema name eventsch already exist, or if you want to use a different database catalog and schema name, modify the following:

- database catalog and schema names in the evenmanager\_script.sql script file.
  - database catalog and schema names in the hibernate.properties file in the <My SASH Home>\hibernate\samples\web\src\main\resources directory of the SAMPLES.zip file.
- 

## Deploying Event Manager on NonStop

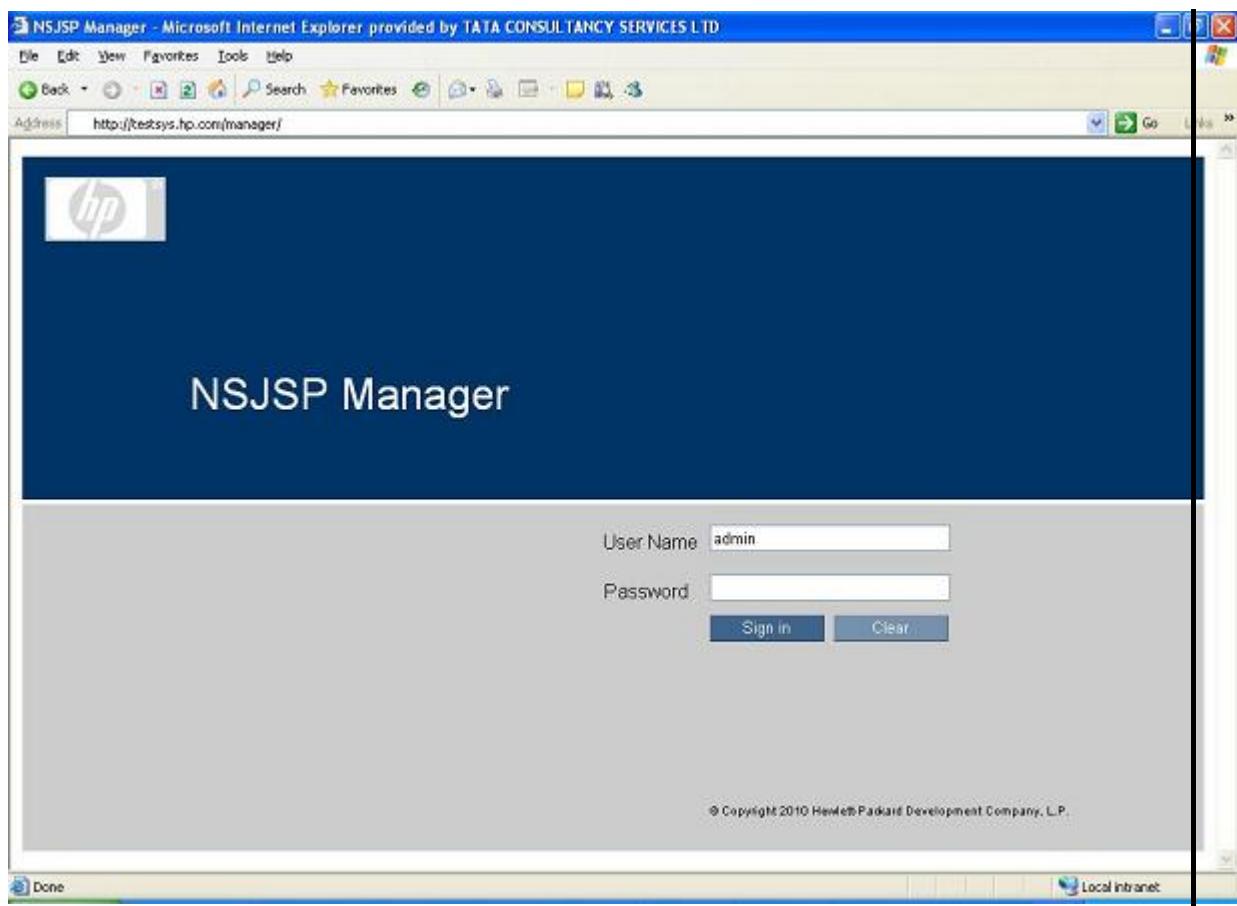
To deploy EventManager on a NonStop system, complete the following steps:

1. Go to [http://<IP Address of the iTP WebServer>:<port#>/<manager\\_directory>](http://<IP Address of the iTP WebServer>:<port#>/<manager_directory>).

The **NSJSP Manager Login** screen appears.

Figure 48 shows the **NSJSP Manager Login** screen.

**Figure 48 NSJSP Manager Login Screen**

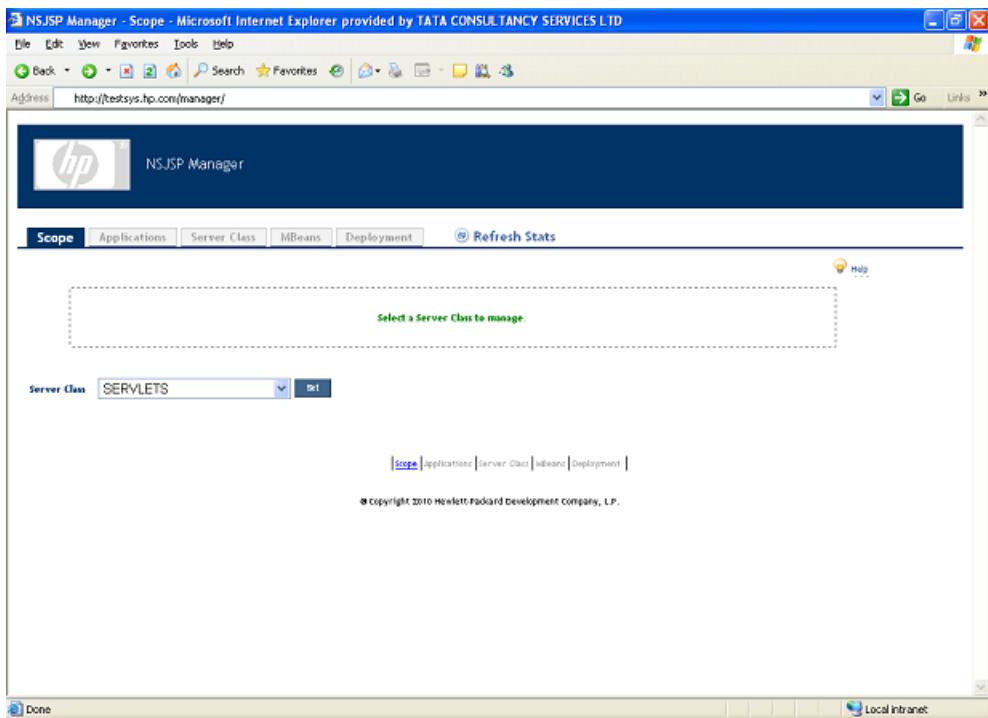


2. Enter the **User name:** and **Password:** and click **OK**.

The **NSJSP Manager** screen appears.

Figure 49 shows the **NSJSP Manager** screen.

**Figure 49 NSJSP Manager Screen**

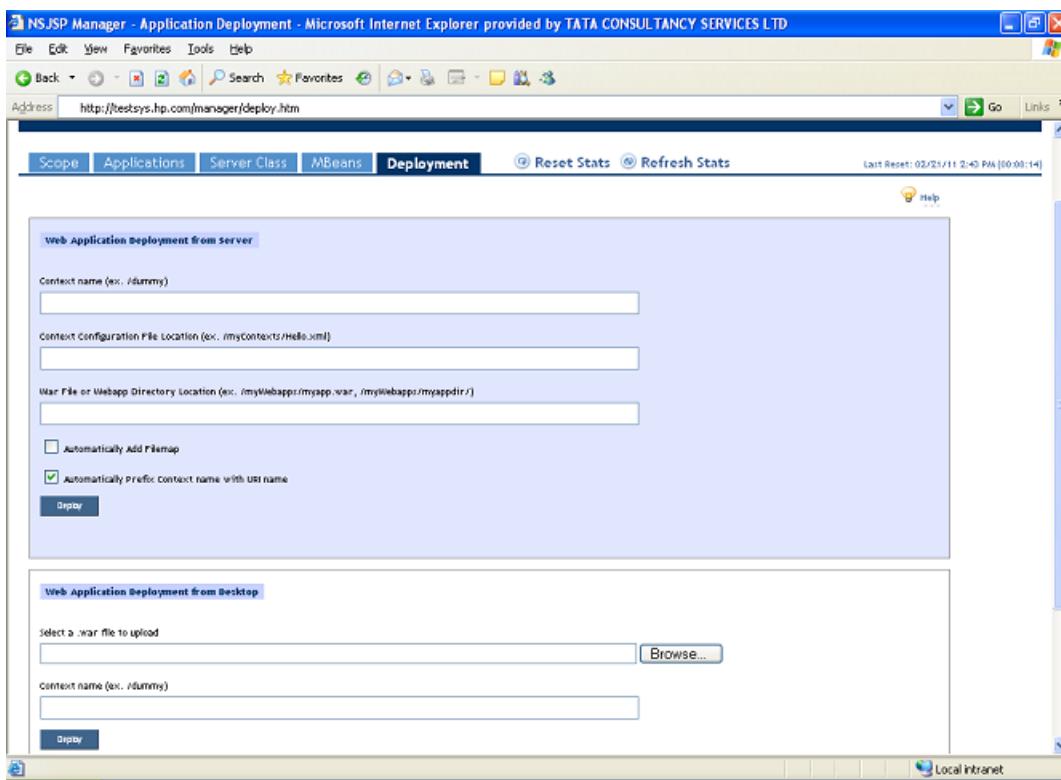


3. Select **SERVLETS** for the **Server Class** and click **Set**.

The **Host (in server.xml)** tab is enabled and populated with **localhost**. Click **Set**, which enables the **Deployment** tab on the **NSJSP Manager** screen.

Figure 50 shows the **Deployment** tab of **NSJSP Manager**.

**Figure 50 NSJSP Manager Screen - Deployment tab**



4. In the **Deployment** tab, complete the following steps in the **Web Application Deployment from Desktop** section:
  1. In the **Select a .war file to upload** field, click **Browse...** and locate the `hibernate-tutorial-web-3.5.0-CR-2.war` file on the Windows system.
  2. **(Optional)** In the **Context name** field, enter a name for the application context.
  3. Click **Deploy**.

EventManager is deployed on NSJSP and is listed under **Applications** tab of the **NSJSP Manager** screen.

---

**NOTE:** HP recommends that you use the NSJSP manager for deployment. Deployment using the FTP or any other mechanism is not recommended. For more information on using the NSJSP manager, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

---

## Running EventManager on NonStop

To run EventManager on the NonStop system, click the `/<servlet directory>/hibernate-tutorial-web-3.5.0-CR-2` path under **Applications** in the **NSJSP Web Application Manager** screen.

You can now add new events such as birthdays, meetings, reminders, and so on.

# 8 Configuring Hibernate Applications on NonStop Systems

This chapter provides information about configuring Hibernate applications on NonStop systems. This chapter includes the following sections:

- “NonStop Platform Configurations” (page 52)
- “Hibernate Framework Configurations for NonStop Systems” (page 180)
- “Module File Caching Configurations” (page 190)
- “Enabling Browse Access” (page 191)

## NonStop Platform Configurations

On a NonStop system, an application developed using the Hibernate framework is deployed as a web application in the NSJSP container. Thus, it inherits all the NSJSP properties and configurations, and runs as any other NSJSP web application running on a NonStop system. This section discusses the following tasks:

- “Determining the Application Parameters” (page 170)
- “Determining the Maximum Capacity of NSJSP Instance” (page 52)
- “Configuring iTP WebServer for Hibernate Applications” (page 171)
- “Configuring NSJSP for Hibernate Applications” (page 175)

### Determining the Application Parameters

You need to determine the following parameters for your application:

- Average Response Time
- Average Load
- Maximum Load

For a unit response time, the number of active requests to iTP WebServer is a maximum of its static capacity during the average load conditions. During peak load conditions, the number of active requests to the iTP WebServer is a maximum of its dynamic capacity.

For example,

If the average response time of your Hibernate application is 1 second and the iTP WebServer is configured to handle 100 requests statically and 50 requests dynamically, then during average load conditions, the number of active requests to the iTP WebServer can be a maximum of 100. During peak load conditions, 50 more requests can be served dynamically and hence, the active requests to iTP WebServer can be a maximum of 150. Further requests are queued up and remain inactive until started by `httpd`.

The other parameters, such as Average Load and Maximum Load, help you decide the configuration of the static and dynamic `httpd` processes of the iTP WebServer.

### Determining the Maximum Capacity of NSJSP Instance

Before configuring the NSJSP parameters, you must determine the capacity of a single instance of NSJSP. To determine the maximum load for a single instance of NSJSP, it is important to first configure the relevant TS/MP and server parameters of NSJSP (of the single instance) to their maximum limit in the following way:

1. Set the value of `Numstatic` to 1. This limits the number of static instances of NSJSP to 1.
2. Set the value of `Maxservers` to 1. This limits the number of NSJSP instances that can be started to 1. This implies that TS/MP cannot start more than one instance of NSJSP.

3. Set the value of Maxlinks to 250. This is the maximum number of links to the server process (NSJSP process). This means that the server process must be capable of processing 250 requests simultaneously.
4. Set the value of TANDEM\_RECEIVE\_DEPTH to 250 (because the values of Maxlinks and TANDEM\_RECEIVE\_DEPTH should be equal). A value of 250 means that the NSJSP process is able to read a maximum of 250 messages simultaneously from its \$RECEIVE file.
5. Configure the Executor element in the *<NSJSP Deployment Directory>/conf/server.xml* file on OSS. The Executor element is used by the Connector element as a thread pool. Set maxThreads = 300, minSpareThreads = 10 and maxIdleTime = 30000. Using an Executor element helps monitor the number of active threads at any given time. A value of 300 for maxThreads ensures that you have enough threads to process all incoming requests (a maximum of 250) simultaneously. A value of 30000 for maxIdleTime ensures that if a thread is idle for more than 30 seconds, that thread will be stopped.

After configuring the parameters, you might want to use a tool that can simulate the HTTP clients and can handle the HTTP cookies. The tool reveals the number of HTTP clients that a single instance of NSJSP can handle and indicates the number of simultaneous HTTP requests that NSJSP is capable of handling.

---

**NOTE:** There are a number of HTTP client simulators available, for example, Apache JMeter, HP LoadRunner, and Radview Webload. These tools provide a good interface to monitor the test results. You can use any of these tools to determine the maximum handling capacity of each NSJSP instance.

To arrive at the required numbers, complete the following steps:

1. Run the test tool with a single instance of the HTTP client simulator.

---

**NOTE:** The test tool must be capable of displaying the response time of the HTTP requests.

2. Monitor the response time of the HTTP requests and allow your application to attain a steady state (in terms of response time per HTTP request).
3. At steady state, check if the response times are within the Service Level Agreement (SLA).
4. If the response time is within the SLA, increase the number of HTTP client instances and repeat step 2 onwards.
5. If the response time is beyond the acceptable SLA, stop the tests and determine the number of HTTP clients and the number of HTTP requests that NSJSP instance can process simultaneously.

While the test tool can indicate the number of HTTP clients that are processed simultaneously, the number of simultaneous HTTP requests can be arrived at using different means. Following are some of the calculation strategies:

- The number of HTTP requests recorded by the testing tool and idea of the number of requests, which required processing by NSJSP, provides the total number of HTTP requests processed by NSJSP.
- The total number of HTTP requests processed by NSJSP together with the test duration indicates the average number of simultaneous HTTP requests handled by NSJSP.

## Configuring iTP WebServer for Hibernate Applications

The `httpd` process of iTP WebServer is responsible for managing the load and forwarding the requests to their corresponding application servers. The context of this section is limited to the configurations related to `httpd` processes only.

Configuring the `httpd` process involves the following tasks:

- “Configuring `httpd` Processes to Handle Maximum Load” (page 172)
- “Limiting the Maximum Number of Incoming Requests” (page 174)

## Configuring `httpd` Processes to Handle Maximum Load

The `httpd` processes are configured using the `Server` configuration directive in the `<iTP WebServer Deployment Directory>/conf/httpd.config` file on OSS. The `Server` directive controls the creation of the PATHMON environment, where the web server runs.

A typical default configuration of the `httpd` process of the iTP WebServer appears as follows:

```
Server $root/bin/httpd {  
    eval $DefaultServerAttributes  
    CWD [pwd]  
    Arglist -server [HTTPD_CONFIG_FILE]  
    Env TANDEM_RECEIVE_DEPTH=50  
    Priority 170  
    Numstatic 5  
    Maxservers 50  
    MapDefine =TCPIP^RESOLVER^NAME /G/system/ztcpip/resconf  
    MapDefine =TCPIP^PROCESS^NAME $transport  
}
```

The number of `httpd` processes are governed by the following TS/MP attributes:

- `Numstatic`: This specifies the number of static servers running under PATHMON. The static processes run on the system, irrespective of the load.
- `Maxservers`: This specifies the maximum number of server processes that can run under PATHMON.
- `TANDEM_RECEIVE_DEPTH`: This specifies the capacity of each of the configured `httpd` processes.
- (`Maxservers` – `Numstatic`): The difference denotes the number of dynamic servers. A dynamic server is a need-based server. When the iTP WebServer is under heavy load and all the static `httpd` processes are busy, a dynamic server process, `httpd`, is created to support the excess load. These servers are dynamically created by TS/MP and are terminated once the process is complete.

The capacity of the iTP WebServer environment can be summarized as:

- The static capacity of iTP WebServer is [`Numstatic` X `TANDEM_RECEIVE_DEPTH`].
- The dynamic capacity is [(`Maxservers` – `Numstatic`) X `TANDEM_RECEIVE_DEPTH`] requests.
- The total capacity of iTP WebServer is [`Maxservers` X `TANDEM_RECEIVE_DEPTH`].

Example 1:

Assume that the `httpd` process of the iTP WebServer has the following configurations:

```
Server $root/bin/httpd {  
    eval $DefaultServerAttributes  
    CWD [pwd]  
    Arglist -server [HTTPD_CONFIG_FILE]  
    Env TANDEM_RECEIVE_DEPTH=100  
    Priority 170  
    Numstatic 5  
    Maxservers 50  
    MapDefine =TCPIP^RESOLVER^NAME /G/system/ztcpip/resconf  
    MapDefine =TCPIP^PROCESS^NAME $transport  
}
```

When you start the iTP WebServer, five static `httpd` processes are started, governed by the value of `Numstatic`. Because the value of `TANDEM_RECEIVE_DEPTH` is set to 100, each

of the five static processes can handle 100 requests. In this example, the capacity of the iTP WebServer environment can be summarized as follows:

- The static capacity of iTP WebServer is [Numstatic X TANDEM\_RECEIVE\_DEPTH]= 500.
- The dynamic capacity is [(Maxservers - Numstatic) X TANDEM\_RECEIVE\_DEPTH] = 4500.
- The total capacity of iTP WebServer is [Maxservers X TANDEM\_RECEIVE\_DEPTH] = 5000.

Using this configuration, the iTP WebServer can handle 500 simultaneous requests statically. When it receives the 501st request, a dynamic httpd process is created, which functions as a normal httpd process. The capacity of each of the dynamic httpd processes is the value of TANDEM\_RECEIVE\_DEPTH. When the load decreases, the dynamic httpd processes goes down.

For more information on the Server configuration directive of the iTP WebServer, see the *iTP Secure WebServer System Administrator's Guide*.

### Guidelines for Configuring the Server Directive

Before configuring the various attributes of the Server directive, you must determine the following parameters:

- Processors for httpd Processes
- Application parameters: Average Load, Peak Load, and Average Response Time

### Processors for httpd Processes

When you start the iTP WebServer, the static httpd processes equal to the value of Numstatics are started on your NonStop system. By default, all the configured processors are taken into account. If the value of Numstatic is set to n, and the number of configured processors on the system are equal to n, one static httpd process is started on each of the processors. However, if the value of Numstatic is less than the number of configured processors, one static httpd process starts on some of the processors, while the remaining processors will not have any static httpd process.

You can determine the processors on which you intend to start httpd processes and configure the processors directive. The following example will help you understand this better:

Example:

Assume that you have a 16-processor system and you want to start httpd processes on processors 0, 1, 2, 3, 4, 5. For this, the processors configuration attribute must be configured as processors 0 1 2 3 4 5.

In this scenario:

- If you set the value of Numstatic equal to 6, each of the processors 0, 1, 2, 3, 4, 5 will have one static httpd process.
- If you set the value of Numstatic equal to a multiple of 6, each of the processors 0, 1, 2, 3, 4, 5 will have (Numstatic/6) static httpd processes.
- If you set the value of Numstatic equal to 5, any 5 of the processors 0, 1, 2, 3, 4, 5 will have one static httpd process, and remaining processors will not have any httpd process.
- If you set the value of Numstatic equal to 9, any three of the processors 0, 1, 2, 3, 4, 5 will have two static httpd process and each of the remaining three processors will have one httpd process.

After determining the values, you can use any the following approaches to configure the TS/MP parameters of the `httpd` processes:

- Using the default value of `TANDEM_RECEIVE_DEPTH`
- Modifying the value of `TANDEM_RECEIVE_DEPTH`

#### Using the default value of `TANDEM_RECEIVE_DEPTH`

With `TANDEM_RECEIVE_DEPTH` set to the default value of 50, the static processes can be configured on the basis of Average Load and dynamic processes can be configured on the basis of Peak Load on your system. You can use the following approaches while deciding the values of `Numstatic` and `Maxservers`:

- The value of `Numstatic` must be set to  $(\text{Average Load}) / 50$
- The value of `Maxservers` must be set to  $(\text{Peak Load}) / 50$

For example:

If the Average Load and Peak Load on your Hibernate application turn out to values 1000 and 1500 requests and you limit `TANDEM_RECEIVE_DEPTH` to its default value of 50, the `Numstatic` must be set to 20 and `Maxservers` must be set to 30.

---

**NOTE:** It is advisable to set the value of `Numstatic` to, at least, the number of configured processors.

#### Modifying the value of `TANDEM_RECEIVE_DEPTH`

If the number of `httpd` processes configured on the basis of default value of `TANDEM_RECEIVE_DEPTH` does not fulfill your application requirement, you can change the value of `TANDEM_RECEIVE_DEPTH` and calculate the values of `Numstatic` and `Maxservers` accordingly. However, you must consider the following before increasing the value of `TANDEM_RECEIVE_DEPTH`:

- It is recommended to keep the web server running with 80 percent of its capacity. To achieve this, set the value of `TANDEM_RECEIVE_DEPTH` accordingly.  
For example, if your calculation suggests that `TANDEM_RECEIVE_DEPTH` of 80 is required, you must set it to a value of 100.
- The maximum value of `TANDEM_RECEIVE_DEPTH` is 255.
- Do not reduce `TANDEM_RECEIVE_DEPTH` to a value less than 50.

### Limiting the Maximum Number of Incoming Requests

Because of constraints on the available system resources, you might want to limit the number of requests to your iTP WebServer. If you want to configure your iTP WebServer to handle only a certain number of requests at an instance, use the `MaxConnections` configuration directive to specify the maximum number of connections that can be served by the iTP WebServer at any given instance. The iTP WebServer serves the number of requests equal to the multiple of `Numstatic` greater than or equal to the `MaxConnections` count.

#### Syntax:

```
MaxConnections -count <integer value> -replytype <customized/RST>
```

For example:

```
MaxConnections -count 101 -replytype RST
```

Consider the scenario of Example 1 above, where `Numstatic` is 5, with the following `MaxConnections` configuration:

```
MaxConnections -count 101 -replytype customized
```

In this case, the iTP WebServer serves 105 requests (higher multiple of Numstatic nearest to the count value). Here, the 106th request will display the following error message:

Maximum connections reached: The server reached its maximum configured capacity.  
with HTTP response code:

200 OK

To customize the error message, create a new message ID error-maximum-connection. This customized message is displayed if the Message configuration directive is used in the <iTP WebServer Deployment Directory>/conf/httpd.config file with the new message ID. For more information on the MaxConnections configuration directive and creating a customized error message using the Message configuration directive, see the *iTP Secure WebServer System Administrator's Guide*.

---

**NOTE:** To use the MaxConnections configuration directive, your iTP WebServer must be configured for the static environment only, that is, the values of Numstatic and MaxServers of the httpd process must be equal.

---

## Configuring NSJSP for Hibernate Applications

When a Hibernate application runs on a NonStop system, it runs as an instance of NSJSP. Therefore, before configuring the NSJSP environment, it is important to determine the load each instance of NSJSP is expected to handle.

This section describes the following configuration aspects:

- “Configuring SessionBasedLoadBalancing” (page 175)
- “Configuring Connector Threads” (page 176)
- “Configuring TS/MP Specific Parameters” (page 177)
- “Configuring Java Runtime Arguments” (page 179)

### Configuring SessionBasedLoadBalancing

The NSJSP Container maintains sessions in the form of serialized Java objects. Each session object is identified by a unique identifier called the session-ID. The session-ID is sent to the HTTP client either as a cookie or in the form of URL-rewriting. The name of the cookie is JSESSIONID and when the user application creates a session, NSJSP generates the cookie (JSESSIONID) as the name and the session-ID as the cookie value. The session objects can be either kept in the process memory or persisted in a persistent store (for example, a database table). When a session object is kept in a process, it is available only for the process that created it. If it is kept in a persistent store, it is available for any process in the NSJSP environment. When the SessionBasedLoadBalancing feature is enabled, all requests related to a particular session are routed to the process that has the session object in its memory.

To enable the SessionBasedLoadBalancing feature, you must modify the configuration of the servlet.ssc object in the <iTP WebServer Deployment Directory>/conf/servlet.config file on OSS. The servlet.ssc object is configured in the Server directive. The SessionBasedLoadBalancing feature is governed by the -DSaveSessionOnCreation and -DSessionBasedLoadBalancing arguments in the Arglist of the Server directive.

- -DSaveSessionOnCreation

Enables or disables saving the sessions in a persistent store during their creation time.

Syntax:

-DSaveSessionOnCreation= [ true | false ]

---

**NOTE:** The default value of `-DSaveSessionOnCreation` is false.

---

- `-DSessionBasedLoadBalancing`

Enables or disables SessionBasedLoadBalancing.

Syntax:

```
-DSessionBasedLoadBalancing=[ true | false ]
```

---

**NOTE:** The default value of `-DSessionBasedLoadBalancing` is set to true.

---

For example, if you want to enable the SessionBasedLoadBalancing feature for your Hibernate application and save the application sessions in a persistent store, set the Arglist as:

```
Server $server_objectcode {  
    ...  
    ...  
    ...  
    ...  
  
    Arglist -DSessionBasedLoadBalancing=true \  
-DSaveSessionOnCreation=true \  
    ...  
    ...  
    ...  
    ...  
}
```

where,

`server_objectcode` is mapped to the `servlet.ssc` object in the `<iTP WebServer Deployment Directory>/bin` directory on OSS.

While setting these arguments, consider the following:

1. If both the `SaveSessionOnCreation` and `SessionBasedLoadBalancing` options are set to false, and if a Persistent Manager is configured with a persistent store, 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.
2. Enabling or disabling the `SessionBasedLoadBalancing` feature depends on the application requirement. Your Hibernate application might encounter any of the following scenarios:
  - The application depends heavily on the state stored in session objects. Therefore, session objects cannot be lost and the application cannot recover from loss of state.
  - Application response is of prime importance and the application can recover from a loss of state.
  - The application is expected to handle largely varying loads and having a large number of static NSJSP instances is not an option.
  - The session objects must be valid for large durations (such as an entire day).
  - The session objects must be available whenever the application restarts.

For more information on the `SessionBasedLoadBalancing` and `SessionBasedLoadBalancing` configuration directives, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

## Configuring Connector Threads

The Connector is responsible for creating and maintaining the threads that are used for message processing. A thread is allocated for each incoming message; therefore, the number of messages

that can be processed simultaneously can be controlled by limiting the number of threads that NSJSP Connector can spawn.

---

**NOTE:** A single connector can service multiple instances of the container. The connector must be configured such that it can spawn threads sufficient for all the container instances. In terms of configuration, the Host element in the server.xml configuration file represents a container that is serviced by the connector.

---

The NSJSP Connector thread pool can be connected using the following ways:

- “Configuring the Connector Element” (page 177)
- “Configuring the Executor Element” (page 177)

### Configuring the Connector Element

Configure the maxThreads attribute of the Connector element in the <NSJSP Deployment Directory>/conf/server.xml file on OSS. The Connector element is the child element of the Service element.

For example: The following snippet shows the configuration in the <NSJSP Deployment Directory>/conf/server.xml file on OSS, if you want to configure 75 simultaneous connector threads:

```
...
<Service name="NSJSP">
    <Connector protocol="HTTP/1.1"
               connectionTimeout="0"
               acceptCount="25"
               maxThreads="75"/>
...
</Service>
```

### Configuring the Executor Element

You can configure the Executor element as child elements of the Service element in the <NSJSP Deployment Directory>/conf/server.xml file on OSS. The following snippet of the server.xml configuration file shows the configuration of the Executor element and how a Connector element can be configured to use an Executor element:

```
...
<Service name="NSJSP">
    <Executor name="HibernateExec"
              className="org.apache.catalina.core.StandardThreadExecutor"
              namePrefix="HibernateAPP"
              maxThreads="75"
              minSpareThreads="20"
              maxIdleTime="10000"/>
    <Connector executor="HibernateExec"/>
...
...
```

---

**NOTE:** All Executor elements that are configured in NSJSP must implement the Java interface org.apache.catalina.Executor. NSJSP provides a standard implementation of this interface in the class org.apache.catalina.core.StandardThreadExecutor.

---

## Configuring TS/MP Specific Parameters

This section describes the following TS/MP specific configuration parameters:

1. Numstatic – Determines the number of static NSJSP processes.
2. Maxservers – Determines the total number of NSJSP processes.

3. TANDEM\_RECEIVE\_DEPTH – Determines the handling capacity of each instance of NSJSP.
4. Maxlinks – Determines the maximum number of TS/MP links. The number of simultaneous messages that can be delivered to an instance of NSJSP is determined by Maxlinks. NSJSP reads these many number of messages from the \$RECEIVE file simultaneously.
5. Number of Connector threads.

The configurations for Numstatic, Maxservers, TANDEM\_RECEIVE\_DEPTH, and Maxlinks can be set in the Server directive in the *<ITP WebServer Deployment Directory>/conf/servlet.config* file on OSS. Before configuring NSJSP, it is important that you determine the maximum load that each instance of NSJSP is expected to handle. The way these configuration directives drive the NSJSP environment depends on whether the SessionBasedLoadBalancing feature is turned ON or OFF.

- “[NSJSP Configured with SessionBasedLoadBalancing Turned OFF](#)” (page 178)
- “[NSJSP Configured with SessionBasedLoadBalancing Turned ON](#)” (page 179)

### [NSJSP Configured with SessionBasedLoadBalancing Turned OFF](#)

- The value of Numstatic must be  $[(\text{Average Load}) / (\text{Max load one instance of NSJSP can handle})]$ .
- Maxservers must be  $[(\text{Peak Load}) / (\text{Max load one instance of NSJSP can handle})]$ .
- The value of Maxlinks must be set to the peak load that one instance of NSJSP is expected to handle.
- TANDEM\_RECEIVE\_DEPTH must be twice the value of Maxlinks. Normally, to handle  $n$  number of messages, you need  $n$  number of threads. In extreme cases, all the  $n$  requests may timeout. While the same number of threads processing the timeout requests could still be busy, you need the same  $n$  number of threads to process the new incoming requests. Therefore, the number of threads that must be configured is  $(n + n = 2n)$ .
- The maximum number of connector threads, maxThreads must be equal to the value of TANDEM\_RECEIVE\_DEPTH.

For example:

The parameters Maxlinks, TANDEM\_RECEIVE\_DEPTH, and the connector threads are closely linked to each other. These parameters are explained using examples. The following examples assume that the httpd processes are configured to serve 100 simultaneous HTTP requests, and all 100 HTTP requests are serviced by your Hibernate application deployed in NSJSP (that is, there is no static content in the application). Also, the peak load that one single instance of NSJSP can handle is 25.

Because SessionBasedLoadBalancing is turned OFF, all messages between HTTPD and NSJSP flow through TS/MP. The number of simultaneous messages that can be delivered to a single instance of NSJSP is determined by the value of Maxlinks; set the value of Maxlinks to 25. NSJSP can now read 25 messages from its \$RECEIVE queue simultaneously. In an extreme case, all the 25 messages time out; there must be enough space in the \$RECEIVE queue to accommodate 25 more messages. Therefore, \$RECEIVE must be opened with a Receive Depth (controlled by TANDEM\_RECEIVE\_DEPTH) of (25+25). Thus, you must set the value of TANDEM\_RECEIVE\_DEPTH to 50. The reason for setting TANDEM\_RECEIVE\_DEPTH to a value of 50 can be explained as:

To handle 25 messages, you need 25 threads. In an extreme case where all the 25 requests time out, you need another 25 threads to process the new incoming 25 requests because the threads processing the timeout requests could still be busy. Therefore, the number of threads that must be configured is  $(25+25 = 50)$ .

## NSJSP Configured with SessionBasedLoadBalancing Turned ON

- The value of Numstatic must be  $\lceil (\text{Peak Load}) / (\text{Max load one instance of NSJSP can handle}) \rceil$ .
- The value of Maxservers must be  $\lceil (\text{Peak Load}) / (\text{Max load one instance of NSJSP can handle}) \rceil$ . This ensures that no dynamic process is created, so that the session object is not lost when there are many file system calls.
- The value of Maxlinks must be set to the peak load that one instance of NSJSP is expected to handle.
- Arriving at a definite value for TANDEM\_RECEIVE\_DEPTH is difficult when NSJSP is configured with SessionBasedLoadBalancing turned ON. Deciding an optimum value for TANDEM\_RECEIVE\_DEPTH requires a good understanding of the application. The following example provides an insight into making this decision.
- The maximum number of connector threads, maxThreads must be equal to the value of TANDEM\_RECEIVE\_DEPTH.

For example:

In this example, it is evident how a single instance of NSJSP, although having Maxlinks set to 25, can serve all the 100 requests.

Because SessionBasedLoadBalancing is turned ON, all messages between HTTPD and NSJSP flow through TS/MP and file system calls. The number of simultaneous messages that can be delivered to an instance of NSJSP is determined by the value of Maxlinks; set the value of Maxlinks to 25.

With Maxlinks set to 25, a single instance of NSJSP can handle 25 concurrent requests through TS/MP, all being the first requests of web dialogs. The first call on a web dialog is always delivered to NSJSP through TS/MP and there will not be any file system call. After the 25 requests are serviced, subsequent requests on those 25 web dialogs are delivered to NSJSP through file system I/O operations.

At this stage, all 25 links to the NSJSP instance are free to process more incoming requests delivered through TS/MP. Therefore, the NSJSP instance must be able to handle 25 more concurrent requests, all being the first requests of web dialogs. After the 25 requests are also processed, the subsequent requests on these new connections arrive at NSJSP through file system I/O. At this stage, NSJSP could be handling up to 50 concurrent requests and all these requests are being delivered through file system calls from httpd and not through TS/MP. Hence, the 25 links are free to process new web dialogs. Therefore, a single instance of NSJSP can serve all the 100 requests.

At this point, one instance of NSJSP could be processing requests from all the possible 100 connections to the httpd processes. This means that there could be a scenario where all the 100 requests time out. Therefore, there must be enough space in the \$RECEIVE file to handle (100+100) messages. Therefore, TANDEM\_RECEIVE\_DEPTH must have a value of 200. The reason for setting TANDEM\_RECEIVE\_DEPTH to a value of 200 can be explained as:

To handle 100 requests, you need 100 threads. In an extreme case of all the 100 requests timing out, you need another 100 threads to process the new incoming 100 requests because the threads processing the timeout requests could still be busy. Therefore, the number of threads that need to be configured is  $(100+100 = 200)$ .

## Configuring Java Runtime Arguments

The configurations for Java runtime arguments can be done in the *<ITP WebServer Deployment Directory>/conf/servlet.config* file on OSS. The sscaux object is configured under the Server directive. Java runtime arguments are populated in the Arglist. Some of the important

Java runtime arguments that you must consider during the deployment of your Hibernate applications are:

- “[-Xmx](#)” ([page 180](#))
- “[-Xss](#)” ([page 180](#))
- “[-Xnogc](#)” ([page 180](#))

There are other Java runtime arguments supported by NSJSP. For more information, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

#### [-Xmx](#)

Sets the maximum size of the memory allocation pool, which is the garbage collected heap.

Syntax:

```
-Xmx maximum-heap-size [ k | m ]
```

where,

maximum-heap-size

is the maximum size of the memory allocated for the garbage collected. It must be greater than or equal to 1000 bytes.

k

sets the value of maximum-heap-size to be read in kilobytes.

m

sets the value of maximum-heap-size to be read in megabytes.

#### [-Xss](#)

Sets the maximum stack size that can be used by a Java thread.

Syntax:

```
-Xmx maximum-stack-size
```

where,

maximum-stack-size

is the maximum size of the stack trace in kilobytes.

#### [-Xnogc](#)

Is an optional argument to stop the Java class garbage collection.

Syntax:

```
-Xnogc
```

By default, the Java runtime reclaims space for unused Java classes. Including this optional argument might prevent any memory-leak problems.

## Hibernate Framework Configurations for NonStop Systems

This section discusses the NonStop specific Hibernate configurations for the following:

- “[JDBC Driver for SQL/MX Database](#)” ([page 181](#))
- “[Database Transaction Management](#)” ([page 188](#))
- “[Connection Pooling](#)” ([page 189](#))

All the configurations for JDBC, Transaction Management, and Connection Pooling are configured in either the `hibernate.properties` or the `hibernate.cfg.xml` file.

## JDBC Driver for SQL/MX Database

The JDBC driver to be used when a Hibernate application connects to the SQL/MX database must be specified in the `hibernate.cfg.xml` or `hibernate.properties` file created while developing the Hibernate application.

This section discusses the following:

- “Configuring the JDBC Type 2 Driver for SQL/MX Database” (page 181)
- “Configuring JDBC Type 4 Driver for SQL/MX Database” (page 185)

### Configuring the JDBC Type 2 Driver for SQL/MX Database

In a typical Hibernate application, you can configure the JDBC Type 2 driver for the SQL/MX database using either one of the following preferred options:

- “Configuring the JDBC Type 2 Driver for the SQL/MX Database using `hibernate.cfg.xml`” (page 181)
- “Configuring JDBC Type 2 Driver for SQL/MX Database using `hibernate.properties`” (page 183)

#### Configuring the JDBC Type 2 Driver for the SQL/MX Database using `hibernate.cfg.xml`

The advantage of using this approach is the externalization of mapping file names to the configuration. To complete the database operation in a Hibernate application using the JDBC Type 2 driver for the SQL/MX database, complete the following steps:

1. “Specifying the JDBC Type 2 Driver for SQL/MX Database” (page 181)
2. “Defining the Connection URL” (page 182)
3. “Establishing the Connection” (page 182)
4. “Defining the Hibernate Dialect for SQL/MX Database” (page 182)
5. “Specifying the Mapping Resources” (page 182)
6. “Opening a Session for Database Operation” (page 182)
7. “Closing the Session” (page 183)

#### Specifying the JDBC Type 2 Driver for SQL/MX Database

One of the advantages of using the JDBC driver is that the SQL/MX database server does not require any changes. Instead, the JDBC Type 2 driver translates calls written in Java to the specific format required by the database server.

To specify the JDBC Type 2 driver, enter `com.tandem.sqlmx.SQLMXDriver` as the JDBC Type 2 driver class for the SQL/MX database in the `hibernate.cfg.xml` file under the `<session-factory>` tag as shown:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">com.tandem.sqlmx.SQLMXDriver</property>
...
...
...
</session-factory>
</hibernate-configuration>
```

---

**NOTE:** The significance of the `<session-factory>` tag is explained in “Opening a Session for Database Operation” (page 182).

---

## Defining the Connection URL

After you have specified the JDBC Type 2 driver, enter the location of the SQL/MX server. URLs referring to SQL/MX use the `jdbc:` protocol and have the server host, port number embedded within the URL.

To define the connection URL, enter the connection URL as `jdbc:sqlmx://` for the SQL/MX database in the `hibernate.cfg.xml` file under the `<session-factory>` tag as shown:

```
<property name="connection.url">jdbc:sqlmx://</property>
```

## Establishing the Connection

If you plan to use the JDBC Type 2 driver, do not specify the username and password

Add the following syntax in the `hibernate.cfg.xml` file under the `<session-factory>` tag as shown:

```
<property name="connection.username"></property>
<property name="connection.password"></property>
```

## Defining the Hibernate Dialect for SQL/MX Database

Enter `org.hibernate.dialect.SqlmxDialect` as the Hibernate dialect class name for SQL/MX database, in the `hibernate.cfg.xml` file under the `<session-factory>` tag.

```
<property name="dialect">
  org.hibernate.dialect.SqlmxDialect
</property>
```

This dialect file allows Hibernate to generate SQL that is optimized for the SQL/MX database.

## Specifying the Mapping Resources

Specify all the mapping files (.hbm) in the `<mapping resource>` tag as shown:

```
<!-- Mapping files -->
<mapping resource="<Name of the .hbm file>\" />
```

For example:

If the mapping file used in your application is `Employee.hbm.xml`, specify it as shown:

```
<!-- Mapping files -->
<mapping resource="Employee.hbm.xml" />
```

The basic configuration is ready for you to perform database operation using Hibernate.

## Opening a Session for Database Operation

To open a new session for database transaction, complete the following steps:

1. ["Configuring SessionFactory in the hibernate.cfg.xml File" \(page 182\)](#)
2. ["Creating a New Session from the SessionFactory in your Java Program" \(page 183\)](#)

## Configuring SessionFactory in the `hibernate.cfg.xml` File

Add the following lines in the `hibernate.cfg.xml` file to configure the Hibernate SessionFactory.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    ...
    ...
    ...
    ...
    ...
  </session-factory>
</hibernate-configuration>
```

```
</session-factory>  
</hibernate-configuration>
```

---

**NOTE:** The above configuration uses the hibernate-configuration-3.0.dtd file.

### **Creating a New Session from the SessionFactory in your Java Program**

To create a new session from SessionFactory, add the following lines of code in your Java program:

```
SessionFactory sessionFactory;  
Configuration cfg = new Configuration().configure();  
sessionFactory = cfg.buildSessionFactory();  
Session session = sessionFactory.openSession();
```

---

**NOTE:** To view the complete configuration snippet of hibernate.cfg.xml, see “[JDBC Configuration](#)” (page 227).

### **Closing the Session**

Once the database operation is complete, close the session to deallocate the memory.

Add the following line of code when you complete the database operation.

```
session.close();
```

## **Configuring JDBC Type 2 Driver for SQL/MX Database using hibernate.properties**

To complete the database operation in a Hibernate application using the JDBC Type 2 driver for the SQL/MX database, complete the following steps:

1. “[Specifying the JDBC Type 2 Driver for SQL/MX Database](#)” (page 183)
2. “[Defining the Connection URL](#)” (page 183)
3. “[Establishing the Connection](#)” (page 183)
4. “[Defining the Hibernate Dialect for SQL/MX Database](#)” (page 184)
5. “[Opening a Session for Database Operation](#)” (page 184)
6. “[Specifying the Mapping Resources](#)” (page 184)
7. “[Closing the Session](#)” (page 184)

### **Specifying the JDBC Type 2 Driver for SQL/MX Database**

To specify the JDBC Type 2 driver, enter com.tandem.sqlmx.SQLMXDriver as the JDBC Type 2 driver class for the SQL/MX database in the hibernate.properties file as shown:

```
hibernate.connection.driver_class_com.tandem.sqlmx.SQLMXDriver
```

### **Defining the Connection URL**

After you have specified the JDBC Type 2 driver, enter the location of the SQL/MX server. URLs referring to SQL/MX use the jdbc: protocol embedded within the URL.

To define the connection URL, enter jdbc:sqlmx:// as the connection URL for the SQL/MX database in the hibernate.properties file as shown:

```
hibernate.connection.url jdbc:sqlmx://
```

### **Establishing the Connection**

If you plan to use the JDBC Type 2 driver, do not specify the username and password.

Add the properties for the username and password for your NonStop system in the hibernate.properties file as shown:

```
hibernate.connection.username  
hibernate.connection.password
```

## Defining the Hibernate Dialect for SQL/MX Database

Enter `org.hibernate.dialect.SqlmxDialect` as the class name of a Hibernate dialect file in the `hibernate.properties` file as shown:

```
hibernate.dialect org.hibernate.dialect.SqlmxDialect
```

This dialect file allows Hibernate to generate SQL optimized for SQL/MX database.

## Opening a Session for Database Operation

To open a new session for database transaction, complete the following steps:

1. “Configuring SessionFactory in your Java Program” (page 184)
2. “Creating a New Session from the SessionFactory in Java” (page 184)

### Configuring SessionFactory in your Java Program

Add the following lines to your Java program to create Hibernate SessionFactory.

```
private SessionFactory factory;
```

```
...
...
...
public void databaseCon()
{
    Configuration cfg = new Configuration();
    factory = cfg.buildSessionFactory();
    ...
    ...
    ...
}
```

### Creating a New Session from the SessionFactory in Java

Add the following lines to your Java program to create a new session from the SessionFactory:

```
Session s = factory.openSession();
```

This will create the session which will be used further in your application to perform database operation.

---

**NOTE:** To view the complete configuration snippet of the `hibernate.properties` file, see “JDBC Configuration” (page 227).

---

## Specifying the Mapping Resources

Specify all the mapping files (.hbm) in your Java program as shown.

```
Configuration cfg = new Configuration()
    .addResource("Name of mapping file#1")
    .addResource("Name of mapping file#2");
```

For example,

If the mapping file used in your application is `Item.hbm.xml` and `Bid.hbm.xml`, specify it as shown.

```
Configuration cfg = new Configuration()
    .addResource("Item.hbm.xml")
    .addResource("Bid.hbm.xml");
```

The basic configuration is ready for you to perform database operation using Hibernate.

## Closing the Session

Once the database operation is complete, close the session to deallocate the memory.

Add the following line of code when you complete the database operation.

```
session.close();
```

## Configuring JDBC Type 4 Driver for SQL/MX Database

In a typical Hibernate application, you can configure the JDBC Type 4 driver for SQL/MX database using either one of the following preferred options:

- “Configuring JDBC Type 4 Driver for SQL/MX Database using `hibernate.cfg.xml`” (page 185)
- “Configuring JDBC Type 4 Driver for SQL/MX Database using `hibernate.properties`” (page 187)

### Configuring JDBC Type 4 Driver for SQL/MX Database using `hibernate.cfg.xml`

The advantage of using this approach is the externalization of mapping file names to configuration. To complete the database operation in a Hibernate application using the JDBC Type 4 driver for the SQL/MX database, complete the following steps:

1. “Specifying the JDBC Type 4 Driver for SQL/MX Database” (page 185)
2. “Defining the Connection URL” (page 185)
3. Establishing the Connection
4. “Defining the Hibernate Dialect for SQL/MX Database” (page 186)
5. “Specifying the Mapping Resources” (page 186)
6. “Opening a Session for Database Operation” (page 186)
7. “Closing the Session” (page 187)

### Specifying the JDBC Type 4 Driver for SQL/MX Database

To specify the JDBC Type 4 driver for SQL/MX database, enter

`com.tandem.t4jdbc.SQLMXDriver` as the name of the JDBC Type 4 driver class for the SQL/MX database in the `hibernate.cfg.xml` file under the `<session-factory>` tag as shown below.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">com.tandem.t4jdbc.SQLMXDriver</property>
...
...
...
...
</session-factory>
</hibernate-configuration>
```

---

**NOTE:** The significance of the `<session-factory>` tag is discussed in “Opening a Session for Database Operation” (page 186).

---

### Defining the Connection URL

After you have specified the JDBC Type 4 driver, enter the location of the SQL/MX server. URLs referring to SQL/MX use the `jdb:tcp` protocol and have the server host, port number embedded within the URL.

To define the connection URL, enter `jdb:tcp://<HP NonStop System IP Address>:<Port No.>` as the connection URL for the SQL/MX database in the `hibernate.cfg.xml` under the `<session-factory>` tag as shown:

```
<property name="connection.url">jdb:tcp://<HP NonStop System IP Address>:<Port No.></property>
```

## Establishing the Connection

To establish the connection, specify the username and password of your NonStop system in the hibernate.cfg.xml file under the `<session-factory>` tag as shown.

```
<property name="connection.username"><HP NonStop Username></property>
<property name="connection.password"><HP NonStop Password></property>
```

## Defining the Hibernate Dialect for SQL/MX Database

Enter `org.hibernate.dialect.SqlmxDialect` as the class name of a Hibernate dialect file in the `hibernate.cfg.xml` file under the `<session-factory>` tag.

```
<property name="dialect">  
    org.hibernate.dialect.SqlmxDialect  
</property>
```

This dialect file allows Hibernate to generate SQL that is optimized for the SQL/MX database.

## Specifying the Mapping Resources

Specify all the mapping files (.hbm) in the `<mapping resource>` tag as shown:

```
<!-- Mapping files -->
<mapping resource="<Name of the .hbm file>" />
```

For example:

If the mapping file used in your application is `Employee.hbm.xml`, specify it as shown:

```
<!-- Mapping files -->
<mapping resource="Employee.hbm.xml" />
```

The basic configuration is ready for you to perform database operation using Hibernate.

## Opening a Session for Database Operation

To open a new session for database transaction, complete the following steps:

1. "Configuring SessionFactory in the hibernate.cfg.xml File" (page 186)
  2. Creating a New Session from the SessionFactory in Java

## Configuring SessionFactory in the hibernate.cfg.xml File

Add the following lines in the hibernate.cfg.xml file to configure the Hibernate SessionFactory.

**NOTE:** The above configuration uses the hibernate-configuration-3.0.dtd file.

## Creating a New Session from the SessionFactory in Java

Add the following to create a new session from the SessionFactory:

```
SessionFactory sessionFactory;
Configuration cfg = new Configuration().configure();
sessionFactory = cfg.buildSessionFactory();
Session session = sessionFactory.openSession();
```

---

**NOTE:** To view the complete configuration snippet of the `hibernate.cfg.xml` file, see “[JDBC Configuration](#)” (page 227).

### Closing the Session

Once the database operation is complete, close the session to deallocate the memory.

Add the following line of code when you complete the database operation.

```
session.close();
```

### Configuring JDBC Type 4 Driver for SQL/MX Database using `hibernate.properties`

To complete the database operation in a Hibernate application using the JDBC Type 4 driver for the SQL/MX database, complete the following steps:

1. Specifying the JDBC Type 4 Driver for SQL/MX Database
2. “Defining the Connection URL” (page 187)
3. “Establishing the Connection” (page 187)
4. “Defining the Hibernate Dialect for SQL/MX Database” (page 187)
5. “Opening a Session for Database Operation” (page 187)
6. “Specifying the Mapping Resources” (page 188)
7. “Closing the Session” (page 188)

### Specifying the JDBC Type 4 Driver for SQL/MX Database

To specify the JDBC Type 4 driver for SQL/MX database, enter

`com.tandem.t4jdbc.SQLMXDriver` as the JDBC Type 4 driver class for the SQL/MX database in the `hibernate.properties` file as shown:

```
hibernate.connection.driver_class com.tandem.t4jdbc.SQLMXDriver
```

### Defining the Connection URL

After you have specified the JDBC Type 4 driver, enter the location of the SQL/MX server. URLs referring to SQL/MX use the `jdb: protocol` and have the server host, port number embedded within the URL.

To define the connection URL, enter `jdb:t4sqlmx://<HP NonStop System IP Address>:<Port No.>` as the connection URL for the SQL/MX database in the `hibernate.properties` file as shown:

```
hibernate.connection.url jdb:t4sqlmx://<HP NonStop System IP Address>:<Port No.>
```

### Establishing the Connection

To establish the connection, specify the username and password of your NonStop system in the `hibernate.properties` as shown.

```
hibernate.connection.username <HP NonStop Username>  
hibernate.connection.password <HP NonStop Password>
```

### Defining the Hibernate Dialect for SQL/MX Database

Enter `org.hibernate.dialect.SqlmxDialect` as the class name of a Hibernate dialect file in the `hibernate.properties` file as shown below.

```
hibernate.dialect org.hibernate.dialect.SqlmxDialect
```

This property allows Hibernate to generate SQL optimized for SQL/MX database.

### Opening a Session for Database Operation

To open a new session for database transaction, complete the following steps:

1. “[Configuring SessionFactory in your Java Program](#)” (page 188)
2. “[Creating a New Session from the SessionFactory in Java](#)” (page 188)

## Configuring SessionFactory in your Java Program

Add the following lines to your Java program to create Hibernate SessionFactory.

```
private SessionFactory factory;  
  
...  
...  
...  
public void databaseCon()  
{  
    Configuration cfg = new Configuration();  
    factory = cfg.buildSessionFactory();  
    ...  
    ...  
    ...  
}
```

## Creating a New Session from the SessionFactory in Java

Add the following lines to your Java program to create a new session from the SessionFactory:

```
Session s = factory.openSession();
```

This will create the session which will be used further to perform database operation.

---

**NOTE:** To view the complete configuration snippet of the hibernate.properties file, see "JDBC Configuration" (page 227).

---

## Specifying the Mapping Resources

Specify all the mapping files (.hbm) in your Java program as shown:

```
Configuration cfg = new Configuration()  
    .addResource("<Name of mapping file#1>")  
    .addResource("<Name of mapping file#2>");
```

For example,

If the mapping file used in your application is Item.hbm.xml and Bid.hbm.xml, specify it as shown:

```
Configuration cfg = new Configuration()  
    .addResource("Item.hbm.xml")  
    .addResource("Bid.hbm.xml");
```

The basic configuration is ready for you to perform database operation using Hibernate.

## Closing the Session

Once the database operation is complete, close the session to deallocate the memory.

Add the following line of code when you complete the database operation.

```
session.close();
```

## Database Transaction Management

Hibernate works in any environment that uses JTA, in fact, we recommend to use JTA whenever possible as it is the standard Java transaction interface. You get JTA built-in with all J2EE/JEE application servers, and each datasource you use in such a container is automatically handled by a JTA TransactionManager.

---

**NOTE:**

- NonStop Server for Java supports transactions by means of the NonStop Server for JTA, which is an implementation of the Sun Microsystems JTA.
  - NonStop Server for JTA implements the package, javax.transaction.
  - NonStop Server for Java includes the JTA package com.tandem.jta.
- 

## Configurations using JTA

The application code must implement JTA user transaction by using JTAFactory provided with the NonStop Server for Java Transaction API package com.tandem.jta. This can be achieved by following the below given steps:

1. Define a JTA user transaction.

```
javax.transaction.UserTransaction tx = null;
```

2. Initialize the user transaction.

```
tx = com.tandem.jta.JTAFactory.getUserTransaction();
```

3. Begin the transaction.

```
tx.begin();
```

4. Commit or rollback the transaction depending upon the status of the operations performed.

```
tx.commit();
```

or

```
tx.rollback();
```

## Configurations using JDBC

To use JDBC Transaction Manager, complete the following steps:

1. Open the session.

```
Session session = factory.openSession();
```

2. Initialize the user transaction.

```
Transaction tx = null;
```

3. Begin the transaction.

```
tx = session.beginTransaction();
```

4. Perform the database operation.

```
session.load(...);
```

```
session.persist(...);
```

5. Commit or rollback the transaction depending upon the status of the operations performed.

```
tx.commit();
```

or

```
tx.rollback();
```

## Connection Pooling

Connection Pooling is a cache of database connections maintained in the database memory so that the database connections can be reused when the database receives any future requests for data. In this case, whenever a database operation needs to be performed, a request for an existing database connection is made to a pool or any object that holds the existing database connections. Once that particular database operation is completed, the connection is returned to the pool.

Different types of connection pooling are:

- Connection Pooling provided by Hibernate
- Connection Pooling using C3P0
- Connection Pooling using JDBC T2 or T4 driver

In order to enable connection pooling in your application, many vendors provide their own customized Datasource Interfaces. However, we recommend the use of Apache C3P0 as the Datasource Interface to be used with Hibernate applications. Hibernate framework distribution contains the necessary files to use Apache C3P0 0.9.1.2 as the Datasource Interface for connection pooling.

To configure connection pooling using Apache C3P0, complete the following steps:

1. Add the c3p0-0.9.1.jar in your application CLASSPATH.

---

**NOTE:** c3p0-0.9.1.jar is available along with the Hibernate distribution package at <Hibernate Home>/lib directory.

---

2. Set some properties in the hibernate.cfg.xml or hibernate.properties file of your Hibernate application to configure Apache C3P0 connection pooling. The details of these properties are provided in [http://www.mchange.com/projects/c3p0/index.html#appendix\\_a](http://www.mchange.com/projects/c3p0/index.html#appendix_a)

---

**NOTE:** Hibernate provides additional configurations related to JDBC, Caching, and Transaction Management. These configurations are optional and are discussed in [http://www.hibernate.org/hib\\_docs/reference/en/html/configuration-optional.html#configuration-optional-properties](http://www.hibernate.org/hib_docs/reference/en/html/configuration-optional.html#configuration-optional-properties).

---

## Module File Caching Configurations

The Module File Caching (MFC) feature shares the prepared statement plans among the NonStop SQL/MX database connections and helps in reducing resource consumption. MFC is disabled by default; therefore, it must be configured to enable it.

To use the MFC feature, you must perform the following activities:

- “Configuring NonStop SQL/MX DataSource for MFC” (page 190)
- “Modifying the Hibernate Application” (page 191)

---

**NOTE:** JDBC Type 4 driver offers MFC feature.

---

## Configuring NonStop SQL/MX DataSource for MFC

To configure MFC on a NonStop system, complete the following steps:

1. Enter the SQL/MX Connectivity Service (MXCS) subsystem using the SQL/MX Conversational Interface (MXCI) subsystem:

```
mxci>> mode mxcs;
```

2. Configure a datasource for the user specified association service (Mxoas service):

```
mxci>> add ds $mcbs."MFC_datasource";
```

---

**NOTE:** You can configure any number of servers based on your requirement. If you do not specify the number of servers, the default value (maxserver=5, initserver=1, and idleserver=1) is assumed.

---

3. Stop the datasource:

```
mxci>> stop ds MFC_datasource, reason 'test';
```

---

**NOTE:** This step must be performed only if you have a configured datasource in the started status.

---

- 4.** To enable the MFC feature, add the statement\_module\_caching environment variable to the datasource:

```
mxci>> add evar $mcbs."MFC_datasource".statement_module_caching, type set,  
        value 'true';
```

**NOTE:** MFC can have only two values: true or false.

- 5.** Add the compiled\_module\_location environment variable to the datasource:

```
mxci>> add evar $mfc."MFC_datasource".compiled_module_location, type set,  
        value 'default';
```

**NOTE:** The compiled\_module\_location can be set to a value other than default, such as, /usr/sash. However, you must ensure that the directory is created before specifying it.

```
mxci>> add evar $mfc."MFC_datasource".compiled_module_location, type set,  
        value '/usr/sash';
```

- 6.** Start the datasource:

```
>> start ds $mfc."MFC_datasource";
```

For more details, see the *JDBC Type 4 Driver 2.0 Programmer's Reference*.

## Modifying the Hibernate Application

**NOTE:** If the default datasource is configured for MFC, do not modify the Hibernate application.

To use the MFC feature , modify the Hibernate application using the following steps:

### hibernate.properties file

- Modify the hibernate.properties file as follows:
  - To set serverDataSource to the datasource name on which MFC is configured, add the following:  
`hibernate.connection.serverDataSource MFC_datasource`

### hibernate.cfg.xml file

Modify the hibernate.cfg.xml file as follows:

- To modify the hibernate.properties file, complete the following steps:
  - To set serverDataSource to the datasource name on which MFC is configured, add the following:  
`<property name="connection.serverDataSource">MFC_datasource</property>`

## Enabling Browse Access

Browse Access is a SQL/MX specific feature, which allows you to read uncommitted data from a table. The feature uses the Hibernate interceptor, which intercepts SQL queries and appends for browse access string based on the configurations defined in the query filtering criteria of BrowseAccessIncludeExcludeList.xml file. You can use the Browse Access feature in Hibernate applications, and Spring and Hibernate applications.

## Hibernate Applications

For Hibernate applications, the interceptor cannot be injected on the Hibernate Configuration object through the configuration. Code must be changed to inject the interceptor on the Configuration object. The changes required to enable Browse Access feature are:

1. Inject the interceptor on the Configuration object. The interceptor can be injected in one of the following ways:

- a. Specify a session-scoped interceptor by opening a session with a call to one of the overloaded SessionFactory.openSession() methods, which accepts an Interceptor. For example:

```
Configuration cfg = new Configuration().configure();
SessionFactory sessFactory= cfg.buildSessionFactory();
Session session =
sessFactory.openSession(org.hibernate.dialect.SqlmxInterceptor.getInterceptorInstance());
```

- b. Specify a SessionFactory-scoped interceptor on the Hibernate Configuration object prior to building the SessionFactory. All session factories built with this Configuration object have the interceptor set. For example:

```
Configuration cfg = new Configuration().configure();
cfg.setInterceptor(SqlmxInterceptor.getInterceptorInstance());
SessionFactory sessFactory= cfg.buildSessionFactory();
```

2. Include Regex Patterns by creating the BrowseAccessIncludeExcludeList.xml file, and add it to the CLASSPATH (place this file with hibernate.properties/ hibernate.cfg.xml files). Copy the following into BrowseAccessIncludeExcludeList.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0_22" class="java.beans.XMLDecoder">
<object class="org.hibernate.dialect.SqlmxInterceptor">
<void property="applyBrowseAccessTo">
<!-- INCLUDE LIST- Please Specify a regular expression
(regex pattern)here. You can specify more than one regex by
adding any number of method tags, for ex: <void
method="add"> <string>regex</string> </void>. If any of
these regex matches with the query browse access will be
enabled -->
<object class="java.util.ArrayList">
<void method="add">
<string>select.*</string>
</void>

<void method="add">
<string></string>
</void>
</object>
</void>
</object>
</java>
```

By default, the interceptor intercepts all the select queries and enables Browse Access. If there is a need to disable Browse Access for any of the tables or queries, update the BrowseAccessIncludeExcludeList.xml file.

## Spring and Hibernate Applications

For Spring and Hibernate applications, you can enable Browse Access through the configuration. The steps to enable Browse Access are:

1. Create a bean for the class org.hibernate.dialect.SqlmxInterceptor and inject applyBrowseAccessTo property with the Regex Patterns in the bean definition file. For example:

```
<bean id="Interceptor"
      class="org.hibernate.dialect.SqlmxInterceptor">
    <property name="applyBrowseAccessTo">
      <list>
        <!--
          INCLUDE LIST- Please Specify a regular expression
          (regex pattern) here. You can specify more than one
          regex by adding any number of value tags, for ex:
          <value>regex</value>. If any of these regex
          matches with the query browse access will be enabled
        -->

        <value>select.*</value>
      </list>
    </property>
</bean>
```

2. In the SessionFactory bean, inject the entityInterceptor property with the bean created in the previous step. For example:

```
<bean id="mySessionFactory"
      class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="entityInterceptor" ref="Interceptor" />
    <property name="hibernateProperties">
      <value>
        hibernate.connection.url= .....
        hibernate.connection.driver_class= .....
        hibernate.connection.username= .....
        .....
        .....
        .....
      </value>
    </property>
</bean>
```

---

**NOTE:** You can also define an exclude list by defining Regex Patterns under doNotApplyBrowseAccessTo list.

The include exclude list for BrowseAccessIncludeExcludeList.xml is interpreted as follows:

1. First, entries under doNotApplyBrowseAccessTo list are checked. If any match is found, then no further checks will be made, and Browse Access is not applied for the query.
2. Entries in applyBrowseAccessTo list are then considered for a match, when there are no Regex Pattern matches in doNotApplyBrowseAccessTo list. Browse Access is enabled when a match is found in applyBrowseAccessTo list.

To define the doNotApplyBrowseAccess list:

- For Hibernate applications, include the following changes in your BrowseAccessIncludeExcludeList.xml file and modify as required.

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0_22" class="java.beans.XMLDecoder">
    <object class="org.hibernate.dialect.SqdmxInterceptor">
        <void property="applyBrowseAccessTo">
            .....
            .....
        </void>
        <void property="doNotApplyBrowseAccessTo"-->
            <!--EXCLUDE LIST-Please Specify a regular expression
            (regex pattern) here. You can specify more than one regex
            by adding any number of method tags, for ex: <void
            method="add"> <string>regex</string> </void>.
            If any of these regex matches with the query
            browse access will not be enabled -->
            <object class="java.util.ArrayList">
                <void method="add">
                    <string></string>
                </void>
            </object>
        </void>
    </object>
</java>
```

- For Spring and Hibernate applications, include the following changes in your bean definition file and modify the values as required.

```
<bean id="Interceptor"
      class="org.hibernate.dialect.SqdmxInterceptor">
    <property name="applyBrowseAccessTo">
        <value> ... </value>
        <value> ... </value>
    </property>
    <property name="doNotApplyBrowseAccessTo">
        <list>
            <!-- EXCLUDE LIST-Please Specify a regular expression
            (regex pattern) here. You can specify more than one regex by
            adding any number of value tags, for ex:
            <value>regex</value>. If any of these regex matches with
            the query browse access will not be enabled -->
            <!-- The following regex pattern ignores the EMP tables for browse access -->
```

```
<value>.* EMP .*</value>
<value> ... </value>
<value> ... </value>
</list>
</property>
</bean>
```

---

# 9 Getting Started with Hibernate

This chapter explains how to develop an application using the Hibernate framework. It describes the steps required to build a basic employee management system on the Windows system and run it on the NonStop system.

## Prerequisites

Before getting started, make sure that you have the requisite software installed on the NonStop and Windows system.

### NonStop System

The following software must be installed on the NonStop system:

- JDBC Type 2 driver for NonStop SQL/MX version T1275H50 or later
- NonStop SQL/MX version T1050H23 or later
- NSJ version T2766H60 or later

### Windows System

The following software must be installed on the Windows system:

- JDK version 1.5 or later
- JDBC Type 4 driver for NonStop SQL/MX version T1249V11 or later
- Eclipse Galileo IDE version 3.3.1.1 or later

---

**NOTE:** For more information about installing the software required on NonStop and Windows system, see “[Prerequisites](#)” (page 18).

## Overview of EmployeeInfo

EmployeeInfo is a basic employee management system developed using the Hibernate framework.

You can use the EmployeeInfo application to:

- Add an employee record and employee details, such as Employee ID, First Name, Last Name, Age, and email ID.
- Retrieves information about existing employees.

In this application:

- The persistence services are developed using the Hibernate ORM Mapping tool.
- The employee data is stored in the NonStop SQL/MX database.
- An Employee ID is generated automatically using the automatic key generation feature in Hibernate.

This section describes the following steps required to develop, set up, and run the EmployeeInfo application:

- “[Developing EmployeeInfo on Windows using the Eclipse Galileo IDE](#)” (page 197)
- “[Setting Up the NonStop Environment](#)” (page 218)
- “[Running EmployeeInfo on NonStop](#)” (page 220)

# Developing EmployeeInfo on Windows using the Eclipse Galileo IDE

## NOTE:

- It is not mandatory for you to use the Eclipse Galileo IDE. You can use an IDE that supports Java.
- The screen captures in this section are based on Eclipse Galileo IDE version 3.3.1.1. The screen captures might look different if you use a different version of Eclipse.

The following activities are required to develop the EmployeeInfo application using the Eclipse Galileo IDE:

- “Creating the Eclipse Workspace” (page 197)
- “Creating a New Java Project” (page 198)
- “Adding Dependency JAR Files to the Project Library” (page 201)
- “Creating the Package for the EmployeeInfo Application” (page 204)
- “Creating the Employee.java Class File” (page 204)
- “Creating the Main Class” (page 206)
- “Implementing the Business Logic” (page 207)
- “Creating the Hibernate Mapping File” (page 209)
- “Setting Hibernate Configurations” (page 212)
- “Creating the EmployeeInfo Application JAR File” (page 216)

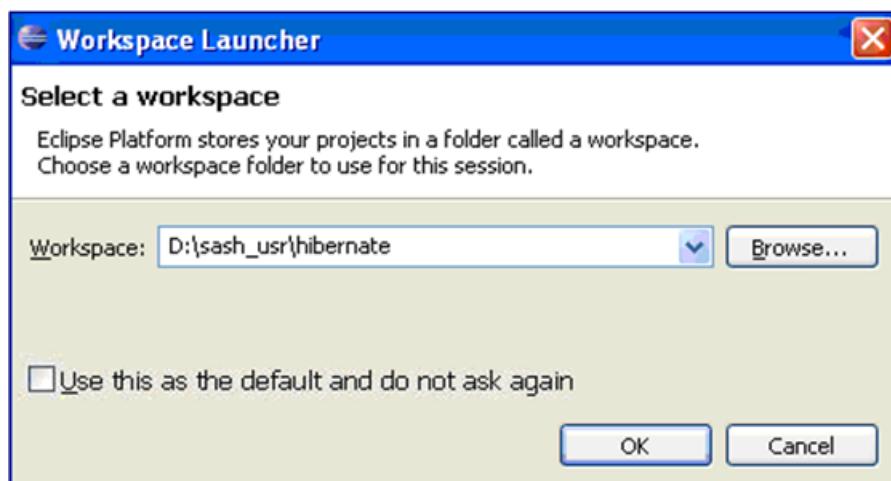
## Creating the Eclipse Workspace

To create a new workspace using the Eclipse Galileo IDE, complete the following steps:

1. To open the Eclipse workspace, double-click the `eclipse.exe` file in *<Eclipse IDE Installation Directory>*.

The Workspace Launcher dialog box appears. By default, the workspace is set to the existing workspace, if already created.

**Figure 51 Workspace Launcher Dialog Box**



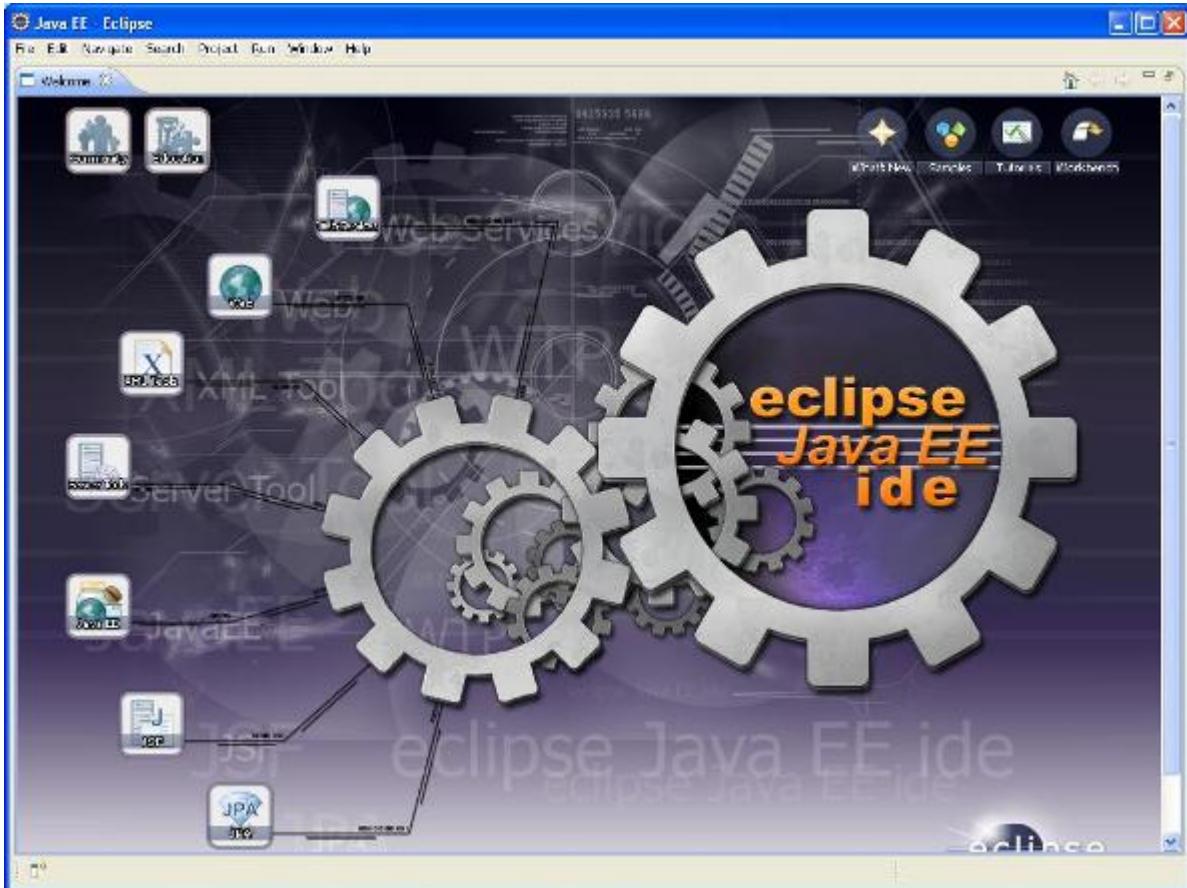
2. Click **OK**.

To create a new workspace, click **Browse** and select the folder you want to use as the workspace.

The Eclipse SDK Welcome screen appears.

**NOTE:** D:\sash\_usr\hibernate is the sample workspace used to develop the EmployeeInfo application.

## Figure 52 Eclipse SDK Welcome Screen



Close the welcome screen. The workspace is created.

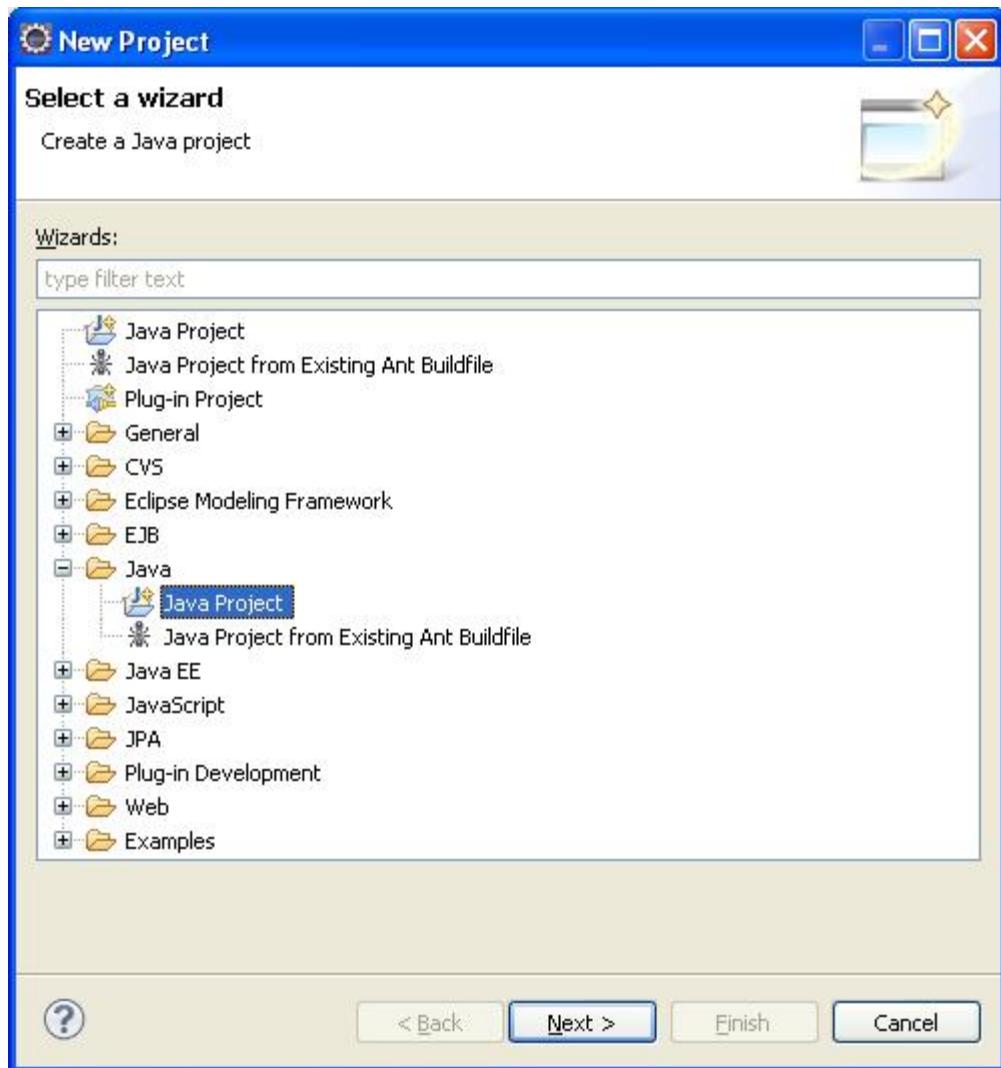
**NOTE:** The source code for the EmployeeInfo application is available in the `SAMPLES.zip` file.

# Creating a New Java Project

To create a new Java project for the EmployeeInfo application, complete the following steps:

1. Click **File > New > Project** to open a new Eclipse project dialog.  
The New Project dialog box appears.
  2. From the list of folders, select **Java > Java Project** and click **Next**.

**Figure 53 New Project Dialog Box**



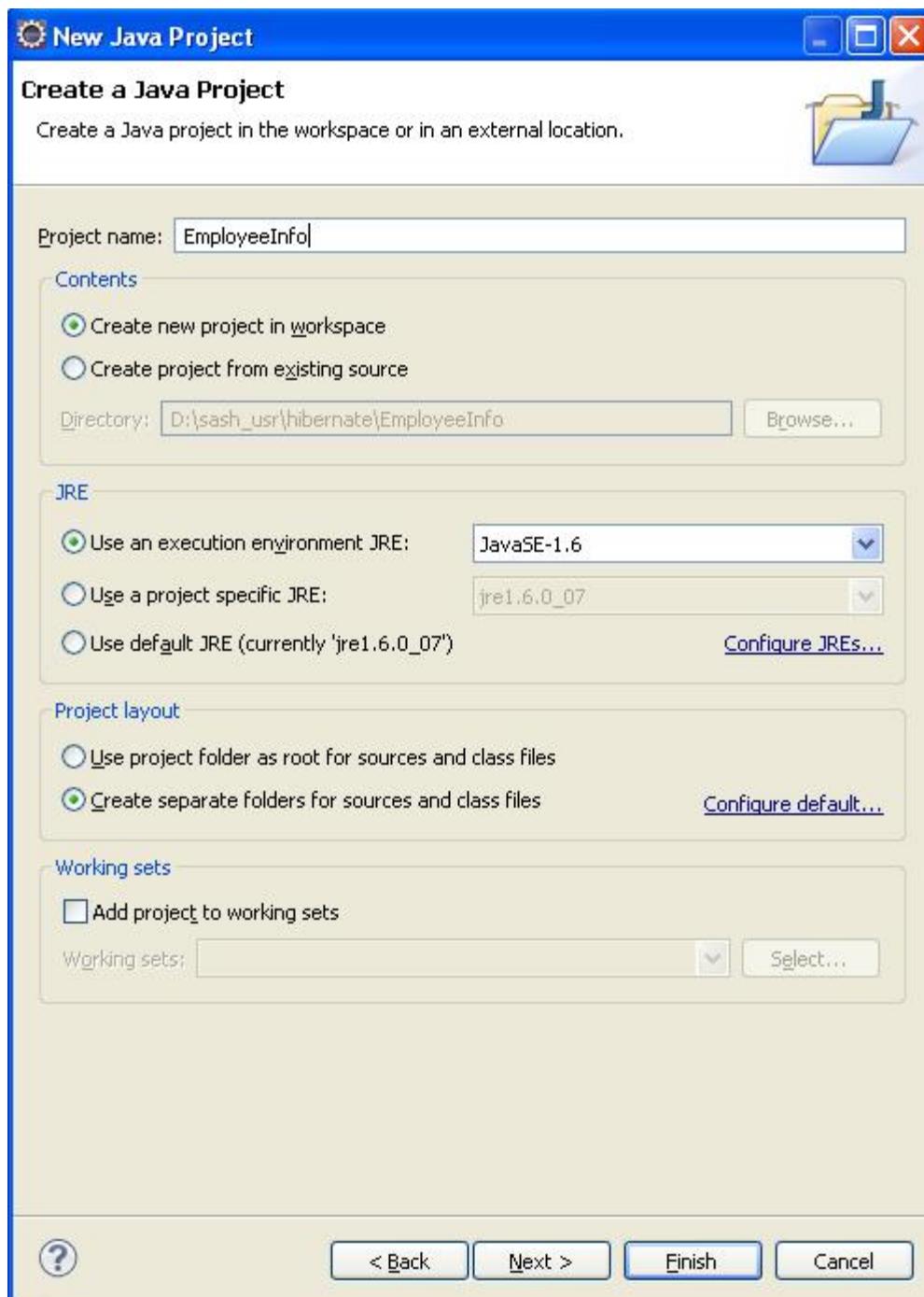
The New Java Project dialog box appears.

3. In the **Project name** field, type `EmployeeInfo`. Ensure that the directory for this project is created inside the workspace that you have selected or created.

**NOTE:**

- In this application, `EmployeeInfo` is the name of the Java project.
- The options in the JRE and Project layout sections are selected by default.

**Figure 54 New Java Project Dialog Box**



4. Verify that the JRE system library is set to JRE version 1.6 or later.

**NOTE:** The JRE version used in this example is 1.6.

If JRE version is not 1.6 or later:

1. In the JRE dialog box, click **Configure default**.

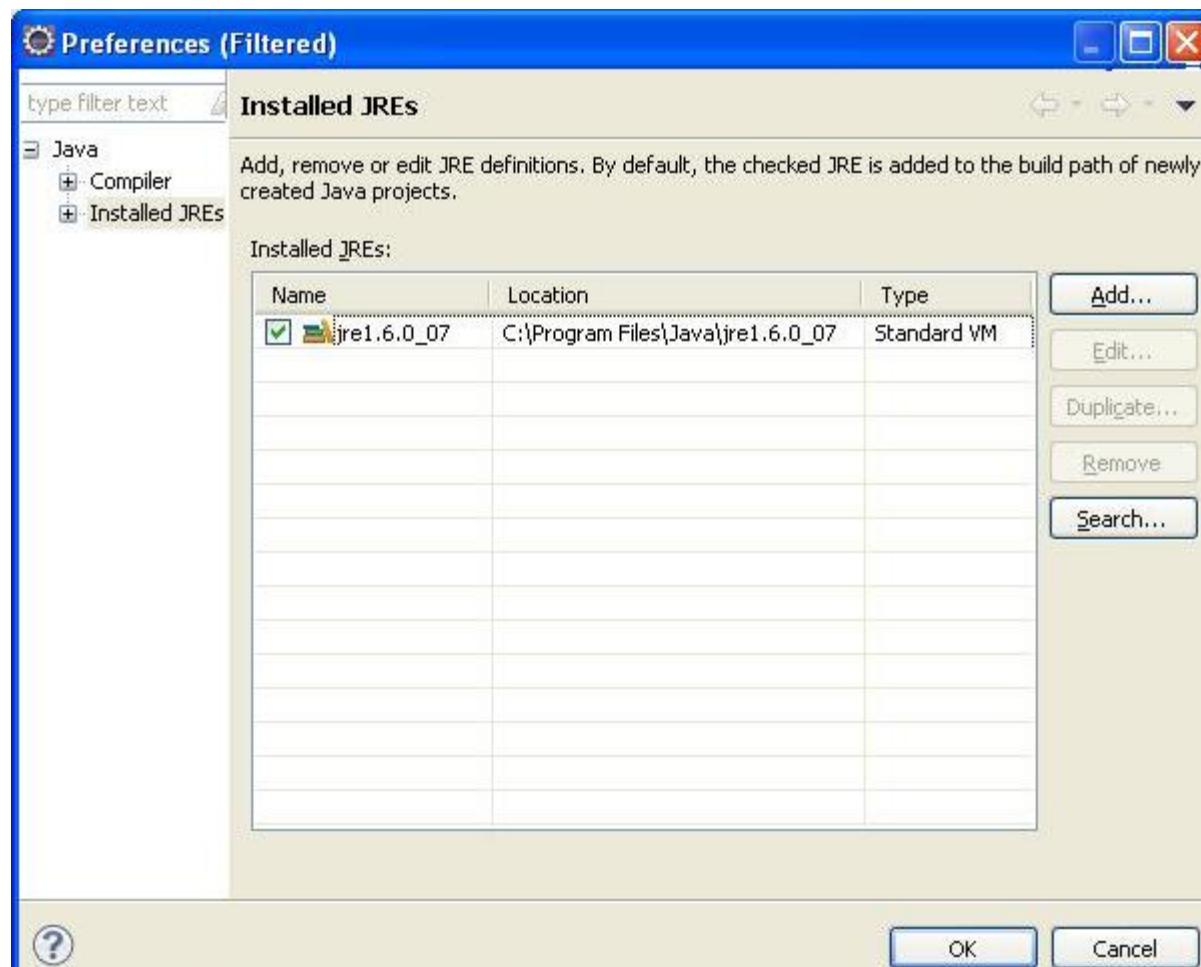
The Preferences screen appears.

2. Under the type filter text, select **Java > Installed JREs**.

A list of JREs that are exposed to the Eclipse IDE is displayed as shown in [JRE Options Dialog Box](#).

3. Select the **JRE 1.6** check box and click **OK**.

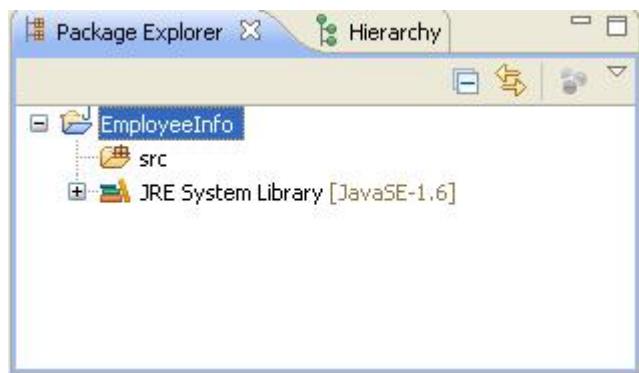
## Figure 55 JRE Options Dialog Box



5. Click **Finish** in Figure 54 to create the Java project.

The structure of the EmployeeInfo Java project appears in the Java Project Structure.

**Figure 56 Java Project Structure Dialog Box**



## Adding Dependency JAR Files to the Project Library

To compile and link the project, add the Hibernate dependency JAR files, listed in [Table 7](#), to the Java Build Path of the EmployeeInfo project.

**Table 7 Hibernate Dependency JAR Files**

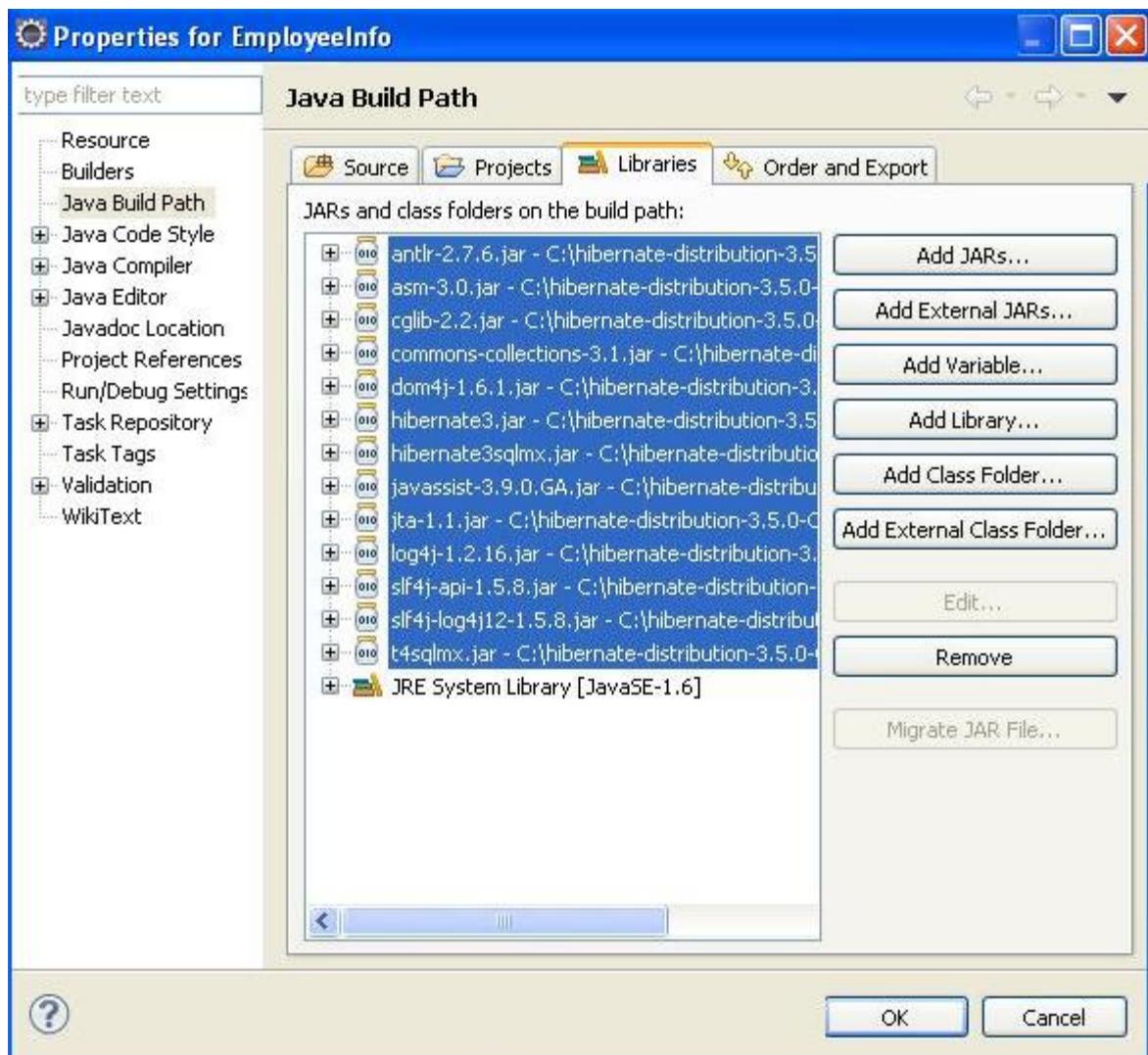
Dependency JAR Files	Source Location
antlr-2.7.6.jar	<Hibernate_Home>\lib\required
asm-3.1.jar	< Maven Repository >\asm\asm\3.0
cgleib-2.1.3.jar	<Hibernate_Home>\lib\bytecode\cglib
commons-collections-3.1.jar	<Hibernate_Home>\lib\required
commons-logging-1.0.4.jar	< Maven Repository >\commons-logging\commons-logging\1.0.4
dom4j-1.6.1.jar	<Hibernate_Home>\lib\required
hibernate3.jar	<Hibernate_Home>
hibernate3sqlmx.jar	<Hibernate_Home>\lib
jta-1.1.jar	<Hibernate_Home>\lib\required
log4j-1.2.14.jar	< Maven Repository >\log4j\log4j\1.2.14

To add the dependency JAR files in the EmployeeInfo project library path:

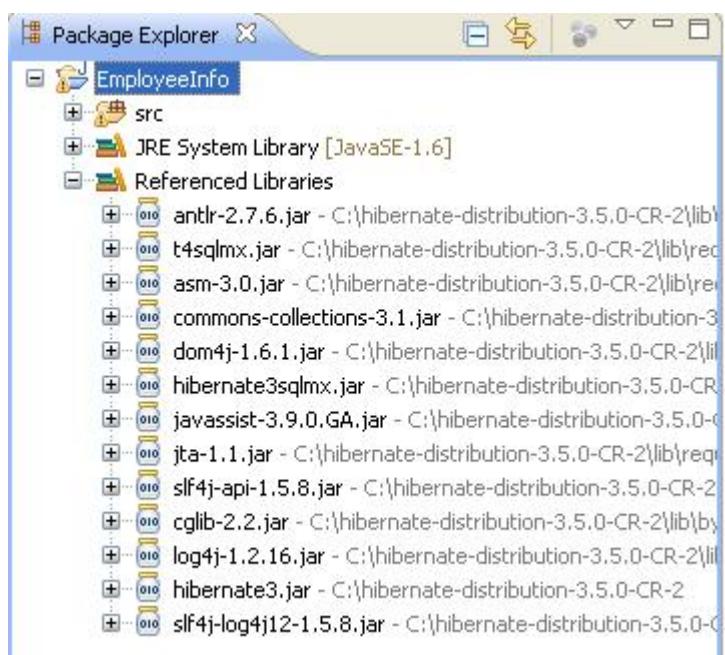
1. On the Project Explorer frame, right-click **EmployeeInfo** and select **Properties**.  
The Project Properties dialog box appears.
2. From the type filter text, select **Java Build Path**.
3. Click the **Libraries** tab.
4. Click **Add External JARs** to add each of the dependency JAR files listed in [Table 7](#). Browse to specify the location for each of the dependency JAR files.
5. Click **OK**.

The projects library path is populated.

**Figure 57 EmployeeInfo Project Libraries Dialog Box**



**Figure 58 EmployeeInfo Project Structure Dialog Box**

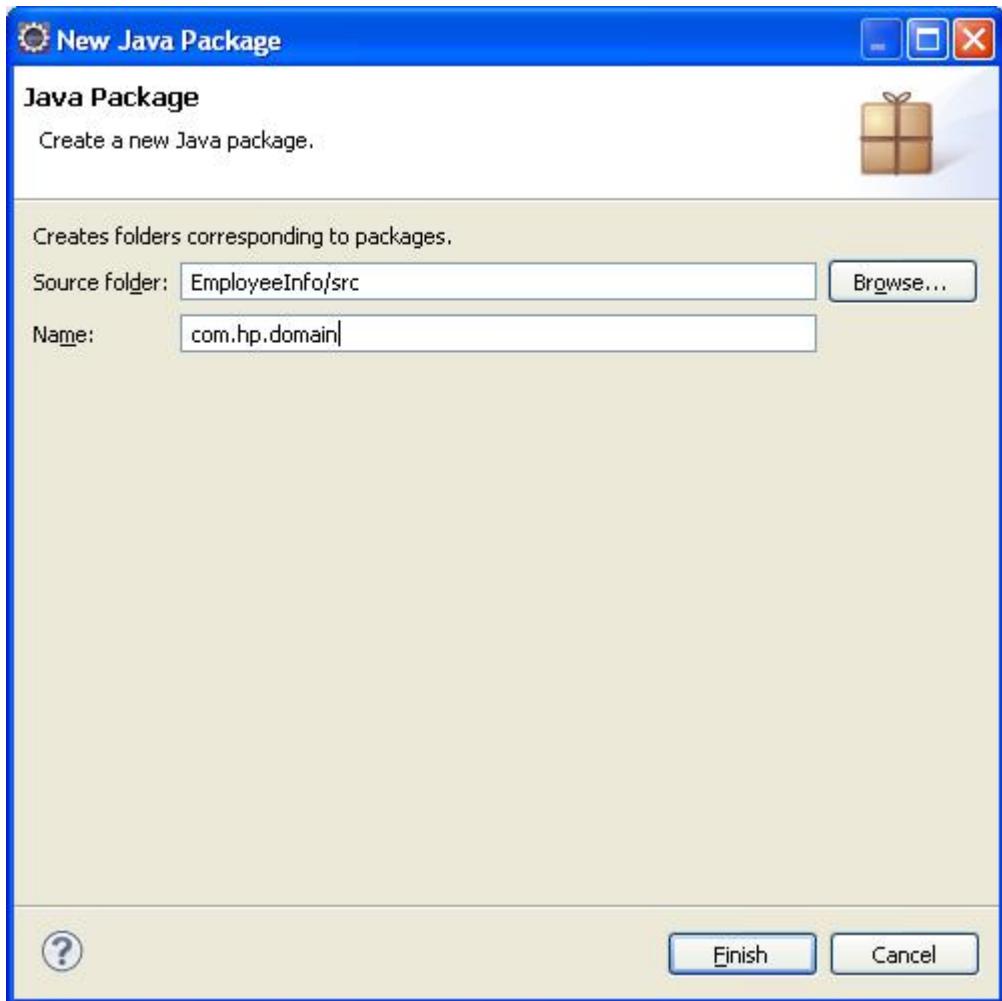


## Creating the Package for the EmployeeInfo Application

To create the package for the EmployeeInfo application, complete the following steps:

1. On the Project Explorer frame, right-click **EmployeeInfo** and select **New > Package**.  
The New Java Package dialog box appears.
2. In the Name field, type `com.hp.domain` and click **Finish**.

**Figure 59** New Java Package Dialog Box

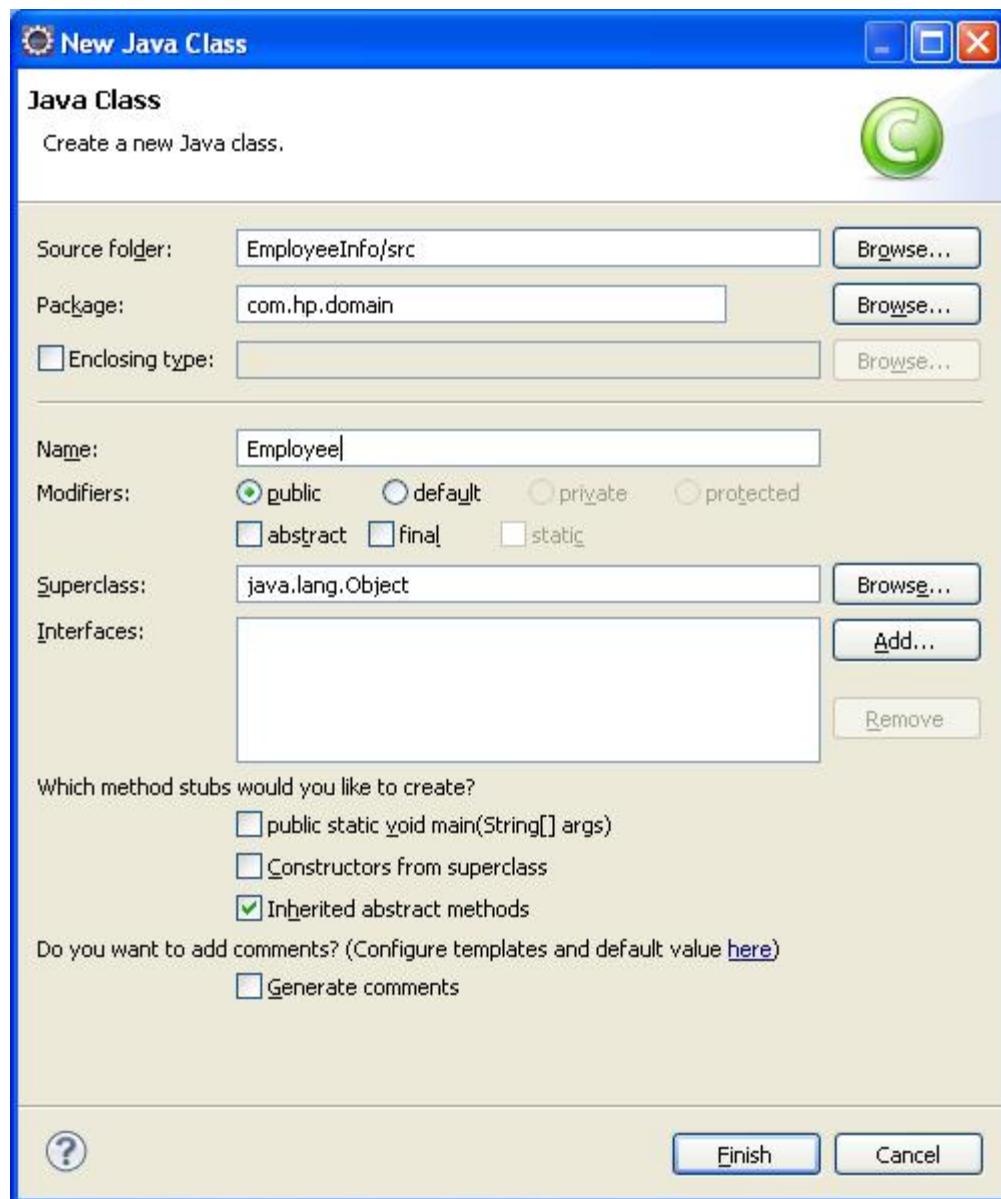


## Creating the Employee.java Class File

To create the `Employee.java` class file, complete the following steps:

1. On the Project Explorer frame, right-click the `com.hp.domain` package and select **New > Class**.  
The New Java Class dialog box appears.
2. In the Name field, type `Employee` and click **Finish**. Verify that the Source folder is `EmployeeInfo/src`.

**Figure 60 New Java Class Dialog Box**



The Employee.java class file is created.

3. Modify the Employee.java class file to add the application attributes and their getter and setter methods, so that it appears as follows:

```
package com.hp.domain;

public class Employee {
    private int id;
    private String fname;
    private String lname;
    private int age;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
        this.fname = fname;
    }
    public String getLname() {
        return lname;
    }
    public void setLname(String lname) {
        this.lname = lname;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

## Creating the Main Class

To create a Main class, complete the following steps:

1. Create the com.hp.imp Java package, as explained in “[Creating the Package for the EmployeeInfo Application](#)” (page 204).
2. Create Test.java main class file in the com.hp.imp package, as explained in “[Creating the Employee.java Class File](#)” (page 204).
3. Replace the contents of the Test.java file with the following code to implement the functionality for the Hibernate session factory and the Main method:

```
package com.hp.imp;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import com.hp.domain.Employee;

public class Test {

    public Session openSession() {
        SessionFactory sessionFactory;
        Configuration cfg = new Configuration().configure();
        sessionFactory = cfg.buildSessionFactory();
        Session session = sessionFactory.openSession();
        return session;
    }

    public void testFindByHQL(TestManager manager) {
```

```

Employee emp = manager.getEmpl();
System.out.println("Employee found using HQL" + "\n"
    + "\to_id=" + emp.getId());
}

public void createEmp(TestManager manager) {
    manager.createEmpl();
}

public static void main(String[] args) {
    Test client = new Test();
    Session session = client.openSession();
    TestManager manager = new TestManager(session);
    client.createEmp(manager);
    client.testFindByHQL(manager);
    session.close();
}

}

```

The Test.java main class file is created.

## Implementing the Business Logic

The business logic for the EmployeeInfo application is implemented in the TestManager.java class file. To do so, complete the following steps:

1. Create the TestManager.java class file in the com.hp.imp package, as explained in ["Creating the Employee.java Class File" \(page 204\)](#).
2. Add the method to perform an insert operation on the database using Hibernate.
3. Add the method to perform a select operation on the database using the HQL.

The TestManager.java class must appear as:

```

package com.hp.imp;

import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.hp.domain.Employee;

public class TestManager {

    private Session session = null;
    private Transaction tx = null;

    public TestManager(Session session) {
        if (session == null)
            throw new RuntimeException(
                "Invalid session object. Cannot instantiate the EmpManager");
        this.session = session;
    }

    public Employee getEmpl() {
        System.out.println(System.currentTimeMillis());
        List l = session.createQuery("from Employee").list();

        java.util.Iterator emps = l.iterator();

```

```

Employee emp = null;
do {
    if (emps.hasNext()) {
        emp = (Employee) emps.next();
        System.out.println("-----" + emp.getId());
        System.out.println("-----" + emp.getFname());
    }
} while (emps.hasNext());
return (emp);
}

public void createEmpl() {
    try {
        tx = session.beginTransaction();
        Employee emp = new Employee();

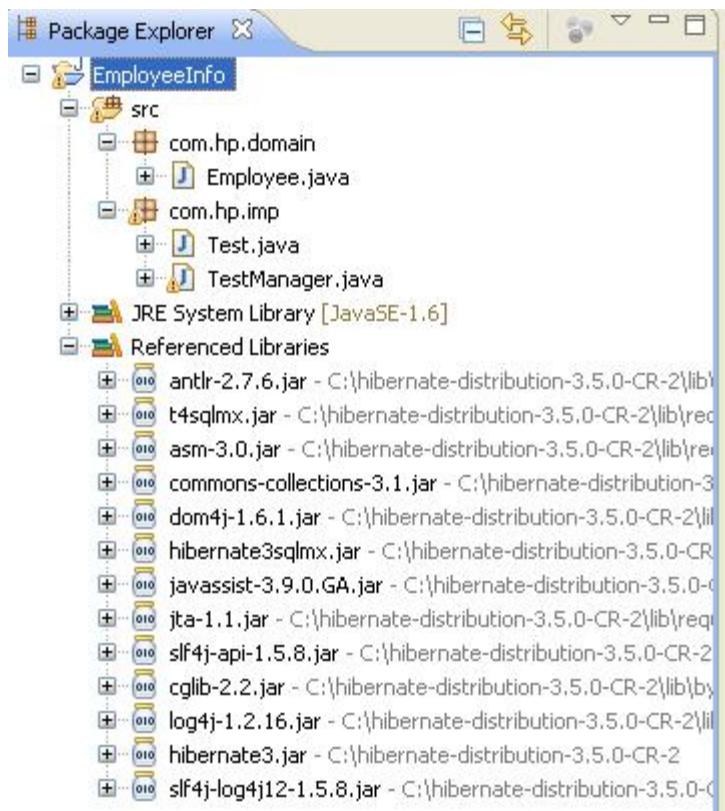
        emp.setAge(23);
        emp.setFname("Test");
        emp.setLname("Data");
        session.save(emp);

        tx.commit();
    } catch (HibernateException e) {
        if (tx != null)
            try {
                tx.rollback();
            } catch (HibernateException e1) {
                System.out.println("rollback not successful");
            }
    }
}
}

```

After adding the necessary packages and the classes under them, the EmployeeInfo project appears as:

**Figure 61 EmployeeInfo Project Dialog Box**



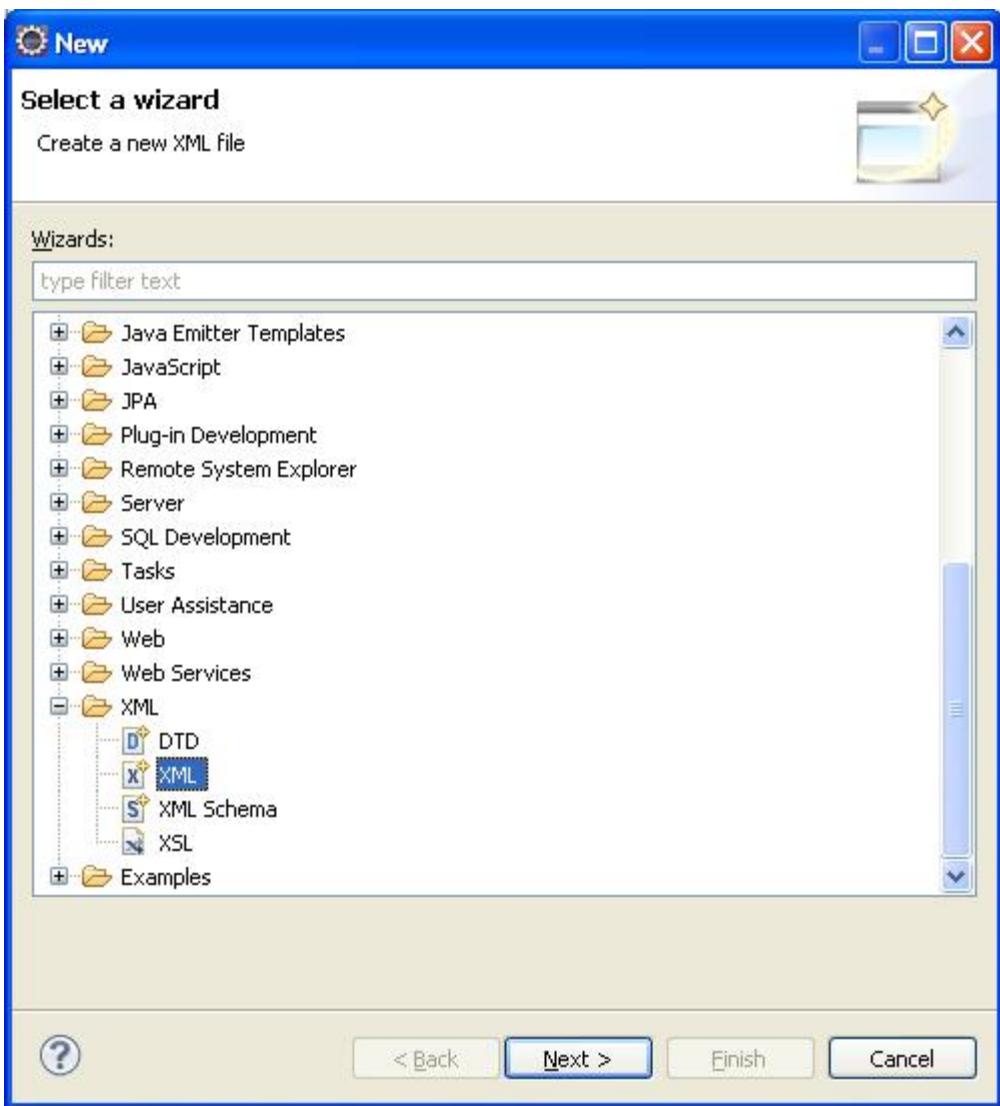
### Creating the Hibernate Mapping File

The Hibernate mapping file (`.hbm`) maps the employee objects to their respective columns in the SQL/MX database tables.

To create the `Employee.hbm.xml` file in the `EmployeeInfo/src` directory:

1. On the Project Explorer frame, right-click **EmployeeInfo** and select **New > Other**.  
The New dialog box appears.
2. From the list of folders, select **XML > XML** and click **Next**.

**Figure 62 New Dialog Box**

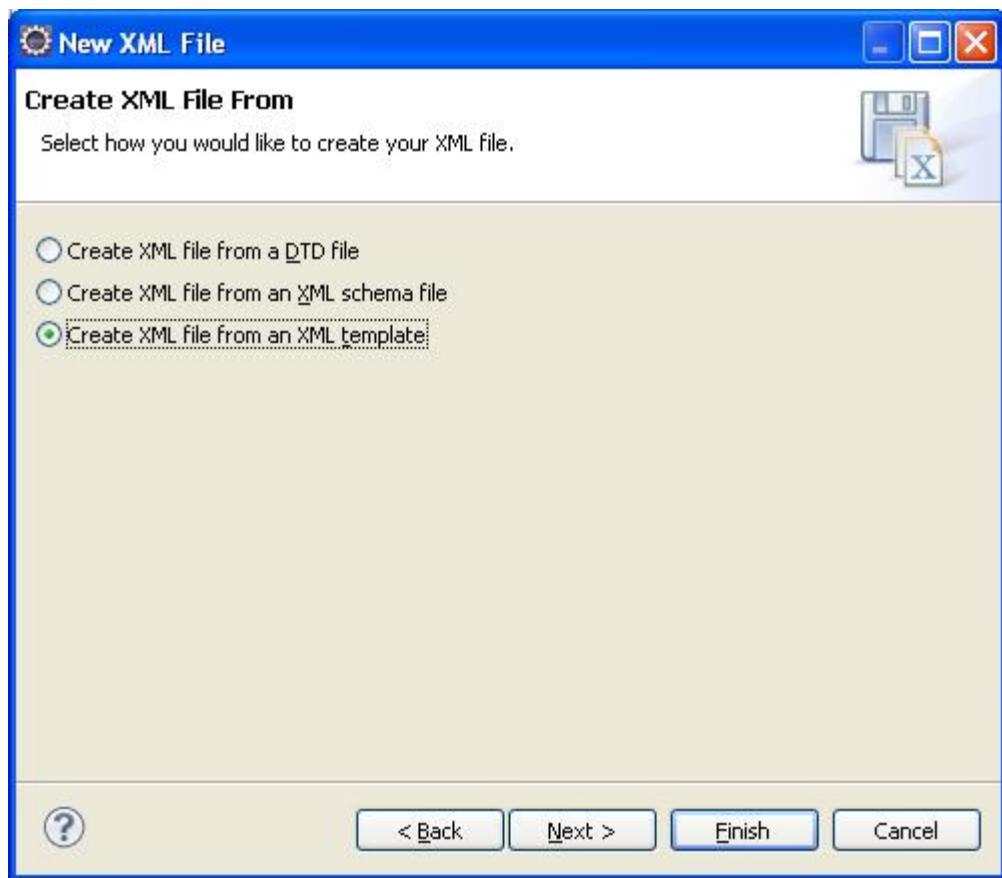


The New XML File: Create XML dialog box appears.

3. In the File name field, type `Employee.hbm.xml` and ensure that the parent folder is set to EmployeeInfo/src. Click **Next**.

The New XML File: Create XML dialog box appears again.

**Figure 63 New XML File: Create XML Dialog Box**



4. Select the **Create XML from an XML template** and click **Finish**.

The Employee.hbm.xml file is created in the EmployeeInfo/src directory.

5. Modify the Employee.hbm.xml file by adding a Hibernate mapping for the Employee.java class.

The Employee.hbm.xml file now appears as:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping
  package="com.hp.domain">

  <class name="Employee" table="Emp" >

    <id name="id">
      <generator class="increment"/>
    </id>

    <property name="fname"
      not-null="true"
      length="15"
      column="fname"/>

    <property name="lname"
      not-null="true"
      length="15"
      column="lname"/>

  </class>
</hibernate-mapping>
```

```
length="15"
column="lname"/>

<property name="age" column="age" />

</class>
</hibernate-mapping>
```

---

**NOTE:** The generator class for tag `<id>` is `increment`. This is to use the auto-increment feature of Hibernate.

---

## Setting Hibernate Configurations

To set up Hibernate configurations, create the following files:

- `hibernate.cfg.xml`
- `hibernate.properties`

`hibernate.cfg.xml`

Hibernate uses the `hibernate.cfg.xml` file to set up the environment.

To create and modify `hibernate.cfg.xml`:

1. Create the `hibernate.cfg.xml` file in the `EmployeeInfo/src` directory, as explained in “[Creating the Hibernate Mapping File](#)” (page 209).

2. Modify the hibernate.cfg.xml file:

- Add a reference for the Hibernate mapping file under the `<session-factory>` tag.  
`<mapping resource="Employee.hbm.xml" />`
- Add a reference for the Hibernate dialect file by including the following property tag under the `<session-factory>` tag.  
`<property name="dialect">  
 org.hibernate.dialect.SqlmxDialect  
</property>`
- Enable auto-creation of database tables using the Hibernate mapping file by including the following property tag under the `<session-factory>` tag.  
`<property name="hibernate.hbm2ddl.auto">create</property>`

The hibernate.cfg.xml file now appears as:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>

    <property name="hibernate.hbm2ddl.auto">create</property>
    <property name="show_sql">true</property>

    <!-- SQL Dialect to use. Dialects are database specific -->
    <property name="dialect">
        org.hibernate.dialect.SqlmxDialect
    </property>
    <!-- Mapping files -->
    <mapping resource="Employee.hbm.xml" />
</session-factory>
</hibernate-configuration>
```

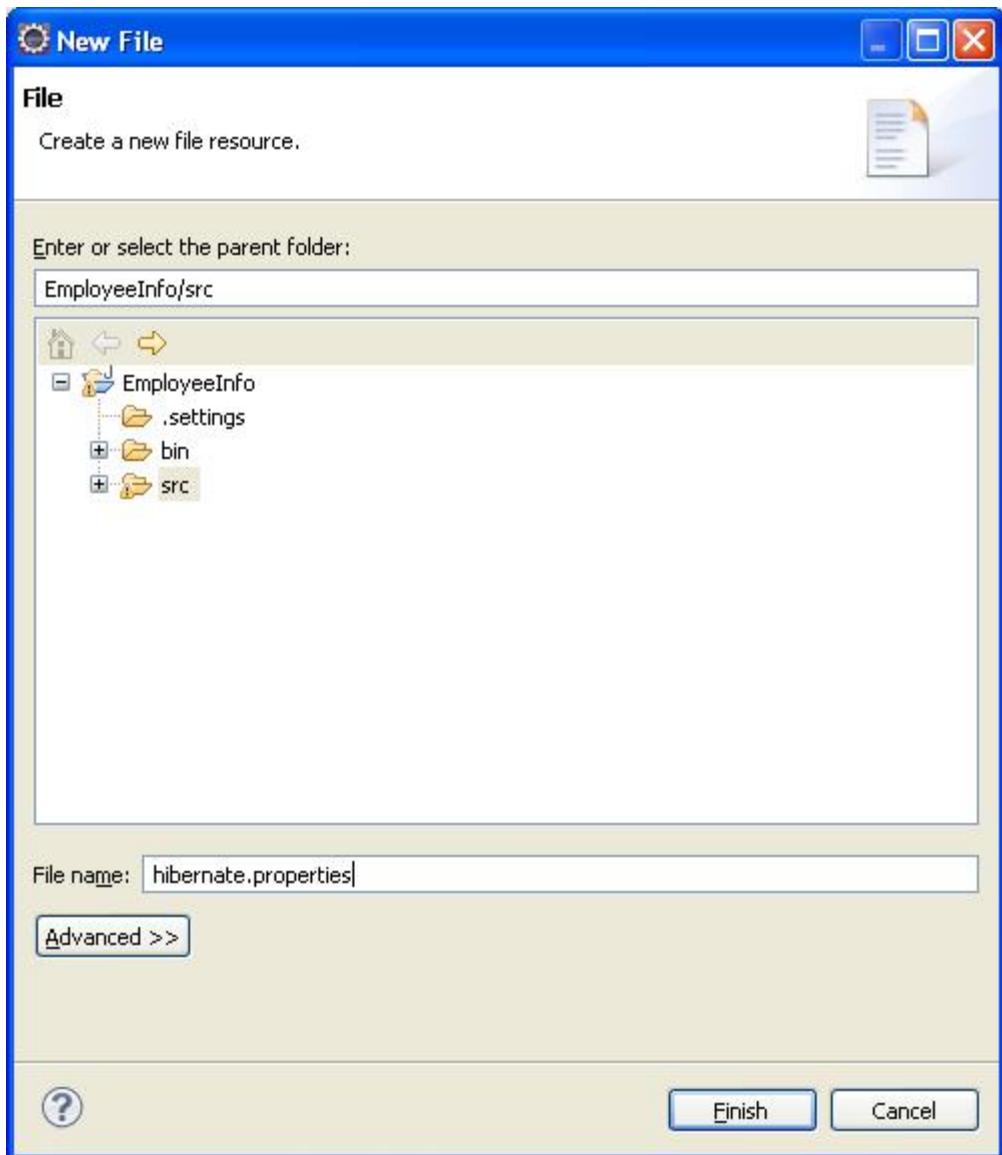
#### hibernate.properties

The hibernate.properties file includes information on the SQL/MX database.

To create the hibernate.properties file:

1. On the Project Explorer frame, right-click **EmployeeInfo** and select **New > File**.  
The New File dialog box appears.
2. Change the parent folder to the `EmployeeInfo/src` directory. In the **File name** field, type `hibernate.properties`. Click **Finish**.  
The hibernate.properties file is created.

**Figure 64 New File Dialog Box**



3. Modify the hibernate.properties file depending on the JDBC driver type you plan to use.

- To use the JDBC Type 2 driver, the SQL/MX settings for JDBC Type 2 driver in the hibernate.properties file must appear as:

```
-----
# SQL/MX Settings for JDBC Type 2 Driver

hibernate.connection.driver_class=com.tandem.sqlmx.SQLMXDriver
hibernate.connection.url=jdbc:sqlmx://
hibernate.connection.username=
hibernate.connection.password=
hibernate.connection.catalog=employeeinfocat
hibernate.connection.schema=employeeinfosch
```

---

**NOTE:** Because JDBC Type 2 driver is located on the NonStop system, you need not type the username and password in the jdbc.username and jdbc.password fields.

- To use the JDBC Type 4 driver, the SQL/MX settings for JDBC Type 4 driver in the hibernate.properties file must appear as:

```
-----
# SQL/MX Settings for JDBC Type 4 Driver

hibernate.connection.driver_class= com.tandem.t4jdbc.SQLMXDriver
hibernate.connection.url= jdbc:t4sqlmx:// <HP NonStop System IP Address>:<Port No.>
hibernate.connection.username=<HP NonStop Username>
hibernate.connection.password=<HP NonStop Password>
hibernate.connection.catalog=employeeinfocat
hibernate.connection.schema=employeeinfosch
```

---

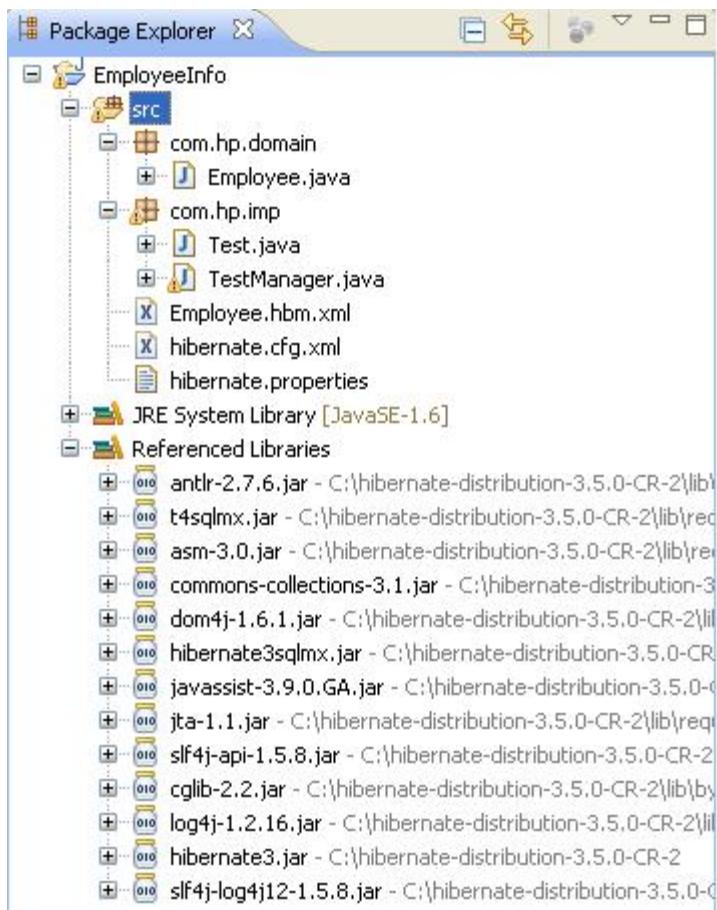
**NOTE:**

- If you are using the JDBC Type 4 driver, type the JDBC URL (NonStop system IP Address and Port Number of the JDBC data source), NonStop system username, and password.
- The name of the database catalog used in the example is employeeinfocat and the schema name is employeeinfosch. If these database catalog and schema names conflict with any of the existing catalog and schema names on the NonStop system, modify the database catalog and schema names in the hibernate.properties file.

---

This completes the development of the EmployeeInfo project. The structure of the EmployeeInfo project must appear as follows:

**Figure 65 EmployeeInfo Project Structure Dialog Box**



## Creating the EmployeeInfo Application JAR File

To create the JAR file of the EmployeeInfo application, complete the following steps:

1. On the Project Explorer frame, right-click **EmployeeInfo** and select **Export**.  
The Export dialog box appears.
2. From the list of folders, select **Java > JAR file**.

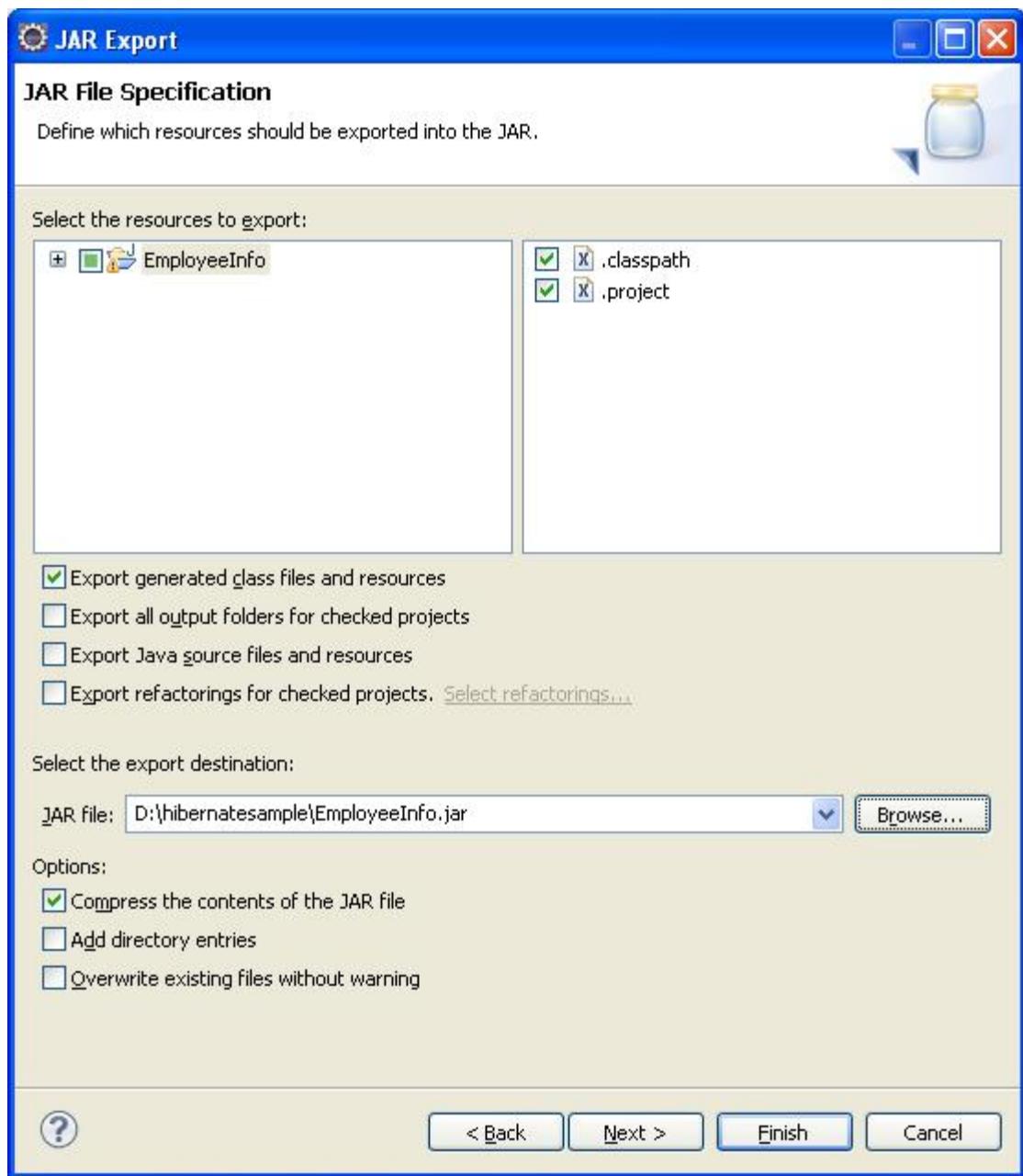
**Figure 66 Export Dialog Box**



3. Click **Next**.

The JAR Export dialog box appears. Browse to the location where you want to save the JAR file on the Windows system.

**Figure 67 The JAR Export Dialog Box**



4. Click **Finish**.

The EmployeeInfo application JAR file (`EmployeeInfo.jar`) is created.

## Setting Up the NonStop Environment

To set up NonStop environment, complete the following steps:

1. Create the SQL/MX database catalog and schema on the NonStop system with names specified in the `hibernate.properties` file.

The MXCI commands to create the database catalog and schema are as follows:

```
mxci>> create catalog employeeinfocat LOCATION <node.$vol>;  
mxci>> create schema employeeinfocat.employeeinfosch  
AUTHORIZATION "<user>" LOCATION ZSD<subvol reference>;
```

---

**NOTE:** By default, the `hibernate.properties` file has the catalog name as `employeeinfocat` and schema name as `employeeinfosch`. If you have modified the catalog name and schema name in the `hibernate.properties` file, run the MXCI commands.

---

2. Copy the created project JAR file `EmployeeInfo.jar` from the Windows system to `<NonStop SASH Home>/hibernate/my_samples/employeeinfo` on the NonStop system.
3. Copy the `ei_setenv` script from `<My SASH Home>\hibernate\getting-started\envsetup` to `<NonStop SASH Home>/hibernate/my_samples/employeeinfo` on the NonStop system.

**NOTE:** To view the contents of the `ei_setenv` script, see “[Hibernate Environment Setup Script](#)” (page 228).

---

4. Edit the `ei_setenv` script:

1. Set the value of the `hibernatehome` variable to the `<NonStop Hibernate Home> OSS` directory.

For example, if `<NonStop Hibernate Home>` is `/usr/tandem/sash/hibernate-distribution-3.5.1-Final`, the `hibernatehome` variable must be set to `hibernatehome=/usr/tandem/sash/hibernate-distribution-3.5.1-Final`.

2. Set the value of the `mavenrepository` variable to the `<NonStop Hibernate Maven Repository> OSS` directory.

For example, if `<NonStop Hibernate Maven Repository>` is `/usr/tandem/sash/.m2`, the `mavenrepository` variable must be set to `mavenrepository=/usr/tandem/sash/.m2`.

3. Set the value of the JDBC driver locations.
    1. To use the JDBC Type 2 driver, set the value of the `t2jdbc` variable to the `<JDBC T2 Installation Directory>` OSS directory.  
For example, if `<JDBC T2 Installation Directory>` is `/usr/tandem/jdbcMx/current`, the `t2jdbc` variable must be set to `t2jdbc=/usr/tandem/jdbcMx/current`.
    2. To use the JDBC Type 4 driver:
      - Copy the JDBC Type 4 driver JAR file, `t4sqlmx.jar`, from `<JDBC T4 Installation Directory>\lib` the Windows directory to an OSS directory on the NonStop system.
      - Replace `<NonStop T4 Location>` with the absolute path of the OSS directory where you have kept the `t4sqlmx.jar` file.  
For example, if `<NonStop T4 Location>` is `/home/sash_usr/sash/jdbct4`, the `t4jdbc` variable must be set to `t4jdbc=/home/sash_usr/sash/jdbct4`.
- 

**NOTE:** To view the contents of the `ei_setenv` script file, see “[Hibernate Environment Setup Script](#)” (page 228).

3. Set the execute permissions of the `ei_setenv` script file using the OSS command:  
`OSS> chmod +x ei_setenv`
5. Run the `ei_setenv` script file to set the JDBC environment and include the dependency JAR files in the CLASSPATH.
  - To use the JDBC Type 2 driver, type:  
`OSS> ./ei_setenv T2`
  - To use the JDBC Type 4 driver, type:  
`OSS> ./ei_setenv T4`

## Running EmployeeInfo on NonStop

To run the EmployeeInfo on NonStop systems, complete the following steps:

1. Go to the `<NonStop SASH Home>/hibernate/my_samples/employeeinfo` directory on the NonStop system.
2. Un-JAR the contents of the application JAR file `EmployeeInfo.jar`:  
`OSS> jar -xvf EmployeeInfo.jar`
3. Run the EmployeeInfo application:  
`OSS> java com.hp.imp.Test`

The following output appears:

```
Hibernate: select max(id) from Emp
Hibernate: insert into Emp (fname, lname, age, id) values (?, ?, ?, ?)
1237276999876
Hibernate: select employee0_.id as id0_, employee0_.fname as fname0_, employee0_.
.lname as lname0_, employee0_.age as age0_ from Emp employee0_
-----1
-----Test
Employee found using HQL
    o_id=1
```

# E Customizing Sample Applications

## Customizing Caveat Emptor

To customize the Caveat Emptor sample application to run on NonStop systems, three files were added and one file was modified.

### Added Files

- `hibernate.properties`
- `setenv`
- `Caveatemptor_script.sql`

### Modified File

- `Main.java`

## Added Files

### `hibernate.properties`

(*<My SASH Home>\hibernate\samples\eg\src\main\resources\hibernate.properties*)

This file provides hibernate configuration to Caveat Emptor.

- The content of the `hibernate.properties` file is:

```
##-----  
# SQL/MX Settings for JDBC Type 2 Driver  
#hibernate.dialect org.hibernate.dialect.SqlmxDialect  
#hibernate.connection.driver_class com.tandem.sqlmx.SQLMXDriver  
#hibernate.connection.url jdbc:sqlmx://  
#hibernate.connection.username  
#hibernate.connection.password  
#hibernate.connection.catalog auctioncat  
#hibernate.connection.schema auctionsch  
  
##-----  
# SQL/MX Settings for JDBC Type 4 Driver  
#hibernate.dialect=org.hibernate.dialect.SqlmxDialect  
#hibernate.connection.driver_class com.tandem.t4jdbc.SQLMXDriver  
#hibernate.connection.url jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.>  
#hibernate.connection.username <HP NonStop Username>  
#hibernate.connection.password <HP NonStop Password>  
#hibernate.connection.catalog auctioncat  
#hibernate.connection.schema auctionsch  
  
#For JDBC Type 4 Driver:  
#  
#<HP NonStop System IP Address> - This is the IP address of your NonStop system  
#<Port No.> - This is the Port Number of JDBC Data Source  
#<HP NonStop Username> - This is the HP NonStop system UserName  
#<HP NonStop Password> - This is the HP NonStop system Password
```

### `setenv`

(*<My SASH Home>\hibernate\samples\eg\setup\setenv*)

This script file sets the Hibernate and JDBC libraries in the NonStop environment.

The content of this script file is:

```
#!/bin/ksh  
hibernatehome=<NonStop Hibernate Home>  
mavenrepository=<NonStop Hibernate Maven Repository>  
t4jdbc=<NonStop SASH Home>  
t2jdbc=<JDBC T2 Installation Directory>  
  
Usage="\n      Usage: ./setenv <T2/T4>\n\n      For JDBC/MX Type2 driver use 'T  
2'\n      For JDBC/MX Type4 driver use 'T4'\n"  
  
usage()
```

```

{
echo $Usage
exit
}

if [ $# -ne 1 ] ;
then
usage
fi

if [ $1 != "T2" ] && [ $1 != "T4" ] ;
then
usage
fi

#####
#CLASSPATHs for JDBC/MX Type 2 Drivers

if [ $1 = T2 ] ;
then
export CLASSPATH=$CLASSPATH:$t2jdbc/lib/jdbcMx.jar
export CLASSPATH=$CLASSPATH:$t2jdbc/lib/libjdbcMx.so
export _RLD_LIB_PATH=$t2jdbc/lib
fi

#####
#CLASSPATHs for JDBC/MX Type 4 Drivers

if [ $1 = T4 ] ;
then
export CLASSPATH=$CLASSPATH:$t4jdbc/hibernate/samples/eg/t4sqlmx.jar
fi

#####
#CLASSPATHs for hibernate and third-party libraries

export CLASSPATH=$CLASSPATH:$mavenrepository/antlr/antlr/2.7.6/antlr-2.7.6.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/asm/asm/3.1/asm-3.1.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/cglib/cglib/2.2/cglib-2.2.jar
export
CLASSPATH=$CLASSPATH:$mavenrepository/commons-collections/commons-collections/3.2/commons-collections-3.2.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/commons-logging/commons-logging/1.0.4/commons-logging-1.0.4.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/dom4j/dom4j/1.6.1/dom4j-1.6.1.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/javassist/javassist/3.9.0.GA/javassist-3.9.0.GA.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/javax/transaction/jta/1.1/jta-1.1.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/log4j/log4j/1.2.14/log4j-1.2.14.jar
export CLASSPATH=$CLASSPATH:$hibernatehome/hibernate3.jar
export CLASSPATH=$CLASSPATH:$hibernatehome/lib/hibernate3sqlmx.jar
export CLASSPATH=$CLASSPATH:$hibernatehome/lib/required/slf4j-api-1.5.8.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/org/slf4j/slf4j-log4j12/1.5.8/slf4j-log4j12-1.5.8.jar

#####

```

## Caveatemptor\_script.sql

(<My SASH Home>\hibernate\samples\eg\dbconfig\caveatemptor\_script.sql)  
This script file creates database catalog, schema, and database tables for Caveat Emptor.

The content of this file is:

```

log caveat.log;
drop schema auctioncat.auctionsch cascade;
drop catalog auctioncat;
create catalog auctioncat LOCATION <node.$vol>;
create schema auctioncat.auctionsch AUTHORIZATION "<user>" LOCATION ZSD<subvol reference>;
set schema auctioncat.auctionsch;
create table auctionitem (id bigint not null, seller bigint not null,
shortDescription varchar(200) not null, description varchar(1000),
ends timestamp, "CONDITION" integer, successfulBid bigint, primary key
(id)) LOCATION <$datavol> ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;
create table auctionuser (id bigint not null, userName varchar(10) not null, "password" varchar(15) not
null, email varchar(255), firstName varchar(50) not null, "initial" char(1), lastName varchar(50) not null,
primary key (id)) LOCATION <$datavol> ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;
create table bid (id bigint not null, isBuyNow char(1) not null, item bigint not null, amount float not
null, "datetime" timestamp not null, bidder bigint not null, primary key (id)) LOCATION <$datavol>
ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;
create unique index auctionitem_seller on auctionitem (seller, shortDescription)
LOCATION <$datavol> ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;
alter table auctionitem add constraint FKEECF7FD6F65B1FAF foreign key (successfulBid) references bid;
alter table auctionitem add constraint FKEECF7FD61107FE9B foreign key (seller)
references auctionuser;
create unique index auctionuser_username on auctionuser (userName)
LOCATION <$datavol> ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;
create unique index auctionitem_item_amount on bid (item, amount)
LOCATION <$datavol> ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;
alter table bid add constraint FK17CFDF43A3910 foreign key (bidder)
references auctionuser;

```

```

alter table bid add constraint FK17CFDEE6E7E98 foreign key (item)
references auctionitem;
create table hibernate_unique_key (next_hi integer )
LOCATION <$datavol> ATTRIBUTE_EXTENT (16, 64), MAXEXTENTS 160;
insert into hibernate_unique_key values (1);

```

## Modified Files

### Main.java

(*<My SASH Home>\hibernate\samples\eg\src\main\java\org\hibernate\auction>Main.java*)  
This class file performs all database operations for Caveat Emptor.

#### Changes to the Main.java file

Changed the value of `Environment.HBM2DDL_AUTO` property to `false` to avoid auto creation of database tables by Hibernate.

---

**NOTE:** The default value of `Environment.HBM2DDL_AUTO` is `create`.

---

Before the change:

```

Configuration cfg = new Configuration()
    .addClass(AuctionItem.class)
    .addClass(Bid.class)
    .addClass(User.class)
    .setProperty(Environment.HBM2DDL_AUTO, "create");
//cfg.setProperty("hibernate.show_sql", "true");

```

After the change:

```

Configuration cfg = new Configuration()
    .addClass(AuctionItem.class)
    .addClass(Bid.class)
    .addClass(User.class)
    .setProperty(Environment.HBM2DDL_AUTO, "false");
//cfg.setProperty("hibernate.show_sql", "true");

```

## Customizing EventManager

To customize the EventManager sample application to run on NonStop systems, two directories and two files were added, and one file was modified.

### Added Directories:

- `/dbconfig`
- `eventmanager_script.sql`

### Added Files

- `hibernate.properties`
- `setenv`

### Modified File

- `Hibernate.cfg.xml`

## Added Directories

### /dbconfig

(*<My SASH Home>\hibernate\samples\web\dbconfig*)

This directory contains the database script that is required to set up the EventManager database on SQL/MX.

### /etc

(*<My SASH Home>\hibernate\samples\web*)

This directory is included in the `SAMPLES.zip` file and contains the `hibernate3sqlmx.jar` file.

## Added Files

### `hibernate.properties`

(*<My SASH Home>\hibernate\samples\web\src\main\resources\hibernate.properties*)

This file provides hibernate configuration to EventManager.

- The content of the `hibernate.properties` file is:

```
##-----  
# SQL/MX Settings for JDBC Type 2 Driver  
#hibernate.dialect org.hibernate.dialect.SqlmxDialect  
#hibernate.connection.driver_class com.tandem.sqlmx.SQLMXDriver  
#hibernate.connection.url jdbc:sqlmx://  
#hibernate.connection.username  
#hibernate.connection.password  
#hibernate.connection.catalog eventcat  
#hibernate.connection.schema eventsch  
  
#-----  
# SQL/MX Settings for JDBC Type 4 Driver  
#hibernate.dialect org.hibernate.dialect.SqlmxDialect  
#hibernate.connection.driver_class com.tandem.t4jdbc.SQLMXDriver  
#hibernate.connection.url jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.>  
#hibernate.connection.username <HP NonStop Username>  
#hibernate.connection.password <HP NonStop Password>  
#hibernate.connection.catalog eventcat  
#hibernate.connection.schema eventsch  
  
hibernate.show_sql true  
hibernate.hbm2ddl.auto false  
  
hibernate.cache.provider_class org.hibernate.cache.NoCacheProvider  
hibernate.current_session_context_class org.hibernate.context.ManagedSessionContext  
  
#For JDBC Type 4 Driver:  
#  
#<HP NonStop System IP Address> - This is the IP address of your NonStop system  
#<Port No.> - This is the Port Number of JDBC Data Source  
#<HP NonStop Username> - This is the HP NonStop system UserName  
#<HP NonStop Password> - This is the HP NonStop system Password
```

### `eventmanager_script.sql`

(*<My SASH Home>\hibernate\samples\web\dbconfig\eventmanager\_script.sql*)

This script file creates database catalog, schema, and database tables for EventManager.

The content of this file is:

```
log caveat.log;  
  
drop schema eventcat.eventsch cascade;  
drop catalog eventcat;  
  
create catalog eventcat LOCATION <node.$vol>;  
create schema eventcat.eventsch AUTHORIZATION "<user>" LOCATION ZSD<subvol reference>;  
set schema eventcat.eventsch;  
  
create table EVENTS (EVENT_ID bigint not null, EVENT_DATE timestamp, title varchar(255),  
primary key (EVENT_ID))LOCATION <$datavol> ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;  
  
create table PERSON (PERSON_ID bigint not null, age integer, firstname varchar(255),  
lastname varchar(255), primary key (PERSON_ID)) LOCATION <$datavol> ATTRIBUTE EXTENT (16, 64),  
MAXEXTENTS 160;  
  
create table PERSON_EMAIL_ADDR (PERSON_ID bigint not null, EMAIL_ADDR varchar(255))  
LOCATION <$datavol> ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;  
  
create table PERSON_EVENT (EVENT_ID bigint not null, PERSON_ID bigint not null,  
primary key (PERSON_ID, EVENT_ID)) LOCATION <$datavol> ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;  
  
create unique index eventcat_events on EVENTS (EVENT_ID) LOCATION <$datavol>  
ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;
```

```

alter table PERSON_EMAIL_ADDR add constraint FKA54215FEB0A1327A foreign key (PERSON_ID)
references PERSON;

alter table PERSON_EVENT add constraint FKAD91D910A8B41A9A foreign key
(EVENT_ID) references EVENTS;

create unique index eventcat_person on PERSON(PERSON_ID) LOCATION <$datavol>
ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;

create unique index eventcat_person_event on PERSON_EVENT(EVENT_ID, PERSON_ID)
LOCATION <$datavol> ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;

create unique index eventcat_emails on PERSON_EMAIL_ADDR(PERSON_ID)
LOCATION <$datavol> ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;

create table hibernate_unique_key (next_hi integer ) LOCATION <$datavol>
ATTRIBUTE EXTENT (16, 64), MAXEXTENTS 160;

insert into hibernate_unique_key values (1);

```

## Modified File

### Hibernate.cfg.xml

(*<My SASH Home>\hibernate\samples\web\src\main\resources*)

This is the hibernate configuration file for EventManager that contains hibernate connection properties and hbm resource mapping.

#### Changes to the Hibernate.cfg.xml file

This file was modified to remove the default connection-related hibernate properties.

Before the change:

```

<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <!-- Database connection settings -->
    <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
    <property name="connection.url">jdbc:hsqldb:hsq://localhost</property>
    <property name="connection.username">sa</property>
    <property name="connection.password"></property>

    <!-- JDBC connection pool (use the built-in) -->
    <property name="connection.pool_size">2</property>

    <!-- SQL dialect -->
    <property name="dialect">org.hibernate.dialect.HSQLDialect</property>

    <!-- Enable Hibernate's current session context -->
    <property name="current_session_context_class">
      org.hibernate.context.ManagedSessionContext</property>

    <!-- Disable the second-level cache -->
    <property name="cache.provider_class">
      org.hibernate.cache.NoCacheProvider</property>

    <!-- Echo all executed SQL to stdout -->
    <property name="show_sql">true</property>

    <!-- Drop and re-create the database schema on startup -->
    <property name="hbm2ddl.auto">create</property>

    <mapping resource="org/hibernate/tutorial/domain/Event.hbm.xml"/>
    <mapping resource="org/hibernate/tutorial/domain/Person.hbm.xml"/>

  </session-factory>

</hibernate-configuration>

```

After the change:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <session-factory>
    <mapping resource="org/hibernate/tutorial/domain/Event.hbm.xml"/>
    <mapping resource="org/hibernate/tutorial/domain/Person.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

# F JDBC Configuration

This appendix describes the consolidated JDBC Type 2 driver configuration and the JDBC Type 4 driver configuration in `hibernate.cfg.xml` and `hibernate.properties` files.

- JDBC Type 2 driver configurations

The consolidated JDBC Type 2 driver configuration in `hibernate.cfg.xml` is as follows:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">com.tandem.sqlmx.SQLMXDriver</property>
        <property name="connection.url">jdbc:sqlmx://</property>
        <property name="connection.username"></property>
        <property name="connection.password"></property>
        <property name="dialect">org.hibernate.dialect.SqlmxDialect</property>
        <mapping resource=<Name of the .hbm file>" />
    </session-factory>
</hibernate-configuration>
```

The SQL/MX settings for JDBC Type 2 driver in `hibernate.properties` appear as follows:

```
#####
# SQL/MX Settings for JDBC Type 2 Driver
hibernate.dialect=org.hibernate.dialect.SqlmxDialect
hibernate.connection.driver_class=com.tandem.sqlmx.SQLMXDriver
hibernate.connection.url=jdbc:sqlmx://
hibernate.connection.username=
hibernate.connection.password=
```

- JDBC Type 4 driver configurations

The consolidated JDBC Type 4 driver configuration in `hibernate.cfg.xml` is as follows:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">com.tandem.t4jdbc.SQLMXDriver</property>
        <property name="connection.url">jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.></property>
        <property name="connection.username"><HP NonStop Username></property>
        <property name="connection.password"><HP NonStop Password></property>
        <property name="dialect">org.hibernate.dialect.SqlmxDialect</property>
        <mapping resource=<Name of the .hbm file>" />
    </session-factory>
</hibernate-configuration>
```

The SQL/MX settings for JDBC Type 4 driver in `hibernate.properties` appear as follows:

```
#####
# SQL/MX Settings for JDBC Type 4 Driver
hibernate.dialect=org.hibernate.dialect.SqlmxDialect
hibernate.connection.driver_class=com.tandem.t4jdbc.SQLMXDriver
hibernate.connection.url=jdbc:t4sqlmx://<HP NonStop System IP Address>:<Port No.>
hibernate.connection.username=<HP NonStop Username>
hibernate.connection.password=<HP NonStop Password>
```

# G Hibernate Environment Setup Script

The `ei_setenv` script file is used for setting up the Hibernate CLASSPATH and JDBC environments.  
The content of the file is as follows:

```
#!/bin/ksh
hibernatehome=<NonStop Hibernate Home>
mavenrepository=<NonStop Hibernate Maven Repository>
t4jdbc=<NonStop T4 Location>
t2jdbc=<JDBC T2 Installation Directory>

Usage=\n      Usage: ./ei_setenv T2<T2/T4>\n\n      For JDBC/MX Type2 driver use 'T
2'\n      For JDBC/MX Type4 driver use 'T4'\n

usage()
{
echo $Usage
exit
}

if [ $# -ne 1 ] ;
then
usage
fi

if [ $1 != "T2" ] && [ $1 != "T4" ] ;
then
usage
fi

#####
#CLASSPATHs for JDBC/MX Type 2 Drivers

if [ $1 = T2 ];
then
export CLASSPATH=$CLASSPATH:$t2jdbc/lib/jdbcMx.jar
export CLASSPATH=$CLASSPATH:$t2jdbc/lib/libjdbcMx.so
export _RLD_LIB_PATH=$t2jdbc/lib
fi

#####
#CLASSPATHs for JDBC/MX Type 4 Drivers

if [ $1 = T4 ];
then
export CLASSPATH=$CLASSPATH:$t4jdbc/t4sqlmx.jar
fi

#####
#CLASSPATHs for hibernate and third-party libraries

export CLASSPATH=$CLASSPATH:$mavenrepository/antlr/antlr/2.7.6/antlr-2.7.6.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/asm/asm/3.1/asm-3.1.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/cglib/cglib/2.2/cglib-2.2.jar
export
CLASSPATH=$CLASSPATH:$mavenrepository/commons-collections/commons-collections/3.2/commons-collections-3.2.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/commons-logging/commons-logging/1.0.4/commons-logging-1.0.4.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/dom4j/dom4j/1.6.1/dom4j-1.6.1.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/javassist/javassist/3.9.0.GA/javassist-3.9.0.GA.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/javax/transaction/jta/1.1/jta-1.1.jar
export CLASSPATH=$CLASSPATH:$mavenrepository/log4j/log4j/1.2.14/log4j-1.2.14.jar
export CLASSPATH=$CLASSPATH:$hibernatehome/hibernate3.jar
export CLASSPATH=$CLASSPATH:$hibernatehome/lib/hibernate3sqlmx.jar
export CLASSPATH=$CLASSPATH:$hibernatehome/lib/required/slf4j-api-1.5.8.jar
export CLASSPATH=$CLASSPATH:$hibernatehome/lib/slf4j-log4j12-1.5.0.jar

#####
```

---

## Part III MyFaces Framework

Part III describes how a MyFaces framework can be customized to work on a NonStop system. It includes the following chapters:

- [“MyFaces Overview” \(page 231\)](#)

This chapter provides an overview of the MyFaces framework, its architecture, key features, advantages and message flow.

- [“Installing MyFaces Framework” \(page 233\)](#)

This chapter helps you to install the MyFaces framework libraries and enables you to deploy and run sample MyFaces applications on your NonStop system.

- [“Configuring MyFaces Applications on NonStop Systems” \(page 240\)](#)

This chapter helps you to configure the MyFaces application on your NonStop system.

- [“Getting Started with MyFaces” \(page 252\)](#)

This chapter helps you to develop a web application on your Windows system using MyFaces and deploy and run the web application on your NonStop system.

- [“Installing MyFaces Trinidad Framework Libraries on NonStop” \(page 285\)](#)

This appendix helps you to install MyFaces Trinidad Framework Libraries on your NonStop system.

- [“Installing MyFaces Tomahawk Framework Libraries on NonStop” \(page 286\)](#)

This appendix describes helps you to install MyFaces Tomahawk Framework Libraries on your NonStop system.

---

# Contents

<b>10 MyFaces Overview.....</b>	<b>231</b>
MyFaces Projects.....	231
MyFaces Trinidad.....	231
MyFaces Tomahawk.....	231
MyFaces Application on NonStop.....	232
<b>11 Installing MyFaces Framework.....</b>	<b>233</b>
Prerequisites.....	233
NonStop System.....	233
Windows System.....	233
Downloading MyFaces Framework Libraries on Windows.....	233
Deploying and Running Sample MyFaces Application on NonStop.....	234
myfaces-components.....	234
<b>12 Configuring MyFaces Applications on NonStop Systems.....</b>	<b>240</b>
NonStop Platform Configurations.....	240
Determining the Application Parameters.....	240
Determining the Maximum Capacity of NSJSP Instance.....	240
Configuring iTP WebServer for MyFaces Applications.....	241
Configuring NSJSP for MyFaces Applications.....	245
<b>13 Getting Started with MyFaces.....</b>	<b>252</b>
Prerequisites.....	252
NonStop System.....	252
Windows System.....	252
Overview of SkinSelector.....	252
Developing SkinSelector on Windows using the Eclipse Galileo IDE.....	252
Deploying SkinSelector on NonStop.....	278
Running SkinSelector on NonStop.....	282
<b>H Installing MyFaces Trinidad Framework Libraries on NonStop.....</b>	<b>285</b>
Downloading MyFaces Trinidad Distribution.....	285
Copying MyFaces Trinidad Runtime Libraries from Windows to NonStop.....	285
<b>I Installing MyFaces Tomahawk Framework Libraries on NonStop.....</b>	<b>286</b>
Downloading MyFaces Tomahawk Distribution on Windows.....	286
Copying MyFaces Tomahawk Runtime Libraries from Windows to NonStop.....	286

# 10 MyFaces Overview

The Apache MyFaces project is an implementation of the JavaServer Faces (JSF) specifications. It also provides a set of JSF components that go beyond the JSF specifications. The MyFaces "core" project is an implementation of the JSF specifications and its sub-projects add features that work with MyFaces core or any other implementation of JSF specifications.

**NOTE:** Any mention to Apache MyFaces in this document refers to MyFaces version 2.0.2 and any mention of JSF refers to JSF version 2.0.

The Apache MyFaces project provides:

- A JavaServer Faces implementation (MyFaces API, MyFaces Impl modules)
- Component libraries for building web applications using JSF (for example, MyFaces Tomahawk, MyFaces Trinidad)
- Implementation and extension packages to JSFs (for example, MyFaces Extensions Validator)

## MyFaces Projects

Apart from the MyFaces core project, the Apache MyFaces Trinidad and MyFaces Tomahawk sub-projects are certified for use on the NonStop platform. For more information about MyFaces projects, visit <http://myfaces.apache.org/index.html>.

### MyFaces Trinidad

MyFaces Trinidad includes a comprehensive component library that provides a set of extended services by adding new components to the default components of the MyFaces core framework. MyFaces Trinidad provides the following features:

- Rich set of components, validators, and converters, such as Dialog Framework, Date Restriction validator, and so on.
- Efficient implementations of client-side state saving.
- Rich DHTML client-side renderers.
- Partial Page Rendering (PPR).
- Client-side converters/validators.
- Bidirectional language support.
- Accessibility - support for Section 508.

For information on MyFaces Trinidad, see <http://myfaces.apache.org/trinidad/index.html>.

### MyFaces Tomahawk

MyFaces Tomahawk provides custom components that are fully compatible with JSF. These components extend the functionality provided by the core components of JSF. MyFaces Tomahawk provides the following features:

- Support for Tiles to build page templates and plug in reusable page components. For more information on using Tiles, see <http://myfaces.apache.org/tomahawk/tiles.html>.
- Advanced GUI components, such as CAPTCHA, JSCook\_Menu, and so on.
- Support for enhanced components, such as Extended\_Data\_Table, HtmlSelectManyCheckbox, PanelGrid, and so on.

For information on MyFaces Tomahawk, see <http://myfaces.apache.org/tomahawk/index.html>.

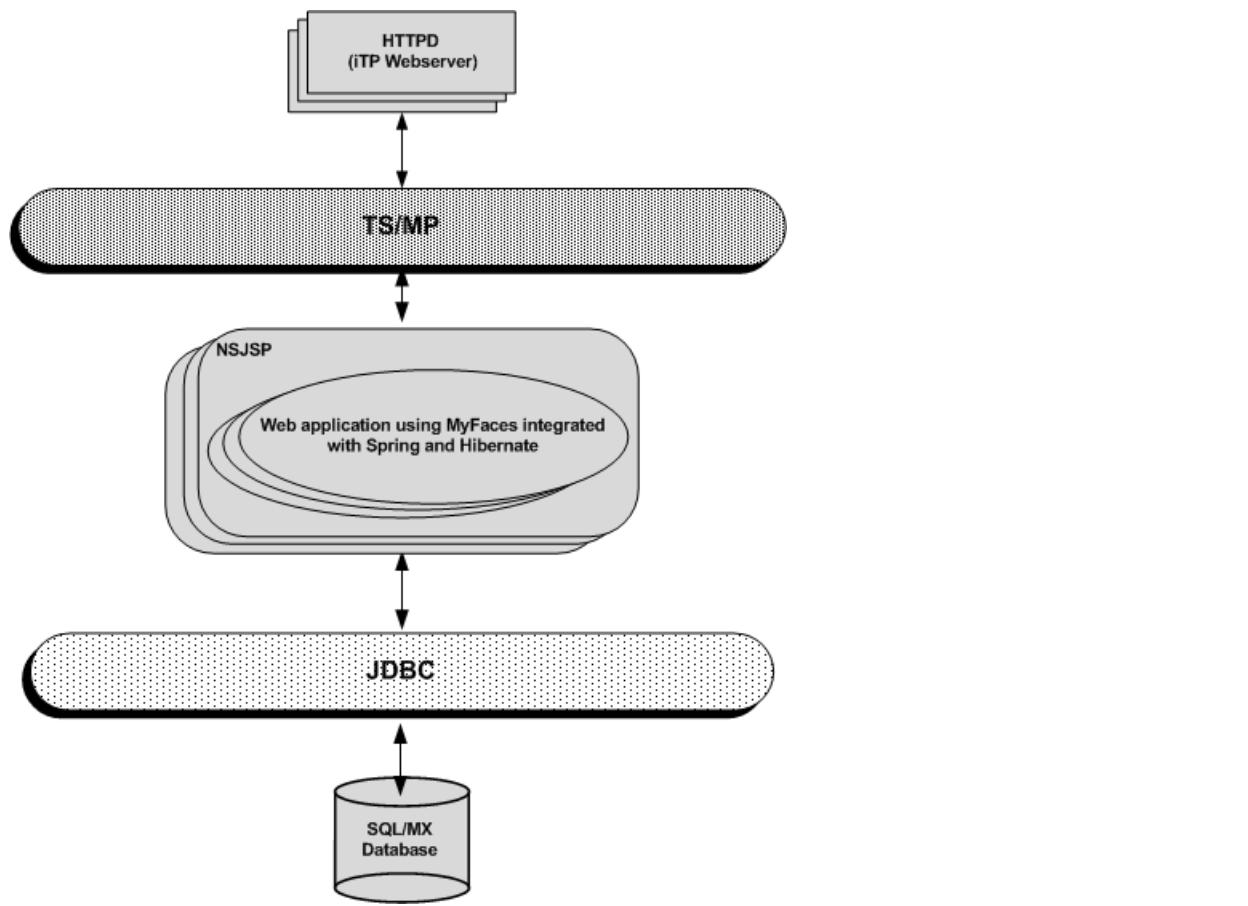
## MyFaces Application on NonStop

MyFaces projects help you to design the user interface for your web applications. Because the SPRING core is certified on NonStop and the Web Module of the SPRING core includes integration components for any JSF implementation, the preferred way to use MyFaces is to integrate it with SPRING. The web application thus developed can be deployed on NSJSP (the servlet container on NonStop). The application thus inherits all the unique advantages of NonStop through NSJSP.

**NOTE:** MyFaces can be integrated with the Spring framework using the Web module of Spring. For more information on Spring and MyFaces integration, see "[Integrating Frameworks](#)" (page 346).

Figure 68 shows the typical use of MyFaces projects to create web applications on NonStop.

**Figure 68 Message Flow of MyFaces Application**



In Figure 68, a typical web application, which can access a SQL/MX database, is developed by integrating the MyFaces, Spring, and Hibernate frameworks. In the web application:

- MyFaces implements the presentation layer.
- Spring implements the business logic for the application (using features such as Inversion of Control (IoC), Dependency Injection (DI), and aspect-oriented programming (AOP)).
- Hibernate implements the persistence layer.

Apart from the set of built-in components provided by the MyFaces core framework, you can also develop custom components and validators depending on your application requirements. The subsequent chapters discuss installing and configuring MyFaces to customize it for your web application development requirements.

# 11 Installing MyFaces Framework

This chapter discusses the steps required to complete the following activities:

1. "Downloading MyFaces Framework Libraries on Windows" (page 233)
2. "Deploying and Running Sample MyFaces Application on NonStop" (page 234)

## Prerequisites

Before you begin installing the MyFaces framework to design the UI for your web applications, make sure that you have the requisite software installed on your NonStop and Windows system.

### NonStop System

The following software must be installed on your NonStop system:

- NonStop iTP WebServer version T8996H02 or later
- NSJSP version T1222H60 or later
- NSJ version T2766H60 or later

### Windows System

The following software must be installed on your Windows system:

- Java Development Kit (JDK) version 1.5 or later
- Maven version 2.2.1

**NOTE:** For more information about installing the software required on NonStop and Windows system, see "Prerequisites" (page 18).

## Downloading MyFaces Framework Libraries on Windows

**NOTE:** Throughout the chapter, references are made to the following directories:

- <MyFaces Home>: This is the directory on your Windows system where the MyFaces distribution files are extracted.
- <My SASH Home>: This is the directory on your Windows system where the contents of the SAMPLES file (distributed as a part of the NS Samples for Java Frameworks - T0874 and available for download in Scout for NonStop Servers) is extracted.

To download the MyFaces Core distribution on your Windows system, complete the following steps:

1. Go to <http://archive.apache.org/dist/myfaces/binaries/>.

A web page displaying a list of distributions available for download appears.

2. Download the myfaces-core-2.0.2-bin.zip file (for Windows) or the myfaces-core-2.0.2-bin.tar.gz file (for UNIX).

---

**NOTE:**

- The `myfaces-core-2.0.2-bin.zip` and `myfaces-core-2.0.2-bin.tar.gz` files include the MyFaces framework libraries and third party libraries that are required to build MyFaces applications. These files only differ in their compression formats.
  - The sample application (Tomahawk) discussed in this section has been verified using MyFaces version 2.0.2 only.
- 

3. Extract the `myfaces-core-2.0.2-bin.zip` or `myfaces-core-2.0.2-bin.tar.gz` file into a directory on your Windows system.

This directory will be referred as `<MyFaces Home>`. Among other directories and files, `<MyFaces Home>` contains the following sub-directory:

`\lib`

includes the MyFaces core libraries `myfaces-impl-2.0.2.jar`, `myfaces-api-2.0.2.jar`, and third-party libraries usually required to build MyFaces applications.

## Deploying and Running Sample MyFaces Application on NonStop

By default, the MyFaces distribution package does not include any sample application. To verify if the framework has been installed successfully and to demonstrate some basic features of MyFaces, a sample application called `myfaces-components`, is provided in the `SAMPLES` file.

---

**NOTE:** The `SAMPLES.zip` file is distributed as a part of the NS Samples for Java Frameworks - T0874 in the `T0874AAB.BIN` file in Scout for NonStop Servers. For information on how to install the `T0874AAB.BIN` file from Scout, see [https://h20453.www2.hp.com/scout/download\\_help.htm](https://h20453.www2.hp.com/scout/download_help.htm).

Before you deploy the sample applications, complete the following steps:

1. Download the `SAMPLES.zip` file from Scout for NonStop servers.
2. Add the `.zip` extension to it.  
The file is renamed as `SAMPLES.zip`.
3. Extract the `SAMPLES.zip` file to a location on the Windows system.

The `NS-Samples-for-Java-Frameworks` folder appears.

---

**NOTE:** The absolute path of the `NS-Samples-for-Java-Frameworks` folder is referred as `<My SASH Home>`.

---

This section describes the following steps for the `myfaces-components` application:

- Building the sample application on Windows system
- Deploying the sample application on NonStop system
- Running the sample application on NonStop system

### myfaces-components

The `myfaces-components` sample application demonstrates the use of various MyFaces components such as `DataList`, and `TabbedPane`.

This section describes the following steps for the `myfaces-components` sample application.

- “Building `myfaces-components` on Windows” (page 235)
- “Deploying `myfaces-components` on NonStop” (page 235)
- “Running `myfaces-components` on NonStop” (page 237)

## Building myfaces-components on Windows

To build myfaces-components on your Windows system, complete the following steps:

1. Go to the `<My SASH Home>\myfaces\samples\myfaces-components` directory on your Windows system. Among the other files, this directory consists of the following sub-directories:

`\src`

includes the source files for the myfaces-components sample application.

`\src\main\webapp`

includes the necessary configuration files required to build the application WAR file.

2. Build myfaces-components (`myfaces-components-0.0.1-SNAPSHOT.war`) using Maven:

- a. Go to the `<My SASH Home>\myfaces\samples\myfaces-components` directory on your Windows system, using the command:

command prompt> cd `<My SASH Home>\myfaces\samples\myfaces-components`

- b. Build the myfaces-components application web archive, using the command:

command prompt> mvn install

- c. After the successful build of myfaces-components, a new directory named `target` is created in the `<My SASH Home>\myfaces\samples\myfaces-components` directory. The application WAR file (`myfaces-components-0.0.1-SNAPSHOT.war`) is created in the `target` directory.

This completes building of the myfaces-components application WAR file on your Windows system.

## Deploying myfaces-components on NonStop

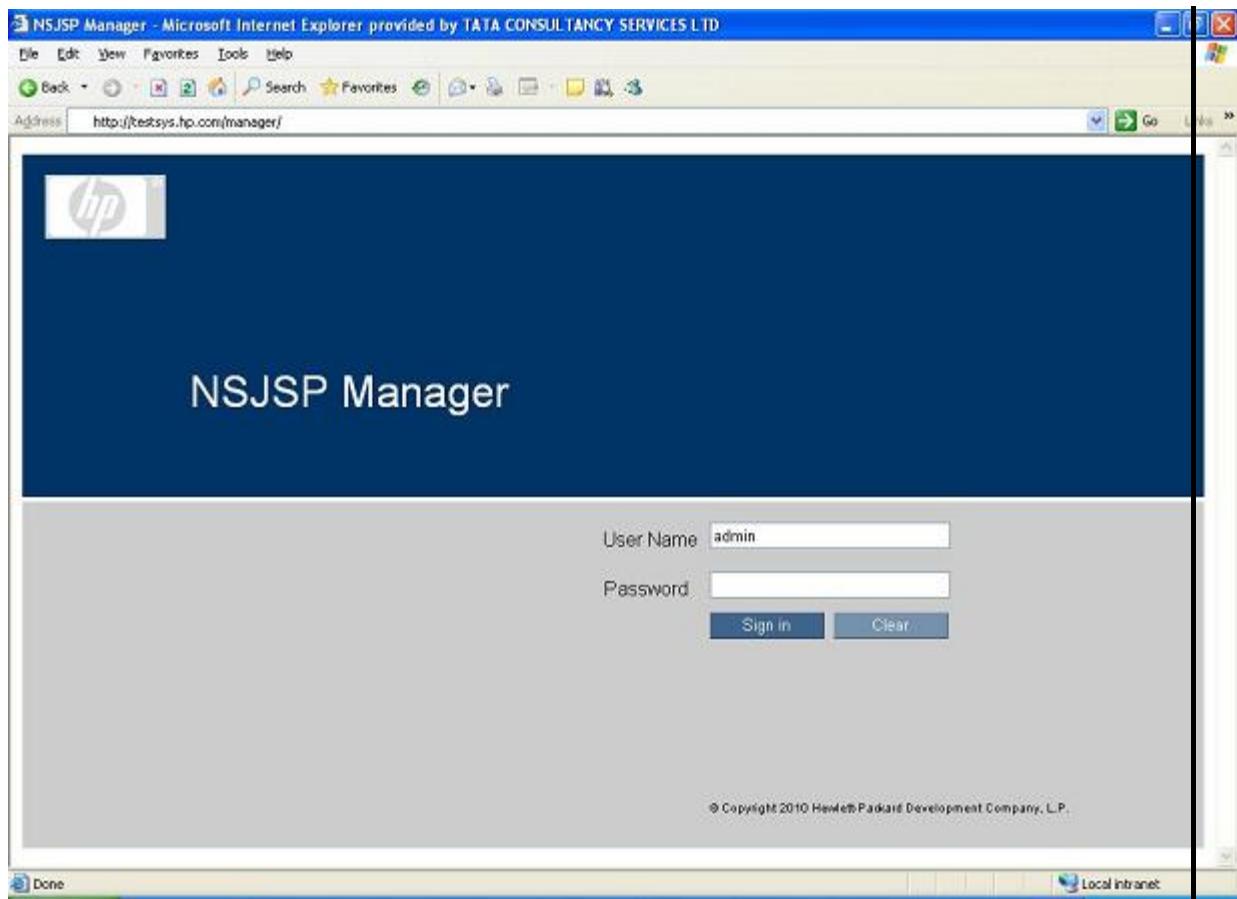
To deploy myfaces-components on your NonStop system, complete the following steps:

1. Go to [http://<IP Address of the iTP WebServer>:<port#>/<manager\\_directory>](http://<IP Address of the iTP WebServer>:<port#>/<manager_directory>).

The **NSJSP Manager Login** screen appears.

Figure 69 shows the **NSJSP Manager Login** screen.

**Figure 69 NSJSP Manager Login Screen**

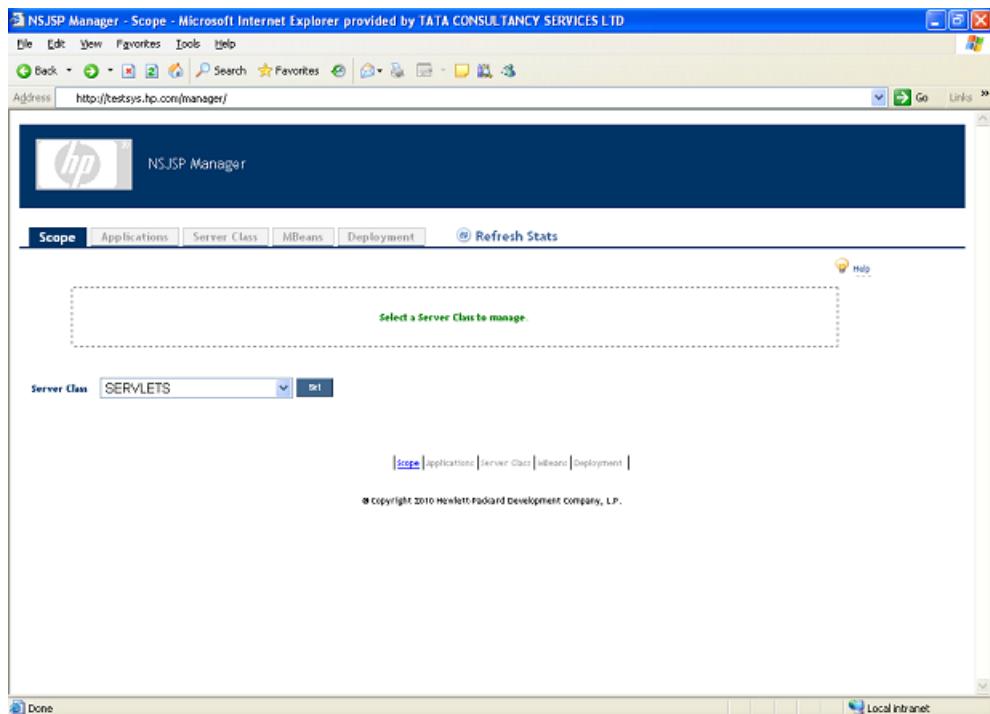


2. Enter the **User name:** and **Password:** and click **OK**.

The **NSJSP Manager** screen appears.

Figure 70 shows the **NSJSP Manager** screen.

**Figure 70 NSJSP Manager Screen**

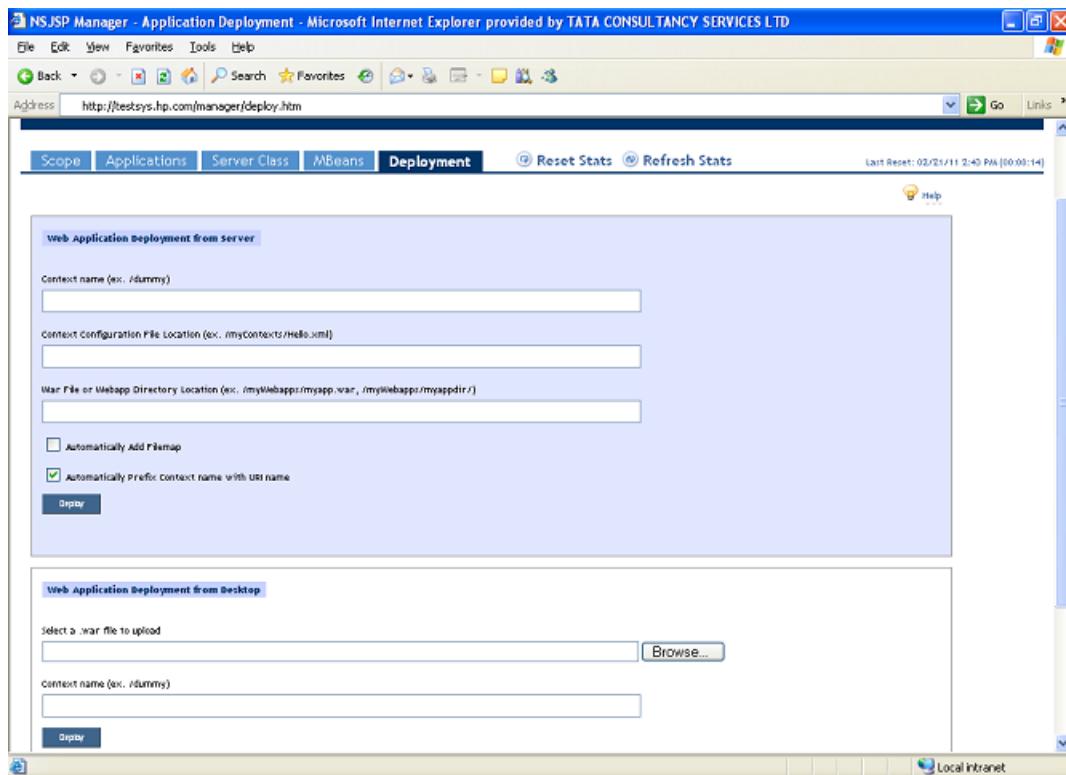


3. Select **SERVLETS** for the **Server Class** and click **Set**.

The **Host (in server.xml)** tab is enabled and populated with **localhost**. Click **Set**, which enables the **Deployment** tab on the **NSJSP Manager** screen.

Figure 71 shows the **Deployment** tab of **NSJSP Manager**.

**Figure 71 NSJSP Manager Screen - Deployment tab**



4. In the **Deployment** tab, complete the following steps in the **Web Application Deployment from Desktop** section:

1. In the **Select a .war file to upload** field, click **Browse...** and locate the **myfaces-components-0.0.1-SNAPSHOT.war** file on the Windows system.
2. **(Optional)** In the **Context name** field, enter a name for the application context.
3. Click **Deploy**.

The **myfaces-components** application is deployed on NSJSP and is listed under **Applications** tab of the **NSJSP Manager** screen.

---

**NOTE:** HP recommends that you use the NSJSP manager for deployment. Deployment using the FTP or any other mechanism is not recommended. For more information on using the NSJSP manager, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

---

### Running myfaces-components on NonStop

To run **myfaces-components** on your NonStop system, click the **/<servlet directory>/myfaces-components-0.0.1-SNAPSHOT** path under **Applications** in the **NSJSP Manager** screen.

You can now access the following web pages of myfaces-components:

- `http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/myfaces-components-0.0.1-SNAPSHOT/date.faces`

This page demonstrates the use of the `t:inputDate` tag from the MyFaces Tomahawk library. The `t:inputDate` tag is a date component that can be used to enter date/time information in a form.

- `http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/myfaces-components-0.0.1-SNAPSHOT/dataList.faces`

This page demonstrates the use of the `t:dataList` tag from the MyFaces Tomahawk library. The `t:dataList` tag renders its data as a HTML list element. The `layout` attribute defines how each item is rendered. Legal values are: `simple`, `unorderedList`, and `orderedList`.

- `http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/myfaces-components-0.0.1-SNAPSHOT/newspaperTable.faces`

This page demonstrates the use of the `t:newspaperTable` tag from the MyFaces Tomahawk library. The `t:newspaperTable` tag is a data table that renders table text across several columns (like a newspaper layout). The number of columns can be set using the `newspaperColumns` attribute.

- `http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/myfaces-components-0.0.1-SNAPSHOT/order.faces`

This page demonstrates the use of `t:inputSecret`, `t:inputText`, `t:validateEmail`, `t:validateCreditCard`, and `t:validateEqual` tags from MyFaces Tomahawk library.

- `t:inputSecret`: The `t:inputSecret` tag renders an `<input type="password">` HTML tag. When the user types a string into this field, a row of asterisks is displayed instead of the text the user types. The `t:inputSecret` tag extends the JSF `inputSecret` tag. It supports conditional rendering based on user roles.

- `t:inputText`: This tag extends the JSF `inputSecret` tag. It supports conditional rendering based on user roles.

- `t:validateEmail`: This tag is a custom validator that checks the syntax of an email address using the Jakarta Commons validation library.

- `t:validateCreditCard`: This tag allows you to validate various credit cards, including Amex, Visa, Mastercard, and Discover, using the Jakarta Commons validation library.

- `t:validateEqual`: This tag allows you to validate user input by comparing it with another backing bean property. This is useful, for example, when validating a password or an e-mail address twice.

- `http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/myfaces-components-0.0.1-SNAPSHOT/tabbedPanes.faces`

This page demonstrates the use of the `t:panelTabbedPane` and `t:panelTab` tags from the MyFaces Tomahawk library.

- `t:panelTabbedPane`: This tag generates a panel container with a number of tabs. This component can be fully customized using CSS style attributes.

- `t:panelTab`: This tag renders an individual panel tab as a HTML button. It must be contained within a `t:panelTabbedPane` tag.

- `http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/myfaces-components-0.0.1-SNAPSHOT/popup.faces`

This page demonstrates the use of the `t:popup` tag from the MyFaces Tomahawk library. The `t:popup` tag renders a browser pop-up window, which displays a number of possible

mouse or keyboard events, including `onclick`, `ondblclick`, `onmouseover`, `onkeypress`, `onkeydown`, and `onkeyup`.

# 12 Configuring MyFaces Applications on NonStop Systems

This chapter provides information about configuring JavaServer Faces (MyFaces) applications on NonStop systems. This chapter includes the following section:

- “NonStop Platform Configurations” (page 240)

## NonStop Platform Configurations

On a NonStop system, an application developed using the MyFaces framework is deployed as a web application in the NSJSP container. Thus, it inherits all the NSJSP properties and configurations, and runs as any other NSJSP web application running on a NonStop system. This section discusses the steps to complete the following activities:

- “Determining the Application Parameters” (page 240)
- “Determining the Maximum Capacity of NSJSP Instance” (page 240)
- “Configuring iTP WebServer for MyFaces Applications” (page 241)
- “Configuring NSJSP for MyFaces Applications” (page 245)

## Determining the Application Parameters

You need to determine the following parameters for your application:

- Average Response Time
- Average Load
- Maximum Load

For a unit response time, the number of active requests to iTP WebServer is a maximum of its static capacity during the average load conditions. During peak load conditions, the number of active requests to the iTP WebServer is a maximum of its dynamic capacity.

For example,

If the average response time of your MyFaces application is 1 second and iTP WebServer is configured to handle 100 requests statically and 50 requests dynamically, then during average load conditions, the number of active requests to the iTP WebServer can be a maximum of 100. During peak load conditions, 50 more requests can be served dynamically and hence, the active requests to iTP WebServer can be a maximum of 150. Further requests are queued up and remain inactive until started by `httpd`.

The other parameters, such as Average Load and Maximum Load, can help you decide the configuration of the static and dynamic `httpd` processes of the iTP WebServer.

## Determining the Maximum Capacity of NSJSP Instance

Before configuring the NSJSP parameters, you must determine the capacity of a single instance of NSJSP. To determine the maximum load for a single instance of NSJSP, it is important to first configure the relevant TS/MP and server parameters of NSJSP (of the single instance) to their maximum limit in the following way:

1. Set the value of `Numstatic` to 1. This limits the number of static instances of NSJSP to 1.
2. Set the value of `Maxservers` to 1. This limits the number of NSJSP instances that can be started to 1. This implies that TS/MP cannot start more than one instance of NSJSP.
3. Set the value of `Maxlinks` to 250. This is the maximum number of links to the server process (NSJSP process). This means that the server process must be capable of processing 250 requests simultaneously.

4. Set the value of TANDEM\_RECEIVE\_DEPTH to 250 (because the values of Maxlinks and TANDEM\_RECEIVE\_DEPTH should be equal). A value of 250 means NSJSP process is able to read a maximum of 250 messages simultaneously from its \$RECEIVE file.
5. Configure the Executor element in the *<NSJSP Deployment Directory>/conf/server.xml* file on OSS. The Executor element is used by the Connector as a thread pool. Set maxThreads = 300, minSpareThreads = 10 and maxIdleTime = 30000. Using the Executor element helps monitor the number of active threads at any given time. A value of 300 for maxThreads ensures that you have enough threads to process all incoming requests (a maximum of 250) simultaneously. A value of 30000 for maxIdleTime ensures that if a thread is idle for more than 30 seconds, that thread will be stopped.

After configuring the parameters, you can use a tool that can simulate the HTTP clients and can handle HTTP cookies. The tool reveals the number of HTTP clients that a single instance of NSJSP can handle and indicates the number of simultaneous HTTP requests that NSJSP is capable of handling.

**NOTE:** There are a number of HTTP client simulators available, for example, Apache JMeter, HP LoadRunner, and Radview Webload. These tools provide a good interface to monitor the test results. You can use any of these tools to determine the maximum handling capacity of each NSJSP instance.

To arrive at the required numbers, complete the following steps:

1. Run the test tool with a single instance of the HTTP client simulator.

**NOTE:** The test tool must be capable of showing the response time of the HTTP requests.

2. Monitor the response time of the HTTP requests and allow your application to attain a steady state (in terms of response time per HTTP request).
3. At steady state, check if the response times are within the Service Level Agreement (SLA).
4. If the response time is within the SLA, increase the number of HTTP client instances and repeat step 2 onwards.
5. If the response time is beyond the acceptable SLA, stop the tests and determine the number of HTTP clients and the number of HTTP requests that NSJSP instance could process simultaneously.

While the test tool can indicate the number of HTTP clients that were being processed simultaneously, the number of simultaneous HTTP requests can be arrived at by different means. Following are some of the calculation strategies:

- The number of HTTP requests recorded by the testing tool and idea of the number of requests, which required processing by NSJSP, provides the total number of HTTP requests processed by NSJSP.
- The total number of HTTP requests processed by NSJSP together with the test duration indicates the average number of simultaneous HTTP requests handled by NSJSP.

## Configuring iTP WebServer for MyFaces Applications

The `httpd` process of the iTP WebServer is responsible for managing the load and forwarding the requests to their corresponding application servers, the context of this section is limited to the configurations related to `httpd` processes only.

Configuring the `httpd` process involves the following tasks:

- “Configuring `httpd` Processes to Handle Maximum Load” (page 242)
- “Limiting the Maximum Number of Incoming Requests” (page 244)

## Configuring httpd Processes to Handle Maximum Load

The httpd processes are configured using the Server configuration directive in the *<iTP WebServer Deployment Directory>/conf/httpd.config* file on OSS. The Server directive controls the creation of the PATHMON environment, where the webserver runs.

A typical default configuration of the httpd process of the iTP WebServer appears as follows:

```
Server $root/bin/httpd {
    eval $DefaultServerAttributes
    CWD [pwd]
    Arglist -server [HTTPD_CONFIG_FILE]
    Env TANDEM_RECEIVE_DEPTH=50
    Priority 170
    Numstatic 5
    Maxservers 50
    MapDefine =TCPIP^RESOLVER^NAME /G/system/ztcpip/resconf
    MapDefine =TCPIP^PROCESS^NAME $transport
}
```

The number of httpd processes are governed by the following TS/MP attributes:

- Numstatic: This specifies the number of static servers running under PATHMON. The static processes run on the system, irrespective of the load.
- Maxservers: This specifies the maximum number of server processes that can run under PATHMON.
- TANDEM\_RECEIVE\_DEPTH: This specifies the capacity of each of the configured httpd processes.
- (Maxservers – Numstatic): The difference denotes the number of dynamic servers. A dynamic server is a need-based server. When the iTP WebServer is under heavy load and all the static httpd processes are busy, a dynamic server process, httpd, is created to support the excess load. These servers are dynamically created by TS/MP and are terminated once the process is complete.

The capacity of the iTP WebServer environment can be summarized as:

- The static capacity of iTP WebServer is [Numstatic X TANDEM\_RECEIVE\_DEPTH].
- The dynamic capacity is [(Maxservers – Numstatic) X TANDEM\_RECEIVE\_DEPTH] requests.
- The total capacity of iTP WebServer is [Maxservers X TANDEM\_RECEIVE\_DEPTH].

Example 1:

Assume that the httpd process of the iTP WebServer has the following configurations:

```
Server $root/bin/httpd {
    eval $DefaultServerAttributes
    CWD [pwd]
    Arglist -server [HTTPD_CONFIG_FILE]
    Env TANDEM_RECEIVE_DEPTH=100
    Priority 170
    Numstatic 5
    Maxservers 50
    MapDefine =TCPIP^RESOLVER^NAME /G/system/ztcpip/resconf
    MapDefine =TCPIP^PROCESS^NAME $transport
}
```

When you start the iTP WebServer, five static httpd processes will be started, governed by the value of Numstatic. Because the value of TANDEM\_RECEIVE\_DEPTH is set to 100,

each of the five static processes can handle 100 requests. In this example, the capacity of the iTP WebServer environment can be summarized as follows:

- The static capacity of iTP WebServer is [Numstatic X TANDEM\_RECEIVE\_DEPTH]= 500.
- The dynamic capacity is [(Maxservers - Numstatic) X TANDEM\_RECEIVE\_DEPTH] = 4500.
- The total capacity of iTP WebServer is [Maxservers X TANDEM\_RECEIVE\_DEPTH] = 5000.

Using this configuration, the iTP WebServer can handle 500 simultaneous requests statically. As soon as it receives the 501st request, a dynamic httpd process is created, which functions as a normal httpd process. The capacity of each of the dynamic httpd processes is the value of TANDEM\_RECEIVE\_DEPTH. When the load decreases, the dynamic httpd processes goes down.

For more information on the Server configuration directive of iTP WebServer, see the *iTP Secure WebServer System Administrator's Guide*.

### Guidelines for Configuring the Server Directive

Before configuring the attributes of the Server directive, you must determine the following parameters:

- Processors for httpd Processes
- Application parameters: Average Load, Peak Load, and Average Response Time

### Processors for httpd Processes

When you start the iTP WebServer, static httpd processes equal to the value of Numstatics are started on your NonStop system. By default, all the configured processors are taken into account. If the value of Numstatic is set to n, and number of configured processors on the system are equal to n, one static httpd process will be started on each of the processors. However, if the value of Numstatic is less than the number of configured processors, one static httpd process starts on some of the processors, while the remaining processors will not have any static httpd process.

You can determine the processors on which you intend to start httpd processes and configure the processors directive. Following example will help you understand this better:

For example:

Assume that you have a 16-processor system and you want to start httpd processes on processors 0, 1, 2, 3, 4, 5. For this, the processors configuration attribute must be configured as processors 0 1 2 3 4 5.

In this scenario:

- If you set the value of Numstatic equal to 6, each of the processors 0, 1, 2, 3, 4, 5 will have one static httpd process.
- If you set the value of Numstatic equal to a multiple of 6, each of the processors 0, 1, 2, 3, 4, 5 will have (Numstatic/6) static httpd processes.
- If you set the value of Numstatic equal to 5, any 5 of the processors 0, 1, 2, 3, 4, 5 will have one static httpd process, and remaining processor will not have any httpd process.
- If you set the value of Numstatic equal to 9, any three of the processors 0, 1, 2, 3, 4, 5 will have two static httpd process and each of the remaining three processors will have one httpd process.

After determining the values, you can use any the following approaches to configure the TS/MP parameters of the `httpd` processes:

- Using the default value of `TANDEM_RECEIVE_DEPTH`
- Modifying the value of `TANDEM_RECEIVE_DEPTH`

#### [Using the default value of `TANDEM\_RECEIVE\_DEPTH`](#)

With `TANDEM_RECEIVE_DEPTH` set to the default value of 50, the static processes can be configured on the basis of Average Load, and dynamic processes can be configured on the basis of Peak Load on your system. Use the following approaches while deciding the values of `Numstatic` and `Maxservers`:

- The value of `Numstatic` must be set to  $(\text{Average Load})/50$
- The value of `Maxservers` must be set to  $(\text{Peak Load})/50$

For example:

If the Average Load and Peak Load on your MyFaces application turn out to values 1000 and 1500 requests and you limit `TANDEM_RECEIVE_DEPTH` to its default value of 50, the `Numstatic` must be set to 20 and `Maxservers` must be set to 30.

---

**NOTE:** It is advisable to set the value of `Numstatic` at least to the number of configured processors.

#### [Modifying the value of `TANDEM\_RECEIVE\_DEPTH`](#)

If the number of `httpd` processes configured on the basis of default value of `TANDEM_RECEIVE_DEPTH` does not fulfill your application requirement, you can change the value of `TANDEM_RECEIVE_DEPTH` and calculate the values of `Numstatic` and `Maxservers` accordingly. However, consider the following before increasing the value of `TANDEM_RECEIVE_DEPTH`:

- It is recommended to keep the web server running with 80 percent of its capacity. To achieve this, set the value of `TANDEM_RECEIVE_DEPTH` accordingly.  
For example, if your calculation suggests that `TANDEM_RECEIVE_DEPTH` of 80 is required, you must set it to a value of 100.
- The maximum value of `TANDEM_RECEIVE_DEPTH` is 255.
- Do not reduce `TANDEM_RECEIVE_DEPTH` to a value less than 50.

### [Limiting the Maximum Number of Incoming Requests](#)

Because of constraints on the available system resources, you might want to limit the number of requests to your iTP WebServer. If you want to configure your iTP WebServer to handle only a certain number of requests at an instance, use the `MaxConnections` configuration directive to specify the maximum number of connections that can be served by the iTP WebServer at any given instance. The iTP WebServer serves the number of requests equal to the multiple of `Numstatic` greater than or equal to `MaxConnections` count.

#### **Syntax:**

```
MaxConnections -count <integer value> -replytype <customized/RST>
```

For example:

```
MaxConnections -count 101 -replytype RST
```

Consider the scenario of Example 1 above, where `Numstatic` is 5 with the following `MaxConnections` configuration:

```
MaxConnections -count 101 -replytype customized
```

In this case, the iTP WebServer serves 105 requests (higher multiple of Numstatic nearest to the count value). Here, the 106th request will display the following error message:

Maximum connections reached: The server reached its maximum configured capacity.  
with HTTP response code:

200 OK

To customize the error message, create a new message ID error-maximum-connection. This customized message is displayed if the Message configuration directive is used in the <iTP WebServer Deployment Directory>/conf/httpd.config file with the new message ID. For more information on the MaxConnections configuration directive and creating customized error messages using the Message configuration directive, see the *iTP Secure WebServer System Administrator's Guide*.

---

**NOTE:** To use the MaxConnections configuration directive, your iTP WebServer must be configured for the static environment only, that is, the values of Numstatic and MaxServers of the httpd process must be equal.

---

## Configuring NSJSP for MyFaces Applications

When a MyFaces application runs on a NonStop system, it runs as an instance of NSJSP. Therefore, before configuring the NSJSP environment, it is important to determine the load each instance of NSJSP is expected to handle. This section describes the following configuration aspects:

- ["Configuring SessionBasedLoadBalancing" \(page 245\)](#)
- ["Configuring Connector Threads" \(page 246\)](#)
- ["Configuring TS/MP Specific Parameters" \(page 247\)](#)
- ["Configuring Java Runtime Arguments" \(page 249\)](#)
- ["Determining the Maximum Capacity of NSJSP Instance" \(page 250\)](#)

### Configuring SessionBasedLoadBalancing

The NSJSP Container maintains sessions in the form of serialized Java objects. Each session object is identified by a unique identifier called the session-ID. The session-ID is sent to the HTTP client either as a cookie or in the form of URL-rewriting. The name of the cookie is JSESSIONID and when the user application creates a session, NSJSP generates the cookie (JSESSIONID) as the name and session-ID as the cookie value. The session objects can either be kept in the process memory or persisted in a persistent store (for example, a database table). When it is kept in the process, it is available only for the process that created the session object. On the other hand, if it is kept in a persistent store, it is available for any process in the NSJSP environment. When the SessionBasedLoadBalancing feature is enabled, all requests related to a particular session are routed to the process that has the session object in its memory.

To enable the SessionBasedLoadBalancing feature, you must configure the servlet.ssc object by editing the <iTP WebServer Deployment Directory>/conf/servlet.config file on OSS. The servlet.ssc object is configured under the Server directive. The SessionBasedLoadBalancing feature is governed by the -DSaveSessionOnCreation and -DSessionBasedLoadBalancing arguments in the Arglist of the Server directive.

- -DSaveSessionOnCreation

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

Syntax:

-DSaveSessionOnCreation=[ true | false ]

**NOTE:** The default value of `-DSaveSessionOnCreation` is false.

- -DSessionBasedLoadBalancing

Enables or disables session-based load balancing.

Syntax:

-DSessionBasedLoadBalancing= [ true | false ]

**NOTE:** The default value of `-DSessionBasedLoadBalancing` is set to true.

For example, if you want to enable the SessionBasedLoadBalancing feature for your MyFaces application and save the application sessions in a persistent store, set the Arglist as:

```
Server $server_objectcode {  
    ...  
    ...  
    ...  
    ...  
    ...  
  
    Arglist -DSessionBasedLoadBalancing=true \  
    -DSaveSessionOnCreation=true \  
    ...  
    ...  
    ...  
    ...  
}
```

where,

`server_objectcode` is mapped to the `servlet.ssc` object in the `<iTP WebServer Deployment Directory>/bin` directory on OSS.

While setting arguments, consider the following:

1. If both the `SaveSessionOnCreation` and `SessionBasedLoadBalancing` options are set to `false`, and if a Persistent Manager is configured with a persistent store, 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.
  2. Enabling or disabling the `SessionBasedLoadBalancing` feature depends on the application requirement. Your MyFaces application might encounter any of the following scenarios:
    - The application depends heavily on the state stored in session objects. Therefore, session objects cannot be lost and the application cannot recover from loss of state.
    - Application response is of prime importance and the application can recover from a loss of state.
    - The application is expected to handle largely varying loads and having a large number of static NSJSP instances is not an option.
    - The session objects must be valid for large durations (such as an entire day).
    - The session objects must be available whenever the application restarts.

For more information on the `SessionBasedLoadBalancing` and `SessionBasedLoadBalancing` configuration directives, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

## Configuring Connector Threads

The Connector is responsible for creating and maintaining the threads that are used for message processing. A thread is allocated for each incoming message; therefore, the number of messages

that can be processed simultaneously can be controlled by limiting the number of threads the NSJSP Connector can spawn.

---

**NOTE:** A single connector can service multiple instances of the container. The connector must be configured such that it can spawn threads sufficient for all the container instances. In terms of configuration, the Host element in the server.xml configuration file represents a container that is serviced by the connector.

---

The NSJSP Connector thread pool can be connected in the following ways:

- [Configuring the Connector Element](#)
- [Configuring the Executor Element](#)

### Configuring the Connector Element

Configure the maxThreads attribute of the Connector element in the <NSJSP Deployment Directory>/conf/server.xml file on OSS. The Connector element is the child of the Service element.

For example: The following snippet shows the configuration in the <NSJSP Deployment Directory>/conf/server.xml file on OSS, if you want to configure 75 simultaneous connector threads.

```
...
<Service name="NSJSP">
<Connector protocol="HTTP/1.1"
           connectionTimeout="0"
           acceptCount="25"
           maxThreads="75"/>
...
</Service>
```

### Configuring the Executor Element

You can configure the Executor element as child elements of the Service element in the <NSJSP Deployment Directory>/conf/server.xml file on OSS. The following snippet of the server.xml configuration file shows the configuration of the Executor element and how a Connector can be configured to use an Executor element.

```
...
<Service name="NSJSP">
<Executor name="MyFacesExec"
          className="org.apache.catalina.core.StandardThreadExecutor"
          namePrefix="MyFacesAPP" maxThreads="75" minSpareThreads="20"
          maxIdleTime="10000"/>
<Connector executor="MyFacesExec" />
...
...
```

**NOTE:** All Executor elements that are configured in NSJSP must implement the Java interface org.apache.catalina.Executor. NSJSP provides a standard implementation of this interface in the class org.apache.catalina.core.StandardThreadExecutor.

---

## Configuring TS/MP Specific Parameters

This section describes the following TS/MP specific configuration parameters:

1. Numstatic – Determines the number of static NSJSP processes.
2. Maxservers – Determines the total number of NSJSP processes.
3. TANDEM\_RECEIVE\_DEPTH – Determines the handling capacity of each instance of NSJSP.

4. Maxlinks – Determines the maximum number of TS/MP links. The number of simultaneous messages that can be delivered to an instance of NSJSP is determined by Maxlinks. NSJSP reads these many number of messages from the \$RECEIVE file simultaneously.
5. Number of Connector threads.

The configurations for Numstatic, Maxservers, TANDEM\_RECEIVE\_DEPTH, and Maxlinks can be set under the Server directive in the *<iTP WebServer Deployment Directory>/conf/servlet.config* file on OSS. Before trying to configure NSJSP, it is important that you determine the maximum load that each instance of NSJSP is expected to handle. The way these configuration directives drive the NSJSP environment depends on the state of the SessionBasedLoadBalancing feature.

- NSJSP configured with SessionBasedLoadBalancing turned OFF
- NSJSP is configured with SessionBasedLoadBalancing turned ON

#### [NSJSP configured with SessionBasedLoadBalancing turned OFF](#)

- The value of Numstatic must be  $[(\text{Average Load}) / (\text{Max load one instance of NSJSP can handle})]$ .
- Maxservers must be  $[(\text{Peak Load}) / (\text{Max load one instance of NSJSP can handle})]$ .
- The value of Maxlinks must be set to the peak load that one instance of NSJSP is expected to handle.
- TANDEM\_RECEIVE\_DEPTH must be twice the value of Maxlinks. Normally, to handle  $n$  number of messages, you need  $n$  number of threads. In extreme cases, all the  $n$  requests may timeout. While the same number of threads processing the timeout requests can be busy, you need the same  $n$  number of threads to process the new incoming requests. Therefore, the number of threads that must be configured is  $(n + n = 2n)$ .
- The maximum number of connector thread, maxThreads must be equal to the value of TANDEM\_RECEIVE\_DEPTH.

For example:

The parameters Maxlinks, TANDEM\_RECEIVE\_DEPTH, and the Connector threads are closely linked with each other. These parameters are explained using examples. The following examples assume that the httpd processes are configured to serve 100 simultaneous HTTP, and all 100 HTTP requests are serviced by your MyFaces application deployed in NSJSP (that is, there is no static content in the application). Also, the peak load that one single instance of NSJSP can handle is 25.

Because SessionBasedLoadBalancing is turned OFF, all messages between HTTPD and NSJSP flow through TS/MP. The number of simultaneous messages that can be delivered to an instance of NSJSP is determined by the value of Maxlinks; set the value of Maxlinks to 25. NSJSP can now read 25 messages from its \$RECEIVE queue simultaneously. In an extreme case, all the 25 messages time out; there must be enough space in the \$RECEIVE queue to accommodate 25 more messages. Therefore, \$RECEIVE must be opened with a Receive Depth (controlled by TANDEM\_RECEIVE\_DEPTH) of (25+25). Thus, you must set the value of TANDEM\_RECEIVE\_DEPTH to 50. The reason for setting TANDEM\_RECEIVE\_DEPTH to a value of 50 can be explained as:

To handle 25 messages, you need 25 threads. In an extreme case where all the 25 requests time out, you need another 25 threads to process the new incoming 25 requests because the threads processing the timeout requests could still be busy. Therefore, the number of threads that need to be configured is  $(25+25 = 50)$ .

## NSJSP is configured with SessionBasedLoadBalancing turned ON

- The value of Numstatic must be  $[(\text{Peak Load}) / (\text{Max load one instance of NSJSP can handle})]$ .
- The value of Maxservers must be  $[(\text{Peak Load}) / (\text{Max load one instance of NSJSP can handle})]$ . This ensures that no dynamic process is created, so that the session object is not lost when there are numerous file system calls.
- The value of Maxlinks must be set to the peak load that one instance of NSJSP is expected to handle.
- Arriving at a definite value for TANDEM\_RECEIVE\_DEPTH is difficult when NSJSP is configured with SessionBasedLoadBalancing turned ON. Deciding an optimum value for TANDEM\_RECEIVE\_DEPTH requires a good understanding of the application. The following example provides an insight into making this decision.
- Maximum number of connector threads, maxThreads must be equal to the value of TANDEM\_RECEIVE\_DEPTH.

For example:

In this example, it is evident how a single instance of NSJSP, although having Maxlinks set to 25, can serve all the 100 requests.

Because SessionBasedLoadBalancing is turned ON, all messages between HTTPD and NSJSP flow through TS/MP and file system calls. The number of simultaneous messages that can be delivered to an instance of NSJSP is determined by the value of Maxlinks; set the value of Maxlinks to 25.

With Maxlinks set to 25, a single instance of NSJSP can handle 25 concurrent requests through TS/MP, all being the first requests of web dialogs. The first call on a web dialog is always delivered to NSJSP through TS/MP and there will not be any file system call. Once the 25 requests are serviced, subsequent requests on those 25 web dialogs are delivered to NSJSP through file system I/O operations.

At this stage, all 25 links to the NSJSP instance are free to process more incoming requests delivered through TS/MP. Therefore, the NSJSP instance should now be able to handle 25 more concurrent requests all being the first requests of web dialogs. After the 25 requests are processed, the subsequent requests on these new connections arrive at NSJSP through file system I/O. At this stage, NSJSP could be handling up to 50 concurrent requests and all these requests are delivered through file system calls from httpd and not through TS/MP. Hence, the 25 links are free to process new web dialogs. Therefore, a single instance of NSJSP can serve all the 100 requests.

At this point, one instance of NSJSP could be processing requests from all the possible 100 connections to the httpd processes. This means that there could be a scenario where all the 100 requests time out. Therefore, there must be enough space in the \$RECEIVE file to handle (100+100) messages. Therefore, TANDEM\_RECEIVE\_DEPTH must have a value of 200. The reason for setting TANDEM\_RECEIVE\_DEPTH to a value of 200 can be explained as:

To handle 100 requests, you need 100 threads. In an extreme case of all the 100 requests timing out, you need another 100 threads to process the new incoming 100 requests because the threads processing the timeout requests could still be busy. Therefore, the number of threads that need to be configured is  $(100+100 = 200)$ .

## Configuring Java Runtime Arguments

The configurations for Java runtime arguments can be done in the *<ITP WebServer Deployment Directory>/conf/servlet.config* file on OSS. The sscaux object is configured under the Server directive. Java runtime arguments are populated in the Arglist. Some of the important

Java runtime arguments that you must consider during the deployment of your MyFaces application are:

- [-Xmx](#)
- [-Xss](#)
- [-Xnklassgc](#)

There are other Java runtime arguments supported by NSJSP. For more information, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

#### [-Xmx](#)

Sets the maximum size of the memory allocation pool, which is the garbage collected heap.

Syntax:

```
-Xmx maximum-heap-size [ k | m ]
```

where,

maximum-heap-size

is the maximum size of the memory allocated for the garbage collected. It must be greater than or equal to 1000 bytes.

k

sets the value of maximum-heap-size to be read in kilobytes.

m

sets the value of maximum-heap-size to be read in megabytes.

#### [-Xss](#)

Sets the maximum stack size that can be used by a Java thread.

Syntax:

```
-Xmx maximum-stack-size
```

where,

maximum-stack-size

is the maximum size of the stack trace in kilobytes.

#### [-Xnklassgc](#)

This is an optional argument to stop the Java class garbage collection.

Syntax:

```
-Xnklassgc
```

By default, the Java runtime reclaims space for unused Java classes. Including this optional argument might prevent any memory-leak problems.

## Determining the Maximum Capacity of NSJSP Instance

To determine the maximum load for a single instance of NSJSP, it is important to configure the relevant TS/MP and server parameters of NSJSP (of the single instance) to their maximum limit in the following way:

1. Set the value of `Numstatic` to 1. This limits the number of static instances of NSJSP to 1.
2. Set the value of `Maxservers` to 1. This limits the number of NSJSP instances that can be started to 1. This implies that TS/MP cannot start more than one instance of NSJSP.
3. Set the value of `Maxlinks` to 250. This is the maximum number of links to the server process (NSJSP process). This means that the server process must be capable of processing 250 requests simultaneously.

4. Set the value of TANDEM\_RECEIVE\_DEPTH to 250 (because the values of Maxlinks and TANDEM\_RECEIVE\_DEPTH should be equal). A value of 250 means that the NSJSP process is able to read a maximum of 250 messages simultaneously from its \$RECEIVE file.
5. Configure the Executor element in <NSJSP Deployment Directory>/conf/server.xml file on OSS. The Executor element is used by the Connector element as a thread pool. Set maxThreads = 300, minSpareThreads = 10 and maxIdleTime = 30000. Using an Executor element helps to monitor the number of active threads at any given time. A value of 300 for maxThreads ensures that you have enough threads to process all the incoming requests (a maximum of 250) simultaneously. A value of 30000 for maxIdleTime ensures that if a thread is idle for more than 30 seconds, that thread will be stopped.

After making these configurations, you might want to use a tool that can simulate the HTTP clients and can handle the HTTP cookies. The tool reveals the number of HTTP clients that one single instance of NSJSP can handle and indicates the number of simultaneous HTTP requests that NSJSP is capable of handling.

The following are the suggested steps to arrive at the required numbers:

- a. Run the test tool with a single instance of the HTTP client simulator.
- b. The test tool must be capable of displaying the response time of the HTTP requests. Monitor the response time of the HTTP requests and allow your application to attain a steady state (in terms of response time per HTTP Request).
- c. At steady state, check if the response times are within the Service Level Agreement (SLA).
- d. If the response time is within the SLA, increase the number of HTTP client instances and repeat step 2 onwards.
- e. If the response time is beyond the acceptable SLA, stop the tests and determine the number of HTTP clients and the number of HTTP requests that the NSJSP instance can process simultaneously.

While the test tool can indicate the number of HTTP clients that are processed simultaneously, the number of simultaneous HTTP requests can be arrived at using different means. Following are some of the calculation strategies:

- The number of HTTP requests recorded by the testing tool and idea of the number of requests, which required processing by NSJSP, provides the total number of HTTP requests processed by NSJSP.
- The total number of HTTP requests processed by NSJSP together with the test duration indicates the average number of simultaneous HTTP requests handled by NSJSP.

---

**NOTE:** There are a number of HTTP client simulators available, for example, Apache JMeter, HP LoadRunner, and Radview Webload. These tools provide a good interface to monitor the test results. You can use any of these tools for determining the maximum handling capacity of an NSJSP instance.

---

# 13 Getting Started with MyFaces

This chapter describes how to develop a web application using the MyFaces framework. It describes the steps required to build a basic MyFaces sample called SkinSelector on your Windows system and deploy it on your NonStop system.

## Prerequisites

Before getting started, make sure that you have the following software installed on the NonStop and Windows system.

## NonStop System

The following software must be installed on the NonStop system:

- NonStop iTP WebServer version T8996H02 or later
- NSJSP version T1222H60 or later
- NSJ version T2766H60 or later

## Windows System

The following software must be installed on your Windows system:

- JDK version 1.5 or later
- Eclipse Galileo IDE version 3.3.1.1 or a later

**NOTE:** For more information about installing the software required on NonStop and Windows system, see “[Prerequisites](#)” (page 18).

## Overview of SkinSelector

SkinSelector is a basic web application developed using MyFaces. This application demonstrates how to use the basic MyFaces features such as:

- User Interface (UI) Components
- Managed Beans
- Navigation model

This section describes the following steps required to develop, deploy, and run the SkinSelector application:

- “[Developing SkinSelector on Windows using the Eclipse Galileo IDE](#)” (page 252)
- “[Deploying SkinSelector on NonStop](#)” (page 278)
- “[Running SkinSelector on NonStop](#)” (page 282)

## Developing SkinSelector on Windows using the Eclipse Galileo IDE

### NOTE:

- It is not mandatory for you to use the Eclipse Galileo IDE. You can use an IDE that supports Java.
- The screen captures in this section are based on Eclipse Galileo IDE version 3.3.1.1. The screen captures might look different if you use a different version of Eclipse.

The following activities are required to develop SkinSelector using the Eclipse Galileo IDE:

1. "Creating the Eclipse Workspace" (page 253)
2. "Creating a Dynamic Web Project" (page 254)
3. "Creating the `index.jsp` File" (page 258)
4. "Modifying the `web.xml` File" (page 261)
5. "Adding Dependency JAR Files to the Project Library" (page 262)
6. "Creating Views for SkinSelector" (page 265)
7. "Creating the Managed Beans" (page 271)
8. "Creating the Configuration File" (page 275)

## Creating the Eclipse Workspace

Do one of the following:

- Select the workspace you want to use.
- Create a new workspace using the Eclipse IDE.

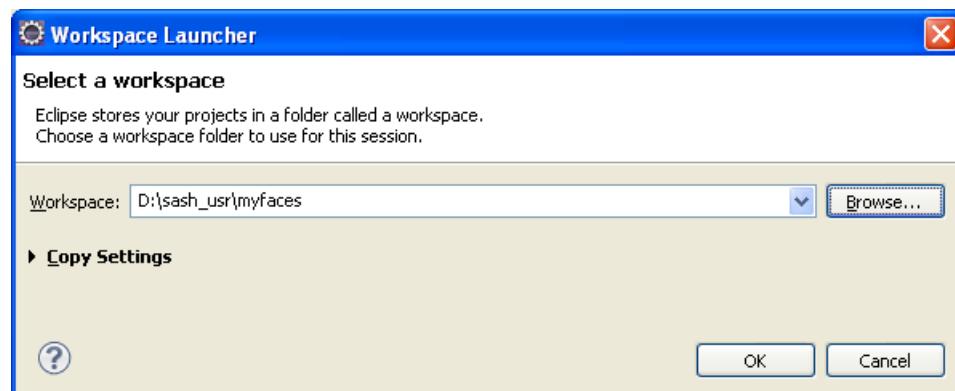
To create a new workspace using the Eclipse IDE, complete the following steps:

1. Double-click the `eclipse.exe` file in *<Eclipse IDE Installation directory>* to open the Eclipse IDE.

The Workspace Launcher dialog box appears. By default, the workspace is set to your existing workspace, if already created.

[Figure 72](#) shows the Workspace Launcher dialog box.

**Figure 72** Workspace Launcher Dialog Box



2. Click **OK**.

If you wish to create a new workspace, click **Browse** and select the folder you want to use as your workspace.

The Eclipse SDK Welcome screen appears.

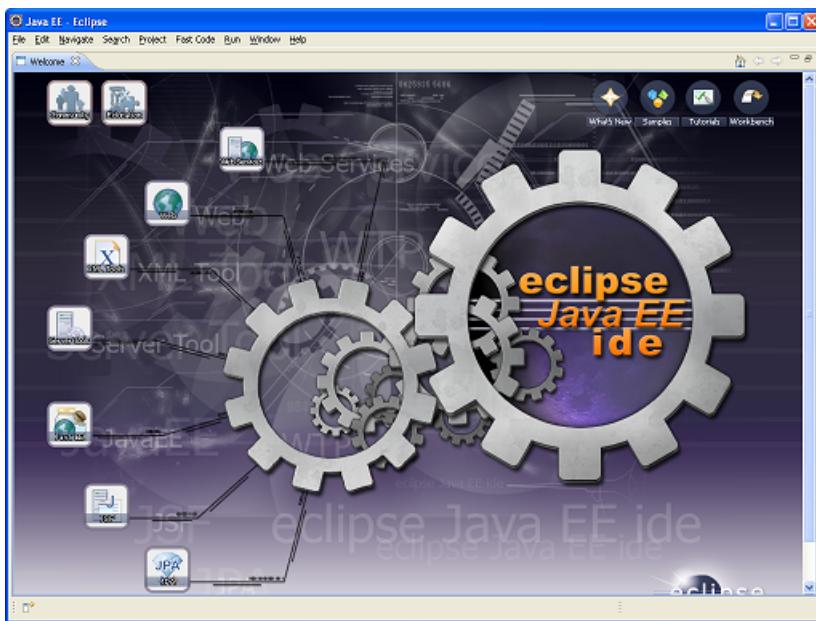
---

**NOTE:** D:\sash\_usr\myFaces is the sample workspace used to develop the SkinSelector application.

---

[Figure 73](#) shows the Eclipse SDK Welcome screen.

**Figure 73 Eclipse SDK Welcome Screen**



Close the welcome screen. The workspace is created.

---

**NOTE:** The source code for the SkinSelector application is located in the `SAMPLES.zip` file.

---

## Creating a Dynamic Web Project

To create a new Eclipse project for the SkinSelector application, complete the following steps:

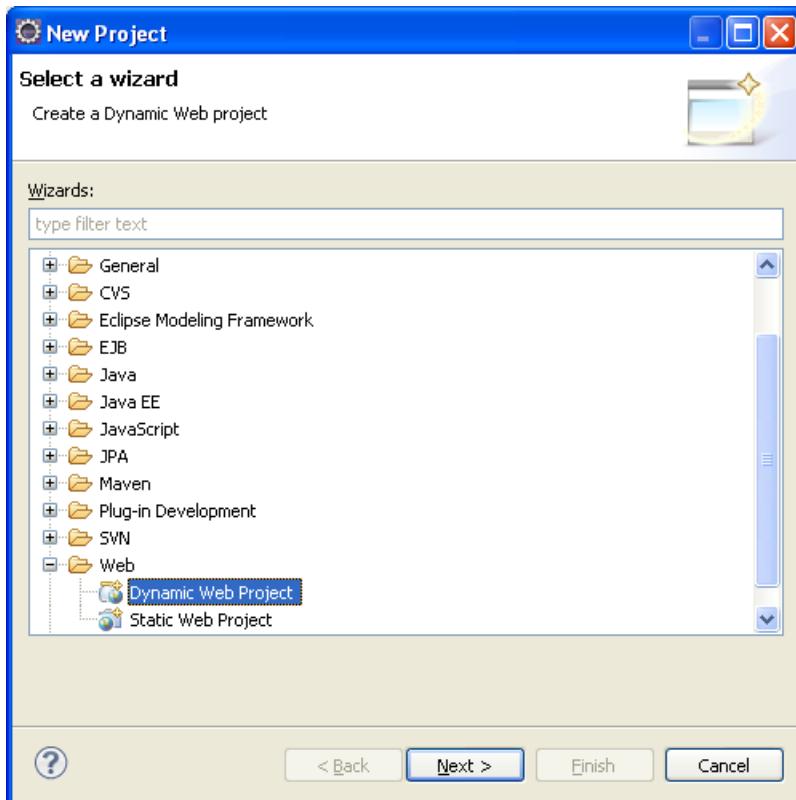
1. Click **File > New > Project**.

The New Project Wizard appears.

2. From the list of folders, select **Web > Dynamic Web Project** and click **Next**.

Figure 74 shows the New Project dialog box.

**Figure 74 New Project Dialog Box**



The New Dynamic Web Project dialog box appears.

3. In the Project name field, enter `SkinSelector` and click **Next**.

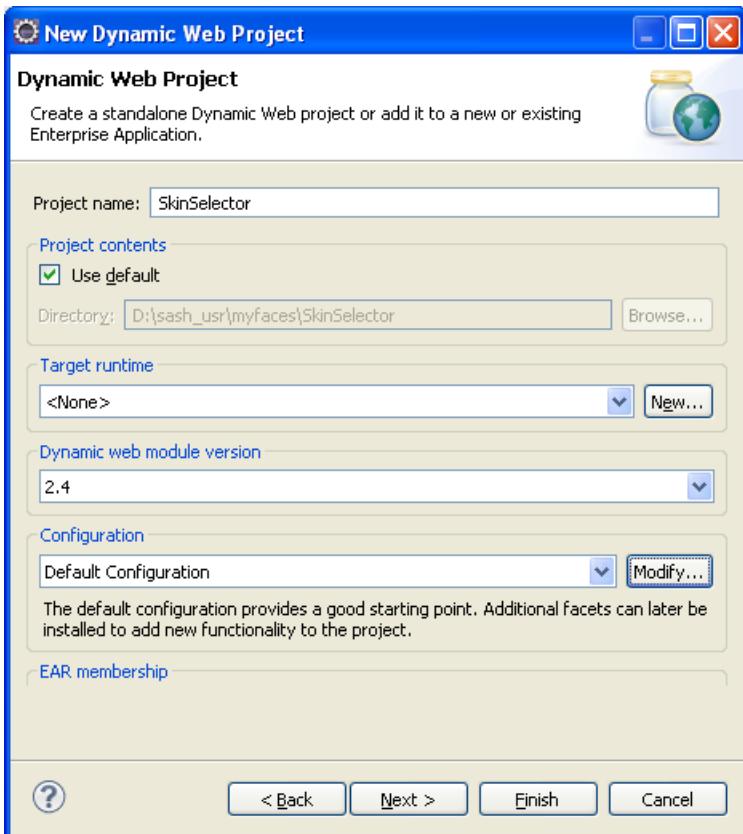
---

**NOTE:** The other fields in the New Dynamic Web Project screen are selected by default.

---

Figure 75 shows the New Dynamic Web Project dialog box.

**Figure 75 New Dynamic Web Project Dialog Box**



The New Dynamic Web Project: Project Facets screen appears.

4. Confirm that the Java version set for your project is 5.0 or later.

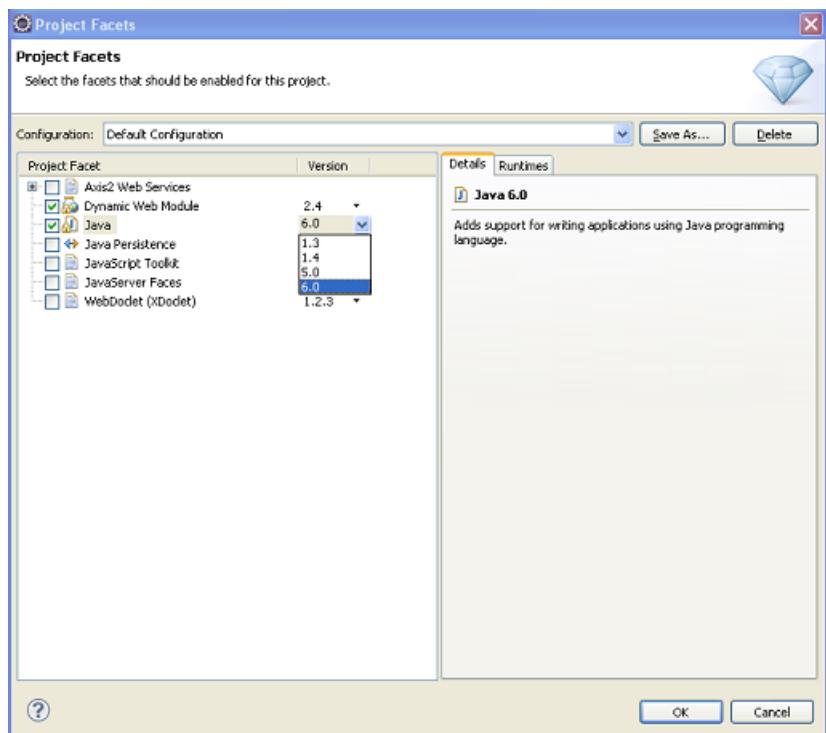
If the version is not 5.0 or later, select **Java** from the Project Facet list. Select 5.0 or a later version from the version list and click **Finish**.

---

**NOTE:** The Java version used in this example is 5.0.

Figure 76 shows the New Dynamic Web Project: Project Facets.

**Figure 76 New Dynamic Web Project: Project Facets**



When prompted, change the perspective to Java EE and click **Yes** to open the new perspective. This step is required because the Dynamic Web Project is associated with the Java EE perspective. The Project Structure appears.

5. In the Project Structure, confirm that the JRE System Library is set to JRE version 1.5 or later. If the JRE version is not set to 1.5, right-click **JRE System Library** to select **JRE 1.5** or later.

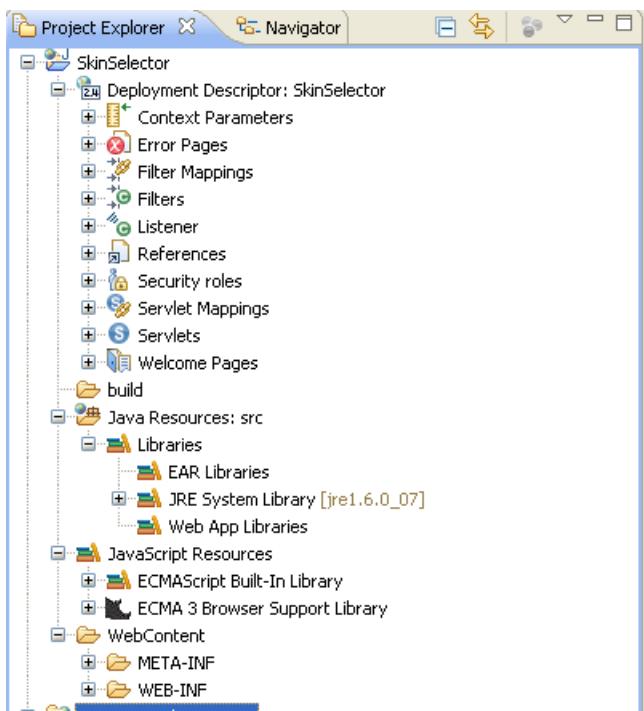
---

**NOTE:** The JRE version used in this example is 1.5.

The SkinSelector project and its directory structure is created.

Figure 77 shows the Project Explorer View.

**Figure 77 Project Explorer View**



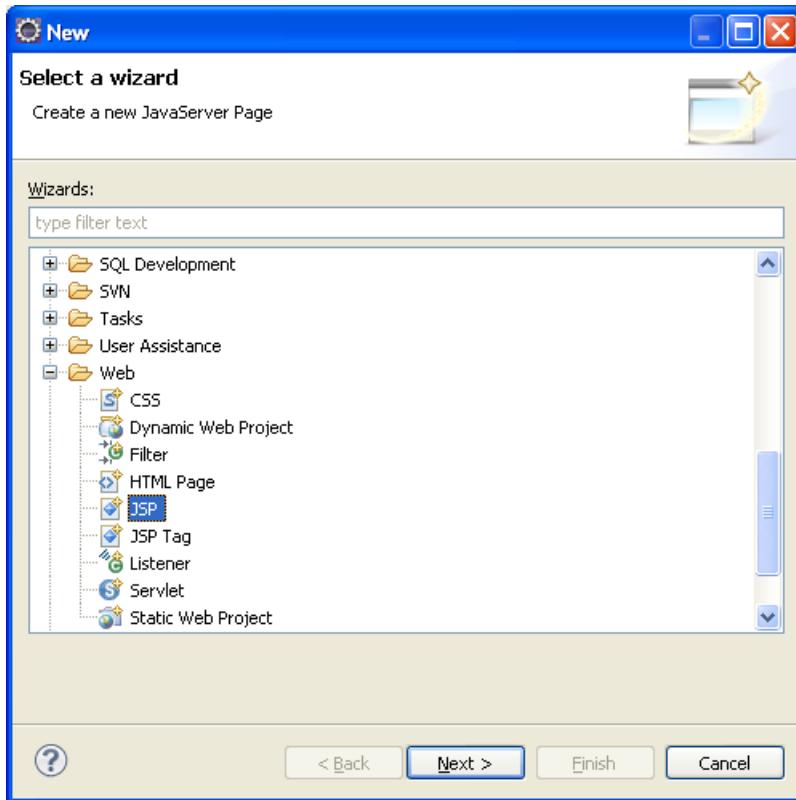
### Creating the `index.jsp` File

To create the `index.jsp` file in the `SkinSelector/WebContent` directory, complete the following steps:

1. On the Project Explorer view, right-click **SkinSelector** and select **New > Other**.  
The New File dialog box appears.
2. From the list of folders, select **Web > JSP** and click **Next**.

Figure 78 shows the New JSP File Selection dialog box.

**Figure 78 New JSP File Selection Dialog Box**

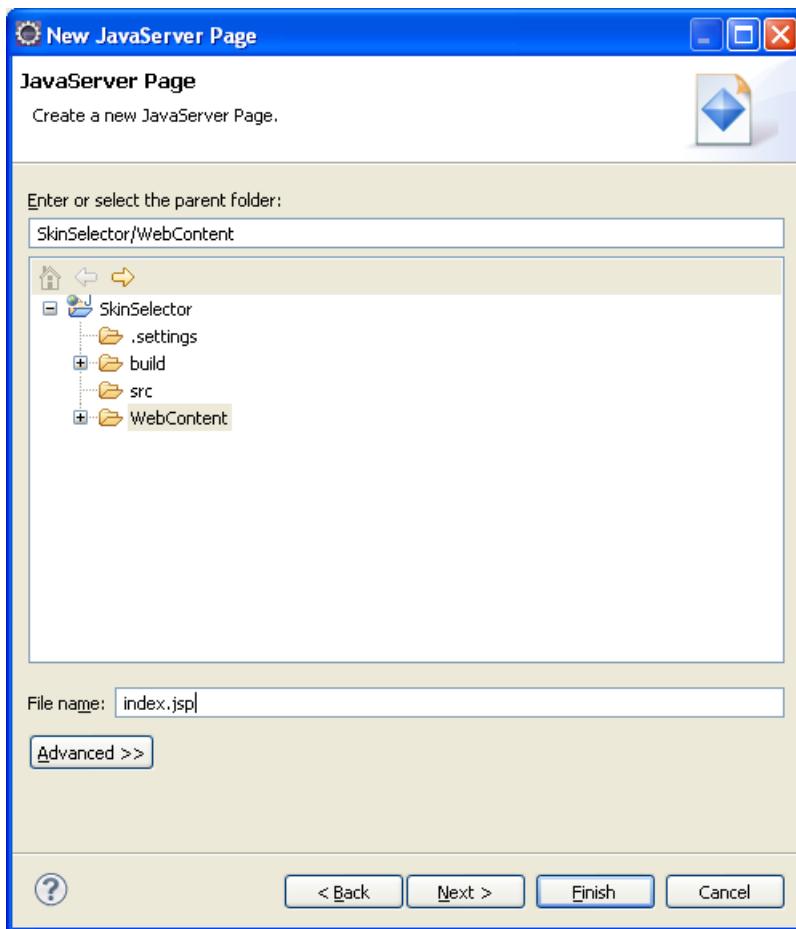


The New JavaServer Page dialog box appears.

3. In the File name field, enter **index.jsp** and ensure that the parent folder is set to SkinSelector/WebContent. Click **Next**.

Figure 79 shows the New JavaServer Page dialog box.

**Figure 79 New JavaServer Page Dialog Box**



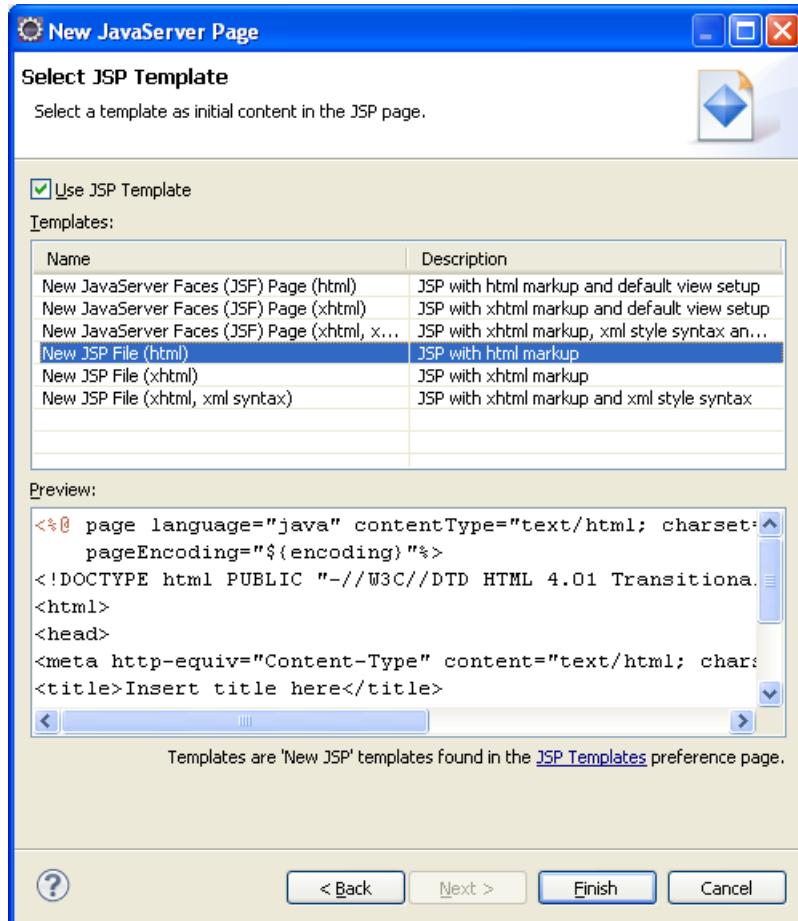
The New JavaServer Page: JSP Template dialog box appears.

4. Select **New JSP File (html)** and click **Finish**.

The template for the `index.jsp` file is generated.

Figure 80 shows the New JavaServer Page: JSP Template dialog box.

**Figure 80 New JavaServer Page: JSP Template Dialog Box**



5. Delete the default code present in the index.jsp file.
6. Add a redirection tag for faces-enabled page customize.jsp.

---

**NOTE:** The [Creating the customize.jsp File](#) section describes how to create the customize.jsp file.

---

The index.jsp file appears as follows:

```
<% response.sendRedirect ("customize.faces"); %>
```

## Modifying the web.xml File

To modify the web.xml file, you must set the dispatcher servlet and its mapping, using the following steps:

1. Double-click the SkinSelector/WebContent/WEB-INF/web.xml file in the Project Explorer frame to open it.

---

**NOTE:** By default, XML files open in the XML Editor. The XML Editor has two views: Design and Source view. Select the Source view.

---

The template for the default web.xml file generated during project creation is:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<display-name>
```

```

SkinSelector</display-name>
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
</welcome-file-list>
</web-app>

```

2. Set the `index.jsp` file as the `<welcome-file>` by deleting other files from the `<welcome-file-list>`.
3. Set the class for the Faces Controller servlet of the SkinSelector application by adding the following information in the `<webapps>` tag:

```

<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

```

4. Specify the URL pattern as `*.faces` in the `<servlet-mapping>` tag as shown below:

```

<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
</servlet-mapping>

```

---

**NOTE:** This servlet definition maps to the URL patterns. Any URL with the `.faces` extension will be routed to the Faces (Controller servlet).

---

After modification, the `web.xml` file appears as:

```

<?xml version="1.0"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
         version="2.4">

    <!-- Faces Servlet -->
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>
            javax.faces.webapp.FacesServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.faces</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>

```

## Adding Dependency JAR Files to the Project Library

Apart from the core MyFaces runtime JAR files, SkinSelector requires some dependency JAR files. Download the following distributions of the dependency JAR files on your Windows system:

---

**NOTE:** If you have already downloaded the servlet-api.jar, commons-el.jar, commons-lang-2.1.jar, standard.jar, and jstl.jar dependency JARs, as described in [Deploying and Running Sample MyFaces Application on NonStop](#) section, you need not download them again.

---

- servlet-api.jar

Download the servlet-api.jar.zip file from <http://www.java2s.com/Code/Jar/STUVWXYZ/Downloadservletapijar.htm>.

- commons-el.jar

Download the commons-el-1.0.zip or commons-el-1.0.tar.gz file from <http://archive.apache.org/dist/commons/el/binaries/>.

Both these files contain binary distributions. The only difference is in their compression format.

---

**NOTE:** Besides the commons-el package files, commons-el-1.0.zip, and commons-el-1.0.tar.gz, you will also find the links to their respective signature files, commons-el-1.0.zip.asc and commons-el-1.0.tar.gz.asc. You need not download the signature files.

---

- commons-lang-2.4.jar

Download commons-lang-2.4-bin.zip or commons-lang-2.4-bin.tar.gz from <http://archive.apache.org/dist/commons/lang/binaries/>.

Both these files contain the binary distributions. The only difference is in their compression format.

---

**NOTE:** Besides the commons-lang package files, commons-lang-2.4-bin.zip, and commons-lang-2.4-bin.tar.gz, you will also find the links to their respective signature files, commons-lang-2.4-bin.zip.asc and commons-lang-2.4-bin.tar.gz.asc. You need not download the signature files.

---

- standard.jar and jstl.jar

Download jakarta-taglibs-standard-1.1.2.zip or jakarta-taglibs-standard-1.1.2.tar.gz from <http://archive.apache.org/dist/jakarta/taglibs/standard/binaries/>.

Both these files contain the binary distributions. The only difference is in their compression format.

---

**NOTE:** Besides the Jakarta-taglibs package files, jakarta-taglibs-standard-1.1.2.zip and jakarta-taglibs-standard-1.1.2.tar.gz, you will also find the links to their respective signature files, jakarta-taglibs-standard-1.1.2.zip.asc and jakarta-taglibs-standard-1.1.2.tar.gz.asc. You need not download the signature files.

---

After the download is complete, extract these components to a location on your Windows system. Assume this location to be <MyFaces Dependencies>.

Because the SkinSelector application uses the MyFaces framework, it requires the following dependency JAR files.

**Table 8 MyFaces Dependency JAR Files**

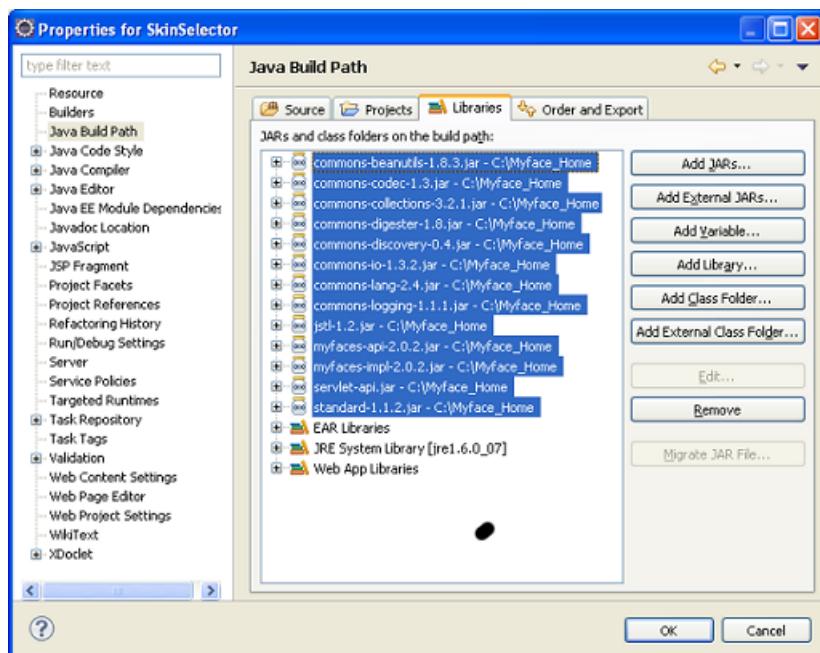
Dependency JAR Files	Source Location
commons-beanutils-1.8.3.jar	<MyFaces Home>\lib
commons-codec-1.3.jar	<MyFaces Home>\lib
commons-collections-3.2.1.jar	<MyFaces Home>\lib
commons-digester-1.8.jar	<MyFaces Home>\lib
commons-discovery-0.4.jar	<MyFaces Home>\lib
commons-logging-1.1.1.jar	<MyFaces Home>\lib
commons-lang-2.4.jar	<MyFaces Home>\lib
jstl.jar	<MyFaces Home>\lib
myfaces-api-2.0.2.jar	<MyFaces Home>\lib
myfaces-impl-2.0.2.jar	<MyFaces Home>\lib
servlet-api.jar	<MyFaces Home>\lib
standard-1.1.2.jar	<MyFaces Home>\lib

To add these dependency JAR files to the SkinSelector project library path, complete the following steps on the Eclipse IDE:

1. On the Project Explorer frame, right-click **SkinSelector** and select **Properties**.  
The SkinSelector Project Libraries - Java Build Path dialog box appears.
2. From the type filter text, select **Java Build Path**.
3. Click the **Libraries** tab.
4. Click **Add External JARs** to add each of the above-mentioned dependency JAR files. Browse to specify the location of each of the dependency JARs.

Figure 81 shows the SkinSelector Project Libraries - Java Build Path dialog box.

**Figure 81 SkinSelector Project Libraries - Java Build Path Dialog Box**



- Click **OK**.

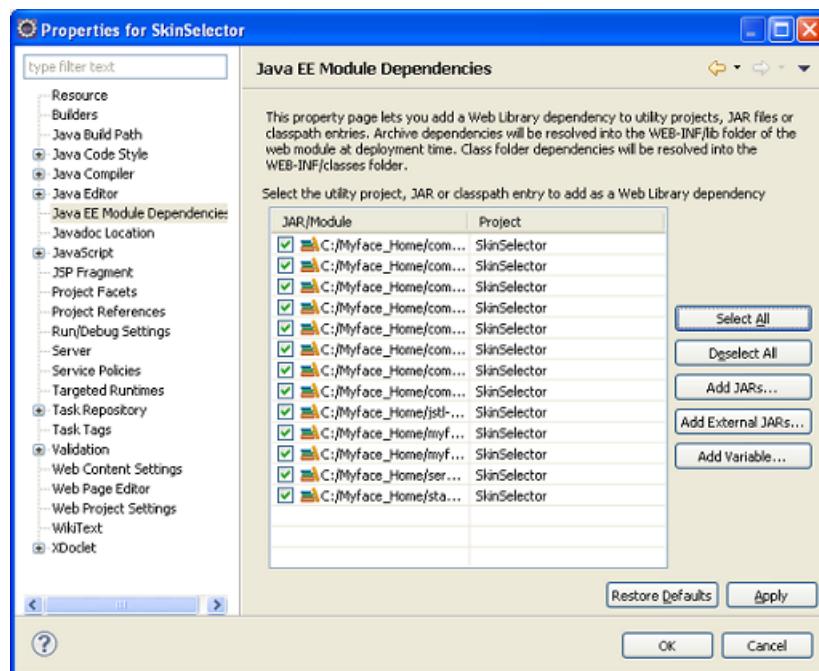
The dependency JAR files are added to the SkinSelector project library path.

To resolve the J2EE module dependency on the JAR files, complete the following steps on the Eclipse IDE:

- On the Project Explorer frame, right-click **SkinSelector** and select **Properties**.  
The SkinSelector Project Libraries - J2EE Module Dependencies dialog box appears.
- From the type filter text, select **J2EE Module Dependencies**.  
The list of added dependency JAR files appears.
- Select the required dependency JARs.

Figure 82 shows the SkinSelector Project Libraries - J2EE Module Dependencies dialog box.

**Figure 82 SkinSelector Project Libraries - J2EE Module Dependencies Dialog Box**



- Click **OK**.

The J2EE module dependency on the JAR files is resolved.

## Creating Views for SkinSelector

Creating Views for SkinSelector involves the following activities:

- "Creating the css and results Folder" (page 265)
- "Creating the style.css File" (page 266)
- "Modifying the style.css File" (page 267)
- "Creating the customize.jsp File" (page 268)
- "Modifying the customize.jsp File" (page 268)
- "Creating the same-color.jsp and show-preview.jsp Files" (page 269)
- "Modifying the same-color.jsp File" (page 269)
- "Modifying the show-preview.jsp File" (page 269)

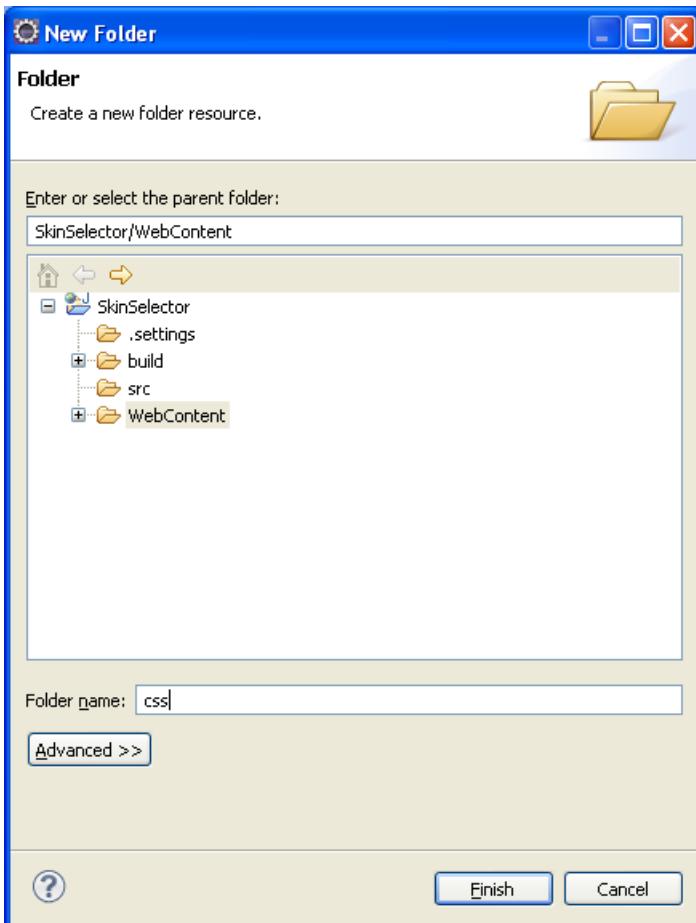
### Creating the css and results Folder

Create two new folders named **css** and **results**, to store the stylesheet file and the result JSP pages respectively.

To create the **css** folder, complete the following steps:

1. On the Project Explorer frame, right-click **SkinSelector** and select **New > Folder**.  
A New Folder dialog box appears.
2. Select the parent folder as **SkinSelector/WebContent** and in the Folder name, enter **css**.  
[Figure 83](#) shows the New Folder dialog box.

[Figure 83 New Folder Dialog Box](#)



3. Click **Finish**.

The css folder is created.

Similarly, create the **results** folder with **SkinSelector/WebContent/WEB-INF** as its parent folder.

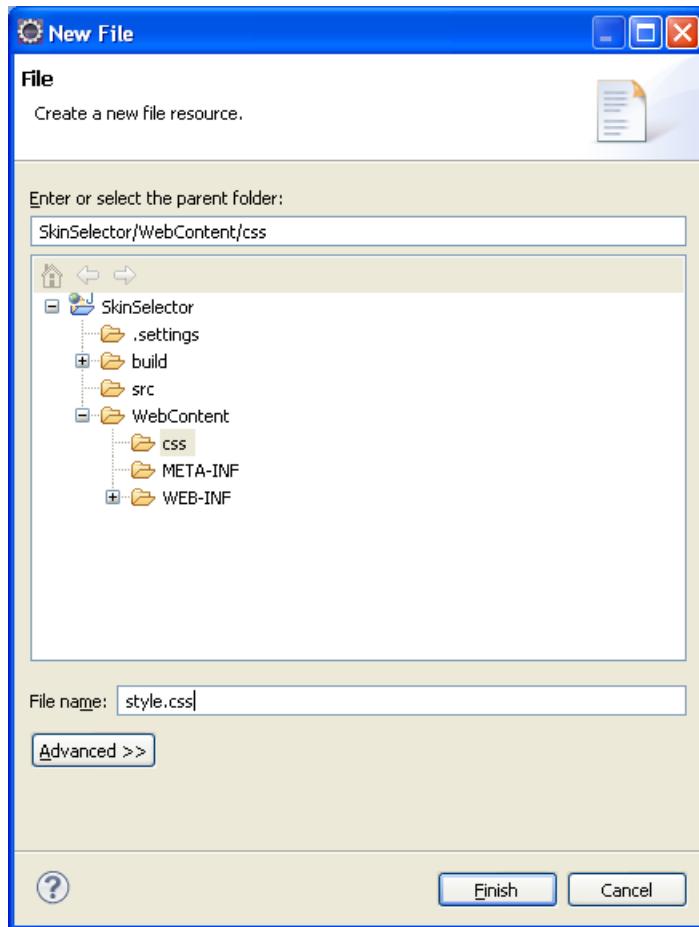
#### [Creating the style.css File](#)

Create a new file **style.css** to describe the presentation of JSP page.

To create the **style.css** file, complete the following steps:

1. On the Project Explorer frame, right-click **SkinSelector** and select **New > File**.  
The New File dialog box appears.
2. In the parent folder, select **SkinSelector/WebContent/css** and in the File name, enter **style.css**.  
[Figure 84](#) shows the New File dialog box.

**Figure 84 New File Dialog Box**



3. Click **Finish**.

The `style.css` file is created.

### Modifying the `style.css` File

To modify the `style.css` file, complete the following steps:

1. Double-click the `style.css` file in the Project Explorer frame to open it.
2. Add the following code in the `style.css` file to describe the presentation of the JSP pages of the SkinSelector:

```
BODY { background-color: #FDF5E6 }
A:hover { color: red }
H2 {
    color: #440000;
    text-align: center;
    font-family: Arial, Helvetica, sans-serif
}
TH.TITLE { background-color: #EF8429;
            font-size: 28px;
            font-family: Arial, Helvetica, sans-serif;
}
.WHITE {
    color: white;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 80%;
    font-weight: bold;
    text-decoration : none;
}
```

```
.DARK { background-color: black;
}
```

## Creating the customize.jsp File

Create a new JSP file named `customize.jsp` in the `SkinSelector\WebContent` to provide User Interface customizations for the SkinSelector application.

To create `customize.jsp`, follow the steps in the [Creating the index.jsp File](#) section.

## Modifying the customize.jsp File

To modify the `customize.jsp` file, complete the following steps:

1. Double-click the `customize.jsp` file in the Project Explorer frame to open it.
2. Replace the default contents of the `customize.jsp` file with the following content:

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Preview Resume</TITLE>
<LINK REL="STYLESHEET"
      HREF=".//css/style.css"
      TYPE="text/css">
</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=5>
    <TR><TH CLASS="TITLE">Preview Resume</TH></TR>
</TABLE>
<P>
<h:form>
    Use RGB instead of color names:
    <h:selectBooleanCheckbox
        valueChangeListener="#{resumeBean.changeColorMode}"
        onclick="submit()"
        immediate="true"/><BR>
    Foreground color:
    <h:selectOneMenu value="#{resumeBean.fgColor}"
                    disabled="#{!resumeBean.colorSupported}">
        <f:selectItems value="#{resumeBean.availableColors}" />
    </h:selectOneMenu>
    <BR>
    Background color:
    <h:selectOneMenu value="#{resumeBean.bgColor}"
                    disabled="#{!resumeBean.colorSupported}">
        <f:selectItems value="#{resumeBean.availableColors}" />
    </h:selectOneMenu><BR>
    <h:commandButton
        value="#{resumeBean.colorSupportLabel}"
        actionListener="#{resumeBean.toggleColorSupport}"
        immediate="true"/>
    <BR><HR WIDTH="25%"><BR>
    Name:
    <h:inputText value="#{resumeBean.name}" /><BR>
    Job Title:
    <h:inputText value="#{resumeBean.jobTitle}" /><P>
    <h:commandButton value="Show Preview"
                     action="#{resumeBean.showPreview}" />
</h:form>
</CENTER></BODY></HTML>
</f:view>
```

---

**NOTE:**

- **JSP directives for using JSF and HTML tag libraries**

The following JSP directives allow the JSP page to use the JSF Core and HTML tag libraries that are provided in the JSF specification's Reference Implementation.

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

- **MyFaces User Interface components**

JSF components, such as `<h:selectBooleanCheckbox>` and `<h:selectOneMenu>` provide Faces enabled User Interface.

`<h:selectOneMenu value="#{resumeBean.bgColor}"` is called the JSF value binding expression and provides direct linkage to the `bgColor` property of the managed bean `resumeBean`.

---

### Creating the same-color.jsp and show-preview.jsp Files

Create two new JSP files, `same-color.jsp` and `show-preview.jsp`, in the `SkinSelector/WebContent/WEB-INF/results` folder, by following the steps explained in the [Creating the index.jsp File](#) section.

#### Modifying the same-color.jsp File

To modify the `same-color.jsp` file, complete the following steps:

1. Double-click the `same-color.jsp` file in the Project Explorer frame to open it.
2. Replace the default content of the `same-color.jsp` file with the following code:

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Color Error</TITLE>
<LINK REL="STYLESHEET"
      HREF=".//css/styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=5>
  <TR><TH CLASS="TITLE">Color Error</TH></TR>
</TABLE>
<P>
You chose
" <h:outputText value="#{resumeBean.fgColor}" />" 
as both the foreground and background color.
<P>
<A HREF="customize.faces">try again</A>.
</CENTER></BODY></HTML>
</f:view>
```

#### Modifying the show-preview.jsp File

To modify the `show-preview.jsp` file, complete the following steps:

1. Double-click the `show-preview.jsp` file in the Project Explorer frame to open it.
2. Replace the default content of the `show-preview.jsp` file with the following code:

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

```

<f:view>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>
Resume for <h:outputText value="#{resumeBean.name}" />
</TITLE></HEAD>
<BODY TEXT="" BGCOLOR="">
<H1 ALIGN="CENTER">
<h:outputText value="#{resumeBean.name}" /><BR>
<SMALL><h:outputText value="#{resumeBean.jobTitle}" />
</SMALL></H1>
Experienced <h:outputText value="#{resumeBean.jobTitle}" />
seeks challenging position doing something.
<H2>Employment History</H2>
Employment History
<H2>Education</H2>
Education
<H2>Publications and Awards</H2>
Publication and Awards
</BODY></HTML>
</f:view>

```

---

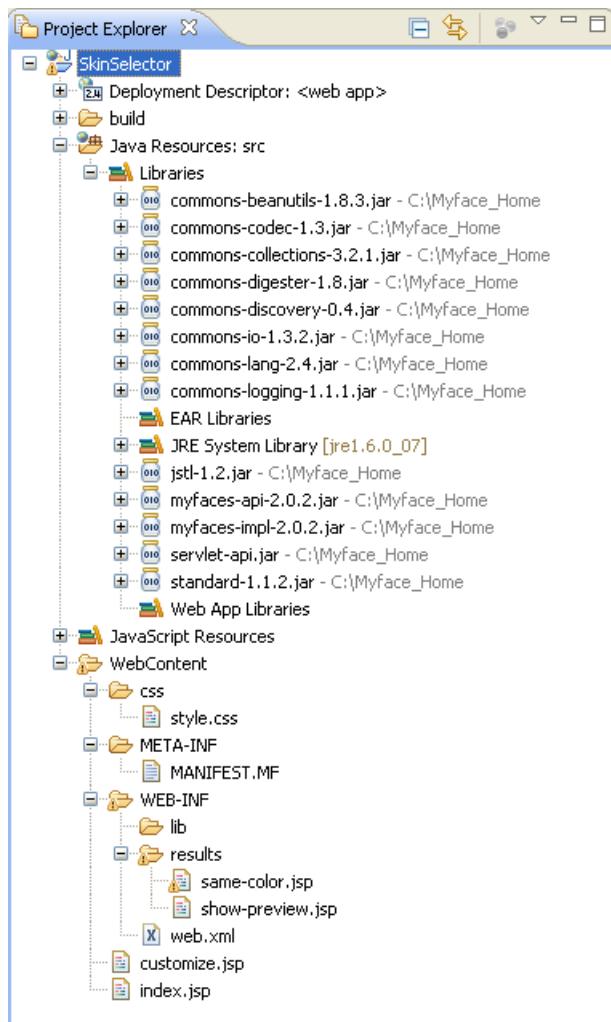
**NOTE:** The same-color.jsp file and the show-preview.jsp file are the resultant JSP pages. These pages are placed in the SkinSelector/WebContent/WEB-INF/results directory to avoid direct access through browser, and can be accessed only through faces mapping.

---

The project structure of SkinSelector, after following all the steps so far, is as follows:

Figure 85 shows the Project Explorer View.

**Figure 85 Project Explorer View**



## Creating the Managed Beans

A managed bean is a Java class for JSF applications. It is a Plain-Old Java Object (POJO) that conforms with the Java Beans naming conventions. For a JSF application to refer to its Java classes, their methods and properties, it must be available in the Java classpath and registered in the `faces-config.xml` () .

For information on `faces-config.xml`, see “[Creating the Configuration File](#)” (page 275).

Creating the Managed Beans, involves the following activities:

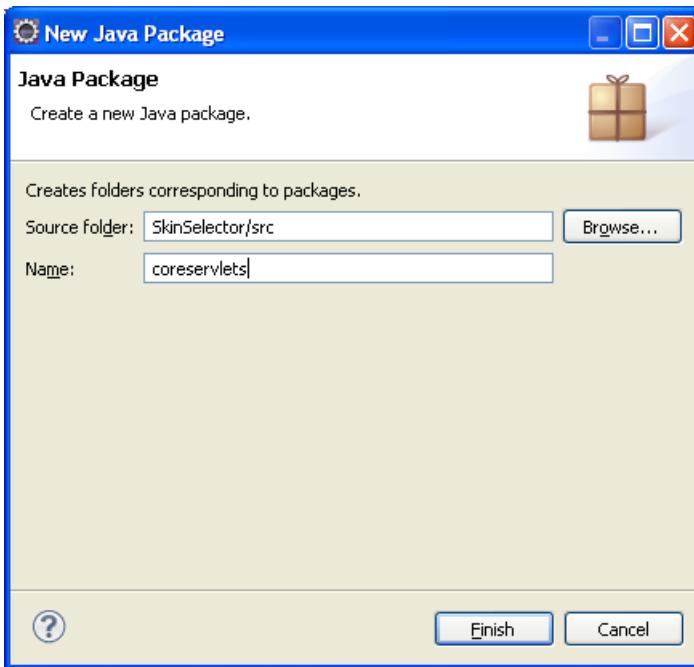
1. “[Creating a New Package](#)” (page 271)
2. “[Creating a New Java Class File](#)” (page 272)
3. “[Modifying the Java Class File](#)” (page 273)

### Creating a New Package

To create a new package named **coreservlets**, complete the following steps:

1. On the Project Explorer frame, right-click **SkinSelector**, and select **New > Package**.  
The New Java Package dialog box appears.
2. In the Source folder, select **SkinSelector/src** and enter the Name as **coreservlets**.  
**Figure 86** shows the New Java Package dialog box.

**Figure 86 New Java Package Dialog Box**



3. Click **Finish**.

The coreservlets package is created.

#### [Creating a New Java Class File](#)

To create a new Java class file in the **coreservlets** package, complete the following steps:

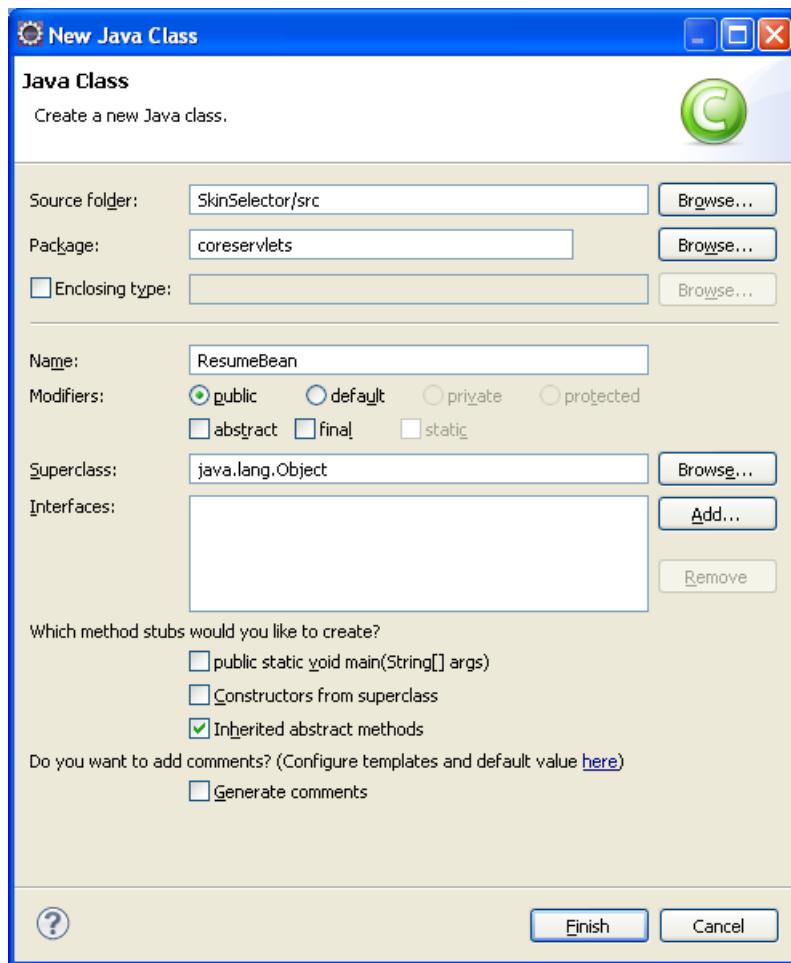
1. On the Project Explorer frame, right-click **coreservlets** package, and select **New > class**.  
The New Java Class dialog box appears.
2. Ensure that the source folder is **SkinSelector/src** and the parent package is **coreservlets**. In the class name field, enter **ResumeBean**.

---

**NOTE:** The other fields in the New Java Class dialog box are selected by default.

Figure 87 shows the New Java Class dialog box.

**Figure 87 New Java Class Dialog Box**



**3. Click **Finish****

The ResumeBean Java class is created.

### Modifying the Java Class File

To modify the ResumeBean.java file, complete the following steps:

1. Double-click the ResumeBean.java file in the Project Explorer frame to open it.
2. Replace the default contents of the ResumeBean.java file with the following code:

```
package coreservlets;

import javax.faces.model.*;
import javax.faces.event.*;
import java.io.*;

public class ResumeBean implements Serializable {
    private String name = "";
    private String jobTitle = "";
    private String fgColor = "BLACK";
    private String bgColor = "WHITE";
    private SelectItem[] availableColorNames =
        { new SelectItem("BLACK"),
          new SelectItem("WHITE"),
          new SelectItem("SILVER"),
          new SelectItem("RED"),
          new SelectItem("GREEN"),
          new SelectItem("BLUE") };
    private SelectItem[] availableColorValues =
```

```

    { new SelectItem("#000000"),
      new SelectItem("#FFFFFF"),
      new SelectItem("#C0C0C0"),
      new SelectItem("#FF0000"),
      new SelectItem("#00FF00"),
      new SelectItem("#0000FF") };
private boolean isColorSupported = true;
private boolean isUsingColorNames = true;

public String getName() { return(name); }

public void setName(String name) {
    this.name = name;
}

public String getJobTitle() { return(jobTitle); }

public void setJobTitle(String jobTitle) {
    this.jobTitle = jobTitle;
}

public String getFgColor() { return(fgColor); }

public void setFgColor(String fgColor) {
    this.fgColor = fgColor;
}

public String getBgColor() { return(bgColor); }

public void setBgColor(String bgColor) {
    this.bgColor = bgColor;
}

public SelectItem[] getAvailableColors() {
    if (isUsingColorNames) {
        return(availableColorNames);
    } else {
        return(availableColorValues);
    }
}

public boolean isColorSupported() { return(isColorSupported); }

public void toggleColorSupport(ActionEvent event) {
    isColorSupported = !isColorSupported;
}

public String getColorSupportLabel() {
    if (isColorSupported) {
        return("Disable Color Customization");
    } else {
        return("Enable Color Customization");
    }
}

public boolean isUsingColorNames() {
    return(isUsingColorNames);
}

public void setUsingColorNames(boolean isUsingColorNames) {
    this.isUsingColorNames = isUsingColorNames;
}

public void changeColorMode(ValueChangeEvent event) {
    boolean flag =

```

```
    ((Boolean)event.getNewValue()).booleanValue();
    setUsingColorNames(!flag);
}

public String showPreview() {
    if (isColorSupported && fgColor.equals(bgColor)) {
        return("same-color");
    } else {
        return("success");
    }
}
```

## Creating the Configuration File

Create the `faces-config.xml` file so that it includes navigation rules and a reference to the managed beans in the SkinSelector/WebContent/WEB-INF directory.

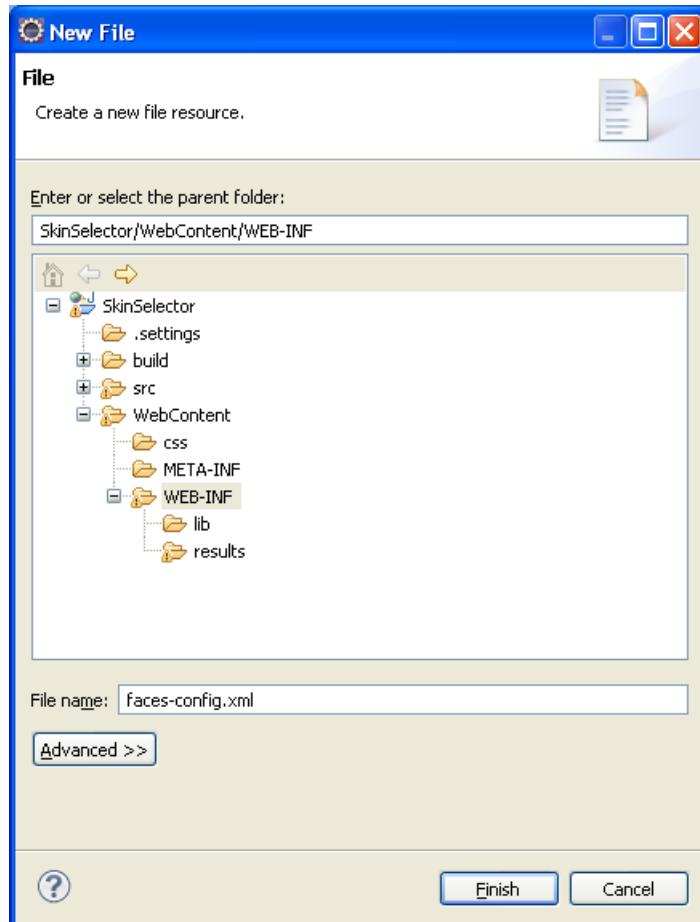
To write the configuration file, complete the following steps:

1. Create a new XML file.
  1. On the Project Explorer frame, right-click **SkinSelector** project, and select **New > File**.  
The New File dialog box appears.

2. In the File name field, enter `faces-config.xml` and select **SkinSelector/WebContent/WEB-INF** as the parent folder.

Figure 88 shows the New File dialog box.

**Figure 88 New File Dialog Box**



The `faces-config.xml` file is created.

**NOTE:** By default, XML files open in the XML Editor. The XML Editor has two views: Design and Source view. Select the Source view.

2. Add the following XML content in the `faces-config.xml` file to describe the navigation model of the application:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
  <managed-bean>
    <managed-bean-name>resumeBean</managed-bean-name>
    <managed-bean-class>coreservlets.ResumeBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>

  <navigation-rule>
    <from-view-id>/customize.jsp</from-view-id>
    <navigation-case>
      <from-outcome>same-color</from-outcome>
      <to-view-id>/WEB-INF/results/same-color.jsp</to-view-id>
    
```

```

</navigation-case>
<navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/WEB-INF/results/show-preview.jsp</to-view-id>
</navigation-case>
</navigation-rule>

</faces-config>

```

---

**NOTE:** The resumeBean is registered as a managed bean in the faces-config.xml file, which is described in the following part of the faces-config.xml:

```

<managed-bean>
    <managed-bean-name>resumeBean</managed-bean-name>
    <managed-bean-class>coreservlets.ResumeBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

```

The navigation rules in the faces-config.xml are described under the `<navigation-rule>` tag:

```

<navigation-rule>
    -----
    <navigation-case>
        -----
        </navigation-case>
    -----
<navigation-rule>

```

To understand the navigation rule consider `customize.jsp`, which has the `<h:commandButton>` component included as:

```

<h:commandButton value="Show Preview"
                  action="#{resumeBean.showPreview}" />

```

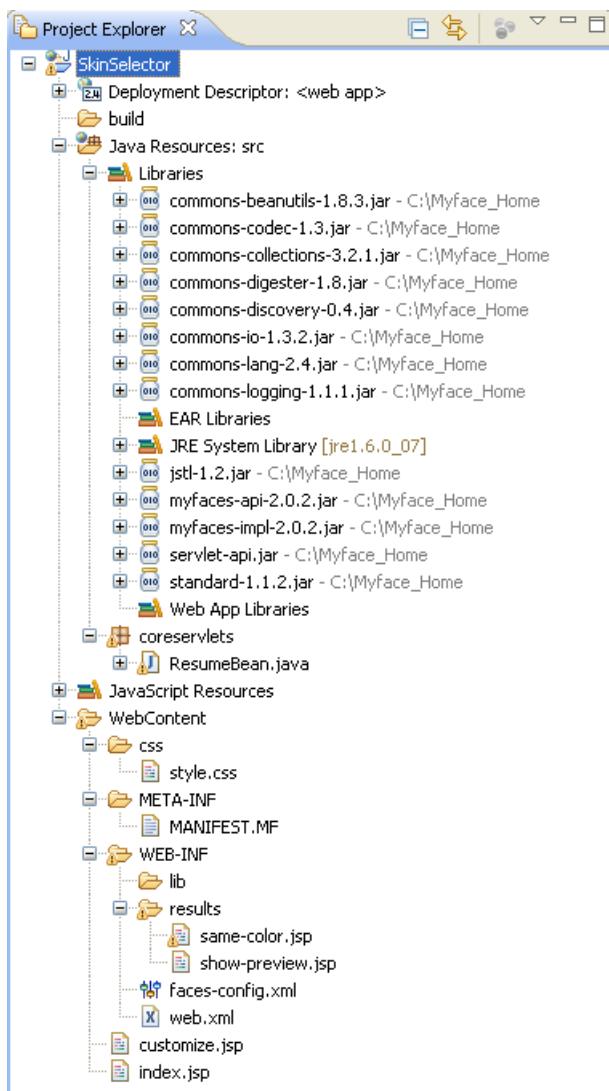
This `commandButton` uses the JSF action "`#{resumeBean.showPreview}`" to navigate to `same-color.jsp` or `show-preview.jsp` depending on the result. The corresponding rules are mentioned in the faces-config.xml file.

---

The SkinSelector project looks as follows:

[Figure 89](#) shows the Project Explorer View.

**Figure 89 Project Explorer View**



## Deploying SkinSelector on NonStop

This section describes the following activities:

1. “Creating the SkinSelector Application WAR File on Windows” (page 278)
2. “Deploying the SkinSelector WAR File in NSJSP on NonStop” (page 280)

### Creating the SkinSelector Application WAR File on Windows

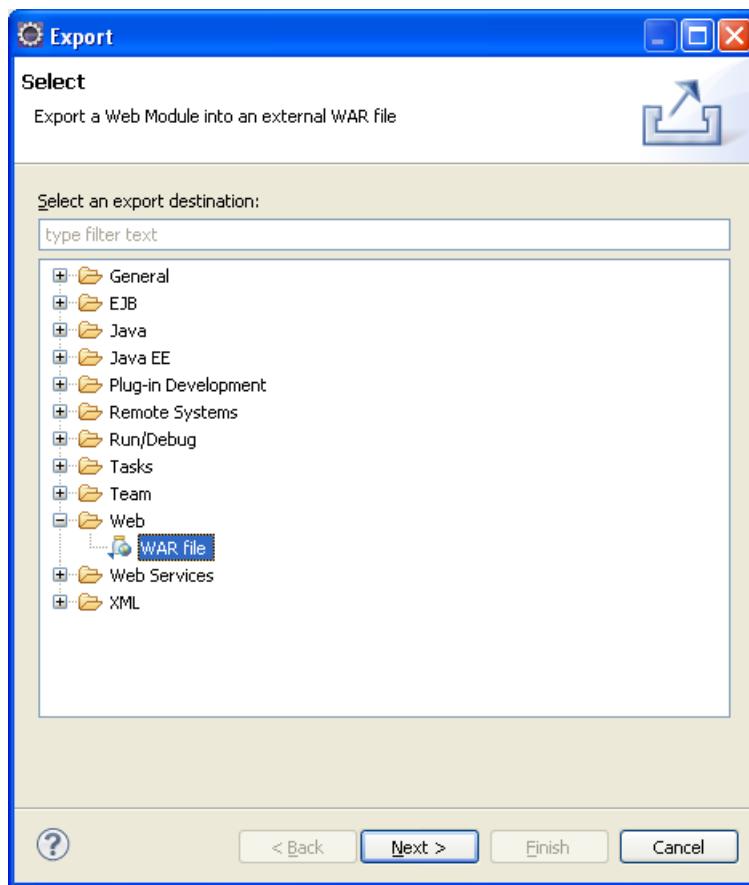
A WAR file is essential to deploy the web application. This file contains the Java class file, JSPs, and configuration XML files.

To create the SkinSelector Application WAR file on Windows, complete the following steps:

1. On the Project Explorer frame, right-click **SkinSelector**, and select **Export > Export**.  
The Export dialog box appears.
2. From the list of folders, select **Web > WAR file**. Click **Next**.

Figure 90 shows the Export dialog box.

**Figure 90 Export Dialog Box**

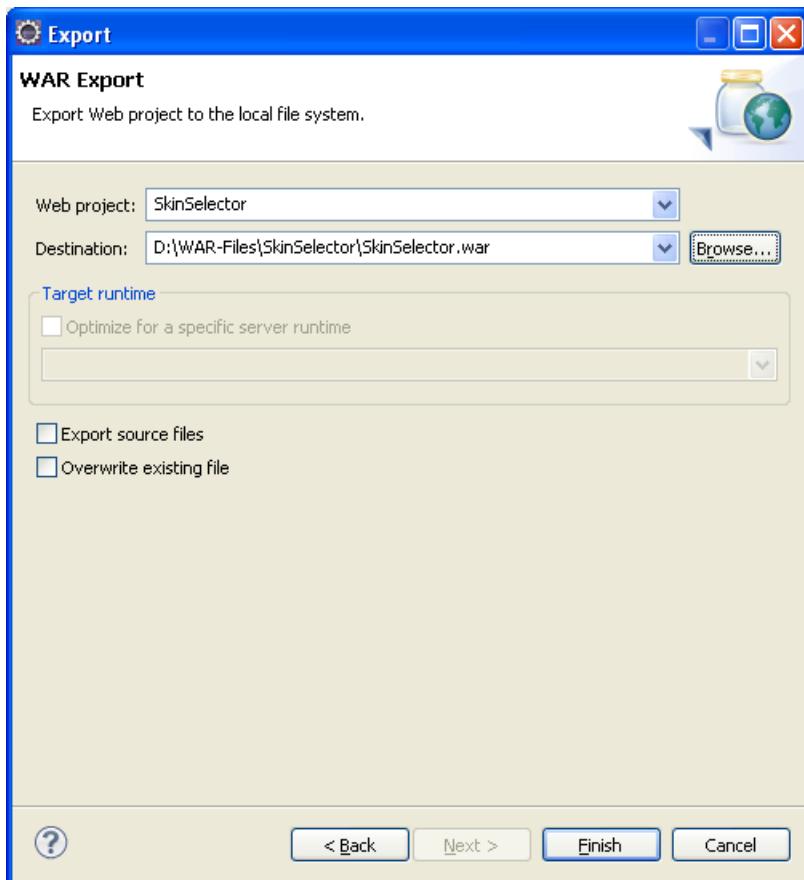


The WAR Export dialog box appears.

3. In the Web module field, enter **SkinSelector** and browse to the destination file to save the WAR file.

Figure 91 shows the WAR Export dialog box.

**Figure 91 WAR Export Dialog Box**



4. Click **Finish**.

The WAR file is created.

**NOTE:** If you have specified an existing name for the WAR file, **Finish** will be disabled. In this case, change the name of the WAR file or select the Overwrite existing file check box.

## Deploying the SkinSelector WAR File in NSJSP on NonStop

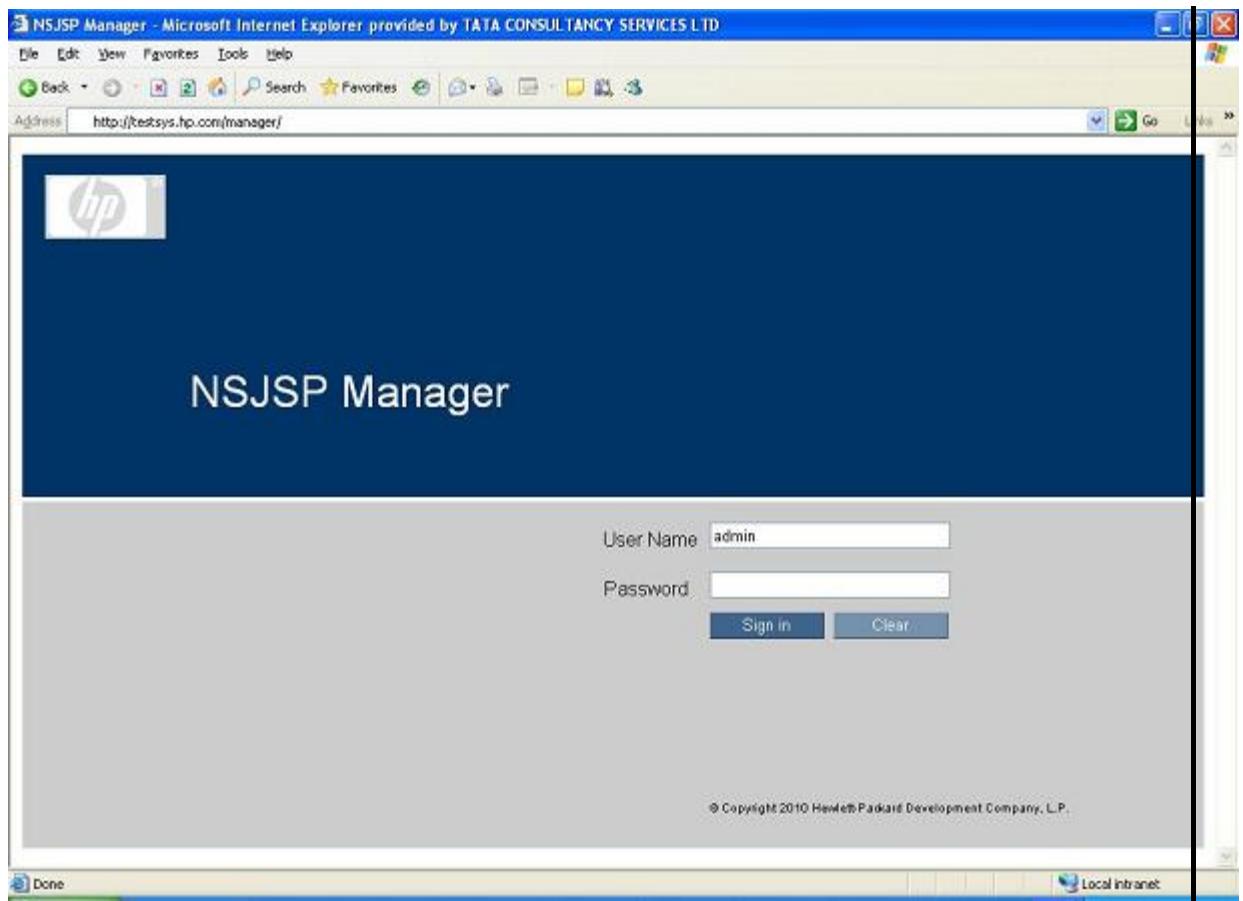
To deploy the SkinSelector WAR file on your NonStop system, complete the following steps:

1. Go to [http://<IP Address of the iTP WebServer>:<port#>/<manager\\_directory>](http://<IP Address of the iTP WebServer>:<port#>/<manager_directory>).

The **NSJSP Manager Login** screen appears.

Figure 92 shows the **NSJSP Manager Login** screen.

**Figure 92 NSJSP Manager Login Screen**

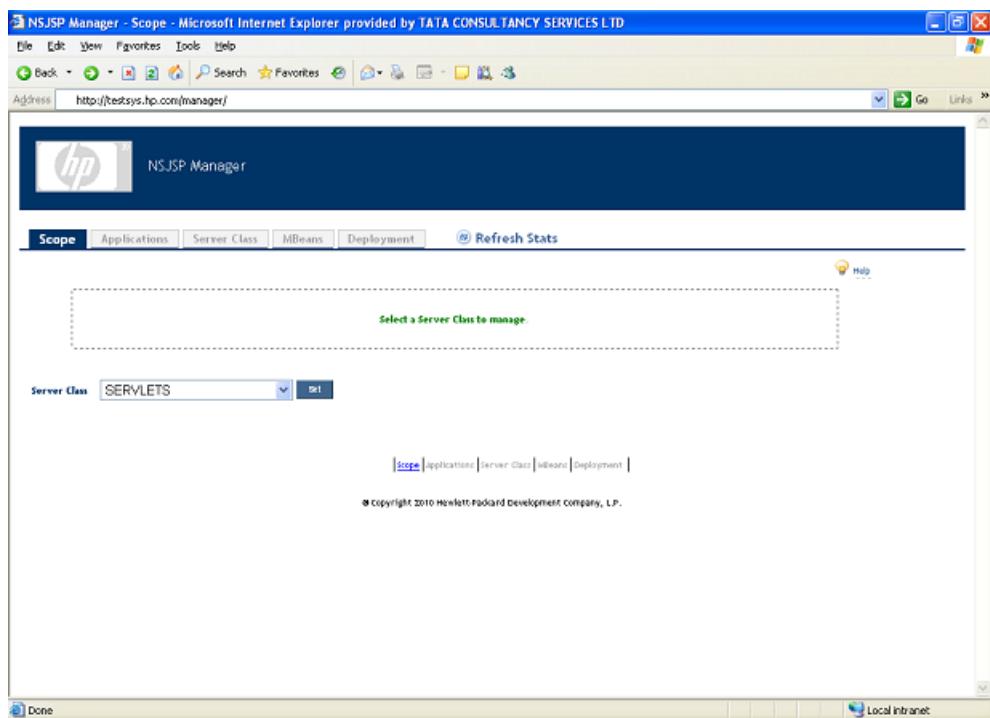


2. Enter the **User name:** and **Password:** and click **OK**.

The **NSJSP Manager** screen appears.

Figure 93 shows the **NSJSP Manager** screen.

**Figure 93 NSJSP Manager Screen**

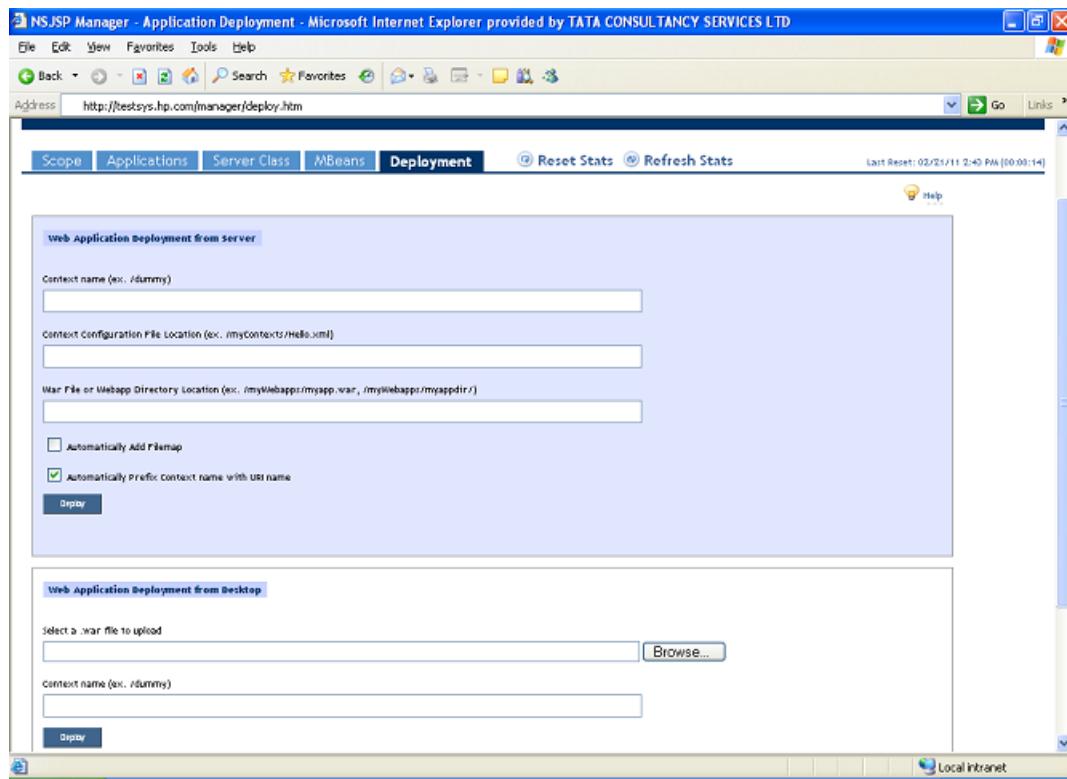


- Select **SERVLETS** for the **Server Class** and click **Set**.

The **Host (in server.xml)** tab is enabled and populated with **localhost**. Click **Set**, which enables the **Deployment** tab on the **NSJSP Manager** screen.

Figure 94 shows the **Deployment** tab of **NSJSP Manager**.

**Figure 94 NSJSP Manager Screen - Deployment tab**



- In the **Deployment** tab, complete the following steps in the **Web Application Deployment from Desktop** section:
  - In the **Select a .war file to upload** field, click **Browse...** and locate the `SkinSelector.war` file on the Windows system.
  - (Optional) In the **Context name** field, enter a name for the application context.
  - Click **Deploy**.

`SkinSelector` is deployed on NSJSP and is listed under **Applications** tab of the **NSJSP Manager** screen.

---

**NOTE:** HP recommends that you use the NSJSP manager for deployment. Deployment using the FTP or any other mechanism is not recommended. For more information on using the NSJSP manager, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

---

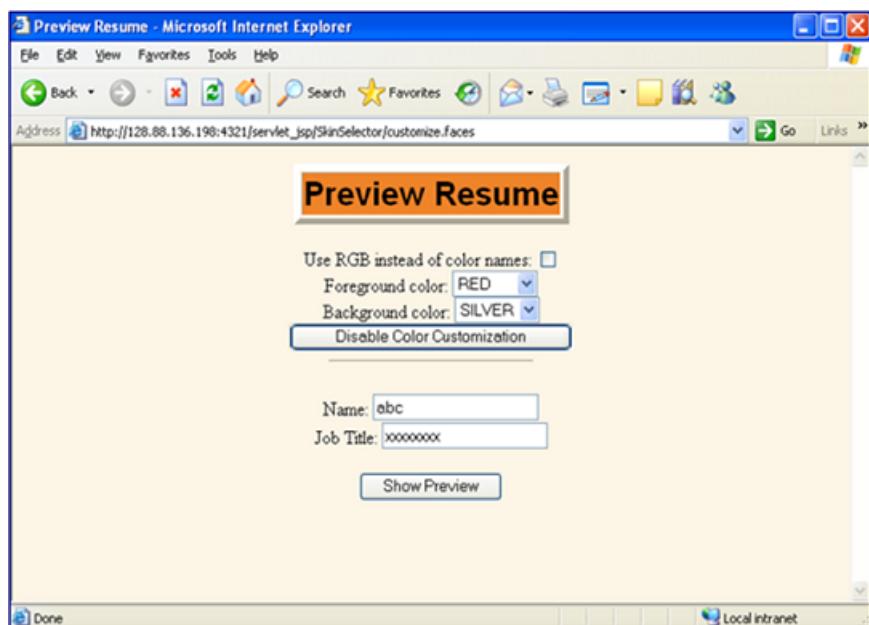
## Running SkinSelector on NonStop

To run SkinSelector on your NonStop system, click the `/<servlet directory>/SkinSelector` path under **Applications** in the NSJSP Web Application Manager screen.

The **Preview Resume** dialog box appears.

Figure 95 shows the **Preview Resume** dialog box.

**Figure 95 Preview Resume Dialog Box**

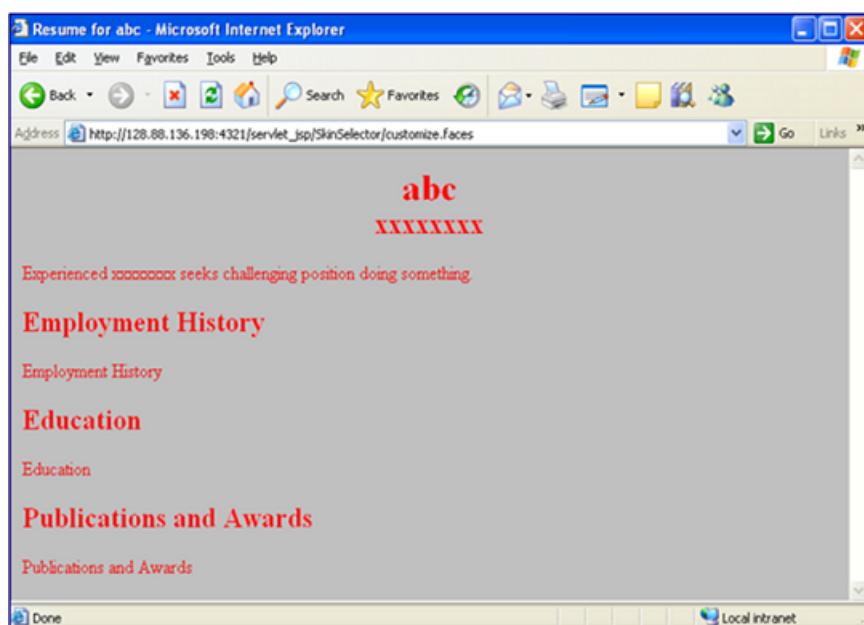


Click **Show Preview**.

- If you select different foreground and background colors, the screen displays the Employee resume (`show-preview.jsp`) appears.

Figure 96 shows the Employee Resume Screen.

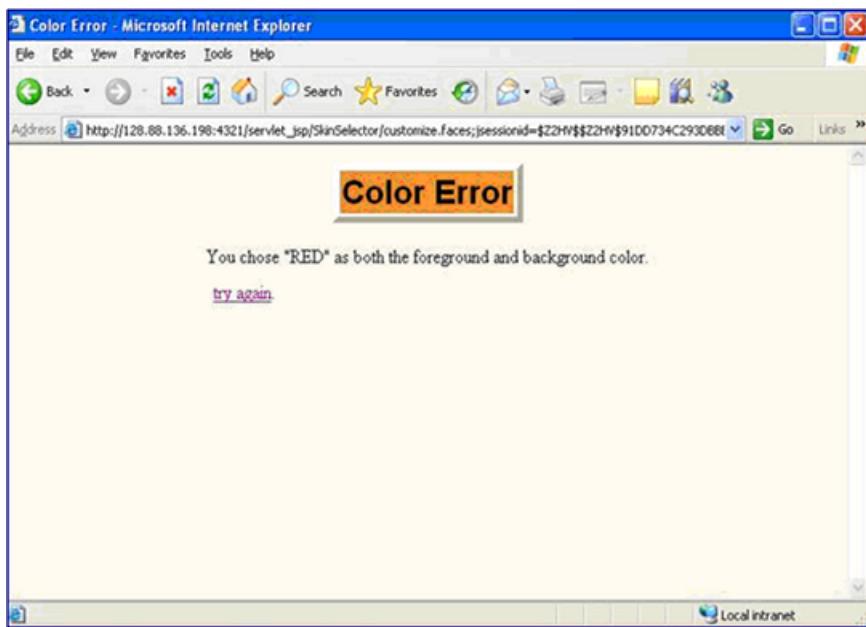
**Figure 96 Employee Resume Screen**



- If you select the same foreground and background color, the try-again option appears on the Color Error page (`same-color.jsp`).

Figure 97 shows the Color Error Screen.

**Figure 97 Color Error Screen**



# H Installing MyFaces Trinidad Framework Libraries on NonStop

Installing MyFaces Trinidad framework libraries on a NonStop system requires the following :

- “[Downloading MyFaces Trinidad Distribution](#)” (page 285)
- “[Copying MyFaces Trinidad Runtime Libraries from Windows to NonStop](#)” (page 285)

## Downloading MyFaces Trinidad Distribution

To download the MyFaces Trinidad distribution on your Windows system, complete the following steps:

1. Go to <http://www.apache.org/dist/myfaces/binaries/>.

A web page showing a list of all MyFaces distributions available for download is displayed.

2. Select trinidad-1.2.11-dist.zip or trinidad-1.2.11-dist.tar.gz.

3. Extract the trinidad-1.2.11-dist.zip or trinidad-1.2.11-dist.tar.gz file into a directory on your Windows system.

This directory will be referred as the *<MyFaces Trinidad Home>*.

**NOTE:** The trinidad-1.2.11-dist.zip and trinidad-1.2.11-dist.tar.gz files include the MyFaces Trinidad framework libraries that are required to build MyFaces Trinidad applications. These files only differ in their compression formats.

## Copying MyFaces Trinidad Runtime Libraries from Windows to NonStop

To copy the MyFaces Trinidad runtime libraries from your Windows system to your NonStop system, complete the following steps:

1. Go to *<MyFaces Trinidad Home>* and create JAR files of the *<MyFaces Trinidad Home>\lib* directory on your Windows system using the following commands:

```
command prompt> cd "<MyFaces Trinidad Home>"  
command prompt> jar -cvf MyFaces-trinidad_lib.jar lib
```

For example:

```
command prompt> cd "C:\trinidad-1.2.11"  
command prompt> jar -cvf MyFaces-trinidad_lib.jar lib
```

2. Create a version-specific MyFaces Trinidad directory *<NonStop MyFaces Trinidad Home>* in the OSS environment on your NonStop system using the command:

```
OSS> mkdir -p <NonStop MyFaces Trinidad Home>
```

For example, create a directory structure /usr/tandem/sash/trinidad-1.2.11

```
OSS> mkdir -p /usr/tandem/sash/Trinidad-1.2.11
```

3. Copy MyFaces-trinidad\_lib.jar from *<MyFaces Trinidad Home>* to *<NonStop MyFaces Trinidad Home>* and extract it using the OSS command:

```
OSS> cd <NonStop MyFaces Trinidad Home>  
OSS> jar -xvf MyFaces-trinidad_lib.jar
```

This completes the installation of MyFaces Trinidad runtime libraries on your NonStop system. You can use these libraries when developing and running MyFaces applications using Trinidad components on a NonStop system, using the steps similar to those for installing the MyFaces framework.

# Installing MyFaces Tomahawk Framework Libraries on NonStop

Installing the MyFaces Tomahawk framework libraries on a NonStop system requires the following:

- “[Downloading MyFaces Tomahawk Distribution on Windows](#)” (page 286)
- “[Copying MyFaces Tomahawk Runtime Libraries from Windows to NonStop](#)” (page 286)

## Downloading MyFaces Tomahawk Distribution on Windows

To download the MyFaces Tomahawk distribution on your Windows system, complete the following steps:

1. Go to <http://www.apache.org/dist/myfaces/binaries/>.  
A web page showing a list of all MyFaces distributions available for download is displayed.
2. Select tomahawk12-1.1.8-bin.zip or tomahawk12-1.1.8-bin.tar.gz.
3. Extract the tomahawk12-1.1.8-bin.zip or tomahawk12-1.1.8-bin.tar.gz file into a directory on your Windows system.

This directory will be referred as the *<MyFaces Tomahawk Home>*.

**NOTE:** The tomahawk12-1.1.8-bin.zip and tomahawk12-1.1.8-bin.tar.gz files include the MyFaces Tomahawk framework libraries that are required to build MyFaces Tomahawk applications. These files only differ in their compression formats.

## Copying MyFaces Tomahawk Runtime Libraries from Windows to NonStop

To copy the MyFaces Tomahawk runtime libraries from your Windows system to your NonStop system, complete the following steps:

1. Go to *<MyFaces Tomahawk Home>* and create JAR files of the *<MyFaces Tomahawk Home>\lib* directory on your Windows system using this command:

```
command prompt> cd "<MyFaces Tomahawk Home>"  
command prompt> jar -cvf MyFaces-tomahawk_lib.jar lib
```

For example:

```
command prompt> cd "C:\tomahawk12-1.1.8"  
command prompt> jar -cvf MyFaces-tomahawk_lib.jar lib
```

2. Create a version-specific MyFaces Tomahawk directory *<NonStop MyFaces Tomahawk Home>* in the OSS environment on your NonStop system using the command:

```
OSS> mkdir -p <NonStop MyFaces Tomahawk Home>
```

For example, create a directory structure /usr/tandem/sash/tomahawk12-1.1.8

```
OSS> mkdir -p /usr/tandem/sash/tomahawk12-1.1.8
```

3. Copy MyFaces-tomahawk\_lib.jar from *<MyFaces Tomahawk Home>* to *<NonStop MyFaces Tomahawk Home>* and extract it using the OSS command:

```
OSS> cd <NonStop MyFaces Tomahawk Home>  
OSS> jar -xvf MyFaces-tomahawk_lib.jar
```

This completes the installation of MyFaces Tomahawk runtime libraries on your NonStop system. You can use these libraries when developing and running MyFaces applications using Tomahawk components on a NonStop system, using the steps similar to those for MyFaces framework.

---

## Part IV Axis2/Java Framework

Part II describes how an Axis2/Java framework can be customized to work on a NonStop system. It includes the following chapters:

- [“Axis2/Java Overview” \(page 289\)](#)

This chapter provides an overview of the Axis2/Java framework, its architecture, key features, advantages, and message flow.

- [“Installing Axis2/Java Framework” \(page 292\)](#)

This chapter helps you to install the Axis2/Java framework and enables you to develop, set up, deploy, and run the web application and its client on your NonStop system.

- [“Configuring Axis2/Java Applications on NonStop Systems” \(page 307\)](#)

This chapter helps you to configure the Axis2/Java application on your NonStop system.

- [“Getting Started with Axis2/Java” \(page 319\)](#)

This chapter helps you to develop a web application on your Windows system using Axis2/Java and deploy and run the web application on your NonStop system.

---

# Contents

<b>14 Axis2/Java Overview.....</b>	<b>289</b>
Apache Axis2/Java Project.....	289
Axis2/Java Framework.....	289
Axis2/Java Modules.....	289
Axis2/Java Applications on NonStop.....	289
<b>15 Installing Axis2/Java Framework.....</b>	<b>292</b>
Prerequisites.....	292
NonStop System.....	292
Windows System.....	292
Installing Axis2/Java on NonStop.....	292
Downloading the Axis2/Java Distribution on Windows.....	292
Building the Axis2/Java Web Archive using Standard Binary Distribution.....	293
Deploying Axis2/Java as a Web Application on NonStop.....	294
Running Axis2/Java on NonStop.....	296
Installing the Sandesha2 Module.....	297
Downloading the Sandesha2 Distribution on Windows.....	297
Modifying the Configuration File on Windows.....	298
Copying the Sandesha2 Binaries to the Axis2/Java Directory.....	298
Modifying the Build File on Windows.....	298
Building the Axis2/Java Web Archive with Sandesha2.....	299
Deploying the Axis2/Java Web Archive with Sandesha2.....	299
Installing the Rampart Module.....	299
Downloading the Axis2/Java Distribution on Windows.....	299
Copying the Rampart Binaries to Axis2/Java.....	300
Building the Axis2/Java Web Archive with Rampart.....	300
Deploying the Axis2/Java Web Archive with Rampart.....	300
Deploying and Running Axis2/Java Sample Applications on NonStop.....	300
FaultHandling.....	301
MTOM.....	303
<b>16 Configuring Axis2/Java Applications on NonStop Systems.....</b>	<b>307</b>
NonStop Platform Configurations.....	307
Determining the Application Parameters.....	307
Determining the Maximum Capacity of an NSJSP Instance.....	307
Configuring iTP WebServer for Axis2/Java Applications.....	308
Configuring NSJSP for Axis2/Java Applications.....	312
Axis2/Java Framework Configurations for NonStop Systems.....	317
Global Configuration.....	317
Service Configuration.....	318
Module Configuration.....	318
<b>17 Getting Started with Axis2/Java.....</b>	<b>319</b>
Prerequisites.....	319
NonStop System.....	319
Windows System.....	319
Overview of TemperatureConverter.....	319
Code-First approach.....	319
Contract-First approach.....	336

# 14 Axis2/Java Overview

The Axis2/Java framework enables you to implement client and server sides of a web service based on the SOAP protocol.

**NOTE:** Any mention of Axis2/Java in this document refers to Axis2/Java version 1.5.2.

## Apache Axis2/Java Project

The Axis2/Java project is an Apache Software Foundation initiative and is a configurable web services, SOAP, and WSDL engine.

For more information on the Axis2/Java framework, see <http://axis.apache.org/axis2/java/core/index.html>.

## Axis2/Java Framework

The Axis2/Java framework enables you to create web services with different features based on your requirements. For example, web services created using Axis2/Java can be configured to be accessed using either the SOAP method or the REST method.

Some of the key functionalities of the Axis2/Java framework are as follows:

- Send, receive, and process SOAP messages with or without attachments.
- Create a Web service from plain Java class.
- Create implementation classes for both server and client using Web Services Description Language (WSDL) files.
- Retrieve WSDL files for a service.
- Create and utilize REST based web services.

## Axis2/Java Modules

Axis2/Java provides a modular architecture that enables easy integration of modules, which can be used to extend the features of the core Axis2/Java engine. The Axis2/Java modules currently certified on HP NonStop are:

- Apache Sandesha2 – extends support for the WS-ReliableMessaging specifications
- Apache Rampart - extends support for the WS-Security

For more information about installing these modules in Axis2/Java, see “[Installing Axis2/Java Framework](#)” (page 292).

## Axis2/Java Applications on NonStop

Axis2/Java can be deployed/run on NonStop as a standalone server or as a servlet under NSJSP. When deployed as a servlet under NSJSP, Axis2/Java inherits the benefits of NSJSP, such as, scalability, hot deployment, easy manageability and so on. Applications developed using Axis2/Java can take advantage of these features on the NonStop platform.

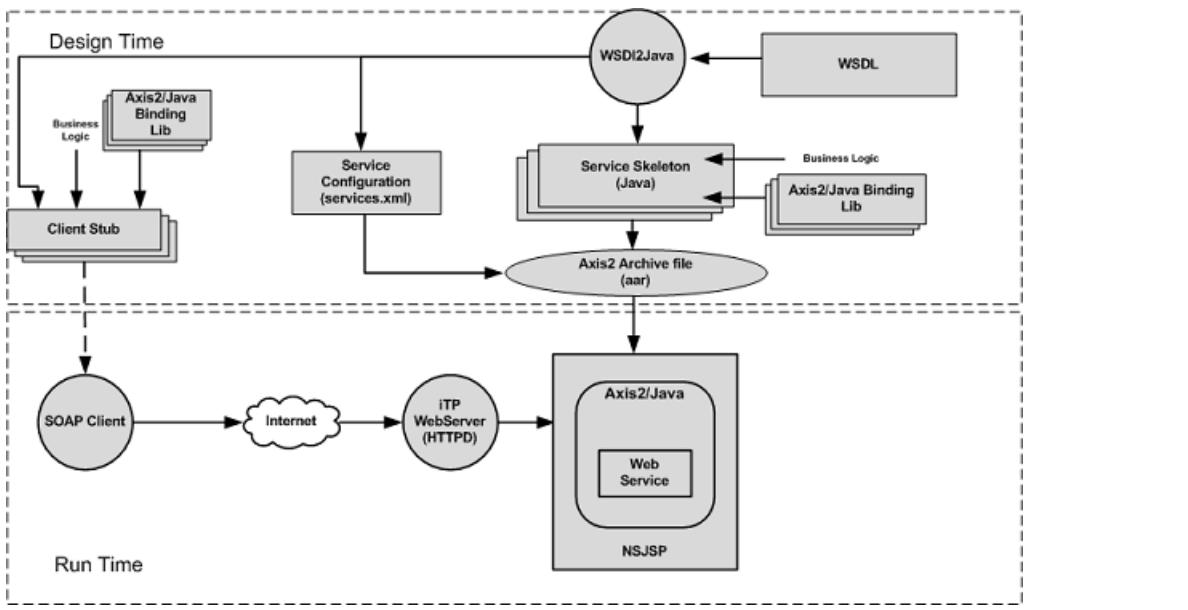
The following are some of the common usage scenarios for Axis2/Java based applications on the NonStop SOAP platform as a follows:

### 1. Developing web services based on existing contract (WSDL)

You can use the Contract-First application development approach to develop web applications based on an existing web services contract (WSDL). Axis2/Java provides the WSDL2Java tool, which simplify the Contract-First application development with Axis2/Java.

[Figure 98](#) shows a typical scenario of the design time activities, deployment tasks, and exposing web services with Axis2/Java.

**Figure 98 Axis2/Java Contract-First Implementation**



At design time, you can use the WSDL2Java tool, which accepts a WSDL file as input, and generates the service skeleton files, client stubs (optional), and service configuration files based on the specifications in the WSDL file.

The generated service skeleton files include Java classes that the service requires to interface with the Axis2/Java server. Thereafter, you need to implement the business logic and build a web archive file, which can then be deployed as a web service.

The web service deployed under NSJSP can be accessed using any SOAP clients on or outside the NonStop platform. These SOAP clients can be developed using the client stubs generated using the WSDL2Java tool or any other SOAP client development methodologies.

The WSDL2Java can also be used to generate SOAP client stubs in Java, based on the WSDL file. The generated client stubs include classes to generate the SOAP message in the SOAP request, consume the SOAP response, and send the SOAP request. You need to implement the client business logic in the client.

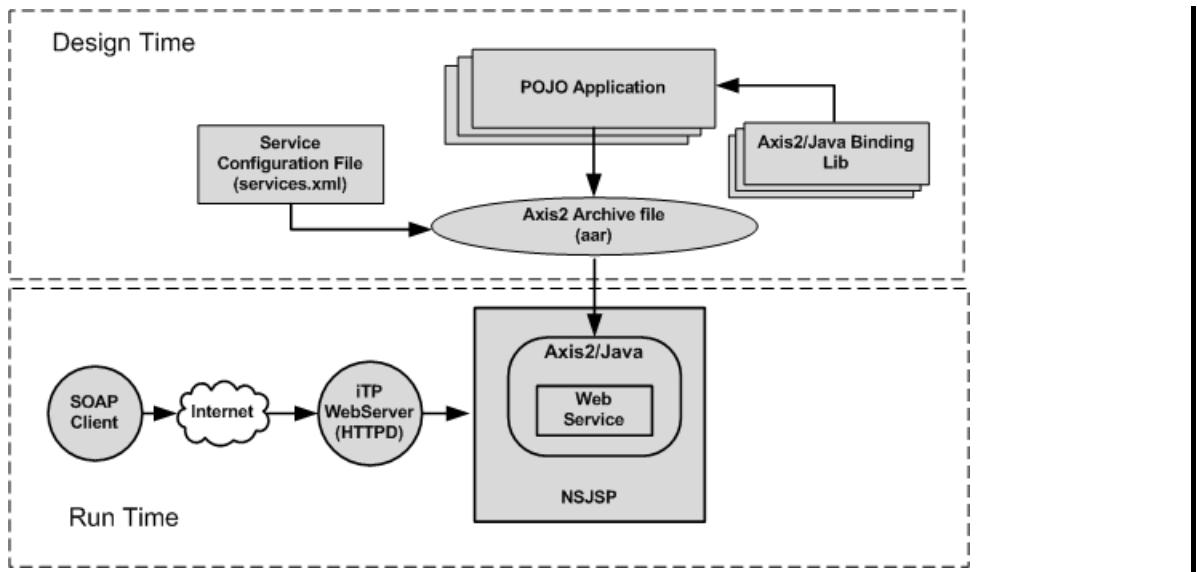
For more information on creating web services and clients using the Contract-First approach, see [“Getting Started with Axis2/Java” \(page 319\)](#).

## 2. Exposing plain Java class as web service

Axis2/Java enables you to expose plain Java class as web services.

[Figure 99](#) shows a typical scenario of design time activities, deployment tasks, and exposing web services using Axis2/Java.

**Figure 99 Axis2/Java Code-First Implementation**



In [Figure 99](#), Code-First Approach is used to design and develop a typical application, which can be exposed as a web service using Axis2/Java.

At design time, the Java class and the service configuration file are created and bundled together to form an archive file (.aar). The Java class contains the business logic and the service configuration file, services.xml contains the web service configuration.

At runtime, this archive file is exposed as a web service. Axis2/Java creates a WSDL which defines the interfaces for the web service and the web service can then be accessed using SOAP clients on or outside the NonStop platform.

For more information on creating web services using this approach, see “[Getting Started with Axis2/Java](#)” (page 319).

# 15 Installing Axis2/Java Framework

This chapter discusses the steps required to complete the following activities:

1. “[Installing Axis2/Java on NonStop](#)” (page 292)
2. “[Deploying and Running Axis2/Java Sample Applications on NonStop](#)” (page 300)

## Prerequisites

Before getting started, make sure that you have the requisite software installed on your NonStop and Windows system.

### NonStop System

The following software must be installed on your NonStop system:

- NonStop iTP WebServer version T8996H02 or later
- NSJSP version T1222H60 or later
- NSJ version T2766H60 or later

### Windows System

The following software must be installed on your Windows system:

- JDK version 1.5 or later
- Ant version 1.7.0 or later

**NOTE:** For detailed information on the above-mentioned software, see the [Prerequisites](#) section.

## Installing Axis2/Java on NonStop

Installing Axis2/Java on a NonStop system requires the following actions:

1. “[Downloading the Axis2/Java Distribution on Windows](#)” (page 292)
2. “[Building the Axis2/Java Web Archive using Standard Binary Distribution](#)” (page 293)
3. “[Deploying Axis2/Java as a Web Application on NonStop](#)” (page 294)
4. “[Running Axis2/Java on NonStop](#)” (page 296)

**NOTE:** Throughout the chapter, references are made to the following directories:

- *<Axis2 Home>*: This is the directory on your Windows system where the Axis2/Java distribution files are extracted.
- *<NonStop Axis2 Home>*: This is the directory on your NonStop system where the Axis2/Java files are located.
- *<My SASH Home>*: This is the directory on your Windows system where the contents of the SAMPLES file (distributed as a part of the NS Samples for Java Frameworks - T0874 and available for download in Scout for NonStop Servers) is extracted.
- *<Sandesha2 Home>*: This is the directory on your Windows system where the Sandesha2 module distribution files are extracted.
- *<Rampart Home>*: This is the directory on your Windows system where the Rampart module distribution files are extracted.

### Downloading the Axis2/Java Distribution on Windows

To download the Axis2/Java distribution on your Windows system, complete the following steps:

1. Go to <http://axis.apache.org/axis2/java/core/download.cgi>.

The **Axis2 1.5.2 Release** web page, displaying a list of distributions available for download appears.

---

**NOTE:** The following distributions are available for download:

- **Standard Binary Distribution:** includes the Axis2/Java binaries and sample applications.
  - **Source Distribution:** includes the source of the Axis2 standard distribution.
  - **Web Archive (WAR) Distribution:** includes the web application to be deployed on servlet containers.
  - **Documents Distribution:** includes the documentation for Axis2/Java.
- 

2. Download the **Standard Binary Distribution** in the zip format.

**NOTE:** The sample applications (FaultHandling and MTOM) discussed in this section have been verified using Axis2/Java version 1.5.2 only.

---

3. Extract the **Standard Binary Distribution** (`axis2-1.5.2-bin.zip`) into a directory on your Windows system.

This directory will be referred as `<Axis2 Home>`. Among other directories, `<Axis2 Home>` contains the following sub-directories:

`\conf`

includes the global configurations needed to run Axis2/Java applications.

`\samples`

includes sample applications demonstrating different Axis2/Java features.

`\lib`

includes dependent libraries required to build Axis2/Java and run samples.

`\repository\modules`

includes module archive files such as addressing modules.

`\repository\services`

includes Axis2/Java WebService archive files.

`\bin`

includes scripts useful for building services and clients, and starting the Axis2/Java server.

`\webapp`

includes the resources to build the Axis2/Java web archive (`axis2.war`).

## Building the Axis2/Java Web Archive using Standard Binary Distribution

To build the Axis2/Java web archive (`axis2.war`) on your Windows system using the Standard Binary Distribution, complete the following steps:

1. Go to `<Axis2 Home>\webapp` on your Windows system using the command:

command prompt> cd `<Axis2 Home>\webapp`

For example:

command prompt> cd C:\axis2-1.5.2\webapp

where,

C:\axis2-1.5.2-bin is the `<Axis2 Home>`

2. Build the Axis2/Java web archive (`axis2.war`) using the command:

command prompt> ant create.war

After successful build, the Axis2/Java web archive (axis2.war) is created in the <Axis2 Home>\dist directory.

## Deploying Axis2/Java as a Web Application on NonStop

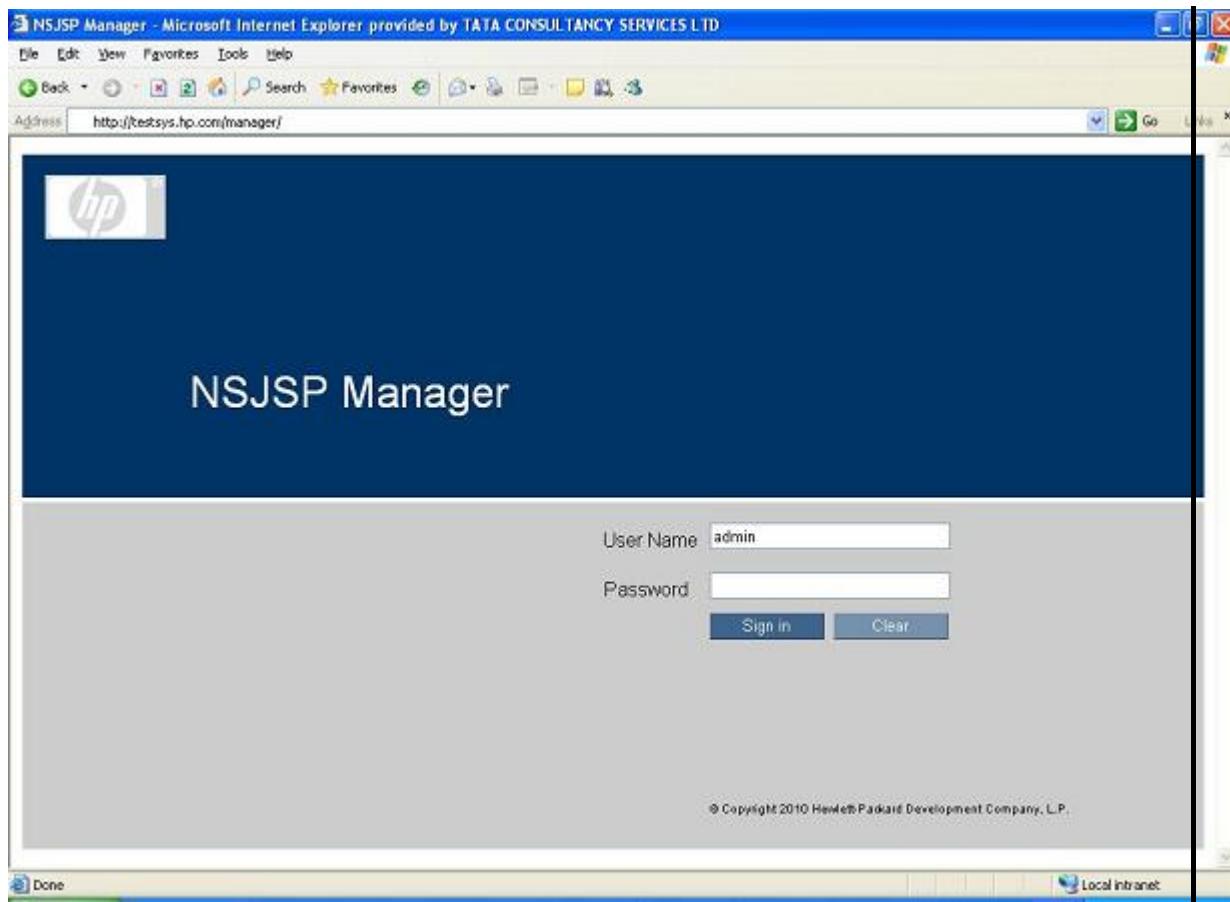
To deploy Axis2/Java on your NonStop system, complete the following steps:

1. Go to [http://<IP Address of the iTP WebServer>:<port#>/<manager\\_directory>](http://<IP Address of the iTP WebServer>:<port#>/<manager_directory>).

The **NSJSP Manager Login** screen appears.

Figure 100 shows the **NSJSP Manager Login** screen.

**Figure 100 NSJSP Manager Login Screen**

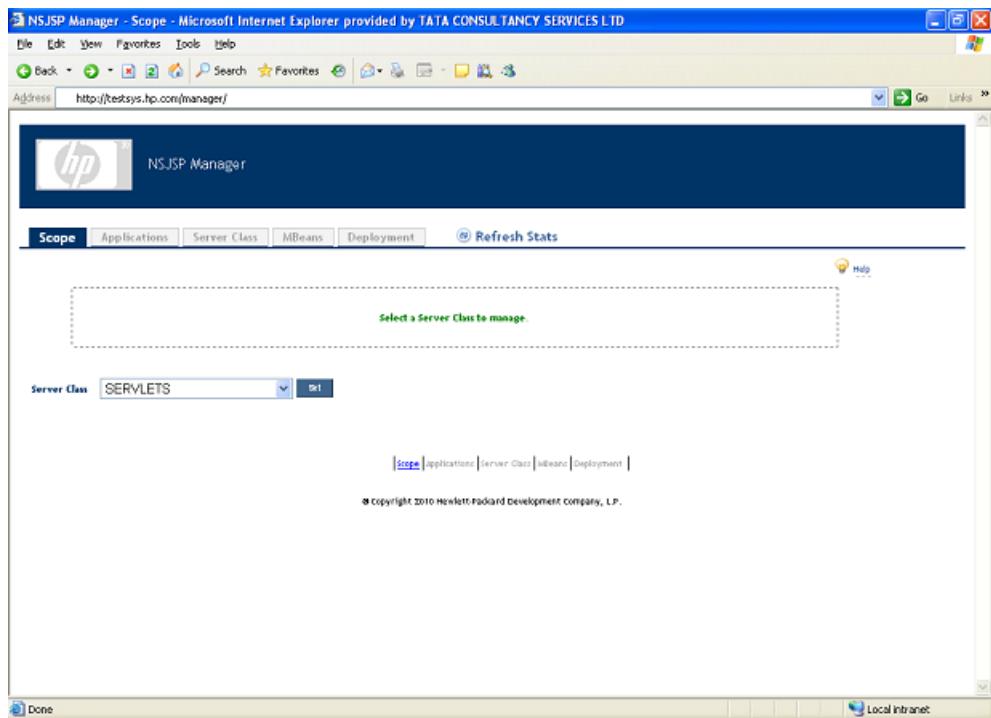


2. Enter the **User name:** and **Password:** and click **OK**.

The **NSJSP Manager** screen appears.

Figure 101 shows the **NSJSP Manager** screen.

**Figure 101 NSJSP Manager Screen**

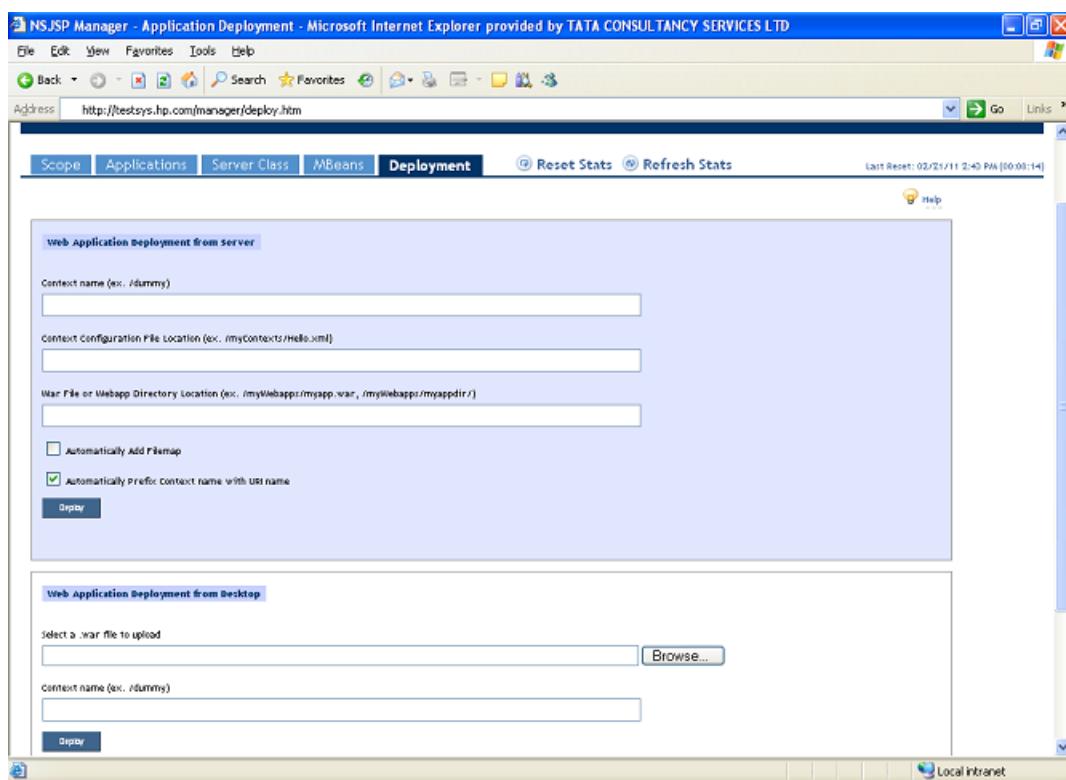


3. Select **SERVLETS** for the **Server Class** and click **Set**.

The **Host (in server.xml)** tab is enabled and populated with **localhost**. Click **Set**, which enables the **Deployment** tab on the **NSJSP Manager** screen.

Figure 102 shows the **Deployment** tab of **NSJSP Manager**.

**Figure 102 NSJSP Manager Screen - Deployment tab**



4. In the **Deployment** tab, complete the following steps in the **Web Application Deployment from Desktop** section:

1. In the **Select a .war file to upload** field, click **Browse...** and locate the axis2.war file on the Windows system.
2. **(Optional)** In the **Context name** field, enter a name for the application context.
3. Click **Deploy**.

Axis2/Java is deployed on NSJSP and is listed under **Applications** tab of the **NSJSP Manager** screen.

**NOTE:** HP recommends that you use the NSJSP manager for deployment. Deployment using the FTP or any other mechanism is not recommended. For more information on using the NSJSP manager, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

## Running Axis2/Java on NonStop

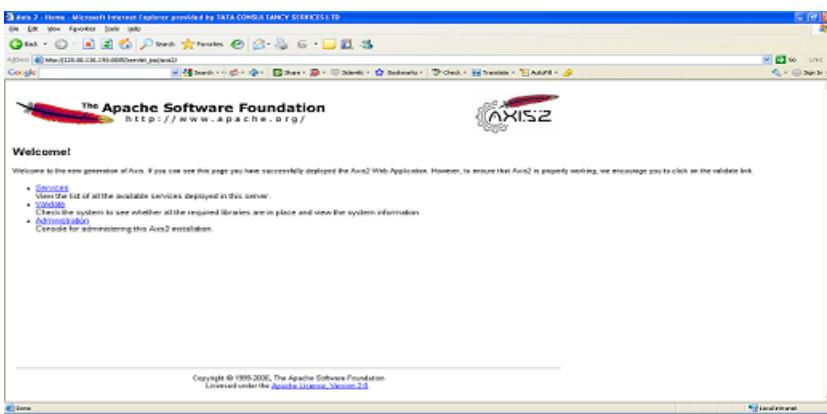
To run Axis2/Java on your NonStop system, complete the following steps:

1. Click the `/<servlet directory>/axis2` path under **Applications** in the **NSJSP Manager** screen.

The Axis2/Java Home Page appears.

Figure 103 shows the Axis2/Java Home Page.

**Figure 103 Axis2/Java Home Page**

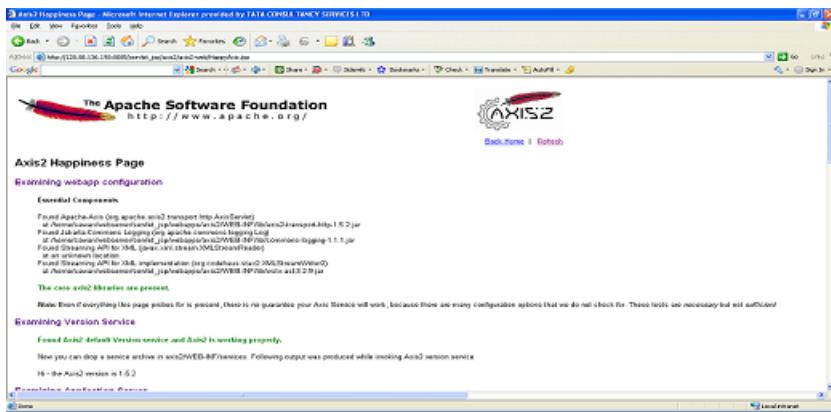


2. Click **Validate** to verify if Axis2/Java has been successfully deployed on your NonStop system. The Axis2/Java deployment is successful if:

- No error or warning is reported.
- The core axis2 libraries are present message is displayed in the **Essential Components** section.
- Found Axis2 default Version service and Axis2 is working properly message is displayed in the **Examining Version Service** section.

Figure 104 shows the Axis2/Java Validate screen.

**Figure 104 Axis2/Java Validate Screen**



## Installing the Sandesha2 Module

The Sandesha2 module is a Web Service-ReliableMessaging (WS-RM) implementation for Axis2/Java. You can use this module to make your web services reliable, or you can invoke already hosted reliable Web services. Installing the Sandesha2 module involves the following tasks:

- “[Downloading the Sandesha2 Distribution on Windows](#)” (page 297)
- “[Modifying the Configuration File on Windows](#)” (page 298)
- “[Copying the Sandesha2 Binaries to the Axis2/Java Directory](#)” (page 298)
- “[Modifying the Build File on Windows](#)” (page 298)
- “[Building the Axis2/Java Web Archive with Sandesha2](#)” (page 299)
- “[Deploying the Axis2/Java Web Archive with Sandesha2](#)” (page 299)

## Downloading the Sandesha2 Distribution on Windows

To download the Sandesha2 distribution on your Windows system, complete the following steps:

1. Access the following web address:

<http://ws.apache.org/sandesha/sandesha2/download.cgi>

The Sandesha2 web page, showing a list of distributions available for download, is displayed.

2. Download the Standard Binary Distribution in the .zip format.

**NOTE:** The following distributions are available for download:

- Standard Binary Distribution: includes the Sandesha2 binaries and sandesha2 module file.
- Source Distribution: includes the source of the Sandesha2 standard distribution.

3. Extract the Standard Binary Distribution (sandesh2-1.3-bin.zip) into a directory on your Windows system.

This directory will be referred as *<Sandesha2 Home>* and contains the following sub-directories:

\docs

includes the Sandesha2 documentation in HTML format.

\samples

includes sample applications demonstrating different Axis2/Java features.

## Modifying the Configuration File on Windows

You must modify the `<Axis2 Home>\conf\axis2.xml` configuration file by adding a user phase named 'RMPhase' to the four flows of the `axis2.xml` file.

The following code snippet shows the user phase added in the configuration file:

```
<axisconfig name="AxisJava2.0">

    <!-- REST OF THE CONFIGURATION-->

    <phaseOrder type="InFlow">
        <phase name="Transport"/>
        <phase name="Security"/>
        <phase name="PreDispatch"/>
        <phase name="Dispatch" />
        <phase name="OperationInPhase"/>
        <phase name="soapmonitorPhase"/>
        <phase name="RMPhase"/>
    </phaseOrder>

    <phaseOrder type="OutFlow">
        <phase name="RMPhase"/>
        <phase name="soapmonitorPhase"/>
        <phase name="OperationOutPhase"/>
        <phase name="PolicyDetermination"/>
        <phase name="MessageOut"/>
        <phase name="Security"/>
    </phaseOrder>

    <phaseOrder type="InFaultFlow">
        <phase name="PreDispatch"/>
        <phase name="Dispatch" />
        <phase name="OperationInFaultPhase"/>
        <phase name="soapmonitorPhase"/>
        <phase name="RMPhase"/>
    </phaseOrder>

    <phaseOrder type="OutFaultFlow">
        <phase name="RMPhase"/>
        <phase name="soapmonitorPhase"/>
        <phase name="OperationOutFaultPhase"/>
        <phase name="PolicyDetermination"/>
        <phase name="MessageOut"/>
    </phaseOrder>

</axisconfig>
```

## Copying the Sandesha2 Binaries to the Axis2/Java Directory

To copy the Sandesha2 binaries and modules to the Axis2/Java directory , complete the following steps:

1. Copy the Sandesha2 module file (`sandesha2-<Version>.mar`) from the `<Sandesha2_Home>` directory to the `<Axis2_Home>/repository/modules` directory.
2. Copy the `Sandesha2-policy-<Version>.jar` and `sandesha2-core--<Version>.jar` files from the `<Sandesha2_Home>` directory to the `<Axis2_Home>/lib` directory.

## Modifying the Build File on Windows

To modify the `<Axis2_Home>\webapp\build.xml` build file, complete the following steps:

1. Open the `<Axis2_Home>\webapp\build.xml` file in any text editor.

2. Modify the `create.war` target and comment the following line:

```
<!-- <exclude name="axis2-codegen*.jar"/> -->
```

This includes `axis2-codegen*.jar` in the WAR file.

## Building the Axis2/Java Web Archive with Sandesha2

Build the Axis2/Java web archive with Sandesha2 using the steps described in “[Building the Axis2/Java Web Archive using Standard Binary Distribution](#)” (page 293).

## Deploying the Axis2/Java Web Archive with Sandesha2

Before you deploy the web archive (`axis2.war`), remove the `axis2.war` file from the `<NSJSP Deployment Directory>/webapps` directory, if the file is already present.

Deploy the Axis2/Java web archive with Sandesha2 using the steps described in “[Deploying Axis2/Java as a Web Application on NonStop](#)” (page 294).

This completes installing the Sandesha2 module in Axis2/Java.

## Installing the Rampart Module

The Rampart module is the security module of Axis2/Java, which you can use to secure your SOAP messages based on the WS-Security stack specifications.

Installing the Rampart module involves the following tasks:

- “[Downloading the Axis2/Java Distribution on Windows](#)” (page 299)
- “[Copying the Rampart Binaries to Axis2/Java](#)” (page 300)
- “[Building the Axis2/Java Web Archive with Rampart](#)” (page 300)
- “[Deploying the Axis2/Java Web Archive with Rampart](#)” (page 300)

## Downloading the Axis2/Java Distribution on Windows

To download the Rampart distribution on your Windows system, complete the following steps:

1. Access the following web address:

<http://axis.apache.org/axis2/java/rampart/download/1.5/download.cgi>

The Rampart web page, displaying a list of distributions available for download appears.

2. Download the Standard Binary Distribution in `.zip` format.

---

**NOTE:** The following distributions are available for download:

- Standard Binary Distribution: includes the Rampart2 binaries and Sandesha2 module file.
- Source Distribution: includes the source of the Rampart2 standard distribution.

3. Extract the Standard Binary Distribution (`rampart-dist-1.5-bin.zip`) into a directory on your Windows system.

This directory will be referred as `<Rampart Home>`. Among other directories, `<Rampart Home>` contains the following sub-directories:

`\samples`

includes sample applications demonstrating Rampart features.

`\lib`

includes the Rampart binaries.

`\modules`

includes the module archive files.

`\docs`

includes the Rampart documentation in HTML format.

## Copying the Rampart Binaries to Axis2/Java

To copy the Rampart binaries to Axis2/Java, complete the following steps:

1. Copy the `rahas-1.5` and `rampart-1.5` module files from the `<Rampart Home>\modules` directory to the `<Axis2 Home>/repository/modules` directory.
2. Copy libraries from the `<Rampart Home>\lib` directory to the `<Axis2 Home>/lib` directory

## Building the Axis2/Java Web Archive with Rampart

Build the Axis2/Java web archive with Rampart using the steps described in “[Building the Axis2/Java Web Archive using Standard Binary Distribution](#)” (page 293).

## Deploying the Axis2/Java Web Archive with Rampart

Before you deploy the web archive (`axis2.war`), remove the `axis2.war` file from the `<NSJSP Deployment Directory>/webapps` directory, if the file is already present.

Deploy the Axis2/Java web archive using the steps described in “[Deploying Axis2/Java as a Web Application on NonStop](#)” (page 294).

This completes installing the Rampart module in Axis2/Java.

## Deploying and Running Axis2/Java Sample Applications on NonStop

This section discusses the steps to deploy the following sample applications distributed with the Axis2/Java binary distribution:

- “[FaultHandling](#)” (page 301)
- “[MTOM](#)” (page 303)

---

**NOTE:** The Axis2/Java distribution includes several sample applications, of which only FaultHandling and MTOM are discussed here for demonstration purposes. To run a sample application that is not discussed in this section, use the building, deploying, and running steps explained for FaultHandling and MTOM as reference.

---

This section describes the following steps for the sample applications:

- Building sample applications on a Windows system
- Deploying sample applications on a NonStop system
- Running sample applications

## FaultHandling

FaultHandling demonstrates Exception Handling using the WSDL faults and invokes BankService webservice when it is run.

The intended users of this application are bank customers.

This section describes the following steps for FaultHandling:

- “[Building FaultHandling on Windows](#)” (page 301)
- “[Deploying FaultHandling on NonStop](#)” (page 301)
- “[Running FaultHandling](#)” (page 303)

### [Building FaultHandling on Windows](#)

To build FaultHandling on your Windows system, complete the following steps:

1. Go to the `<Axis2 Home>\samples\faulthandler` directory on your Windows system, using the command:

```
command prompt> cd <Axis2 Home>\samples\faulthandler
```

For example:

```
command prompt> cd C:\axis2-1.5.2\samples\faulthandler
```

2. Generate the service. To do so, complete the following steps:

- a. Clean the output directory (if already present) using the command:

```
command prompt> ant clean
```

- b. Create the service archive file (`BankService.aar`) in the `<Axis2 Home>\samples\faulthandler\build\service` directory on your Windows system, using the command:

```
command prompt> ant generate.service
```

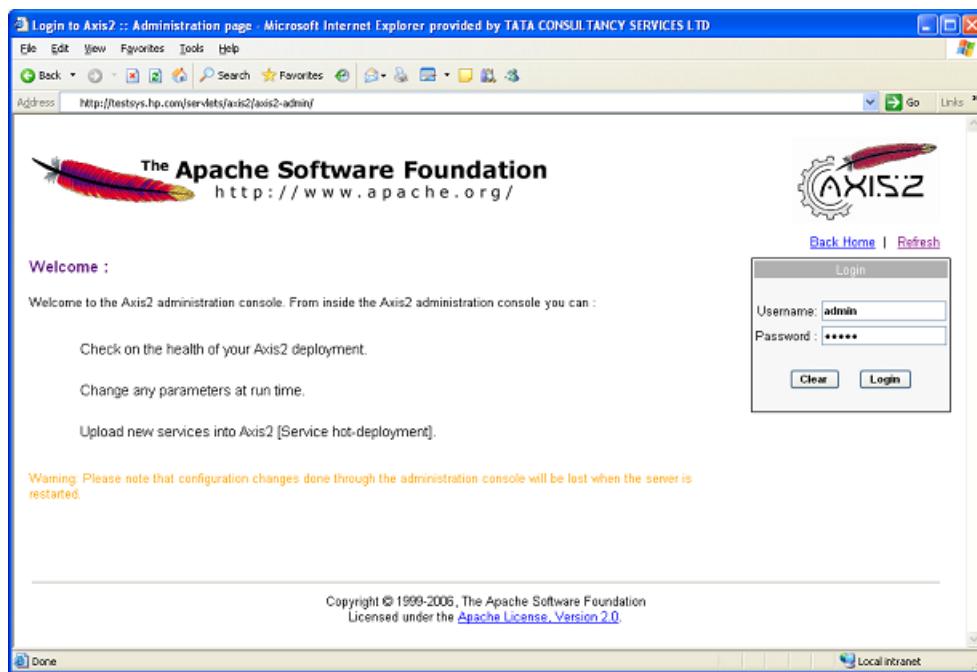
FaultHandling is now ready to be deployed under the Axis2/Java web application on the NonStop system.

### [Deploying FaultHandling on NonStop](#)

To deploy the BankService webservice under the Axis2/Java web application, complete the following steps:

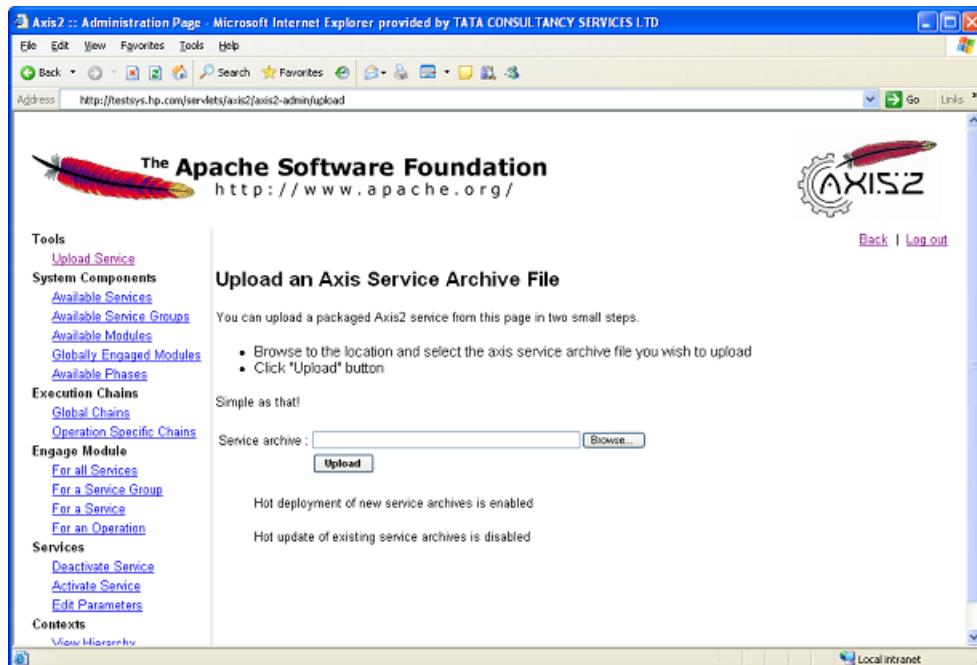
1. Go to the Axis2/Java admin login page using the following URL: [http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/axis2-admin/](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/axis2-admin/).

**Figure 105 Axis2/Java Administration Login Page**



2. Enter the username and password for the admin login to Axis2/Java. The default password for username admin is axis2.
3. In the **Tools** section, click **Upload Service**. In the **Service Archive** field, enter the complete address for the BankService.aar file or click **Browse...** to locate and select the file.

**Figure 106 Axis2/Java Administration Upload Page**



4. Click **Upload** to upload the service in Axis2/Java.

This completes deploying the BankService and it is listed in [http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/services/listServices](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/services/listServices).

## Running FaultHandling

To run FaultHandling, the BankService webservice on your NonStop system must be invoked through a client on your Windows system. To do so, perform the following steps:

1. Go to <Axis2 Home>\samples\faulthandling, using the command:

```
command prompt> cd <Axis2 Home>\samples\faulthandling
```

For example:

```
cd C:\axis2-1.5.2-bin\samples\faulthandling
```

2. You can use the following sample runs to check if BankService is successfully invoked.

- command prompt> ant run.client -Durl=http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/axis2/services/BankService -Daccount=13 -Damt=400

The output must include the following:

```
Account#13 does not exist
```

- command prompt> ant run.client -Durl=http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/axis2/services/BankService -Daccount=88 -Damt=1200

The output must include the following:

```
Account#88 has balance of 1000. It cannot support withdrawal of 1200
```

- command prompt> ant run.client -Durl=http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/axis2/services/BankService -Daccount=88 -Damt=400

The output must include the following:

```
Balance = 600
```

FaultHandling thus displays the following:

- Validity of the specified account number.
- Validity of the withdrawal balance.
- Balance after withdrawal

## MTOM

MTOM demonstrates the capabilities and power of MTOM support of Axis2/Java. When this application is run, the MTOMSample webservice is invoked. In this sample, a user can send a file to the MTOMSample service.

The intended users of this application are those who want to send binary attachments to web services.

This section describes the following steps for MTOM:

- “Building MTOM on Windows” (page 303)
- “Deploying MTOM on NonStop” (page 304)
- “Configuring the client for running MTOM” (page 305)
- “Running MTOM” (page 305)

## Building MTOM on Windows

To build MTOM on a Windows system, complete the following steps:

1. Go to <Axis2 Home>\samples\mtom using the command:

```
command prompt> cd <Axis2 Home>\samples\mtom
```

For example:

command prompt > cd C:\axis2-1.5.2-bin\samples\mtom

**2. Generate the service using the command:**

command prompt> ant generate.service

This command creates the service archive file (`sample-mtom.aar`) in `<Axis2 Home>\samples\mtom\build\service` directory on your Windows system.

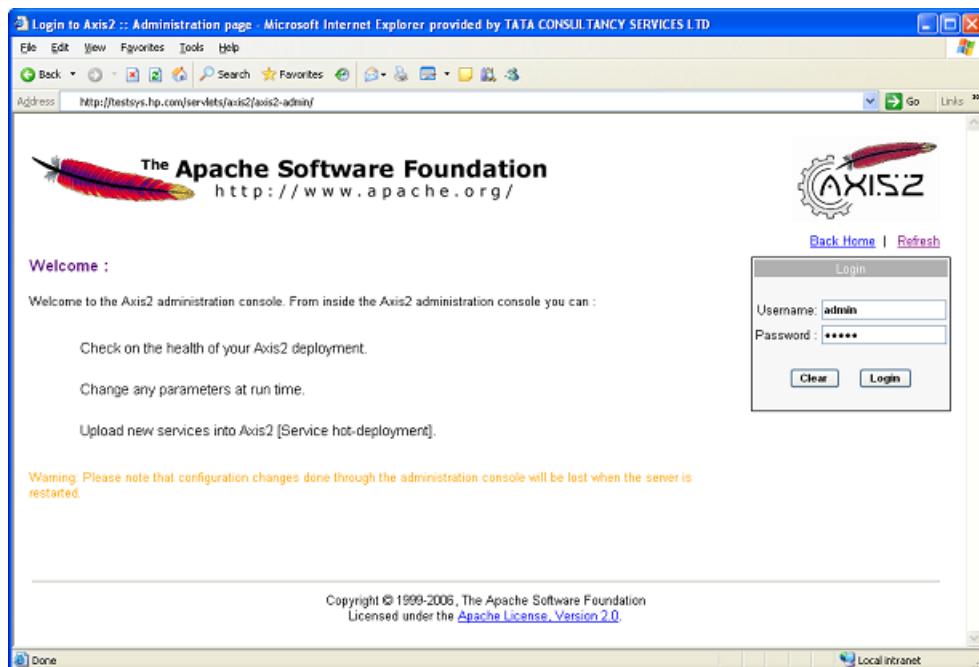
MTOM is now ready to be deployed under the Axis2/Java web application on NonStop system.

## Deploying MTOM on NonStop

To deploy the MTOMSample webservice under Axis2/Java web application, complete the following steps:

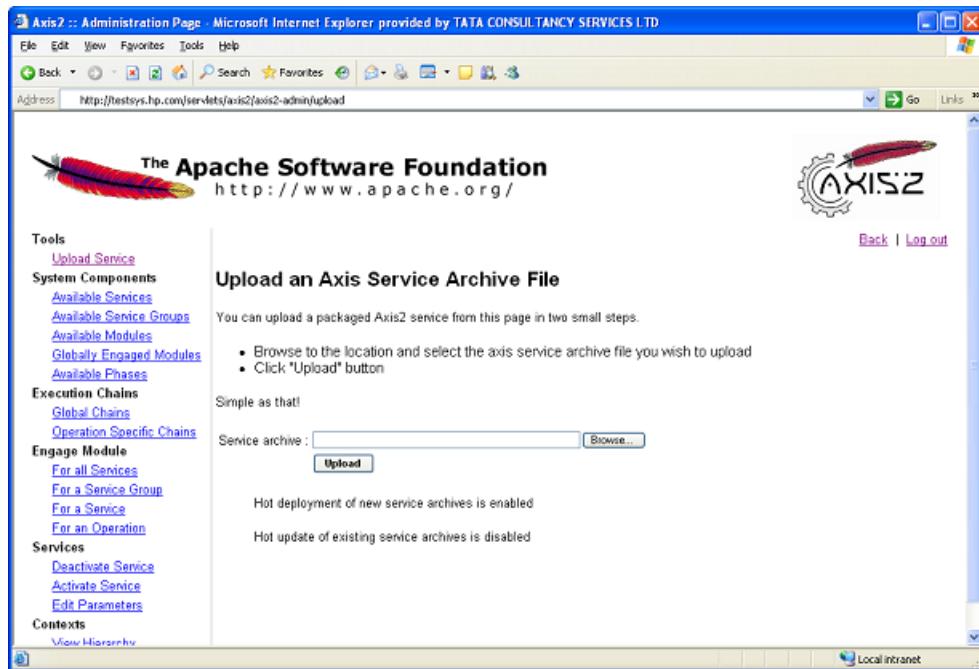
1. Go to the Axis2/Java admin login page using the following URL: [http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/axis2-admin/](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/axis2-admin/).

**Figure 107 Axis2/Java Administration Login Page**



2. Enter the username and password for the admin login to Axis2/Java. The default password for username admin is axis2.
3. In the **Tools** section, click **Upload Service**. In the **Service Archive** field, enter the complete address for the `sample-mtom.aar` file or click **Browse...** to locate and select the file.

**Figure 108 Axis2/Java Administration Upload Page**



- Click **Upload** to upload the service in Axis2/Java.

This completes deploying the MTOMSample and it is listed in [http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/services/listServices](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/services/listServices).

### Configuring the client for running MTOM

Before running MTOM, modify the MTOM Sample WSDL file, using the following steps:

- Go to the `<Axis2 Home>\samples\mtom\resources` directory on your Windows system.
- Update the address location under the `<soap>` tag and `<soap12>` tag in the `MTOMSample.wsdl` file.
  - `<soap>`  
Change the address location <http://localhost:8080/axis2/services/MTOMSample> to [http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/services/MTOMSample](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/services/MTOMSample)
  - `<soap12>`  
Change the address location <http://localhost:8080/axis2/services/MTOMSample> to [http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/services/MTOMSample](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/services/MTOMSample)

### Running MTOM

To run MTOM, the MTOMSample webservice on your NonStop system should be invoked through a client on your Windows system. To do so, complete the following steps:

- Go to `<Axis2 Home>\samples\mtom`, using the command:  
`command prompt> cd <Axis2 Home>\samples\mtom`  
For example:  
`command prompt> cd C:\axis2-1.5.2-bin\samples\mtom`
- Clean the output directory (if already present) using the command:

command prompt> ant clean

**3. Run the client, using the command:**

command prompt> ant run.client -Dfile "<Absolute Windows Path of the file to be transferred>" -Ddest Destination-file-name

For example:

command prompt> ant run.client -Dfile "C:\testfile.txt" -Ddest testfile.txt

Confirm the following:

- a.** The message File saved successfully appears on the command prompt.
- b.** The Destination-file-name file is present in <*NSJSP Deployment Directory*>.

You can thus use MTOM to transfer any file from a Windows system to a NonStop system.

# 16 Configuring Axis2/Java Applications on NonStop Systems

This chapter provides information about configuring Axis2/Java applications on NonStop systems. This chapter includes the following sections:

- “NonStop Platform Configurations” (page 307)
- “Axis2/Java Framework Configurations for NonStop Systems” (page 317)

## NonStop Platform Configurations

On a NonStop system, an application developed using the Axis2/Java framework is deployed as a web application under the NSJSP container. Thus, it inherits all the NSJSP properties and configurations, and runs like any other NSJSP web application running on a NonStop system. This section discusses the following tasks:

- “Determining the Application Parameters” (page 307)
- “Determining the Maximum Capacity of an NSJSP Instance” (page 307)
- “Configuring iTP WebServer for Axis2/Java Applications” (page 308)
- “Configuring NSJSP for Axis2/Java Applications” (page 312)

### Determining the Application Parameters

You need to determine the following parameters for your application:

- Average Response Time
- Average Load
- Maximum Load

For a unit response time, the number of active requests to the iTP WebServer is a maximum of its static capacity during the average load conditions. During peak load conditions, the number of active requests to the iTP WebServer is a maximum of its dynamic capacity.

For example,

If the average response time of your Axis2/Java application is 1 second and iTP WebServer is configured to handle 100 requests statically and 50 requests dynamically, then during average load conditions, the number of active requests to iTP WebServer can be a maximum of 100. During peak load conditions, 50 more requests can be served dynamically and hence, the active requests to the iTP WebServer can be a maximum of 150. Further requests are queued up and remain inactive until started by httpd.

The other parameters, Average Load and Maximum Load, will help you decide the configuration of the static and dynamic httpd processes of the iTP WebServer.

### Determining the Maximum Capacity of an NSJSP Instance

Before configuring NSJSP parameters, you must determine the capacity of a single instance of NSJSP. To determine the maximum load for a single instance of NSJSP, it is important to first configure the relevant TS/MP and server parameters of NSJSP (of the single instance) to their maximum limit in the following way:

1. Set the value of Numstatic to 1. This limits the number of static instances of NSJSP to 1.
2. Set the value of Maxservers to 1. This limits the number of NSJSP instances that can be started to 1. This implies that TS/MP cannot start more than one instance of NSJSP.

3. Set the value of Maxlinks to 250. This is the maximum number of links to the server process (NSJSP process). This means that the server process must be capable of processing 250 requests simultaneously.
4. Set the value of TANDEM\_RECEIVE\_DEPTH to 250 (because the values of Maxlinks and TANDEM\_RECEIVE\_DEPTH should be equal). A value of 250 means that the NSJSP process is able to read a maximum of 250 messages simultaneously from its \$RECEIVE file.
5. Configure the Executor element in the *<NSJSP Deployment Directory>/conf/server.xml* file on OSS. The Executor is used by the Connector as a thread pool. Set maxThreads = 300, minSpareThreads = 10 and maxIdleTime = 30000. Using an executor helps monitor the number of active threads at any given time. A value of 300 for maxThreads ensures that you have enough threads to process all incoming requests (a maximum of 250) simultaneously. A value of 30000 for maxIdleTime ensures that if a thread is idle for more than 30 seconds, that thread will be stopped.

After configuring the parameters, you might want to use a tool that can simulate HTTP clients and can handle HTTP cookies. The tool reveals the number of HTTP clients that one single instance of NSJSP can handle and indicates the number of simultaneous HTTP requests that NSJSP is capable of handling.

---

**NOTE:** There are a number of HTTP client simulators available, for example, Apache JMeter, HP LoadRunner, and Radview Webload. These tools provide a good interface to monitor the test results. You can use any of these tools to determine the maximum handling capacity of each NSJSP instance.

---

To arrive at the required numbers, complete the following steps:

1. Run the test tool with a single instance of the HTTP client simulator.

- 
- NOTE:** The test tool must be capable of displaying the response time of the HTTP requests.
2. Monitor the response time of the HTTP requests and allow your application to attain a steady state (in terms of response time per HTTP request).
  3. At steady state, check if the response times are within the Service Level Agreement (SLA).
  4. If the response time is within the SLA, increase the number of HTTP client instances and repeat step 2 onwards.
  5. If the response time is beyond the acceptable SLA, stop the tests and determine the number of HTTP clients and the number of HTTP requests that NSJSP instance could process simultaneously.

While the test tool can indicate the number of HTTP clients that were being processed simultaneously, the number of simultaneous HTTP requests can be arrived at using different means. Following are some of the calculation strategies:

- The number of HTTP requests recorded by the testing tool and idea of the number of requests, which required processing by NSJSP, provides the total number of HTTP requests processed by NSJSP.
- The total number of HTTP requests processed by NSJSP together with the test duration indicates the average number of simultaneous HTTP requests handled by NSJSP.

## Configuring iTP WebServer for Axis2/Java Applications

The `httpd` process of the iTP WebServer is responsible for managing the load and forwarding the requests to their corresponding application servers, the context of this section is limited to the configurations related to `httpd` processes only.

Configuring the `httpd` process involves the following tasks:

- “Configuring `httpd` Processes to Handle Maximum Load” (page 309)
- “Limiting the Maximum Number of Incoming Requests” (page 311)

## Configuring httpd Processes to Handle Maximum Load

The httpd processes are configured using the Server configuration directive in the <iTP WebServer Deployment Directory>/conf/httpd.config file on OSS. The Server directive controls the creation of the PATHMON environment, where the webserver runs.

A typical default configuration of the httpd process of the iTP WebServer appears as follows:

```
Server $root/bin/httpd {
    eval $DefaultServerAttributes
    CWD [pwd]
    Arglist -server [HTTPD_CONFIG_FILE]
    Env TANDEM_RECEIVE_DEPTH=50
    Priority 170
    Numstatic 5
    Maxservers 50
    MapDefine =TCPIP^RESOLVER^NAME /G/system/ztcpip/resconf
    MapDefine =TCPIP^PROCESS^NAME $transport
}
```

The number of httpd processes are governed by the following TS/MP attributes:

- **Numstatic:** This specifies the number of static servers running under PATHMON. The static processes run on the system, irrespective of the load.
- **Maxservers:** This specifies the maximum number of server processes that can run under PATHMON.
- **TANDEM\_RECEIVE\_DEPTH:** This specifies the capacity of each of the configured httpd processes.
- **(Maxservers – Numstatic):** The difference denotes the number of dynamic servers. A dynamic server is a need-based server. When the iTP WebServer is under heavy load and all the static httpd processes are busy, a dynamic server process, httpd, is created to support the excess load. These servers are dynamically created by TS/MP and are terminated once the process is complete.

The capacity of the iTP WebServer environment can be summarized as:

- The static capacity of iTP WebServer is [Numstatic X TANDEM\_RECEIVE\_DEPTH].
- The dynamic capacity is [(Maxservers – Numstatic) X TANDEM\_RECEIVE\_DEPTH] requests.
- The total capacity of iTP WebServer is [Maxservers X TANDEM\_RECEIVE\_DEPTH].

### Example 1:

Assume that the httpd process of iTP WebServer has the following configurations:

```
Server $root/bin/httpd {
    eval $DefaultServerAttributes
    CWD [pwd]
    Arglist -server [HTTPD_CONFIG_FILE]
    Env TANDEM_RECEIVE_DEPTH=100
    Priority 170
    Numstatic 5
    Maxservers 50
    MapDefine =TCPIP^RESOLVER^NAME /G/system/ztcpip/resconf
    MapDefine =TCPIP^PROCESS^NAME $transport
}
```

When you start the iTP WebServer, five static httpd processes will be started, governed by the value of Numstatic. Because the value of TANDEM\_RECEIVE\_DEPTH is set to 100,

each of the five static processes can handle 100 requests. In this example, the capacity of the iTP WebServer environment can be summarized as follows:

- The static capacity of iTP WebServer is [Numstatic X TANDEM\_RECEIVE\_DEPTH]= 500.
- The dynamic capacity is [(Maxservers - Numstatic) X TANDEM\_RECEIVE\_DEPTH] = 4500.
- The total capacity of iTP WebServer is [Maxservers X TANDEM\_RECEIVE\_DEPTH] = 5000.

Using this configuration, the iTP WebServer can handle 500 simultaneous requests statically. As soon as it receives the 501st request, a dynamic httpd process is created, which functions as a normal httpd process. The capacity of each of the dynamic httpd processes is the value of TANDEM\_RECEIVE\_DEPTH. When the load decreases, the dynamic httpd processes goes down.

For more information on the Server configuration directive of iTP WebServer, see the *iTP Secure WebServer System Administrator's Guide*.

### Guidelines for Configuring the Server Directive

Before configuring the various attributes of the Server directive, you must determine the following parameters:

- Processors for httpd Processes
- Application parameters: Average Load, Peak Load, and Average Response Time

### Processors for httpd Processes

When you start the iTP WebServer, the static httpd processes equal to the value of Numstatics are started on your NonStop system. By default, all the configured processors are taken into account. If the value of Numstatic is set to n, and the number of configured processors on the system are equal to n, one static httpd process is started on each of the processors. However, if the value of Numstatic is less than the number of configured processors, one static httpd process starts on some of the processors, while the remaining processors will not have any static httpd process.

The following example explains how you can determine the processors on which you intend to start httpd processes and configure the processors directive.

#### Example:

Assume that you have a 16-processor system and you want to start httpd processes on processors 0, 1, 2, 3, 4, 5. For this, the processors configuration attribute must be configured as processors 0 1 2 3 4 5.

In this scenario:

- If you set the value of Numstatic equal to 6, each of the processors 0, 1, 2, 3, 4, 5 will have one static httpd process.
- If you set the value of Numstatic equal to a multiple of 6, each of the processors 0, 1, 2, 3, 4, 5 will have (Numstatic/6) static httpd processes.
- If you set the value of Numstatic equal to 5, any 5 of the processors 0, 1, 2, 3, 4, 5 will have one static httpd process, and remaining processors will not have any httpd process.
- If you set the value of Numstatic equal to 9, any three of the processors 0, 1, 2, 3, 4, 5 will have two static httpd process and each of the remaining three processors will have one httpd process.

After determining the values, you can use any the following approaches to make necessary configurations for the TS/MP parameters of the `httpd` processes:

- Using the default value of `TANDEM_RECEIVE_DEPTH`
- Modifying the value of `TANDEM_RECEIVE_DEPTH`

#### [Using the default value of `TANDEM\_RECEIVE\_DEPTH`](#)

With `TANDEM_RECEIVE_DEPTH` set to the default value of 50, the static processes can be configured on the basis of Average Load and dynamic processes can be configured on the basis of Peak Load on your system. You may use the following approaches while deciding the values of `Numstatic` and `Maxservers`:

- The value of `Numstatic` must be set to  $(\text{Average Load})/50$
- The value of `Maxservers` must be set to  $(\text{Peak Load})/50$

For example:

If the Average Load and Peak Load on your Axis2/Java application turn out to values 1000 and 1500 requests and you limit `TANDEM_RECEIVE_DEPTH` to its default value of 50, the `Numstatic` must be set to 20 and `Maxservers` must be set to 30.

---

**NOTE:** It is advisable to set the value of `Numstatic` to, at least, the number of configured processors.

#### [Modifying the value of `TANDEM\_RECEIVE\_DEPTH`](#)

If the number of `httpd` processes configured on the basis of default value of `TANDEM_RECEIVE_DEPTH` does not fulfill your application requirement, you can change the value of `TANDEM_RECEIVE_DEPTH` and calculate the values of `Numstatic` and `Maxservers` accordingly. However, you must consider the following before increasing the value of `TANDEM_RECEIVE_DEPTH`:

- It is recommended to keep the web server running with 80 percent of its capacity. To achieve this, set the value of `TANDEM_RECEIVE_DEPTH` accordingly.  
For example, if your calculation suggests that `TANDEM_RECEIVE_DEPTH` of 80 is required, you must set it to a value of 100.
- The maximum value of `TANDEM_RECEIVE_DEPTH` is 255.
- Do not reduce `TANDEM_RECEIVE_DEPTH` to a value less than 50.

### [Limiting the Maximum Number of Incoming Requests](#)

Because of constraints on available system resources, you might want to limit the number of requests to your iTP WebServer. If you want to configure your iTP WebServer to handle only a certain number of requests at an instance, use the `MaxConnections` configuration directive to specify the maximum number of connections that can be served by iTP WebServer at any given instance. The iTP WebServer serves the number of requests equal to the multiple of `Numstatic` greater than or equal to `MaxConnections` count.

#### **Syntax:**

```
MaxConnections -count <integer value> -replytype <customized/RST>
```

For example:

```
MaxConnections -count 101 -replytype RST
```

Consider the scenario of Example 1 above, where `Numstatic` is 5 with the following `MaxConnections` configuration:

```
MaxConnections -count 101 -replytype customized
```

In this case, the iTP WebServer serves 105 requests (higher multiple of Numstatic nearest to the count value). Here, the 106th request displays the following error message:

Maximum connections reached: The server reached its maximum configured capacity.  
with HTTP response code:

200 OK

To customize the error message, create a new message ID error-maximum-connection. This customized message is displayed if the Message configuration directive is used in the *<iTP WebServer Deployment Directory>/conf/httpd.config* file with the new message ID.

For more information on the MaxConnections configuration directive and creating a customized error message using the Message configuration directive, see the *iTP Secure WebServer System Administrator's Guide*.

---

**NOTE:** To use the MaxConnections configuration directive, your iTP WebServer must be configured for static environment only, that is, the values of Numstatic and MaxServers of the httpd process must be equal.

## Configuring NSJSP for Axis2/Java Applications

When an Axis2/Java application runs on a NonStop system, it runs as an instance of NSJSP. Therefore, before configuring the NSJSP environment, it is important to determine the load each instance of NSJSP is expected to handle. This section describes the following configuration aspects:

- ["Configuring SessionBasedLoadBalancing" \(page 312\)](#)
- ["Configuring Connector Threads" \(page 313\)](#)
- ["Configuring TS/MP Specific Parameters" \(page 314\)](#)
- ["Configuring Java Runtime Arguments" \(page 316\)](#)

### Configuring SessionBasedLoadBalancing

The NSJSP Container maintains sessions in the form of serialized Java objects. Each session object is identified by a unique identifier called the session-ID. The session-ID is sent to the HTTP client either as a cookie or in the form of URL-rewriting. The name of the cookie is JSESSIONID and when the user application creates a session, NSJSP generates the cookie (JSESSIONID) as the name and session-ID as the cookie value. The session objects can be either kept in the process memory or persisted in a persistent store (for example, a database table). When it is kept in the process, it is available only for the process that created the session object. On the other hand, if it is kept in a persistent store, it is available for any process under the NSJSP environment. When the SessionBasedLoadBalancing feature is enabled, all requests related to a particular session are routed to the process that has the session object in its memory.

To enable the SessionBasedLoadBalancing feature, you must modify the configurations of the servlet.ssc object by editing the *<iTP WebServer Deployment Directory>/conf/servlet.config* file on OSS. The servlet.ssc object is configured under the Server directive. The SessionBasedLoadBalancing feature is governed by the -DSaveSessionOnCreation and -DSessionBasedLoadBalancing arguments in the Arglist of the Server directive.

- -DSaveSessionOnCreation

Enables or disables saving the sessions in a persistent store during their creation time.

Syntax:

-DSaveSessionOnCreation= [ true | false ]

**NOTE:** The default value of `-DSaveSessionOnCreation` is false.

- -DSessionBasedLoadBalancing

Enables or disables session-based load balancing.

### Syntax:

-DSessionBasedLoadBalancing= [ true | false ]

**NOTE:** The default value of `-DSessionBasedLoadBalancing` is set to `true`.

For example, if you want to enable the SessionBasedLoadBalancing feature for your Axis2/Java application and save the application sessions in a persistent store, set the Arglist as:

Server \$server objectcode {

2

•

```
Arglist -DSessionBasedLoadBalancing=true \
-DSaveSessionOnCreation=true \
```

1

re

where,

`server_objectcode` is mapped to the `servlet.ssc` object in the `<iTP WebServer Deployment Directory>/bin` directory on OSS.

While setting these arguments, consider the following:

1. If both the `SaveSessionOnCreation` and `SessionBasedLoadBalancing` options are set to `false`, and if a Persistent Manager is configured with a persistent store, 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.
  2. Enabling or disabling the `SessionBasedLoadBalancing` feature depends on the application requirement. Your Axis2/Java application might encounter any of the following scenarios:
    - The application depends heavily on the state stored in session objects. Therefore, session objects cannot be lost and the application cannot recover from loss of state.
    - Application response is of prime importance and the application can recover from a loss of state.
    - The application is expected to handle largely varying loads and having a large number of static NSJSP instances is not an option.
    - The session objects must be valid for large durations (such as an entire day).
    - The session objects must be available whenever the application restarts.

For more information on the `SessionBasedLoadBalancing` and `SessionBasedLoadBalancing` configuration directives, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

## Configuring Connector Threads

The Connector is responsible for creating and maintaining the threads that are used for message processing. A thread is allocated for each incoming message; therefore, the number of messages

that can be processed simultaneously can be controlled by limiting the number of threads the NSJSP Connector can spawn.

---

**NOTE:** A single connector can service multiple instances of the container. The connector must be configured such that it can spawn threads sufficient for all the container instances. In terms of configuration, the Host element in the server.xml configuration file represents a container that is serviced by the connector.

---

The NSJSP Connector thread pool can be connected using the following ways:

- [Configuring the Connector Element](#)
- [Configuring the Executor Element](#)

### Configuring the Connector Element

Configure the maxThreads attribute of the Connector element in the <NSJSP Deployment Directory>/conf/server.xml file on OSS. The Connector element is the child of the Service element.

For example: The following snippet shows the configuration in the <NSJSP Deployment Directory>/conf/server.xml file on OSS, if you want to configure 75 simultaneous connector threads.

```
...
<Service name="NSJSP">
<Connector protocol="HTTP/1.1"
           connectionTimeout="0"
           acceptCount="25"
           maxThreads="75"/>
...
</Service>
```

### Configuring the Executor Element

You can configure the Executor element as child elements of the Service element in the <NSJSP Deployment Directory>/conf/server.xml file on OSS. The following snippet of the server.xml configuration file shows the configuration of the Executor element and how a Connector can be configured to use an Executor.

```
...
<Service name="NSJSP">
<Executor name="Axis2/JavaExec"
className="org.apache.catalina.core.StandardThreadExecutor" namePrefix="Axis2/JavaAPP" maxThreads="75"
minSpareThreads="20" maxIdleTime="10000"/>
<Connector executor="Axis2/JavaExec"/>
...
...
```

**NOTE:** All executors that are configured in NSJSP must implement the java interface org.apache.catalina.Executor. NSJSP provides a standard implementation of this interface in the class org.apache.catalina.core.StandardThreadExecutor.

---

## Configuring TS/MP Specific Parameters

This section describes the following TS/MP specific configuration parameters:

1. Numstatic – Determines the number of static NSJSP processes.
2. Maxservers – Determines the total number of NSJSP processes.
3. TANDEM\_RECEIVE\_DEPTH – Determines the handling capacity of each instance of NSJSP.

4. Maxlinks – Determines the maximum number of TS/MP links. The number of simultaneous messages that can be delivered to an instance of NSJSP is determined by Maxlinks. NSJSP reads these many number of messages from the \$RECEIVE file simultaneously.
5. Number of Connector threads.

The configurations for Numstatic, Maxservers, TANDEM\_RECEIVE\_DEPTH, and Maxlinks can be set under the Server directive in the *<iTP WebServer Deployment Directory>/conf/servlet.config* file on OSS. Before configuring NSJSP, it is important that you determine the maximum load that each instance of NSJSP is expected to handle. The way these configuration directives drive the NSJSP environment depends on the state of the SessionBasedLoadBalancing feature.

- NSJSP configured with SessionBasedLoadBalancing turned OFF
- NSJSP is configured with SessionBasedLoadBalancing turned ON

#### [NSJSP configured with SessionBasedLoadBalancing turned OFF](#)

- The value of Numstatic must be  $[(\text{Average Load}) / (\text{Max load one instance of NSJSP can handle})]$ .
- Maxservers must be  $[(\text{Peak Load}) / (\text{Max load one instance of NSJSP can handle})]$ .
- The value of Maxlinks must be set to the peak load that one instance of NSJSP is expected to handle.
- TANDEM\_RECEIVE\_DEPTH must be twice the value of Maxlinks. Normally, to handle  $n$  number of messages, you need  $n$  number of threads. In extreme cases, all the  $n$  requests may timeout. While the same number of threads processing the timeout requests could still be busy, you need the same  $n$  number of threads to process the new incoming requests. Therefore, the number of threads that must be configured is  $(n + n = 2n)$ .
- The maximum number of connector threads, maxThreads must be equal to the value of TANDEM\_RECEIVE\_DEPTH.

For example:

The parameters Maxlinks, TANDEM\_RECEIVE\_DEPTH, and the Connector threads are closely linked with each other. These parameters are explained using examples. The following examples assume that the httpd processes are configured to serve 100 simultaneous HTTP, and all 100 HTTP requests are serviced by your Axis2/Java application deployed under NSJSP (that is, there is no static content in the application). Also, the peak load that one single instance of NSJSP can handle is 25.

Because SessionBasedLoadBalancing is turned OFF, all the messages between HTTPD and NSJSP flow through TS/MP. The number of simultaneous messages that can be delivered to an instance of NSJSP is determined by the value of Maxlinks; set the value of Maxlinks to 25. NSJSP can now read 25 messages from its \$RECEIVE queue simultaneously. In an extreme case, all the 25 messages time out; there must be enough space in the \$RECEIVE queue to accommodate 25 more messages. Therefore, \$RECEIVE must be opened with Receive Depth (controlled by TANDEM\_RECEIVE\_DEPTH) of (25+25). Thus, you must set the value of TANDEM\_RECEIVE\_DEPTH to 50. The reason for setting TANDEM\_RECEIVE\_DEPTH to a value of 50 can be explained as:

To handle 25 messages, you need 25 threads. In an extreme case where all the 25 requests time out, you need another 25 threads to process the new incoming 25 requests because the threads processing the timeout requests could still be busy. Therefore, the number of threads that need to be configured is  $(25+25 = 50)$ .

## NSJSP is configured with SessionBasedLoadBalancing turned ON

- The value of Numstatic must be [(Peak Load)/(Max load one instance of NSJSP can handle)].
- The value of Maxservers must be [(Peak Load)/(Max load one instance of NSJSP can handle)]. This ensures that no dynamic process is created, so that the session object is not lost when there are numerous file system calls.
- The value of Maxlinks must be set to the peak load that one instance of NSJSP is expected to handle.
- Arriving at a definite value for TANDEM\_RECEIVE\_DEPTH is difficult when NSJSP is configured with SessionBasedLoadBalancing turned ON. Deciding on an optimum value for TANDEM\_RECEIVE\_DEPTH requires a good understanding of the application. The following example provides an insight into making this decision.
- The maximum number of connector threads, maxThreads must be equal to the value of TANDEM\_RECEIVE\_DEPTH.

Example:

In this example, it is evident how a single instance of NSJSP, although having Maxlinks set to 25, can serve all the 100 requests.

Because SessionBasedLoadBalancing is turned ON, all messages between HTTPD and NSJSP flow through TS/MP and file system calls. The number of simultaneous messages that can be delivered to an instance of NSJSP is determined by the value of Maxlinks; set the value of Maxlinks to 25.

With Maxlinks set to 25, a single instance of NSJSP can handle 25 concurrent requests through TS/MP, all being the first requests of web dialogs. The first call on a web dialog is always delivered to NSJSP through TS/MP and there will not be any file system call. After the 25 requests are serviced, subsequent requests on those 25 web dialogs are delivered to NSJSP through file system I/O operations.

At this stage, all 25 links to the NSJSP instance are free to process more incoming requests delivered through TS/MP. Therefore, the NSJSP instance must now be able to handle 25 more concurrent requests, all being the first requests of web dialogs. After the 25 requests are processed, the subsequent requests on these new connections arrive at NSJSP through file system I/O. At this stage, NSJSP could be handling up to 50 concurrent requests and all these requests are being delivered through file system calls from httpd and not through TS/MP. Hence, the 25 links are free to process new web dialogs. Therefore, a single instance of NSJSP can serve all the 100 requests.

At this point, one instance of NSJSP could be processing requests from all the possible 100 connections to the httpd processes. This means that there could be a scenario where all the 100 requests time out. Therefore, there must be enough space in the \$RECEIVE file to handle (100+100) messages. This means that TANDEM\_RECEIVE\_DEPTH must have a value of 200. The reason for setting TANDEM\_RECEIVE\_DEPTH to a value of 200 can be explained as:

To handle 100 requests, you need 100 threads. In an extreme case of all the 100 requests timing out, you need another 100 threads to process the new incoming 100 requests because the threads processing the timeout requests could still be busy. Therefore, the number of threads that need to be configured is (100+100 = 200).

## Configuring Java Runtime Arguments

The configurations for Java runtime arguments can be done in the *<iTP WebServer Deployment Directory>/conf/servlet.config* file on OSS. The sscaux object is configured under the Server directive. The Java runtime arguments are populated in the Arglist. Some of the

important Java runtime arguments that you must consider during the deployment of your Axis2/Java applications are:

- [-Xmx](#)
- [-Xss](#)
- [-Xnklassgc](#)

There are other Java runtime arguments supported by NSJSP. For more information, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

#### [-Xmx](#)

Sets the maximum size of the memory allocation pool, which is the garbage collected heap.

Syntax:

`-Xmx maximum-heap-size [ k | m ]`

where,

`maximum-heap-size`

is the maximum size of the memory allocated for the garbage collected. It must be greater than or equal to 1000 bytes.

`k`

sets the value of `maximum-heap-size` to be read in kilobytes.

`m`

sets the value of `maximum-heap-size` to be read in megabytes.

#### [-Xss](#)

Sets the maximum stack size that can be used by a Java thread.

Syntax:

`-Xmx maximum-stack-size`

where,

`maximum-stack-size`

is the maximum size of the stack trace in kilobytes.

#### [-Xnklassgc](#)

Is an optional argument to stop the Java class garbage collection.

Syntax:

`-Xnklassgc`

By default, the Java runtime reclaims space for unused Java classes. Including this optional argument might prevent any memory-leak problems.

## Axis2/Java Framework Configurations for NonStop Systems

This section discusses the NonStop specific Axis2/Java configurations for the following:

- “[Global Configuration](#)” (page 317)
- “[Service Configuration](#)” (page 318)
- “[Module Configuration](#)” (page 318)

## Global Configuration

Global configurations are used to configure the whole system, wherein the `axis2.xml` file provided with the Axis2/Java distribution is configured.

For more information about the various options available for global configuration, see: [http://ws.apache.org/axis2/1\\_2/axis2config.html#Global\\_Configuration](http://ws.apache.org/axis2/1_2/axis2config.html#Global_Configuration)

## Service Configuration

Service configurations are used to configure various services on Axis2/Java. The description of these services is specified in the `services.xml` file.

Each service archive file (`.aar`) must have a `services.xml` file to be a valid service and it must be available in the `META-INF` directory of the archive file.

For more information about the various options for configuring services in Axis2/Java, see: [http://ws.apache.org/axis2/1\\_2/axis2config.html#Service\\_Configuration](http://ws.apache.org/axis2/1_2/axis2config.html#Service_Configuration)

## Module Configuration

Module configurations are used to configure a module in the `module.xml` file. Each module archive file (`.aar`) must have a `module.xml` file to be a valid module, and it must be available in the `META-INF` directory of the archive file.

For more information about configuring modules, see: [http://ws.apache.org/axis2/1\\_2/axis2config.html#Module\\_Configuration](http://ws.apache.org/axis2/1_2/axis2config.html#Module_Configuration)

# 17 Getting Started with Axis2/Java

This chapter explains how to develop a web service using the Axis2/Java framework. It describes the steps required to build a basic TemperatureConverter system on your Windows system and deploy it on your NonStop system.

## Prerequisites

Before getting started, make sure that you have the requisite software installed on your NonStop and Windows system.

## NonStop System

The following software must be installed on your NonStop system:

- NonStop iTP WebServer version T8996H02 or later
- NSJSP version T1222H60 or later
- NSJ version T2766H60 or later

## Windows System

The following software must be installed on your Windows system:

- JDK version 1.5 or later
- Eclipse Galileo IDE version 3.3.1.1 or a later

**NOTE:** For more information about installing the software required on NonStop and Windows system, see “[Prerequisites](#)” (page 18).

## Overview of TemperatureConverter

TemperatureConverter is a web service developed using the Axis2/Java framework. This application enables you to convert temperature from:

- Celsius to Fahrenheit
- Fahrenheit to Celsius

You can use one of the following approaches to build a web service:

- Code-First approach - by writing a Java object and then developing and deploying it as a web service.
- Contract-First approach - by creating a Web Service Description Language (WSDL) file and then using Java to implement it.

**NOTE:** The subsequent sections provide the steps to develop and deploy a sample application, TemperatureConverter, using both Code-First and Contract-First approaches.

## Code-First approach

This section describes the steps to develop, set up, deploy, and run the TemperatureConverter web service and its client using the Code-First approach. The following tasks are described:

1. “[Developing TemperatureConverter Web Service on Windows](#)” (page 320)
2. “[Deploying the TemperatureConverter Web Service on NonStop](#)” (page 321)
3. “[Running TemperatureConverter Web Service on NonStop](#)” (page 323)
4. “[Developing TemperatureConverter Client on Windows using the Eclipse Galileo IDE](#)” (page 324)
5. “[Running TemperatureConverter Client on Windows](#)” (page 335)

## Developing TemperatureConverter Web Service on Windows

Developing the TemperatureConverter web service on a Windows system involves the following activities:

1. "Creating the Web Service Implementation Class" (page 320)
2. "Creating the Deployment Descriptor" (page 320)
3. "Creating an Axis2/Java AAR File" (page 321)

### Creating the Web Service Implementation Class

To create the Web Service implementation class, complete the following steps:

1. Create the TemperatureConverter sub-directory in *<My SASH Home>\axis2\gettingstarted* directory on your Windows system.
2. Create a Java class TemperatureConverter.java in the TemperatureConverter directory using any text editor.
3. Modify the TemperatureConverter.java class to implement c2fConversion and f2cConversion (to convert Celsius to Fahrenheit and Fahrenheit to Celsius respectively) methods.

After incorporating these two methods, the TemperatureConverter.java class appears as follows:

```
/**
 * Temperature Converter Implementation Class
 */
public class TemperatureConverter {
    /**
     * util method to convert celsius to fahrenheit
     * @param cValue : double value of celsius
     * @return calculated value of fahrenheit
     */
    public double c2fConversion(double cValue) {
        return ((cValue * 9.0)/5.0 )+ 32.0;
    }

    /**
     * util method to convert fahrenheit to celsius
     * @param fValue : double value of fahrenheit
     * @return calculated value of celsius
     */
    public double f2cConversion(double fValue) {
        return ((fValue - 32.0) * 5.0) / 9.0;
    }
}
```

Compile the TemperatureConverter.java class using the following steps:

- a. Go to the TemperatureConverter directory using the command:

```
command prompt> cd <My SASH Home>\axis2\gettingstarted\TemperatureConverter
```

- b. Compile TemperatureConverter.java using the command:

```
command prompt> javac TemperatureConverter.java
```

The TemperatureConverter.class file is created in the TemperatureConverter directory.

### Creating the Deployment Descriptor

The services.xml file is the deployment descriptor for any web service deployed under Axis2/Java. The service deployment descriptor is used by the deployment module to configure and deploy the service. Therefore, it is essential to reference TemperatureConverter as a web service in the services.xml file to deploy it in Axis2/Java.

Axis2/Java has a set of built-in MessageReceivers. Some of them can handle only the XML-In and XML-Out scenario, and these are called RawXML MessageReceivers. There are other message receivers called RPC MessageReceivers that can handle JavaBeans, simple Java types, and XML. The RPC MessageReceivers message receiver is used for the sample TemperatureConverter web service.

To create the deployment descriptor, complete the following steps:

1. Create a new directory named META-INF in the <My SASH Home>\axis2\gettingstarted\TemperatureConverter directory on your Windows system.
2. Create the services.xml in META-INF directory (using any text editor) on your Windows system. If you do not want to create your own deployment descriptor file, you can use the following lines of code to create your services.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceGroup>
    <service name="TemperatureConverter">
        <messageReceivers>
            <messageReceiver mep="http://www.w3.org/ns/wsdl/in-out"
                class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
        </messageReceivers>
        <parameter name="ServiceClass">TemperatureConverter</parameter>
        <parameter name="useOriginalwsdl">false</parameter>
        <parameter name="modifyUserWSDLPortAddress">true</parameter>
        <operation name="f2cConversion"
            mep="http://www.w3.org/ns/wsdl/in-out"
            namespace="http://ws.apache.org/axis2">
            <actionMapping>urn:f2cConversion</actionMapping>
        <outputActionMapping>
            http://TemperatureConverterPortType/f2cConversionResponse
        </outputActionMapping>
        </operation>
        <operation name="c2fConversion"
            mep="http://www.w3.org/ns/wsdl/in-out"
            namespace="http://ws.apache.org/axis2">
            <actionMapping>urn:c2fConversion</actionMapping>
            <outputActionMapping>
                http://TemperatureConverterPortType/c2fConversionResponse
            </outputActionMapping>
        </operation>
    </service>
</serviceGroup>
```

### Creating an Axis2/Java AAR File

An Axis2/Java AAR file contains the compiled code of the service implementation class, and the services.xml file. You can use the JAR utility to create the AAR file for TemperatureConverter.

To create an Axis2/Java AAR file, complete the following steps:

1. Go to the <My SASH Home>\axis2\gettingstarted\TemperatureConverter directory using the command:  
command prompt> cd <My SASH Home>\axis2\gettingstarted\TemperatureConverter
2. Create the AAR archive using the command:  
command prompt> jar -cvf TemperatureConverter.aar \*.class META-INF

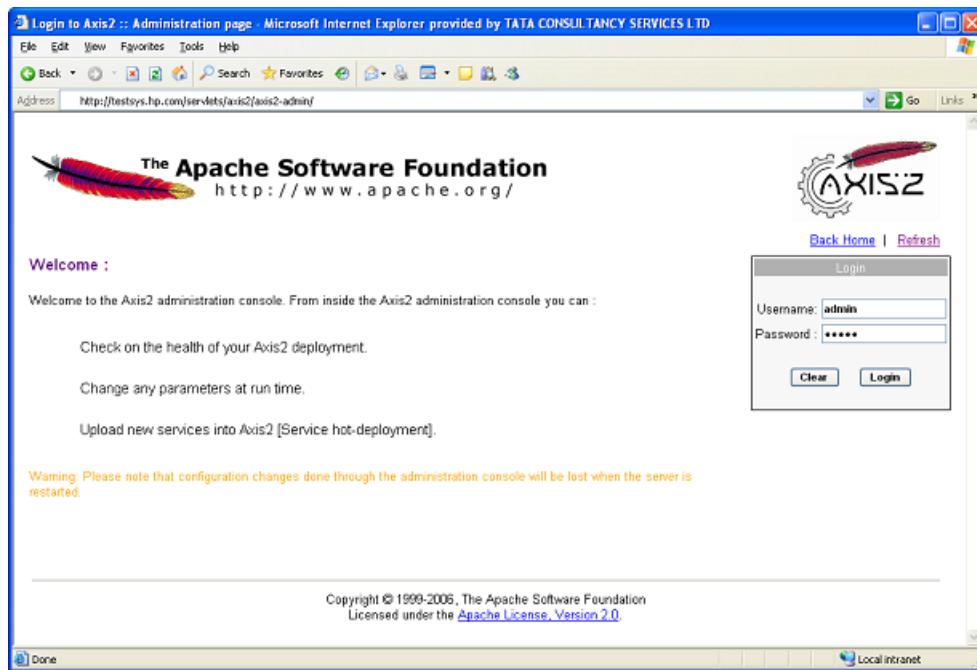
The TemperatureConverter.aar file is created in the TemperatureConverter directory.

### Deploying the TemperatureConverter Web Service on NonStop

To deploy the TemperatureConverter.aar file on your NonStop system, complete the following steps:

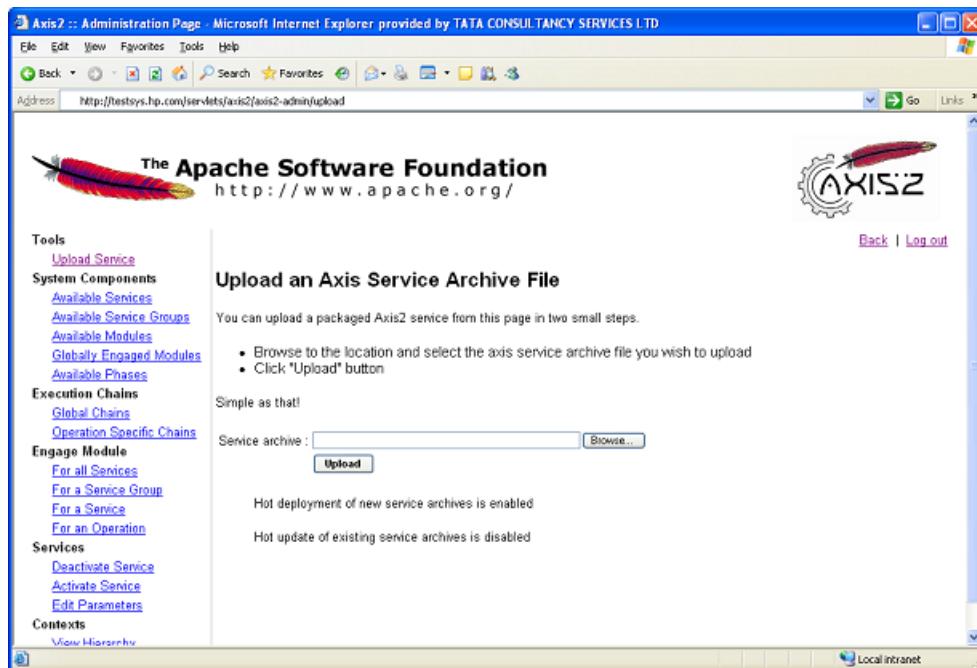
1. Go to the Axis2/Java admin login page using the following URL: [http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/axis2-admin/](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/axis2-admin/).

**Figure 109 Axis2/Java Administration Login Page**



2. Enter the username and password for the admin login to Axis2/Java. The default password for username admin is axis2.
3. In the **Tools** section, click **Upload Service**. In the **Service Archive** field, enter the complete address for the TemperatureConverter.aar file or click **Browse...** to locate and select the file.

**Figure 110 Axis2/Java Administration Upload Page**



4. Click **Upload** to upload the service in Axis2/Java.

This completes deploying TemperatureConverter and it is listed in [http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/services/listServices](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/services/listServices).

## Running TemperatureConverter Web Service on NonStop

To run the TemperatureConverter web service on your NonStop system, complete the following steps:

1. If iTP WebServer is already running, the TemperatureConverter web service must be listed in [http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/services/listServices](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/services/listServices)
2. If iTP WebServer is not running, start the iTP WebServer using the following steps:
  - a. Go to *<iTP WebServer Deployment Directory>/conf* using the command:  
OSS> cd *<iTP WebServer Deployment Directory>/conf*  
For example:  
OSS> cd /home/sash\_usr/webserver/conf
  - b. Start your iTP WebServer, using the command:  
OSS> ./start
  - c. Access the TemperatureConverter web service using the following URL:  
[http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/services/listServices](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/services/listServices)

The TemperatureConverter List Services Screen displays details, such as Version, Service Description, Service Status, and Available Operations.

Figure 111 shows the TemperatureConverter List Services Screen.

**Figure 111 TemperatureConverter List Services Screen**

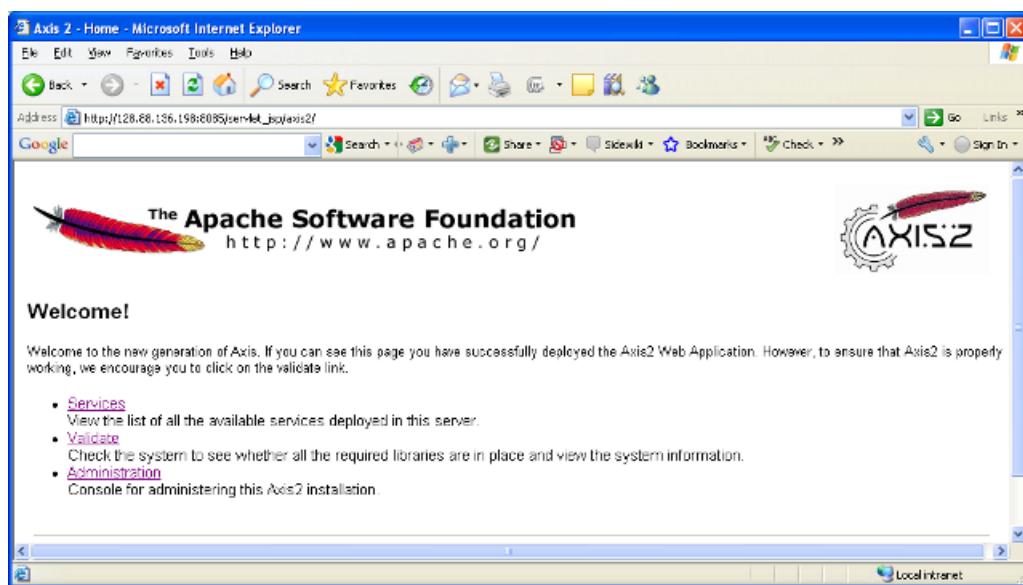


Figure 112 shows the WSDL file for the TemperatureConverter Web Service WSDL.

**Figure 112 TemperatureConverter Web Service WSDL**

The screenshot shows a Microsoft Internet Explorer window displaying the WSDL (Web Services Description Language) document for the TemperatureConverter web service. The URL in the address bar is `http://128.88.136.198:8085/servlet_jsp/axis2/services/TemperatureConverter?wsdl`. The WSDL code is displayed in the main content area, showing XML definitions for the service, its types, and operations like `f2cConversion` and `f2cConversionResponse`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <wsdl:types>
        <xsd:schema attributeFormDefault="qualified" elementFormDefault="qualified">
            targetNamespace="http://ws.apache.org/axis2">
                <xsd:element name="f2cConversion">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element minOccurs="0" name="args0" type="xs:double" />
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="f2cConversionResponse">
                    <xsd:complexType>
                        <xsd:sequence>
```

The WSDL file is required to create the TemperatureConverter web service client that invokes the TemperatureConverter as a web service.

## Developing TemperatureConverter Client on Windows using the Eclipse Galileo IDE

To invoke the TemperatureConverter service running on your NonStop system, you need to create an Axis2/Java client on your Windows system. The APIs of the client application will be used to invoke the web service.

Use the Eclipse Galileo IDE to create an Axis2/Java client for the TemperatureConverter web service. To develop the Axis2/Java client application *AxisClient* for the TemperatureConverter web service, complete the following steps:

1. “Creating the Client Project in Eclipse” (page 324)
2. “Creating the Java Package for Client Project” (page 328)
3. “Creating the Class Files under the Java Package” (page 329)
4. “Modifying the Class File of the Client” (page 330)
5. “Adding Dependency JAR Files” (page 332)

### Creating the Client Project in Eclipse

To create the client project in Eclipse, complete the following steps:

1. Double-click the `eclipse.exe` file in <Eclipse IDE Installation directory> to open the Eclipse IDE.

The **Workspace Launcher** dialog box appears. By default, the workspace is set to your existing workspace, if already created.

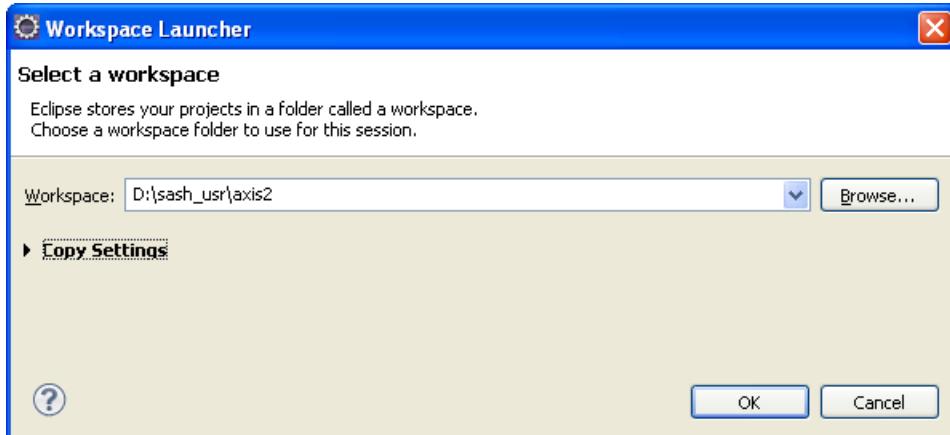
---

**NOTE:** D:\sash\_usr\axis2 is the sample workspace that is used to develop the `AxisClient` client application for the TemperatureConverter web service.

---

Figure 113 shows the **Workspace Launcher** dialog box.

**Figure 113 Workspace Launcher Dialog Box**



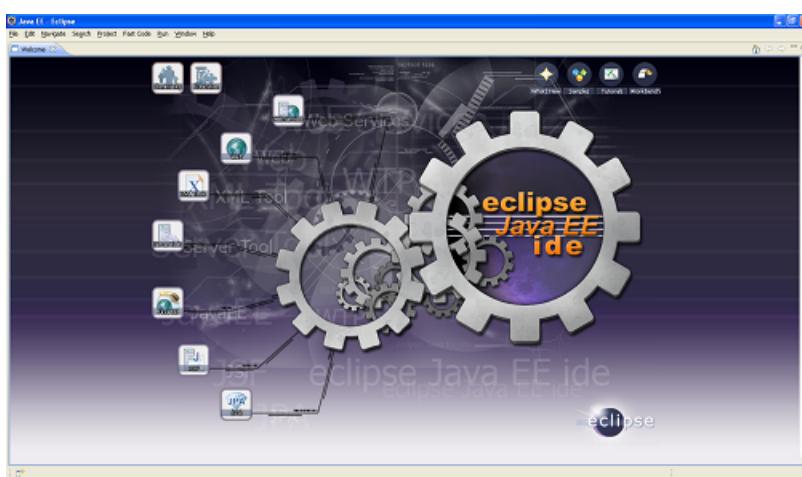
2. Click **OK**.

To create a new workspace, click **Browse** and select the folder you want to use as your workspace and then click **OK**.

The **Eclipse SDK Welcome** screen appears.

Figure 114 shows the **Eclipse SDK Welcome** screen.

**Figure 114 Eclipse SDK Welcome Screen**

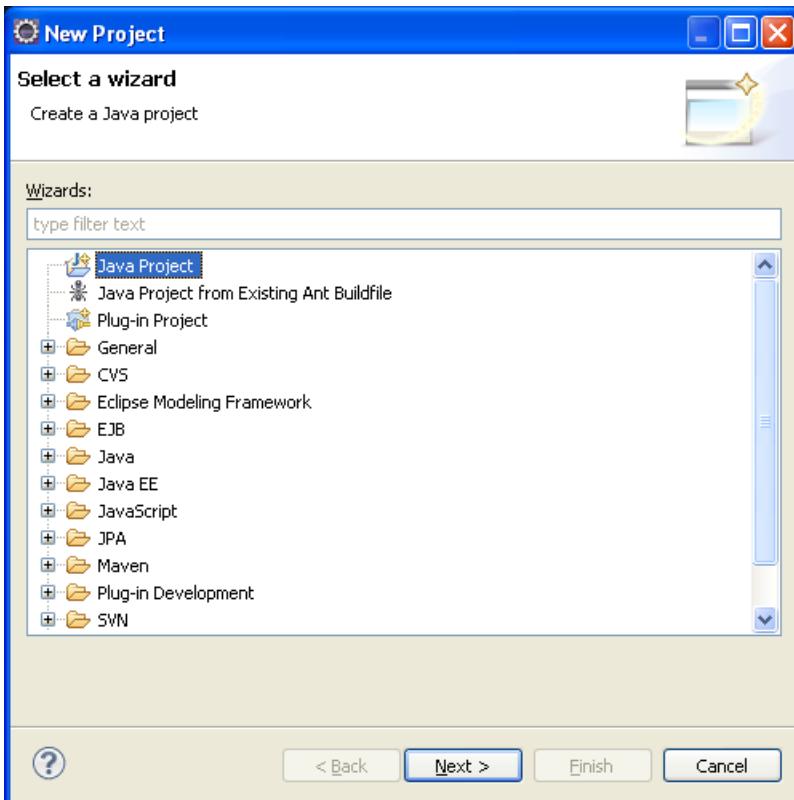


Close the **Welcome** screen. The Workspace is created.

3. Click **File**→**New**→**Java Project** to open the **New Project** dialog box.
4. From the list of folders, select **Java**→**Java Project** and click **Next**.

Figure 115 shows the **New Project** dialog box.

**Figure 115 New Project Dialog Box**



The **New Java Project** dialog box appears.

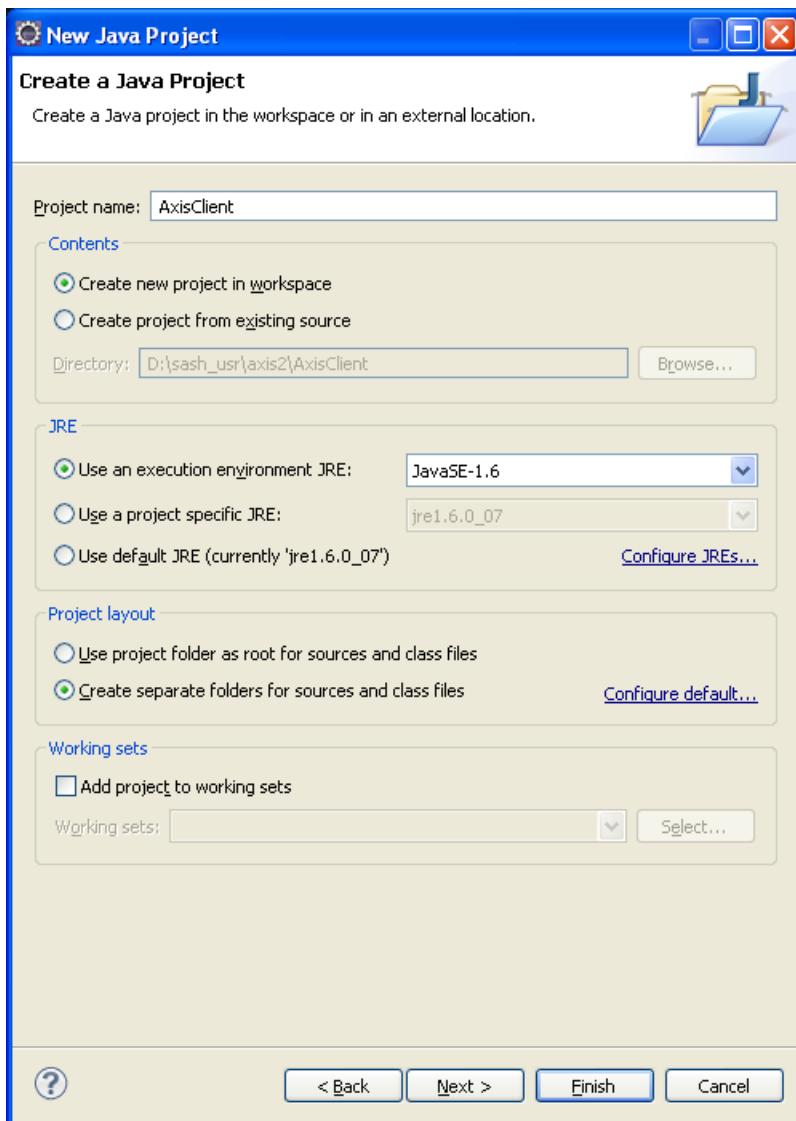
5. In the **Project name** field, enter the name of the project you want to create. Ensure that the directory for this project is created inside the workspace that you have selected or created.

**NOTE:**

- In this sample, *AxisClient* is the name of the Java project and all the references for the project are made to *TemperatureConverterClient*.
- The options in the JRE and Project layout sections are selected by default.

Figure 116 shows the **New Java Project** dialog box.

**Figure 116 New Java Project Dialog Box**



6. Confirm that the JRE System Library is set to JRE version 1.5 or later.

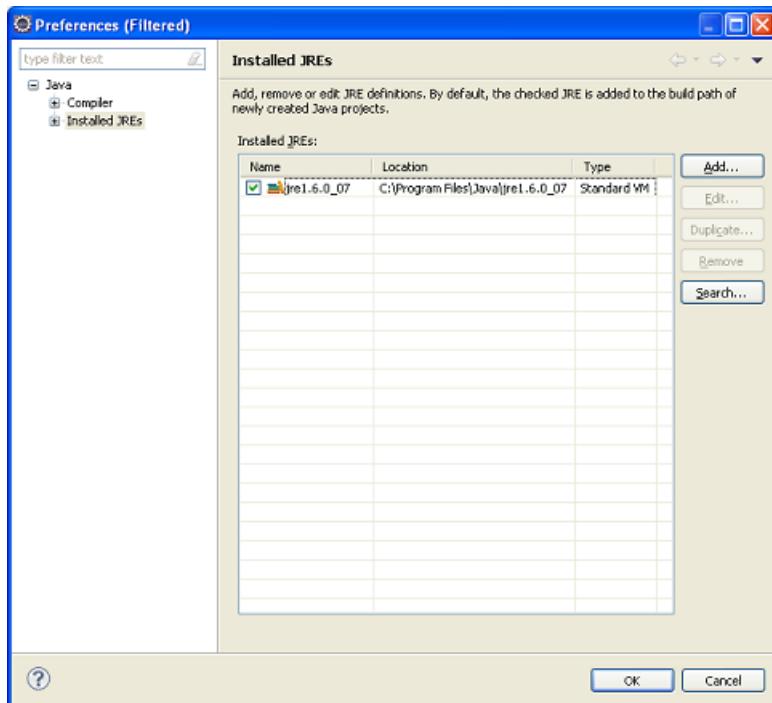
**NOTE:** The JRE version used in this example is 1.6.

If JRE version is not 1.5 or later, complete the following steps to select JRE 1.5 or later:

- a. Select **Windows**→**Preferences**.
- b. From the left pane of the **Preferences** page, select **Java**→**Installed JREs** option.  
The list of JREs that are exposed to the Eclipse IDE appears.
- c. Select JRE 1.5 or later and click **OK**.

Figure 117 shows the JRE Options - Preferences Screen.

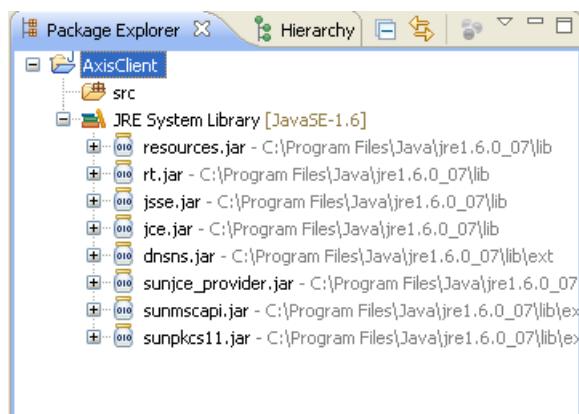
**Figure 117 JRE Options - Preferences Screen**



7. Click **Finish** (in Figure 116) to create the Java project.

Figure 118 shows the structure of the newly created Java Project AxisClient.

**Figure 118 Java Project AxisClient**



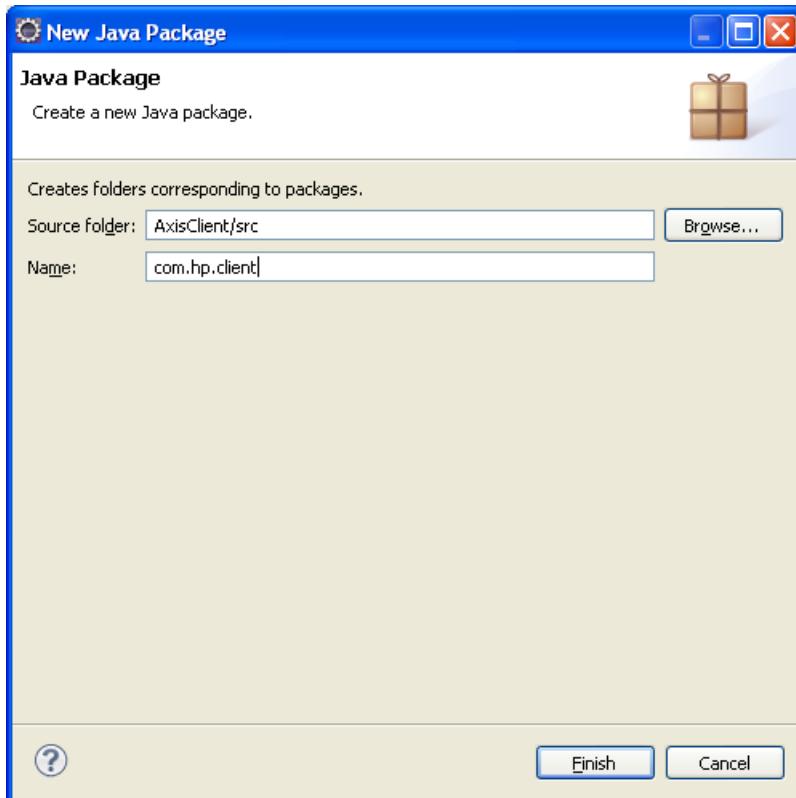
### Creating the Java Package for Client Project

To create the Java Package for client project, complete the following steps:

1. On the **Project Explorer** frame, right-click *AxisClient* and select **New→Package**.  
The **New Java Package** dialog box appears.
2. In the **Name** field, enter **com.hp.client** and ensure that the source folder is set to the *AxisClient/src* directory. Click **Finish**.

Figure 119 shows the **New Java Package** dialog box.

**Figure 119 New Java Package Dialog Box**



The Java Package for Client Project is created.

#### Creating the Class Files under the Java Package

To create the class files under the Java package, complete the following steps:

1. On the **Project Explorer** frame, right-click the **com.hp.client package** and select **New→Class**. The **New Java Class** dialog box appears.
2. In the **Name** field, enter **TemperatureConverterClient**.
3. Select **public static void main(String[ ] args)**.

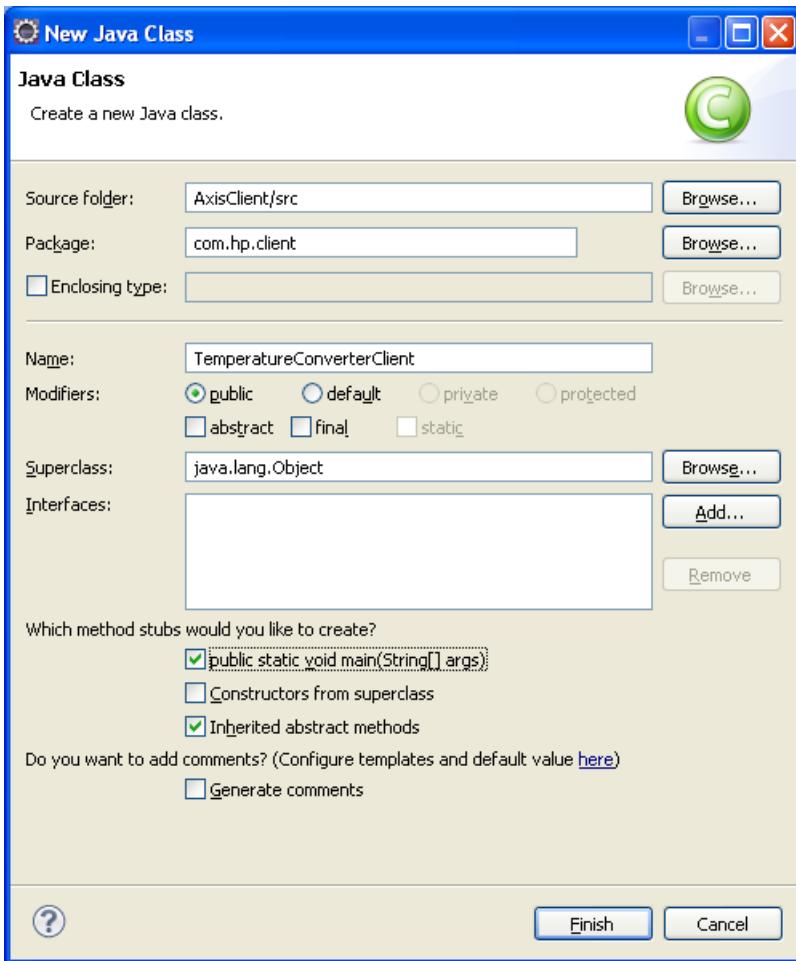
---

**NOTE:** The other options in the screen are selected by default.

---

Figure 120 shows the **New Java Class** dialog box.

**Figure 120 New Java Package Dialog Box**



4. Click **Finish**.

The class files under the Java package are created.

### Modifying the Class File of the Client

To modify the newly created TemperatureConverterClient.java class file, complete the following steps:

1. Create an instance of the ServiceClient under the main() method.

```
ServiceClient client = new ServiceClient();
```

2. Create the following two OMElement methods for payload (request SOAP body or the message) under the main() method:

- createPayLoad1(): For converting Celsius to Fahrenheit
- createPayLoad2(): For converting Fahrenheit to Celsius

---

**NOTE:** OMElement is the Axis2/Java representation of XML.

The following code creates the above-mentioned OMElement methods.

```
public static OMElement createPayLoad1() {  
    OMFactory fac = OMAbstractFactory.getOMFactory();  
    OMNamespace omNs = fac.createOMNamespace("http://ws.apache.org/axis2",  
        "axis2");  
    OMEElement method = fac.createOMELEMENT("c2fConversion", omNs);  
    OMEElement element = fac.createOMELEMENT("value", omNs);  
    element.setText("4");  
    method.addChild(element);
```

```

        return method;
    }

    public static OMElement createPayLoad2() {
        OMFactory fac = OMAbstractFactory.getOMFactory();
        OMNamespace omNs = fac.createOMNamespace(
            "http://ws.apache.org/axis2", "axis2");

        OMEElement method = fac.createOMEElement("f2cConversion", omNs);
        OMEElement element = fac.createOMEElement("value", omNs);
        element.setText("4");
        method.addChild(element);

        return method;
    }

```

3. Create a metadata object called **Options** under the `main()` method and set it to **ServiceClient**.

**NOTE:** The metadata object **Options** contains the target End Point Reference (EPR), SOAP action, and transport data properties to configure the client for the service invocation.

```

Options opts = new Options();
opts.setTo(new EndpointReference("http://<IP Address of iTP WebServer><Port#>
/<servlet directory>/axis2/services/TemperatureConverter"));

```

**NOTE:** Remember to replace *<IP address of iTP WebServer>* with the IP Address and *<Port#>* of the iTP WebServer with the port under which the TemperatureConverter web service is running.

4. Set the SOAP action, provided in the WSDL file, of the TemperatureConverter web service, by adding the following code.

```

// Setting action ,and which can be found from the wsdl of the service
opts.setAction("urn:c2fConversion");
// setting created option into service client
client.setOptions(opts);
OMELEMENT res1 = client.sendReceive(createPayLoad1());
opts.setAction("urn:f2cConversion");
OMELEMENT res2 = client.sendReceive(createPayLoad2());

```

5. Add the following import statements for the objects used in the TemperatureConverterClient.java class:

```

import org.apache.axis2.AxisFault;
import org.apache.axis2.client.ServiceClient;
import org.apache.axiom.om.OMAbstractFactory;

import org.apache.axiom.om.OMELEMENT;
import org.apache.axiom.om.OMFactory;
import org.apache.axiom.om.OMNamespace;

import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.client.Options;

```

After incorporating these changes, your TemperatureConverterClient.java file must appear as:

```

package com.hp.client;

import org.apache.axis2.AxisFault;
import org.apache.axis2.client.ServiceClient;
import org.apache.axiom.om.OMAbstractFactory;

import org.apache.axiom.om.OMELEMENT;
import org.apache.axiom.om.OMFactory;
import org.apache.axiom.om.OMNamespace;

import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.client.Options;

```

```

public class TemperatureConverterClient {

    public static void main(String[] args) throws AxisFault {

        ServiceClient client = new ServiceClient();
        // create option object
        Options opts = new Options();
        // setting target EPR
        opts
            .setTo(new EndpointReference(
                "http://<IP address of iTP WebServer>:<Port#>
                /<servlet directory>/axis2/services/TemperatureConverter"));

        // Setting action ,and which can be found from the wsdl of the service
        opts.setAction("urn:c2fConversion");
        opts.setProperty(Constants.Configuration.ENABLE_REST, Constants.VALUE_TRUE);
        // setting created option into service client
        client.setOptions(opts);

        OMEElement res1 = client.sendReceive(createPayLoad1());
        opts.setAction("urn:f2cConversion");
        OMEElement res2 = client.sendReceive(createPayLoad2());

        System.out.println(res1);
        System.out.println(res2);

    }

    public static OMEElement createPayLoad1() {
        OMFactory fac = OMAbstractFactory.getOMFactory();
        OMNamespace omNs = fac.
            createOMNamespace("http://ws.apache.org/axis2", "axis2");
        OMEElement method = fac.createOMEElement("c2fConversion", omNs);
        OMEElement element = fac.createOMEElement("value", omNs);
        element.setText("4");

        method.addChild(element);
        return method;
    }

    public static OMEElement createPayLoad2() {
        OMFactory fac = OMAbstractFactory.getOMFactory();
        OMNamespace omNs = fac.
            createOMNamespace("http://ws.apache.org/axis2", "axis2");

        OMEElement method = fac.createOMEElement("f2cConversion", omNs);
        OMEElement element = fac.createOMEElement("value", omNs);
        element.setText("4");
        method.addChild(element);

        return method;
    }
}

```

---

**NOTE:** Starting Axis2/Java version 1.1.1, the REST feature is enabled by default. You can configure the REST option using the disableREST flag in the `<Axis2 Home>\config\axis2.xml`. Setting this flag disables REST handling in all the endpoints.

When sending a message, whether the message is RESTful or not, is decided by the value set for the ENABLE\_REST property in the client.

---

### Adding Dependency JAR Files

Add the following Axis2/Java dependency JAR files in the Java Build Path of the *AxisClient* project to compile and link the project.

**Table 9 Axis2/Java Dependency JAR Files**

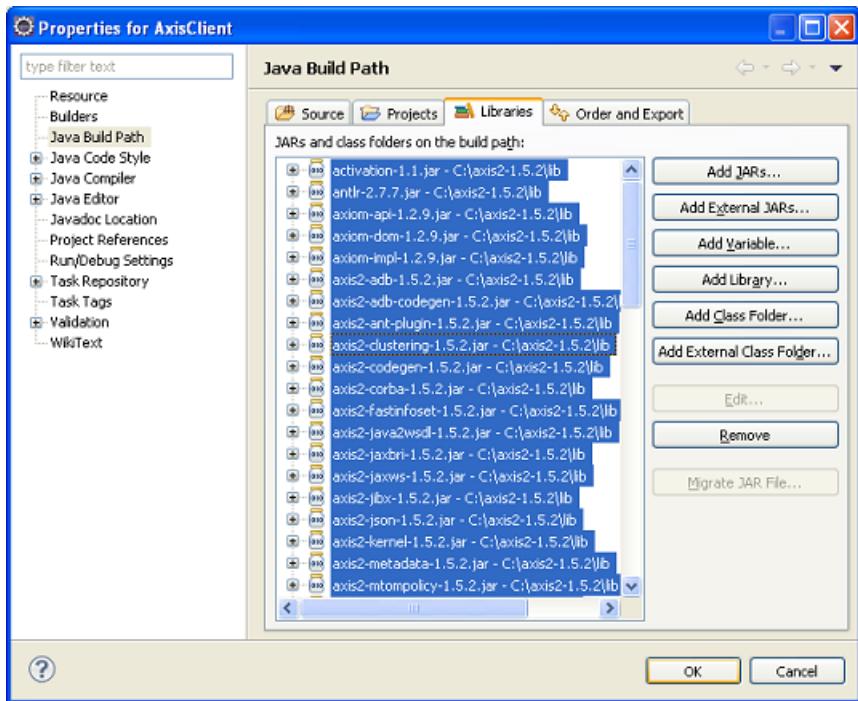
Dependency JAR Files	Source Location
activation-1.1.jar	<Axis2/Java Home>/lib
antlr-2.7.7.jar	<Axis2/Java Home>/lib
axiom-api-1.2.9.jar	<Axis2/Java Home>/lib
axiom-impl-1.2.9.jar	<Axis2/Java Home>/lib
axis2-adb-codegen-1.5.2.jar	<Axis2/Java Home>/lib
axis2-kernel-1.5.2.jar	<Axis2/Java Home>/lib
axis2-transport-http-1.5.2.jar	<Axis2/Java Home>/lib
axis2-transport-local-1.5.2.jar	<Axis2/Java Home>/lib
bcel-5.1.jar	<Axis2/Java Home>/lib
commons-codec-1.3.jar	<Axis2/Java Home>/lib
commons-fileupload-1.2.jar	<Axis2/Java Home>/lib
commons-httpclient-3.1.jar	<Axis2/Java Home>/lib
commons-io-1.4.jar	<Axis2/Java Home>/lib
commons-logging-1.1.1.jar	<Axis2/Java Home>/lib
httpcore-4.0.jar	<Axis2/Java Home>/lib
mail-1.4.jar	<Axis2/Java Home>/lib
neethi-2.0.4.jar	<Axis2/Java Home>/lib
wsdl4j-1.6.2.jar	<Axis2/Java Home>/lib
XmlSchema-1.4.3.jar	<Axis2/Java Home>/lib

To add the dependency JAR files in the projects library path, complete the following activities on the Eclipse IDE:

1. On the **Project Explorer** frame, right-click *AxisClient* and select **Properties**.  
The **Properties** for the *AxisClient* dialog box appears.
2. From the left pane of the **Properties** page, select **Java Build Path**.
3. Click the **Libraries** tab.
4. Click **Add External JARs** to add each of the dependency JAR files listed in **Table 9**. Browse to specify the location of each of the dependency JAR files.
5. Click **OK**.

Figure 121 shows the **Properties** dialog box for *AxisClient*.

**Figure 121 Properties for AxisClient Dialog Box**

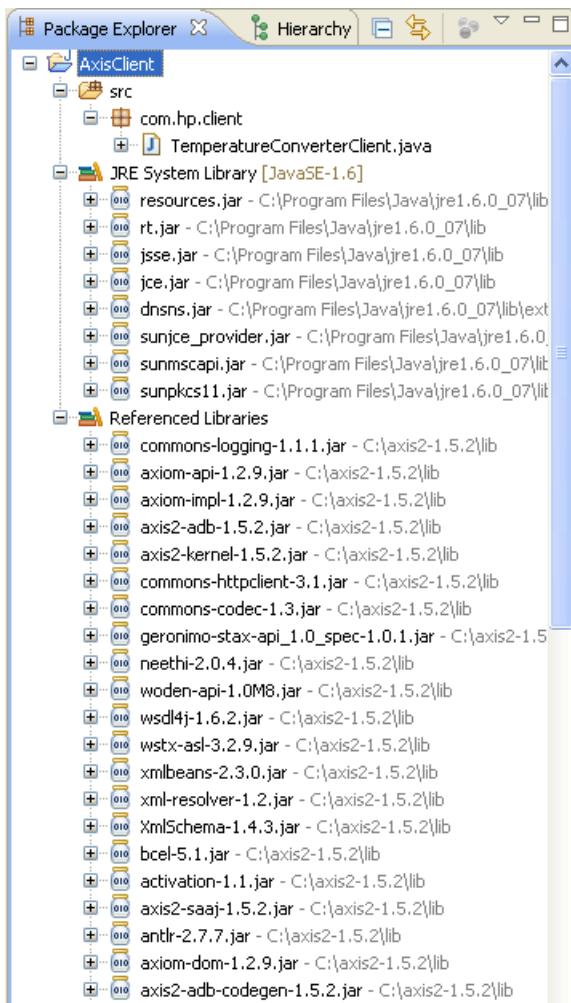


This completes the development of the *AxisClient* project on the Windows system.

The *AxisClient* project structure appears as shown.

Figure 122 shows the *AxisClient* project structure in the project explorer window.

**Figure 122 AxisClient Project Structure Dialog Box**



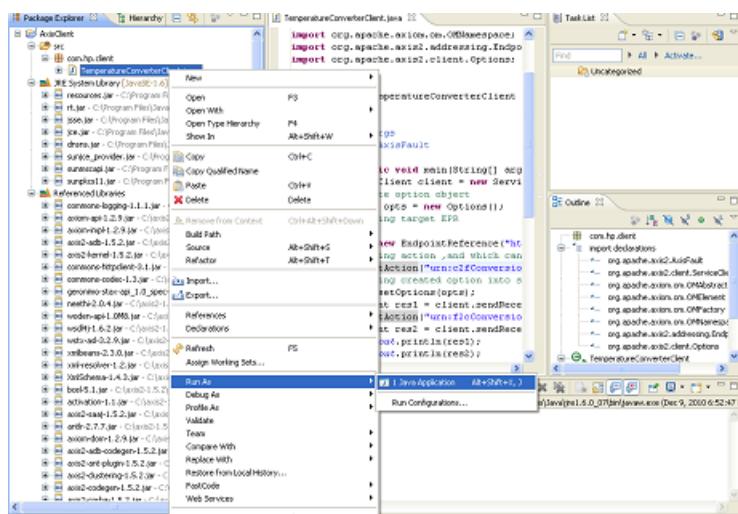
## Running TemperatureConverter Client on Windows

To run the TemperatureConverter Client on a Windows system:

On the **Project Explorer** frame, right-click the TemperatureConverterClient.java file in the **AxisClient** Java Project and select **Run As→Java Application**.

Figure 123 shows the Java Eclipse SDK - Options.

**Figure 123 Java Eclipse SDK - Options**

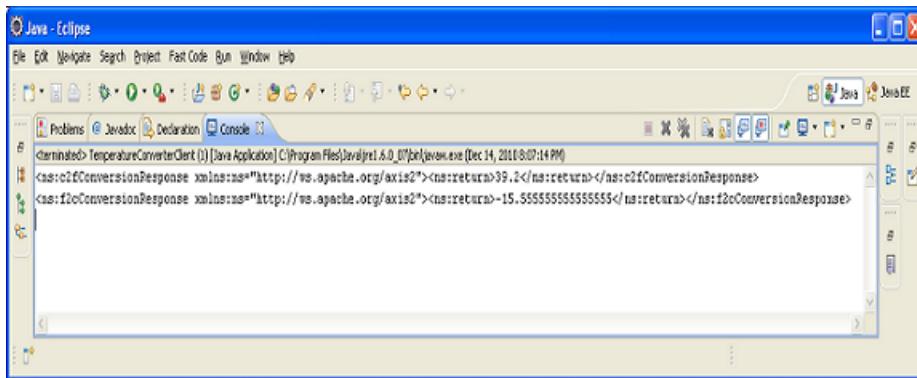


The following output is displayed on the console.

```
<ns:c2fConversionResponse  
    xmlns:ns="http://ws.apache.org/axis2">  
        <ns:return>39.2</ns:return>  
</ns:c2fConversionResponse>  
<ns:f2cConversionResponse  
    xmlns:ns="http://ws.apache.org/axis2">  
        <ns:return>-15.55555555555555</ns:return>  
</ns:f2cConversionResponse>
```

Figure 124 shows the output in the console.

**Figure 124 Console**



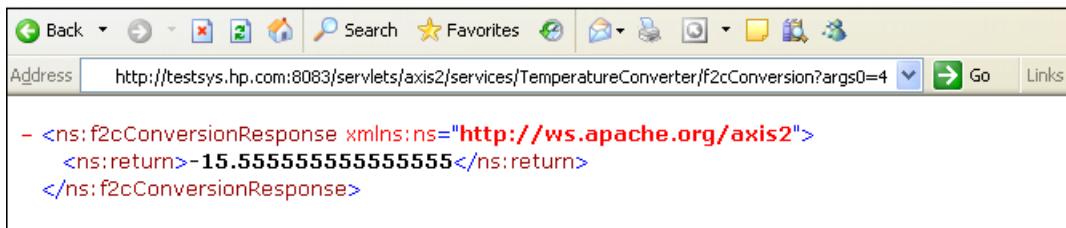
### Accessing a REST Web Service using HTTP GET

You can access the Axis2/Java based web services using HTTP GET. The generic REST based URL format to access a web service is:

```
http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/axis2/services/<Service name>/<Method name>?<service parameters>=<parameter value>
```

Figure 125 shows the address format to access the TemperatureConverter web service using the REST method.

**Figure 125 Java Eclipse SDK - Options**



## Contract-First approach

This section describes the steps to develop, set up, deploy, and run the TemperatureConverter web service and its client, using Contract-First approach. The following tasks are described:

1. "Developing the TemperatureConverter Web Service on Windows" (page 337)
2. "Deploying the TemperatureConverter Web Service on NonStop" (page 339)
3. "Running the TemperatureConverter Web Service on NonStop" (page 340)
4. "Developing the TemperatureConverter Client on Windows using the Eclipse Galileo IDE" (page 342)
5. "Modifying the Client Stub File" (page 344)
6. "Running the TemperatureConverter Client on Windows" (page 345)

## Developing the TemperatureConverter Web Service on Windows

Developing the TemperatureConverter web service on a Windows system involves the following activities:

1. Generating the web service skeleton file using WSDL2Java
2. Update service skeleton with business logic
3. Creating an Axis2/Java AAR File

### Generating the web service skeleton file using WSDL2Java

To generate the web service skeleton file, complete the following steps:

1. Create the TemperatureConverter-Contract-First sub-directory in *<My SASH Home>\axis2\gettingstarted* directory on your Windows system.
2. Create a WSDL file for the TemperatureConverter application. You can use one of the following approaches to create a WSDL file:
  1. Use an existing WSDL file.  
If you want to use an existing WSDL file, save the file as `TemperatureConverter.wsdl` in the *<My SASH Home>\axis2\gettingstarted\TemperatureConverter-Contract-First* directory.
  2. Create a new WSDL file.  
Open any text editor that you have on your Windows system. Save the file as `TemperatureConverter.wsdl` in the *<My SASH Home>\axis2\gettingstarted\TemperatureConverter-Contract-First* directory and include the required contract.

3. Set the bin directory of Axis2/Java in the Path variable using the following command:

```
<command prompt>: set AXIS2_HOME=<Axis2 Home>
```

For example,

```
command>set AXIS2_HOME=C:\axis2-1.5.2-bin
```

```
command>set PATH=%PATH%;%AXIS2_HOME%\bin;
```

4. Run the following command to generate the service skeleton file, an ant build file, and the service descriptor file:

```
<command prompt>: WSDL2Java -uri TemperatureConverter.wsdl -ss -sd true
```

The command generates the following service artifacts:

- `build.xml`

This is an ant file that is used to perform tasks such as, build the Axis2/Java archive file, clean the Axis2/Java archive file, and so on.

- `resources`

This directory contains two files: the WSDL file for the service, `TemperatureConverter.wsdl`, and the service descriptor file, `services.xml` file

- `src`

This is the source folder for the application. By default, the following package structure is created in the `src` directory:

```
org\apache\axis2\
```

**NOTE:** You can also specify a different package structure for your application using the WSDL2Java tool. For more information on the WSDL2Java options, see <http://ws.apache.org/axis/java/reference.html>.

The service skeleton file for your web service is created and you need to implement the required business logic.

### Update the service skeleton file with the business logic

To update the business logic in the service skeleton file, complete the following steps:

1. Modify the <My SASH Home>\axis2\gettingstarted\TemperatureConverter-Contract-First\src\org\apache\ws\axis2\TemperatureConverterSkeleton.java file to implement the c2F conversion and f2CC conversion (to convert Celsius to Fahrenheit and Fahrenheit to Celsius respectively) methods using any text editor.
2. Verify that the skeleton file resembles the following code snippet:

```
* by the Apache Axis2 version: 1.5.2 Built on : Sep 06, 2010 (09:42:01 CEST)
*/
package org.apache.ws.axis2;

/**
 * TemperatureConverterSkeleton java skeleton for the axisService
 */
public class TemperatureConverterSkeleton {

    /**
     * Auto generated method signature
     *
     * @param c2FConversion
     */
    public org.apache.ws.axis2.C2FConversionResponse c2FConversion(
        org.apache.ws.axis2.C2FConversion c2FConversion) {
        double fValue = c2FConversion.getArgs0();
        org.apache.ws.axis2.C2FConversionResponse response = new C2FConversionResponse();
        response.set_return(((fValue - 32.0) * 5.0) / 9.0);

        return response;
    }

    /**
     * Auto generated method signature
     *
     * @param f2CConversion
     */
    public org.apache.ws.axis2.F2CConversionResponse f2CConversion(
        org.apache.ws.axis2.F2CConversion f2CConversion) {
        double fValue = f2CConversion.getArgs0();
        org.apache.ws.axis2.F2CConversionResponse response = new F2CConversionResponse();
        response.set_return(((fValue - 32.0) * 5.0) / 9.0);

        return response;
    }
}
```

### Creating an Axis2/Java AAR File

An Axis2/Java AAR file contains the compiled code of the service implementation class, and the services.xml file. To create an Axis2/Java AAR file, complete the following steps:

1. Go to the <My SASH Home>\Axis2\gettingstarted\TemperatureConverter-Contract-First directory using the command:  
command prompt> cd <My SASH Home>\Axis2\gettingstarted\TemperatureConverter-Contract-First
2. Create the AAR archive using the command:  
command prompt> ant

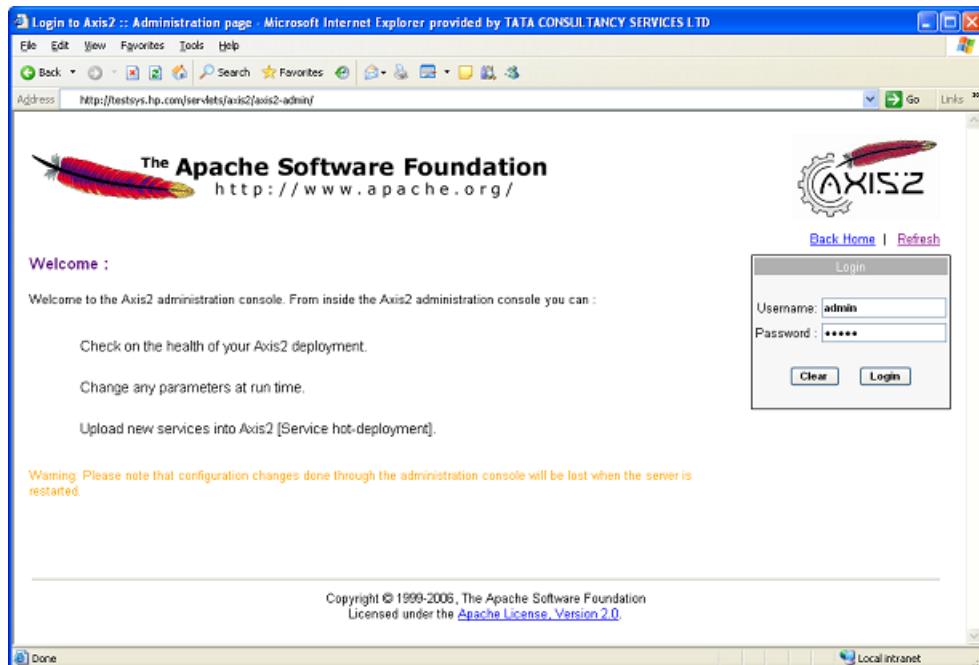
The TemperatureConverter.aar file is created in the <My SASH Home>\Axis2\gettingstarted\TemperatureConverter-Contract-First\build\lib directory.

## Deploying the TemperatureConverter Web Service on NonStop

To deploy the TemperatureConverter.aar file on your NonStop system, complete the following steps:

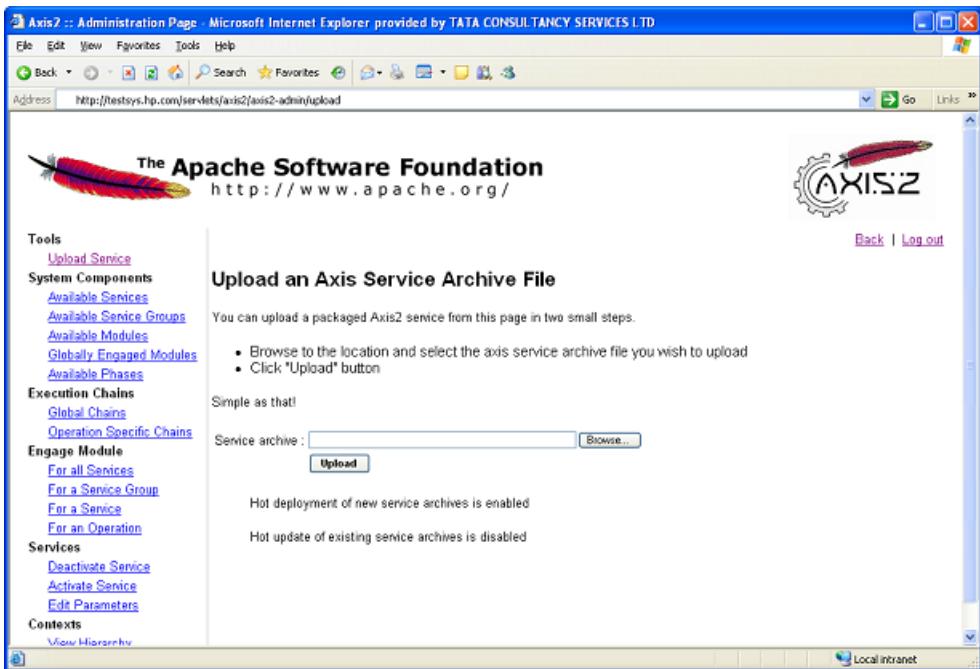
1. Go to the Axis2/Java admin login page using the following URL: [http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/axis2-admin/](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/axis2-admin/).

**Figure 126 Axis2/Java Administration Login Page**



2. Enter the username and password for the admin login to Axis2/Java. The default password for username admin is axis2.
3. In the **Tools** section, click **Upload Service**. In the **Service Archive** field, enter the complete address for the TemperatureConverter.aar file or click **Browse...** to locate and select the file.

**Figure 127 Axis2/Java Administration Upload Page**



4. Click **Upload** to upload the service in Axis2/Java.

This completes deploying TemperatureConverter and it is listed in <http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/axis2/services/listServices>.

### Running the TemperatureConverter Web Service on NonStop

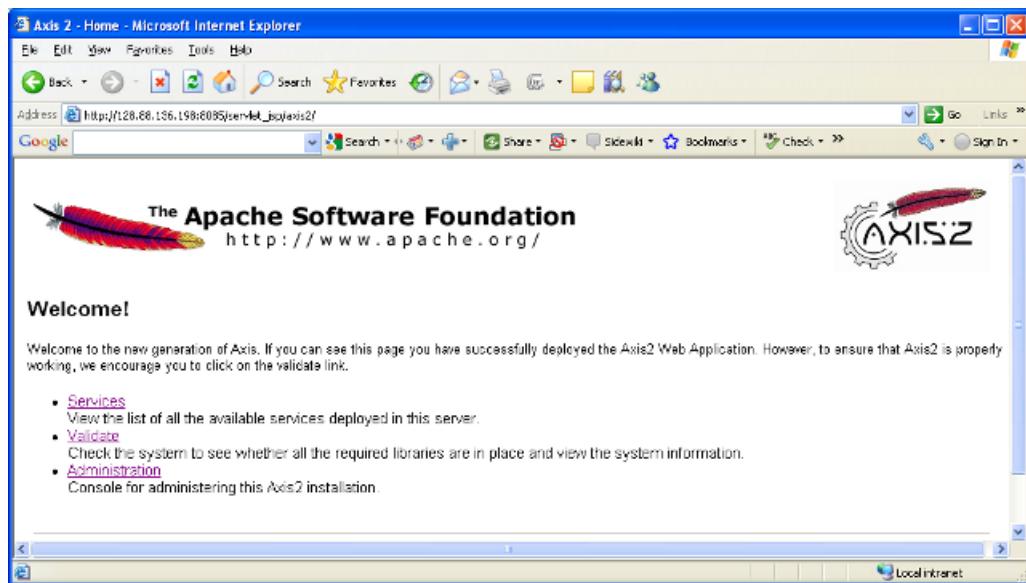
To run the TemperatureConverter web service on your NonStop system, complete the following steps:

1. If iTP WebServer is already running, the TemperatureConverter web service must be listed in <http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/axis2/services/listServices>
2. If iTP WebServer is not running, start the iTP WebServer using the following steps:
  - a. Go to <iTP WebServer Deployment Directory>/conf, using the command:  
OSS> cd <iTP WebServer Deployment Directory>/conf  
For example:  
OSS> cd /home/sash\_usr/webserver/conf
  - b. Start your iTP WebServer, using the command:  
OSS> ./start
  - c. Access the TemperatureConverter web service using the following URL:  
<http://<IP Address of the iTP WebServer>:<port#>/<servlet directory>/axis2/services/listServices>

The TemperatureConverter List Services Screen displays details such as, Version, Service Description, Service Status, and Available Operations.

Figure 128 shows the TemperatureConverter List Services Screen.

**Figure 128 TemperatureConverter List Services Screen**



You can view the auto-generated WSDL file for the TemperatureConverter application by clicking the TemperatureConverter in [Figure 111](#).

[Figure 129](#) shows the WSDL file for the TemperatureConverter web service.

**Figure 129 TemperatureConverter Web Service WSDL File**

The screenshot shows a Microsoft Internet Explorer window displaying the WSDL (Web Services Description Language) XML code for the TemperatureConverter web service. The URL in the address bar is `http://128.88.136.198:8085/servlet_jsp/axis2/services/TemperatureConverter?wsdl`. The XML code is color-coded to highlight different namespaces and elements.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  xmlns:ns1="http://org.apache.axis2/xsd" xmlns:ns="http://ws.apache.org/axis2"
  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  targetNamespace="http://ws.apache.org/axis2">
  <wsdl:documentation>TemperatureConverter</wsdl:documentation>
  - <wsdl:types>
    - <xss:schema attributeFormDefault="qualified" elementFormDefault="qualified">
        targetNamespace="http://ws.apache.org/axis2">
        - <xss:element name="f2cConversion">
            - <xss:complexType>
                - <xss:sequence>
                    <xss:element minOccurs="0" name="args0" type="xss:double" />
                </xss:sequence>
            </xss:complexType>
        </xss:element>
        - <xss:element name="f2cConversionResponse">
            - <xss:complexType>
                - <xss:sequence>
```

You use the WSDL file to create the TemperatureConverter web service client so that the TemperatureConverter is invoked as a web service.

### Developing the TemperatureConverter Client on Windows using the Eclipse Galileo IDE

To invoke the TemperatureConverter service running on your NonStop system, you need to create an Axis2/Java client on your Windows system. The APIs of the client application will be used to invoke the web service.

Use the Eclipse IDE to create an Axis2/Java client for the TemperatureConverter web service.

To develop the Axis2/Java client application *AxisClient* for the TemperatureConverter web service, complete the following steps:

1. “Creating the Client Project in Eclipse” (page 342)
2. “Adding Dependency JAR Files” (page 343)
3. “Creating the Client stub file using WSDL2Java ” (page 343)

#### Creating the Client Project in Eclipse

To create the client project in Eclipse, see “[Developing TemperatureConverter Client on Windows using the Eclipse Galileo IDE](#)” (page 324). The name of the project in this approach is *AxisClientStub*. After creating the project, copy the WSDL file to this project folder.

## Adding Dependency JAR Files

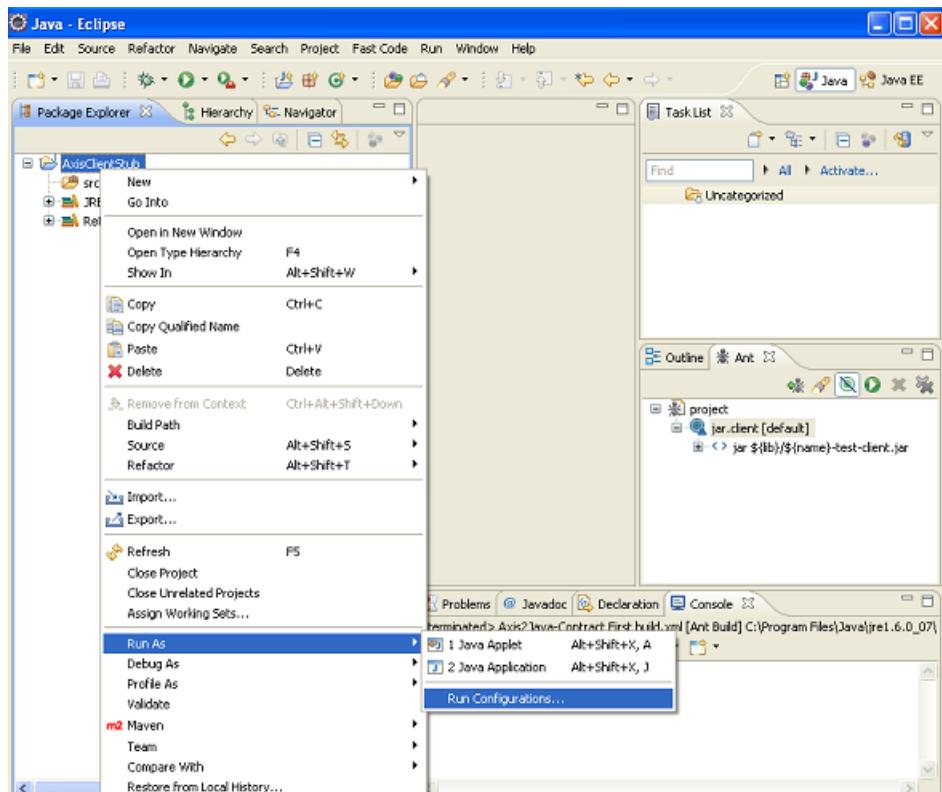
Add all the Axis2/Java dependency JAR files in the Java Build Path of the *AxisClientStub* project to compile and link the project. These JAR files are located in the <Axis2/Java Home>/lib directory.

To add the dependency JAR files to the projects library path, complete the steps described in “[Adding Dependency JAR Files](#)” (page 332).

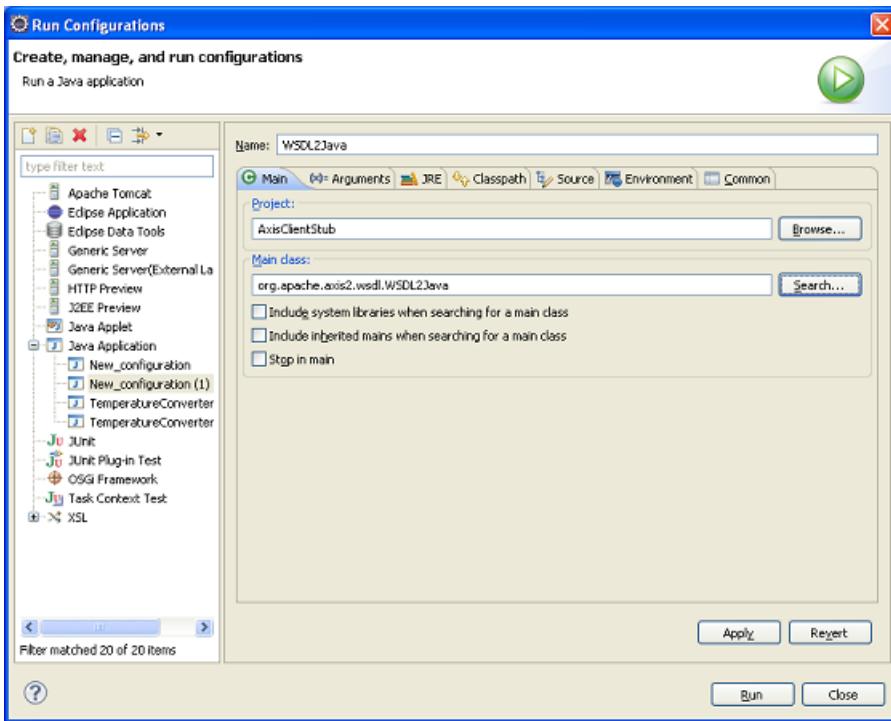
## Creating the Client stub file using WSDL2Java

To create the Client stub, complete the following steps:

1. On the **Project Explorer** frame, right-click **AxisClientStub** and select **Run as→Run Configuration**. The **Run Configuration** dialog box appears.

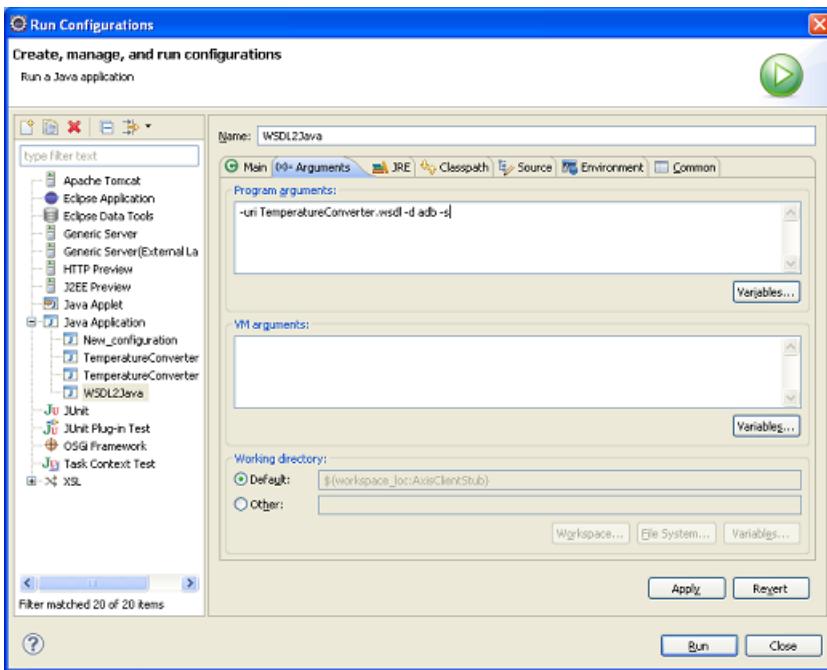


2. Right-click **Java Application** in the Explorer pane and select **New** to create a new configuration to run your Java application. The **Run Configuration** dialog appears. Enter the name for the new run configuration.
3. Click **Browse** to select the project and then, click **Search** to specify the main class of the selected project.



- 4.** Click the **Arguments** tab on the **Run Configuration** dialog and enter the following argument in the **Program arguments:** field:

```
-uri TemperatureConverter.wsdl -d adb -s
```



- 5.** Click **Apply** and then click **Run**.

The client stub file is created in the following location:

AxisClientStub\src\org\apache\ws\axis2.

## Modifying the Client Stub File

Add the following code in the

AxisClientStub\src\org\apache\ws\axis2\TemperatureConverterStub.java  
file:

```
public static void main(String[] args) throws RemoteException {
    TemperatureConverterStub stub = new TemperatureConverterStub();
    F2CConversion f2CConversion = new F2CConversion();
    double param = 1;
    f2CConversion.setArgs0(param);
    stub.f2CConversion(f2CConversion);
    C2FConversion conversion = new C2FConversion();
    double param2 = 2;
    conversion.setArgs0(param2);
    stub.c2FConversion(conversion);
}
```

## Running the TemperatureConverter Client on Windows

To run TemperatureConverter, see “[Running TemperatureConverter Client on Windows](#)” (page 335).

---

## Part V Integrating Frameworks

Part III includes the following chapters:

- “[Using Spring Transaction Manager](#)” (page 348)  
This chapter helps you to use the Spring Transaction Manager.
- “[Integrating Hibernate into Spring](#)” (page 367)  
This chapter helps you to integrate Hibernate with the Spring framework.
- “[Integrating JPA with Hibernate into Spring](#)” (page 383)  
This chapter helps you to integrate JPA with Hibernate with the Spring framework.

---

# Contents

<b>18 Using Spring Transaction Manager.....</b>	<b>348</b>
Why Transaction Management is required.....	348
Spring Transaction Management.....	348
Declarative Transaction Management.....	348
Programmatic Transaction Management.....	348
Example of Using Spring Transaction Manager.....	348
Modifying EmplInfo on Windows using Eclipse Galileo IDE.....	349
Deploying EmplInfo on NonStop.....	362
Running EmplInfo on NonStop.....	366
<b>19 Integrating Hibernate into Spring.....</b>	<b>367</b>
Why Integrate Hibernate into Spring.....	367
Example of Integrating Hibernate into Spring.....	367
Modifying EmplInfo on Windows using Eclipse Galileo IDE.....	368
Deploying EmplInfo on NonStop.....	379
Running EmplInfo on NonStop.....	382
<b>20 Integrating JPA with Hibernate into Spring.....</b>	<b>383</b>
Why Integrate JPA with Hibernate in Spring.....	383
Example of Integrating JPA with Hibernate into Spring.....	383
Modifying EmplInfo on Windows using Eclipse Galileo IDE.....	384
Deploying EmplInfo on NonStop.....	394
Running EmplInfo on NonStop.....	397
<b>21 Integrating Axis2/Java into Spring.....</b>	<b>398</b>
Why Integrate Axis2/Java into Spring.....	398
Example of Integrating Axis2/Java with Spring.....	398
Modifying EmplInfo on Windows using Eclipse Galileo IDE.....	399
Deploying EmplInfo on NonStop.....	404
Running EmplInfo Web Service on NonStop.....	408
Developing EmplInfoClient on Windows.....	408
Running EmplInfoClient on Windows.....	416
<b>22 Integrating MyFaces into Spring.....</b>	<b>421</b>
Why Integrate MyFaces into Spring.....	421
Example of Integrating MyFaces into Spring.....	421
Modifying EmplInfo on Windows using Eclipse Galileo IDE.....	422
Deploying EmplInfo on NonStop.....	444
Running EmplInfo on NonStop.....	448

# 18 Using Spring Transaction Manager

This chapter describes how a Spring application can manage its database transactions using a Spring Transaction Manager. This is demonstrated by modifying the EmplInfo application (developed in “[Getting Started with Spring](#)” (page 75) chapter) using the Eclipse Galileo IDE.

The following topics are discussed in this section:

- “[Why Transaction Management is required](#)” (page 348)
- “[Spring Transaction Management](#)” (page 348)
- “[Example of Using Spring Transaction Manager](#)” (page 348)

## Why Transaction Management is required

A transaction is an inseparable unit of work comprising several operations, all or none of which must be performed to preserve data integrity.

A transaction may contain one or more SQL statements, all of which can be either committed (applied to the database) or rolled back (undone from the database) simultaneously. Thus, whenever a set of operations needs to be implemented or completed simultaneously and if any error occurs during any of the operation, the whole set of operations must be undone. This is done using the Transaction Manager.

## Spring Transaction Management

The Spring framework enables you to leverage the complete transaction support. It provides the following ways for managing transactions:

- “[Declarative Transaction Management](#)” (page 348)
- “[Programmatic Transaction Management](#)” (page 348)

## Declarative Transaction Management

The Declarative Transaction Management in Spring uses the Spring Aspect Oriented Programming (AOP) feature to implement transactions on a particular business logic class. This option has minimum impact on the application code and hence is widely used.

## Programmatic Transaction Management

In Programmatic Transaction Management, the transaction manager is handled within the application code.

A programmatic transaction can be a Java Database Connectivity (JDBC) or Java Transaction API (JTA) transaction. In the Spring framework, programmatic transaction management can be done using:

- **TransactionTemplate**: uses a callback approach to free application code and release transactional resources.
- **PlatformTransactionManagerImplementation**: uses an instance of `org.springframework.transaction.PlatformTransactionManager` and initiates transactions using the `TransactionDefinition` and `TransactionStatus` objects.

## Example of Using Spring Transaction Manager

This section describes how business transactions in a Spring application (EmplInfo) can be managed using the Declarative approach and the Programmatic approach.

## Prerequisites

The prerequisites for using the Spring Transaction Manager are:

- Basic knowledge of the Spring framework and transactions.
- EmplInfo application (explained in the “[Getting Started with Spring](#)” (page 75) chapter) developed on the Windows system.
- Following software installed on the NonStop system:
  - NonStop iTP WebServer version T8996H02 or later
  - NSJSP version T1222H60 or later
  - JDBC Type 2 driver version T1275H50 or later, or JDBC Type 4 driver version T1249V11 or later
  - NonStop SQL/MX version T1050H23 or later
  - NSJ version T2766H60 or later
- Following software installed on the Windows system:
  - JDK version 1.5 or later
  - JDBC Type 4 driver version T1249V11 or later
  - Eclipse Galileo IDE version 3.3.1.1 or later

## Activities involved

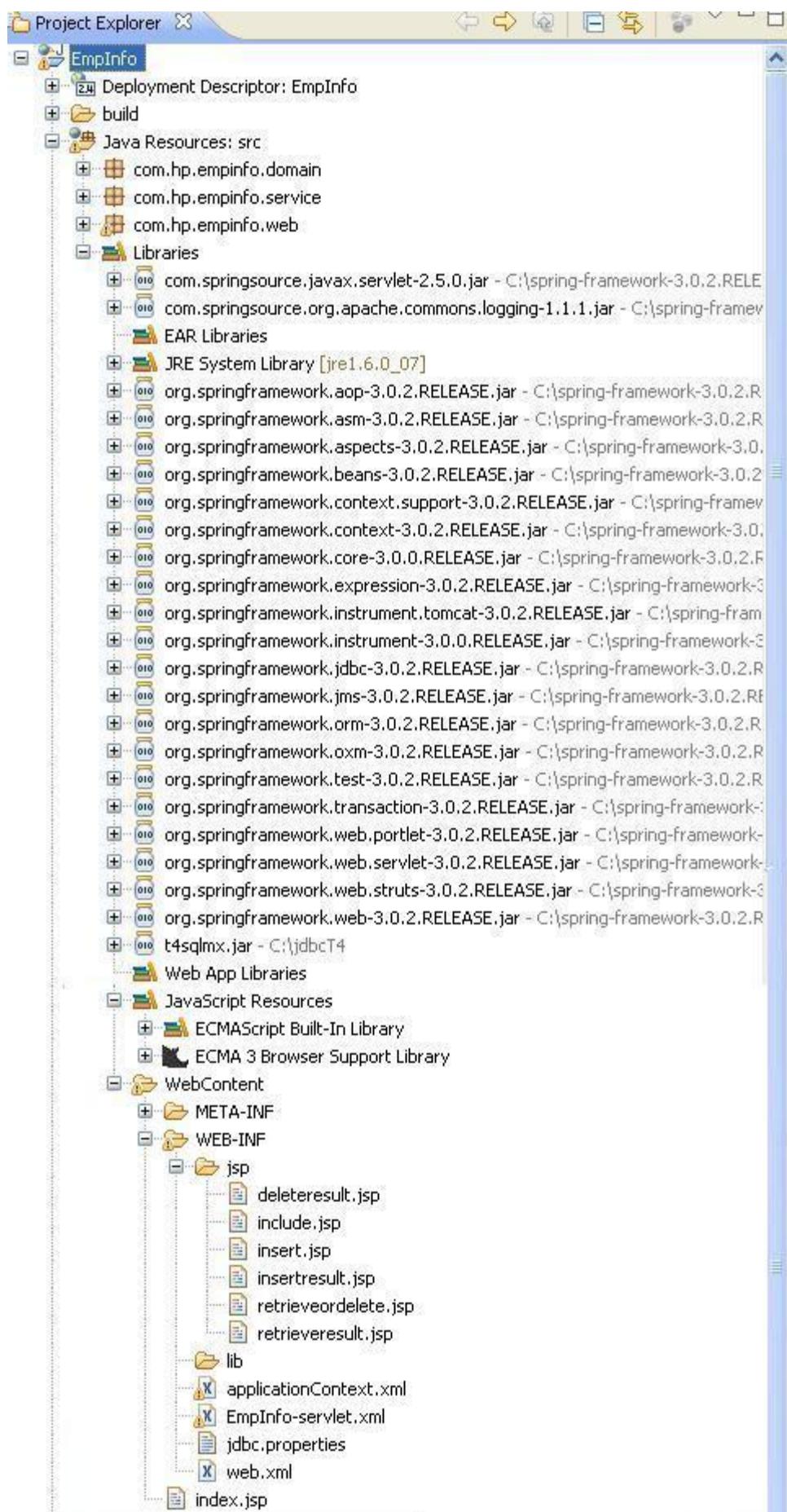
The following activities are required to modify the EmplInfo application so that its database operations are managed by the Spring transaction manager:

- “[Modifying EmplInfo on Windows using Eclipse Galileo IDE](#)” (page 349)
- “[Deploying EmplInfo on NonStop](#)” (page 362)
- “[Running EmplInfo on NonStop](#)” (page 366)

## Modifying EmplInfo on Windows using Eclipse Galileo IDE

EmplInfo, developed in the “[Getting Started with Spring](#)” (page 75) chapter, is the starting point to demonstrate the usage of Spring Transaction Manager.

**Figure 130 Project Explorer View**



Modifying the EmplInfo application involves the following tasks:

1. "Modifying EmplInfo based on the Type of Transaction" (page 351)
  - "For Declarative Transaction" (page 351)
    - "Creating the `IEmployeeDao.java` File" (page 351)
    - "Modifying the `EmployeeController.java` File" (page 353)
    - "Modifying the `applicationContext.xml` File" (page 354)
  - "For Programmatic Transaction" (page 356)
    - "Modifying the `applicationContext.xml` File" (page 356)
    - "Modifying the `EmployeeController.java` File" (page 357)
2. "Adding Dependency JAR Files" (page 359)
  - "For Declarative Transaction" (page 359)
  - "For Programmatic Transaction" (page 361)

## Modifying EmplInfo based on the Type of Transaction

In Declarative Transaction Management, all the transactions are set and managed by the proxy factories configured in the Spring configuration file. Therefore, you must modify the `applicationContext.xml` file.

In Programmatic Transaction Management, transaction management is done directly in the code. Therefore, you must modify the `EmployeeController.java` file.

The following sections describe the modifications required in the `applicationContext.xml` file and the `EmployeeController.java` file.

### For Declarative Transaction

The following activities are involved in the declarative transactions.

#### Creating the `IEmployeeDao.java` File

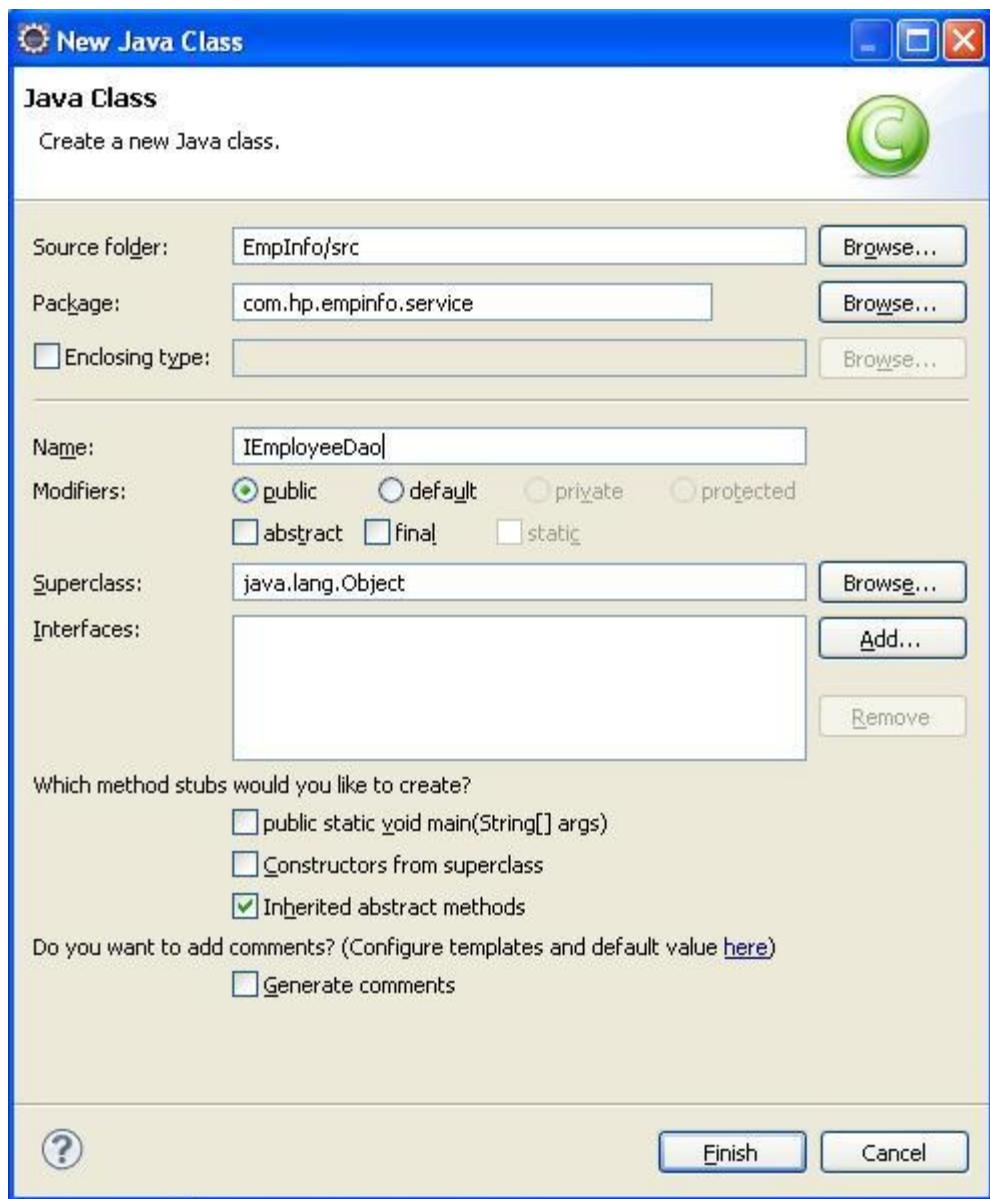
Create a Java interface named `IEmployeeDao.java` to implement the `EmployeeDao.java` class.

The `IEmployeeDao.java` is placed with its implementation class in the `com.hp.empinfo.service` package.

To create the `IEmployeeDao.java` file, complete the following steps:

1. On the Project Explorer frame, right-click **EmplInfo** and select **New > Interface**.  
The New Java Interface dialog box appears.
2. In the **Source folder** field, type `EmplInfo/src` and in the **Package** field, type `com.hp.empinfo.service`.
3. In the **Name** field, type `IEmployeeDao` and click **Finish**.

**Figure 131 New Java Interface Dialog Box**



The IEmployeeDao.java file is created.

4. Modify the IEmployeeDao.java file to declare the methods implemented in the EmployeeDao.java file.

After modification, the IEmployeeDao.java file must appear as:

```
package com.hp.empinfo.service;

import java.sql.SQLException;
import com.hp.empinfo.domain.Employee;

public interface IEmployeeDao {

    public Employee getDetail(int empid) throws SQLException;
    public void insertDetail(int empid, String firstname, String lastname,
                           int age, String email) throws SQLException;
    public String deleteEmployee(int empid) throws SQLException;
```

## Modifying the EmployeeController.java File

Modify EmployeeController.java to replace the instance of the EmployeeDao.java class with the instance of the IEmployeeDao.java interface.

To modify the EmployeeController.java file, complete the following steps:

1. Change an import statement to import the IEmployeeDao.java interface instead of EmployeeDao.java.

```
import com.hp.empinfo.service.IEmployeeDao;
```

2. Replace the EmployeeDao instance with that of IEmployeeDao as:

```
private IEmployeeDao empdao;  
empdao = (IEmployeeDao) wac.getBean("iempdao");
```

After modification, the EmployeeController.java file must appear as:

```
package com.hp.empinfo.web;  
  
import java.sql.SQLException;  
import java.util.HashMap;  
import java.util.Map;  
  
import javax.servlet.ServletException;  
  
import org.springframework.web.context.WebApplicationContext;  
import org.springframework.web.context.support.WebApplicationContextUtils;  
import org.springframework.web.servlet.ModelAndView;  
import org.springframework.web.mvc.SimpleFormController;  
  
import com.hp.empinfo.domain.Employee;  
import com.hp.empinfo.service.IEmployeeDao;  
  
public class EmployeeController extends SimpleFormController {  
  
    private IEmployeeDao empdao;  
  
    public ModelAndView onSubmit(Object command) throws ServletException,  
        SQLException {  
  
        WebApplicationContext wac = WebApplicationContextUtils  
            .getRequiredWebApplicationContext(getApplicationContext());  
  
        empdao = (IEmployeeDao) wac.getBean("iempdao");  
  
        int empid = ((Employee) command).getEmpid();  
        String firstname = ((Employee) command).getFirstname();  
        String lastname = ((Employee) command).getLastname();  
        int age = ((Employee) command).getAge();  
        String email = ((Employee) command).getEmail();  
        String rord = ((Employee) command).getRord();  
  
        if (rord != null && rord.equalsIgnoreCase("Retrieve")) {  
  
            Employee emp1 = empdao.getDetail(empid);  
  
            Map<String, String> model = new HashMap<String, String>();  
            model.put("empid", "" + emp1.getEmpid());  
            model.put("empfn", emp1.getFirstname());  
            model.put("empln", emp1.getLastname());  
            model.put("empage", "" + emp1.getAge());  
            model.put("empemail", emp1.getEmail());  
  
            return new ModelAndView("retrieveresult", "model", model);  
        }  
    }
```

```

if (rord != null && rord.equalsIgnoreCase("Delete")) {
    String str = empdao.deleteEmployee(empid);
    Map<String, String> model = new HashMap<String, String>();
    model.put("del", str);
    return new ModelAndView("deleteresult", "model", model);
}

else {
    empdao.insertDetail(empid, firstname, lastname, age, email);
    Map<String, String> model = new HashMap<String, String>();
    model
        .put("add",
            "Transaction Complete - One Employee Added to Employee Database");
    return new ModelAndView("insertresult", "model", model);
}
}

```

### Modifying the applicationContext.xml File

Modify the applicationContext.xml file to include the following:

- 1. Transaction Manager:** It contains the bean definition for the transaction manager class.

```

<bean id="txManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>

```

---

**NOTE:** Because the EmplInfo application contains only JDBC operations, org.springframework.jdbc.datasource.DataSourceTransactionManager is used as the transaction manager class.

---

- 2. AOP Configurations:** It contains reference to the transaction advisor and the class on which the transaction advise will be applicable.

```

<aop:config>
  <aop:pointcut id="empdaoOperation"
    expression="execution(* com.hp.empinfo.service.EmployeeDao.*(..))"/>
  <aop:advisor advice-ref="txAdvice" pointcut-ref="empdaoOperation"/>
</aop:config>

```

- 3. Transaction Advice:** It contains references to the transaction manager class and transactional details of the getEmployeeDetail, insertEmployee and deleteEmployee methods in the EmployeeDao.java class.

```

<tx:advice id="txAdvice" transaction-manager="txManager">
  <tx:attributes>
    <tx:method name="get*" read-only="true"/>
    <tx:method name="*"/>
  </tx:attributes>
</tx:advice>

```

- 4. AOP Proxy Factory Bean:** It creates an AOP proxy factory bean that contains the business interface and the implementation class as the target.

```

<bean id = "iempdao"
      class = "org.springframework.aop.framework.ProxyFactoryBean">
  <property name = "proxyInterfaces">

```

```

        <value>com.hp.empinfo.service.IEmployeeDao</value>
    </property>
    <property name = "target">
        <ref bean = "empdao"/>
    </property>
</bean>
```

- 5. Bean Instance for the implementation class:** It creates a bean for the implementation class and contains a reference to the data source.

```

<bean id="empdao" class="com.hp.empinfo.service.EmployeeDao">
    <property name="dataSource">
        <ref bean="dataSource" />
    </property>
</bean>
```

- 6. XML namespace and schema location:** The namespace and schema locations of the `<aop:config>` and `<tx:advice>` tags must be defined as follows:

```

xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:aop=http://www.springframework.org/schema/aop"
```

and

```

xsi:schemaLocation="http://www.springframework.org/schema/aop
                    http://www.springframework.org/schema/aop/spring-aop.xsd
                    http://www.springframework.org/schema/tx
                    http://www.springframework.org/schema/tx/spring-tx.xsd"
```

After modification, the `applicationContext.xml` file should appear as:

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/aop
           http://www.springframework.org/schema/aop/spring-aop.xsd
           http://www.springframework.org/schema/tx
           http://www.springframework.org/schema/tx/spring-tx.xsd">
    <bean id="empdao" class="com.hp.empinfo.service.EmployeeDao">
        <property name="dataSource">
            <ref bean="dataSource" />
        </property>
    </bean>
    <bean id = "iempdao"
          class = "org.springframework.aop.framework.ProxyFactoryBean">
        <property name = "proxyInterfaces">
            <value>com.hp.empinfo.service.IEmployeeDao</value>
        </property>
        <property name = "target">
            <ref bean = "empdao"/>
        </property>
    </bean>
    <tx:advice id="txAdvice" transaction-manager="txManager">
        <tx:attributes>
            <tx:method name="get*" read-only="true"/>
            <tx:method name="*"/>
        </tx:attributes>
    </tx:advice>
<aop:config>
```

```

<aop:pointcut id="empdaoOperation"
    expression="execution(* com.hp.empinfo.service.EmployeeDao.*(..))" />
<aop:advisor advice-ref="txAdvice" pointcut-ref="empdaoOperation"/>
</aop:config>
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName">
        <value>${jdbc.driver}</value>
    </property>
    <property name="url">
        <value>${jdbc.url}</value>
    </property>
    <property name="username">
        <value>${jdbc.user}</value>
    </property>
    <property name="password">
        <value>${jdbc.password}</value>
    </property>
    <property name="connectionProperties">
        <props>
            <prop key="catalog">
                ${jdbc.catalog}
            </prop>
            <prop key="schema">
                ${jdbc.schema}
            </prop>
        </props>
    </property>

```

## For Programmatic Transaction

This section describes only the PlatformTransactionManager implementation for Programmatic Transaction.

### Modifying the applicationContext.xml File

Modify the applicationContext.xml file to include the bean definition for the transaction manager class.

---

**NOTE:** Because the EmplInfo application contains only JDBC operations, org.springframework.jdbc.datasource.DataSourceTransactionManager is used as the transaction manager class.

---

Add the following lines of code to include the bean definition for the transaction manager class:

```
<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

After modification, the applicationContext.xml file should appear as:

```

<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName">
            <value>${jdbc.driver}</value>
        </property>
        <property name="url">
            <value>${jdbc.url}</value>
        </property>
        <property name="username">
            <value>${jdbc.user}</value>
        </property>
        <property name="password">

```

```

<value>${jdbc.password}</value>
</property>
<property name="connectionProperties">
<props>
<prop key="catalog">
${jdbc.catalog}
</prop>
<prop key="schema">
${jdbc.schema}
</prop>
</props>
</property>
</bean>
<bean id="empdao" class="com.hp.empinfo.service.EmployeeDao">
<property name="dataSource">
<ref bean="dataSource" />
</property>
</bean>
<bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
<property name="locations" value="/WEB-INF/jdbc.properties" />
</bean>
<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
<property name="dataSource" ref="dataSource"/>
</bean>
</beans>

```

### **Modifying the EmployeeController.java File**

The EmployeeController.java file is modified to use the PlatformTransactionManager implementation as follows:

1. Create an instance of the PlatformTransactionManager class.

```
PlatformTransactionManager txManager;
txManager = (PlatformTransactionManager) wac.getBean("txManager");
```

2. Create instances of the Transaction Definition and Transaction Status as:

```
DefaultTransactionDefinition def = new
DefaultTransactionDefinition();
def.setPropagationBehavior(TransactionDefinition.
PROPAGATION_REQUIRED);
TransactionStatus status = txManager.getTransaction(def);
```

3. To accommodate the above-mentioned instances (in step 1 and 2), include the following imports:

```
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.TransactionDefinition;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.DefaultTransactionDefinition;
```

4. Under the transaction status, include the calls for the methods in EmployeeDao.java.

After modification EmployeeController.java file should appear as:

```
package com.hp.empinfo.web;

import java.util.HashMap;
import java.util.Map;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.TransactionDefinition;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.DefaultTransactionDefinition;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.support.WebApplicationContextUtils;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.SimpleFormController;
import com.hp.empinfo.domain.Employee;
import com.hp.empinfo.service.EmployeeDao;

public class EmployeeController extends SimpleFormController {
```

```

private EmployeeDao empdao;

PlatformTransactionManager txManager;
public ModelAndView onSubmit(Object command) throws Exception {
    WebApplicationContext wac = WebApplicationContextUtils
        .getRequiredWebApplicationContext(getApplicationContext());
    empdao = (EmployeeDao) wac.getBean("empdao");
    txManager = (PlatformTransactionManager) wac.getBean("txManager");

    int empid = ((Employee) command).getEmpid();
    String firstname = ((Employee) command).getFirstname();
    String lastname = ((Employee) command).getLastname();
    int age = ((Employee) command).getAge();
    String email = ((Employee) command).getEmail();
    String rord = ((Employee) command).getRord();
    if (rord != null && rord.equalsIgnoreCase("Retrieve")) {
        DefaultTransactionDefinition def = new DefaultTransactionDefinition();
        def
            .setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);
        TransactionStatus status = txManager.getTransaction(def);
        Employee empl;
        try {
            empl = empdao.getDetail(empid);
        } catch (Exception e) {
            // TODO: handle exception
            txManager.rollback(status);
            throw e;
        }
        txManager.commit(status);
        Map<String, String> model = new HashMap<String, String>();
        model.put("empid", "" + empl.getEmpid());
        model.put("empfn", empl.getFirstname());
        model.put("empln", empl.getLastname());
        model.put("empage", "" + empl.getAge());
        model.put("empemail", empl.getEmail());
        return new ModelAndView("retrieveresult", "model", model);
    }
    if (rord != null && rord.equalsIgnoreCase("Delete")) {
        DefaultTransactionDefinition def = new DefaultTransactionDefinition();
        def
            .setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);
        TransactionStatus status = txManager.getTransaction(def);
        String str;
        try {
            str = empdao.deleteEmployee(empid);
        } catch (Exception e) {
            // TODO: handle exception
            txManager.rollback(status);
            throw e;
        }
        txManager.commit(status);
        Map<String, String> model = new HashMap<String, String>();
        model.put("str", str);
        return new ModelAndView("deleteresult", "model", model);
    }
    else {
        DefaultTransactionDefinition def = new DefaultTransactionDefinition();
        def
            .setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);
        TransactionStatus status = txManager.getTransaction(def);
        try {
            empdao.insertDetail(empid, firstname, lastname, age, email);
        } catch (Exception e) {
            // TODO: handle exception
            txManager.rollback(status);
            throw e;
        }
        txManager.commit(status);
        Map<String, String> model = new HashMap<String, String>();
        model
            .put("add",

```

```

        "Transaction Complete - One Employee Added to Employee Database");
    return new ModelAndView("insertresult", "model", model);
}
}

```

## Adding Dependency JAR Files

Add the dependency JAR files to the build path of EmplInfo, based on the type of transaction as explained below:

### For Declarative Transaction

To incorporate the features of Declarative Transaction in the EmplInfo application, the following JAR files, other than the ones added while developing the EmplInfo application, must be added to the EmplInfo project library path:

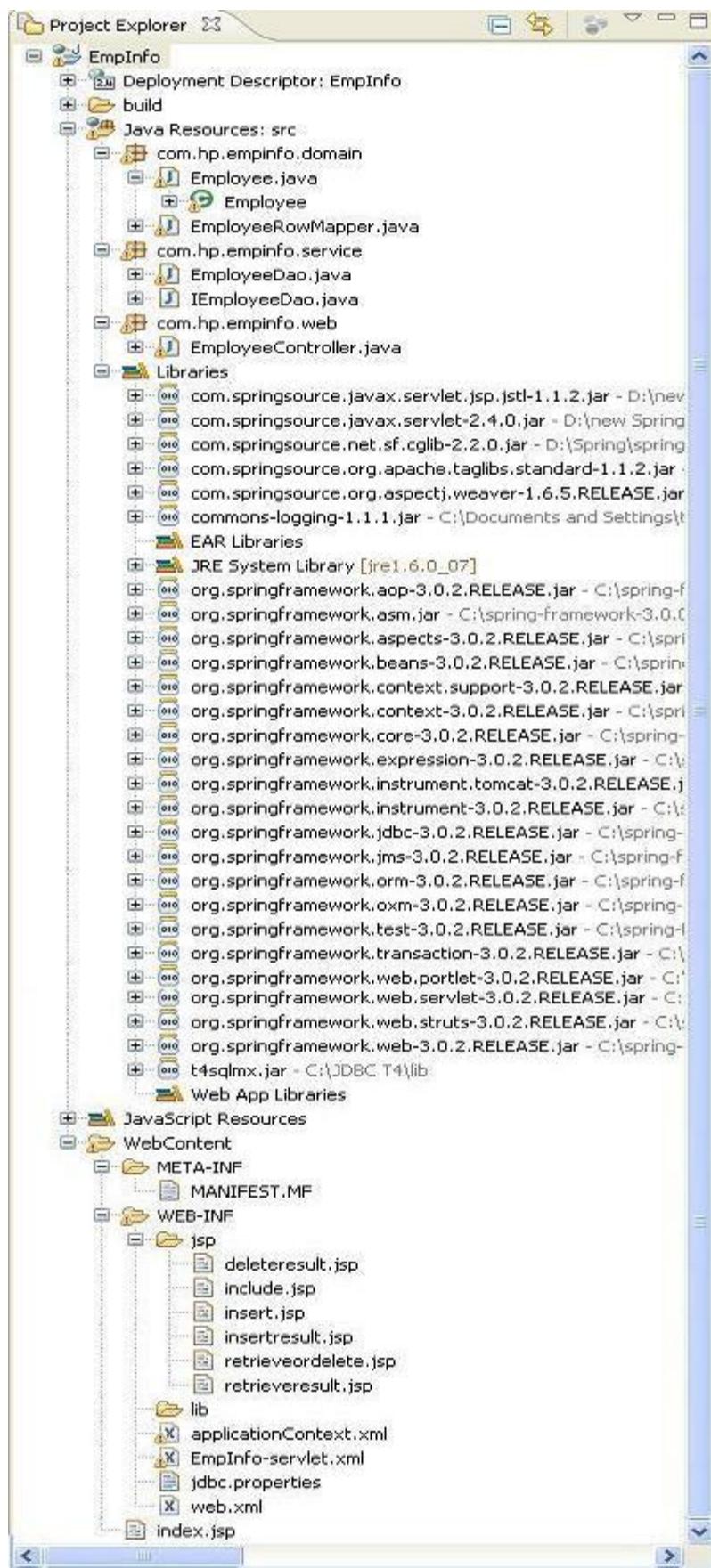
**Table 10 Dependency JAR Files**

Dependency JAR Files	Source Location
com.springsource.org.aspectj.weaver-1.6.5.RELEASE.jar	<Spring Dependency Home>\org.aspectj\com.springsource.org.aspectj.weaver\1.6.5.RELEASE
com.springsource.net.sf.cglib-2.2.0.jar	<Spring Dependency Home>\net.sourceforge.cglib\com.springsource.net.sf.cglib\2.2.0

To add these dependency JAR files in the project library path and to resolve the J2EE module dependency on these JAR files, follow the instructions explained in the [Adding Dependency JAR Files in the Project Library Path](#) section of the “[Getting Started with Spring](#)” (page 75) chapter.

This completes the modifications in the EmplInfo application to incorporate Declarative Transaction.

**Figure 132 Project Explorer View: Declarative Transaction**

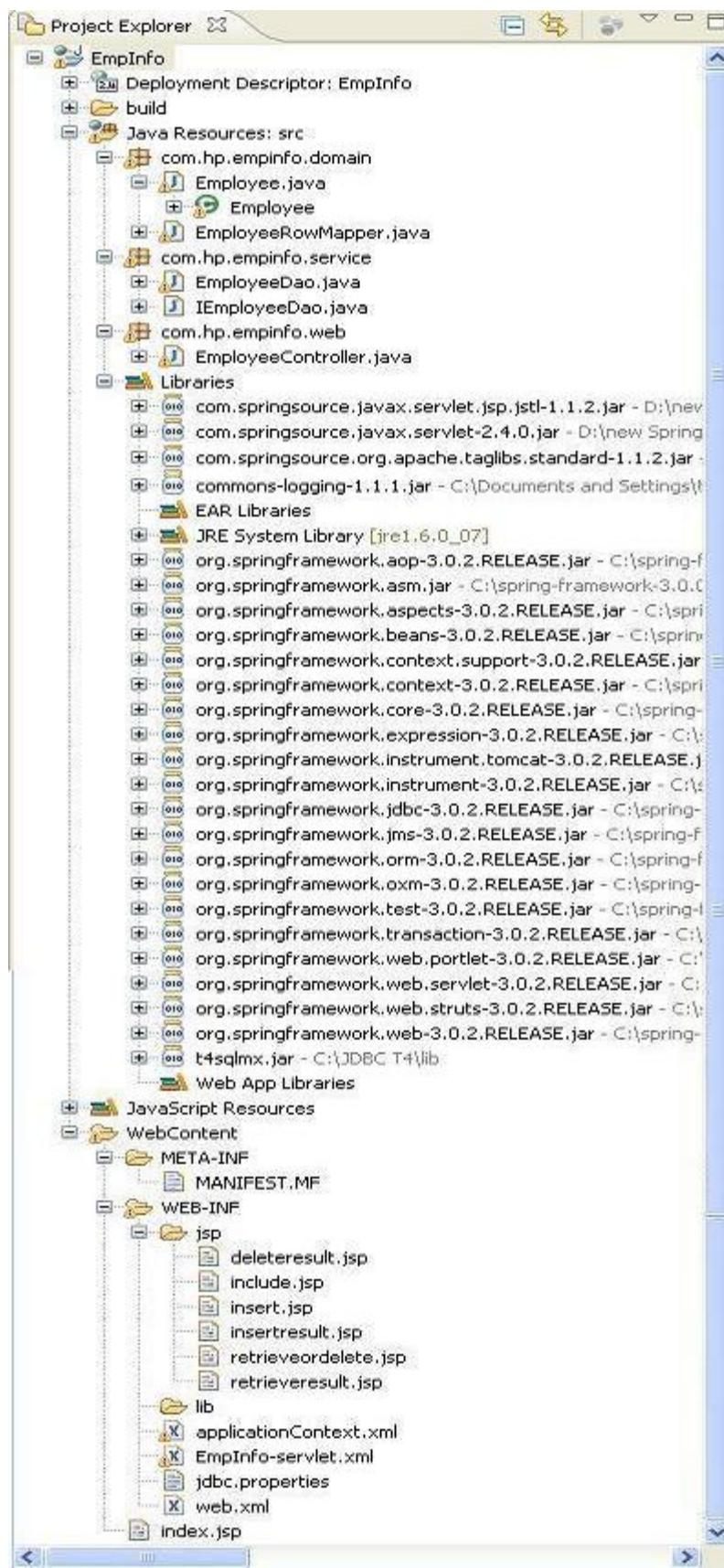


### For Programmatic Transaction

To incorporate the features of Programmatic Transaction in the EmplInfo application, no additional JAR files, other than the ones added while developing the EmplInfo application, are required.

This completes the modifications in the EmplInfo application to incorporate Programmatic Transaction.

**Figure 133 Project Explorer View: Programmatic Transaction**



## Deploying EmpInfo on NonStop

This section describes the following tasks:

1. "Creating the Application WAR File on Windows" (page 363)
2. "Deploying the EmplInfo WAR File in NSJSP on NonStop" (page 364)

## Creating the Application WAR File on Windows

A WAR file is essential to deploy the web application. It includes the property and configuration files, Java class file, and JSPs of the web application.

To create the application WAR file, complete the following steps:

1. On the Project Explorer frame, right-click **EmplInfo** and select **Export > Export**.  
The Export dialog box appears.
2. From the list of folders, select **Web > WAR file** and click **Next**.

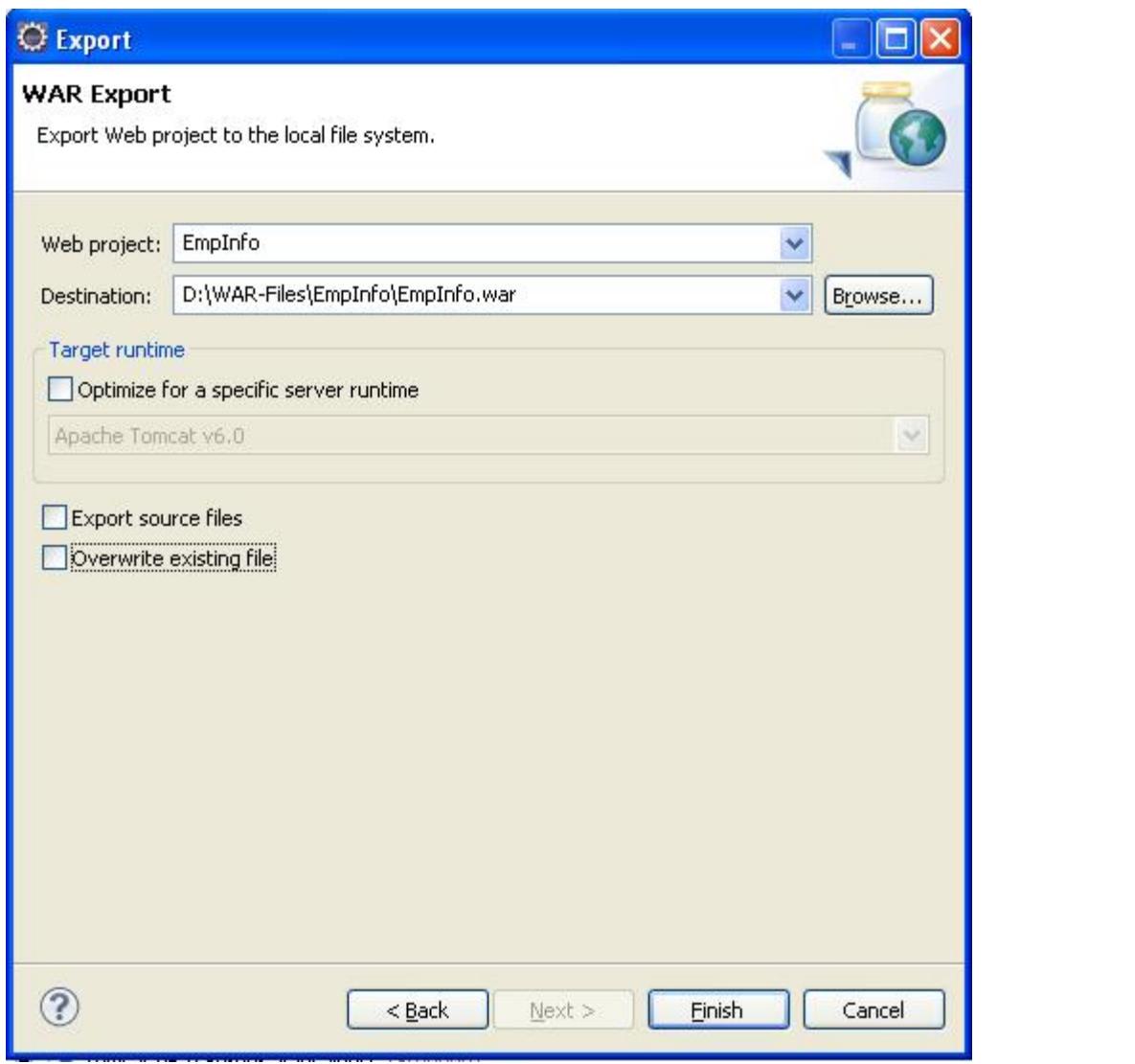
**Figure 134 Export Dialog Box**



The WAR Export dialog box appears.

3. In the **Web project** field, type **EmplInfo** and browse to the destination where you want to save the WAR file.

**Figure 135 WAR Export Dialog Box**



4. Click **Finish**. The WAR file is created.

**NOTE:** If you have specified an existing name for the WAR file, the **Finish** button is disabled. In this case, change the name of the WAR file. If you want use the existing name, select the **Overwrite existing file** check box.

## Deploying the EmplInfo WAR File in NSJSP on NonStop

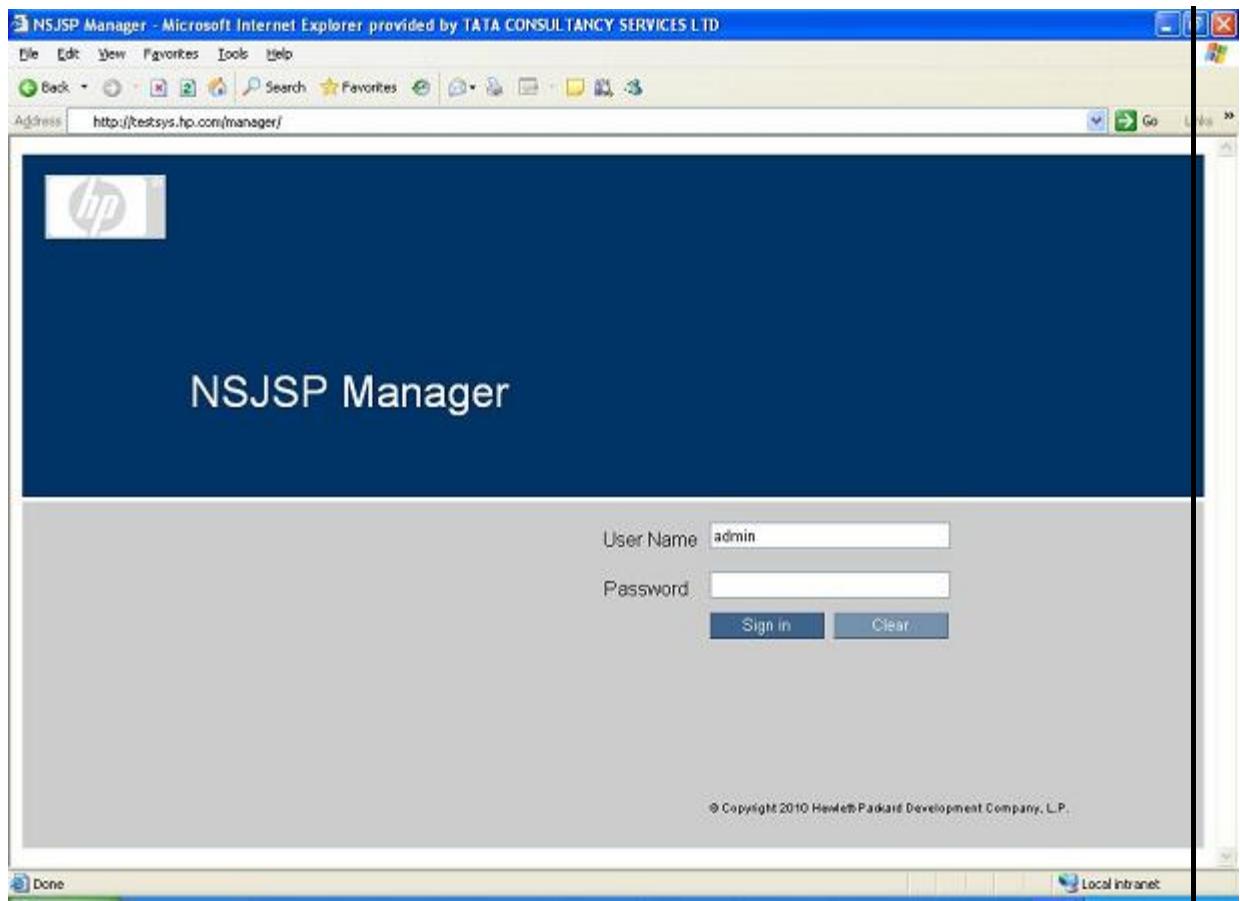
To deploy the EmplInfo WAR file on the NonStop system, complete the following steps:

1. Go to [http://<IP Address of the iTP WebServer>:<port#>/<manager\\_directory>](http://<IP Address of the iTP WebServer>:<port#>/<manager_directory>).

The **NSJSP Manager Login** screen appears.

Figure 136 shows the **NSJSP Manager Login** screen.

**Figure 136 NSJSP Manager Login Screen**

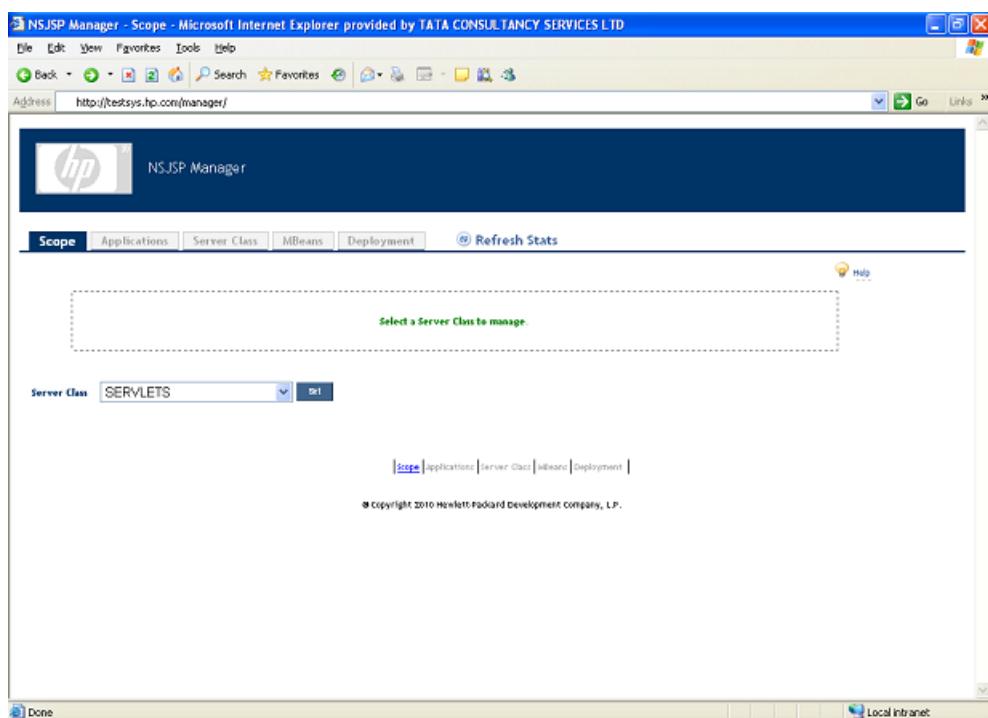


2. Enter the **User name:** and **Password:** and click **OK**.

The **NSJSP Manager** screen appears.

Figure 137 shows the **NSJSP Manager** screen.

**Figure 137 NSJSP Manager Screen**

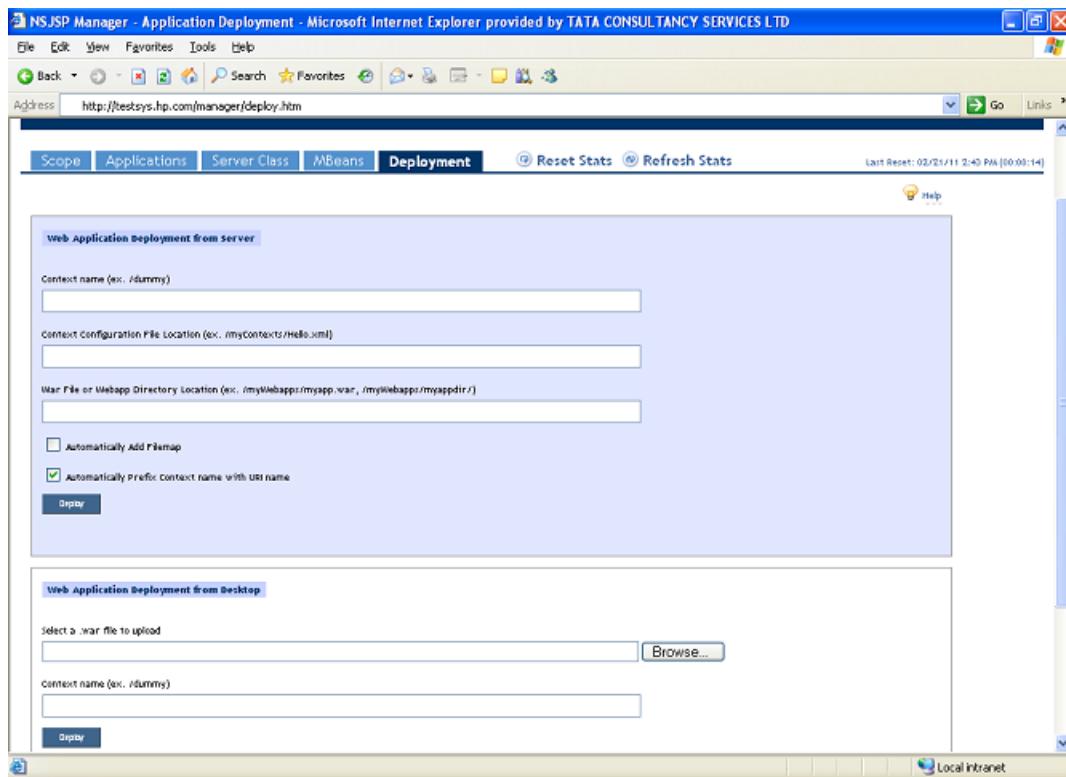


- Select **SERVLETS** for the **Server Class** and click **Set**.

The **Host (in server.xml)** tab is enabled and populated with **localhost**. Click **Set**, which enables the **Deployment** tab on the **NSJSP Manager** screen.

Figure 138 shows the **Deployment** tab of **NSJSP Manager**.

**Figure 138 NSJSP Manager Screen - Deployment tab**



- In the **Deployment** tab, complete the following steps in the **Web Application Deployment from Desktop** section:
  - In the **Select a .war file to upload** field, click **Browse...** and locate the `EmplInfo.war` file on the Windows system.
  - (Optional) In the **Context name** field, enter a name for the application context.
  - Click **Deploy**.

`EmplInfo` is deployed on NSJSP and is listed under **Applications** tab of the **NSJSP Manager** screen.

**NOTE:** HP recommends that you use the NSJSP manager for deployment. Deployment using the FTP or any other mechanism is not recommended. For more information on using the NSJSP manager, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

## Running EmplInfo on NonStop

To run `EmplInfo` on the NonStop system, click the `/<servlet directory>/EmplInfo` path under **Applications** in the **NSJSP Manager** screen.

You can now add, search, and delete employee details as explained in the “[Getting Started with Spring](#)” (page 75) chapter.

# 19 Integrating Hibernate into Spring

This chapter describes how a Spring application can use Hibernate for its database operation. This is demonstrated by modifying the EmplInfo application (developed in the “[Getting Started with Spring](#)” (page 75) chapter) using the Eclipse Galileo IDE.

The following topics are discussed in this section:

- “[Why Integrate Hibernate into Spring](#)” (page 367)
- “[Example of Integrating Hibernate into Spring](#)” (page 367)

## Why Integrate Hibernate into Spring

Hibernate is a powerful ORM tool that enables an application to access data from any database in a platform-independent manner. Therefore, the need for Spring to depend on low-level JDBC details, such as managing connections, dealing with statements and result sets, is greatly reduced. As a result, all necessary details for accessing a particular data source can be configured in XML files.

However, a disadvantage of using Hibernate is that the client application that accesses the database using Hibernate framework will depend on Hibernate APIs, such as Configuration, SessionFactory, and Session. These APIs will continue to get scattered across the code throughout the application. Moreover, the application code must be manually maintained to manage the business objects.

In Spring, the business objects can be highly configurable using the Inversion of Control (IOC) Container. Therefore, the state of an object can be externalized from the application code, thereby enabling the use of Hibernate objects as Spring Beans. Also, the Data Access Object (DAO) support in Spring facilitates data access technologies such as JDBC, Hibernate, or Java Data Objects (JDO) in a consistent way.

The JDBC support in Spring for Object Relational Mapping frameworks provide integration points to the frameworks and other services such as:

- Integrated support for Spring declarative transactions
- Transparent exception handling
- Thread-safe and lightweight template classes
- DAO support classes
- Resource management

Integrating Hibernate with Spring allows the application to use both the Spring features and the Hibernate features of accessing the database.

## Example of Integrating Hibernate into Spring

The EmplInfo application (described in the “[Getting Started with Spring](#)” (page 75) chapter) is a Spring application that uses JDBC for handling database operations.

This section describes how to use Hibernate for handling database operations of the EmplInfo application.

### Prerequisites

The prerequisites for integrating the EmplInfo application with Hibernate are:

- Basic knowledge of the Spring and Hibernate frameworks.
- EmplInfo application (explained in the “[Getting Started with Spring](#)” (page 75) chapter) developed on the Windows system.

- Following software installed on the NonStop system:
  - NonStop iTP WebServer version T8996H02 or later
  - NSJSP version T1222H60 or later
  - JDBC Type 2 driver version T1275H50 or later, or JDBC Type 4 driver version T1249V11 or later
  - NonStop SQL/MX version T1050H23 or later
  - NSJ version T2766H50 or later
- Following software installed on the Windows system:
  - JDK version 1.5 or later
  - JDBC Type 4 driver version T1249V11 or later
  - Eclipse Galileo IDE version 3.3.1.1 or later

## Activities involved

Integrating the EmplInfo application with Hibernate involves the following tasks:

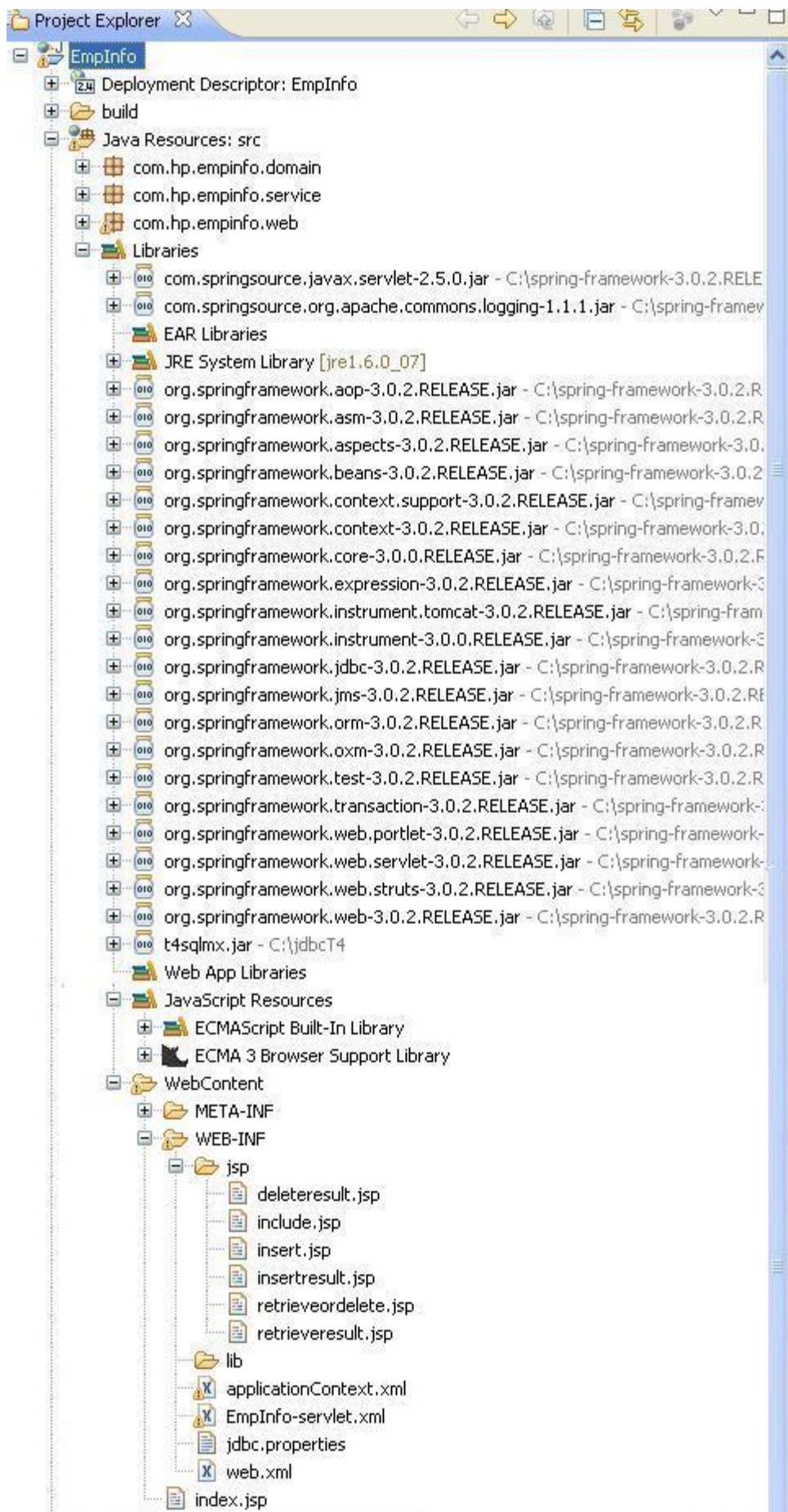
- [“Modifying EmplInfo on Windows using Eclipse Galileo IDE” \(page 368\)](#)
- [“Deploying EmplInfo on NonStop” \(page 379\)](#)
- [“Running EmplInfo on NonStop” \(page 382\)](#)

## Modifying EmplInfo on Windows using Eclipse Galileo IDE

EmplInfo developed in the [“Getting Started with Spring” \(page 75\)](#) chapter is the starting point to demonstrate the integration of Hibernate with Spring applications.

Following is the screen shot of the final structure of the EmplInfo application that was developed in the [“Getting Started with Spring” \(page 75\)](#) chapter:

**Figure 139 Project Explorer View**



Modifying the EmpInfo application involves the following tasks:

- “[Modifying the applicationContext.xml File](#)” (page 370)
- “[Modifying the EmployeeDao.java File](#)” (page 371)
- “[Creating the Employee.hbm.xml File](#)” (page 372)
- “[Removing the EmployeeRowMapper.java File](#)” (page 375)
- “[Adding Dependency JAR Files](#)” (page 376)

## Modifying the applicationContext.xml File

Modify the EmpInfo/WebContent/WEB-INF/applicationContext.xml file, so that it uses the Hibernate SessionFactory to connect to the employee SQL/MX database table using the following steps:

1. Specify the SessionFactory in the applicationContext.xml file as shown:

```
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="mappingResources">
        <list>
            <value>Employee.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">
                org.hibernate.dialect.SqlmxDialect</prop>
```

Example of Integrating Hibernate into Spring 9

```
        <prop key="hibernate.show_sql">true</prop>
    </props>
</property>
<property name="dataSource">
    <ref bean="dataSource"/>
</property>
</bean>
```

2. Modify the bean instance of the EmployeeDao class to add the reference of the Hibernate SessionFactory, as shown:

```
<bean id="empdao" class="com.hp.springapp.service.EmployeeDao">
    <property name="sessionFactory">
        <ref bean="sessionFactory" />
    </property>
</bean>
```

After modification, the applicationContext.xml file should appear as:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName">
            <value>${jdbc.driver}</value>
        </property>
        <property name="url">
            <value>${jdbc.url}</value>
        </property>
        <property name="username">
            <value>${jdbc.user}</value>
```

```

</property>
<property name="password">
    <value>${jdbc.password}</value>
</property>
    <property name="connectionProperties">
<props>
<prop key="catalog">
${jdbc.catalog}
</prop>
<prop key="schema">
${jdbc.schema}
</prop>
</props>
</property>
</bean>
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="mappingResources">
        <list>
            <value>Employee.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.SqLmxDialect</prop>
            <prop key="hibernate.show_sql">true</prop>
        </props>
    </property>
    <property name="dataSource">
        <ref bean="dataSource"/>
    </property>
</bean>
<bean id="empdao" class="com.hp.empinfo.service.EmployeeDao">
    <property name="sessionFactory">
        <ref bean="sessionFactory" />
    </property>
</bean>
<bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
<property name="locations" value="/WEB-INF/jdbc.properties"/>
</bean>
</beans>

```

## Modifying the EmployeeDao.java File

Modify the EmployeeDao.java file (under the com.hp.empinfo.service package), to change the JDBC queries to Hibernate queries for the insert, delete, and retrieve operations, using the following steps:

1. Instantiate the Hibernate SessionFactory as shown:

```

private SessionFactory sessionFactory;

public void setSessionFactory(SessionFactory sessionFactory) {
    this.sessionFactory = sessionFactory;
}

```

2. Add the following import statements to enable the Hibernate SessionFactory:

```

import org.hibernate.SessionFactory;
import org.hibernate.classic.Session;

```

3. Change all the database queries to use the Hibernate SessionFactory.

After modification, the EmployeeDao.java file should appear as:

```

package com.hp.empinfo.service;

import java.sql.SQLException;
import java.util.Iterator;
import java.util.List;
import org.hibernate.SessionFactory;
import org.hibernate.classic.Session;
import com.hp.empinfo.service;

```

```

public class EmployeeDao {
    private SessionFactory sessionFactory;
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }
    public Employee getDetail(int empid) throws SQLException {
        Session session = sessionFactory.openSession();
        List<Employee> l = session.createQuery(
            "from Employee where emp_id=" + empid).list();
        Iterator<Employee> i = l.iterator();
        Employee employee = i.next();
        session.flush();
        return employee;
    }
    public void insertDetail(int empid, String firstname, String lastname,
        int age, String email) throws SQLException {
        Session session = sessionFactory.openSession();
        Employee emp = new Employee();
        emp.setEmpid(empid);
        emp.setFirstname(firstname);
        emp.setLastname(lastname);
        emp.setAge(age);
        emp.setEmail(email);
        session.save(emp);
        session.flush();
    }
    public String deleteEmployee(int empid) {
        Session session = sessionFactory.openSession();
        Employee employee = new Employee();
        employee.setEmpid(empid);
        session.delete(employee);
        session.flush();
        return "Employee deleted";
    }
}

```

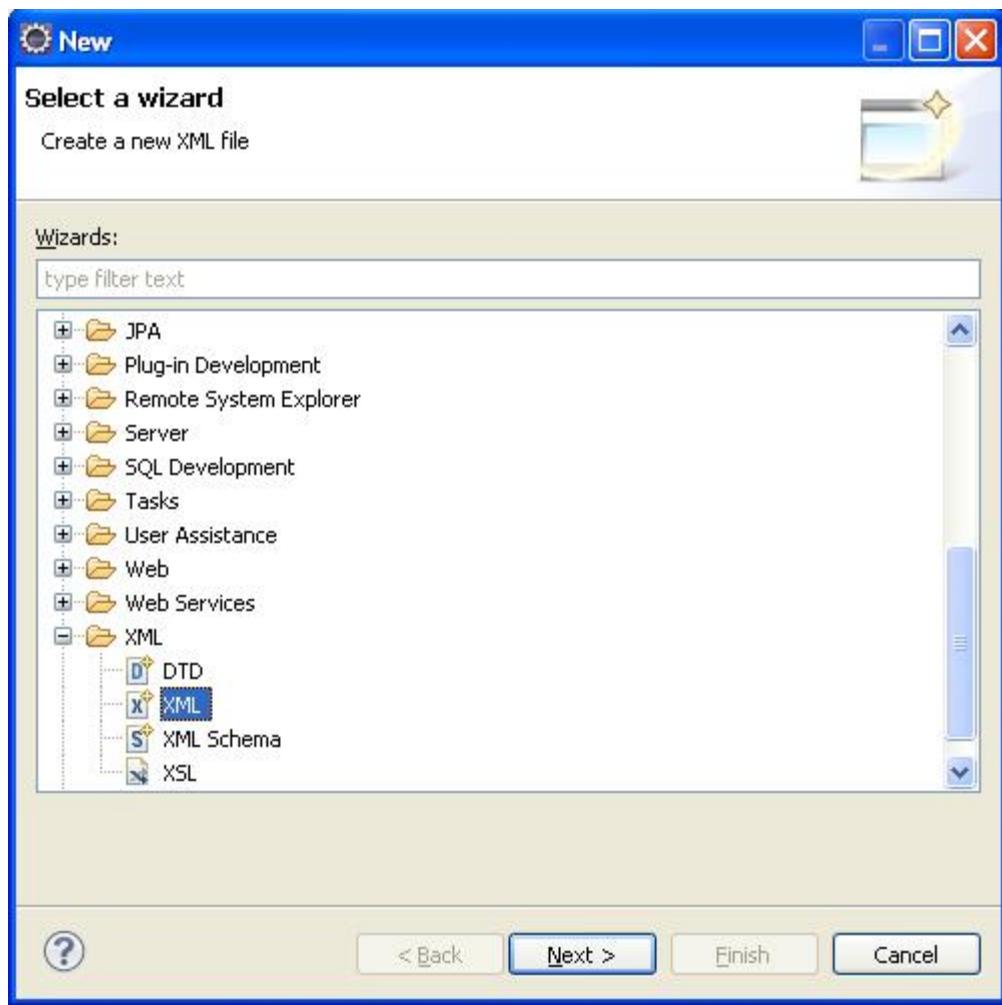
## Creating the Employee.hbm.xml File

Create the Employee.hbm.xml (Hibernate Mapping) file in EmpInfo/src directory to map Java class with the database entities.

To create the Employee.hbm.xml file in the EmpInfo/src directory, complete the following steps:

1. On the Project Explorer frame, right-click **EmpInfo** and select **New > Other**.  
The New File dialog box appears.
2. From the list of folders, select **XML > XML** and click **Next**.

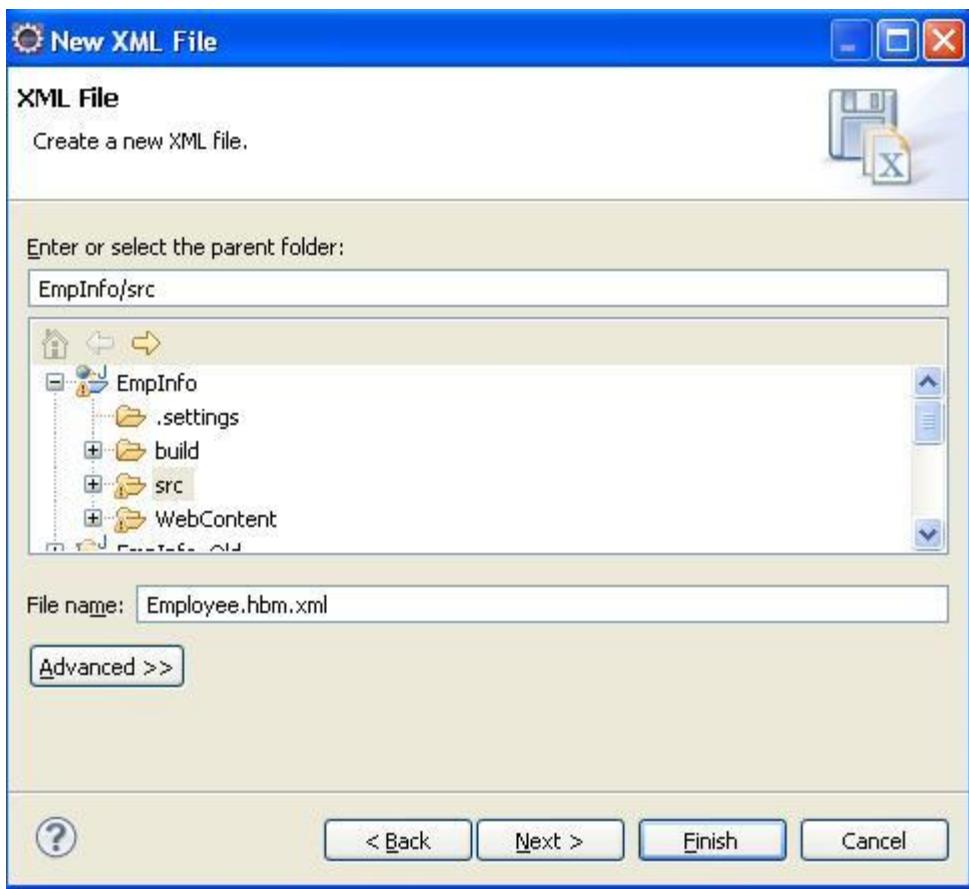
**Figure 140 New File Dialog Box**



The New XML File dialog box appears.

3. In the **File name** field, type `Employee.hbm.xml` and ensure that the parent folder is set to `EmplInfo/src`. Click **Next**.

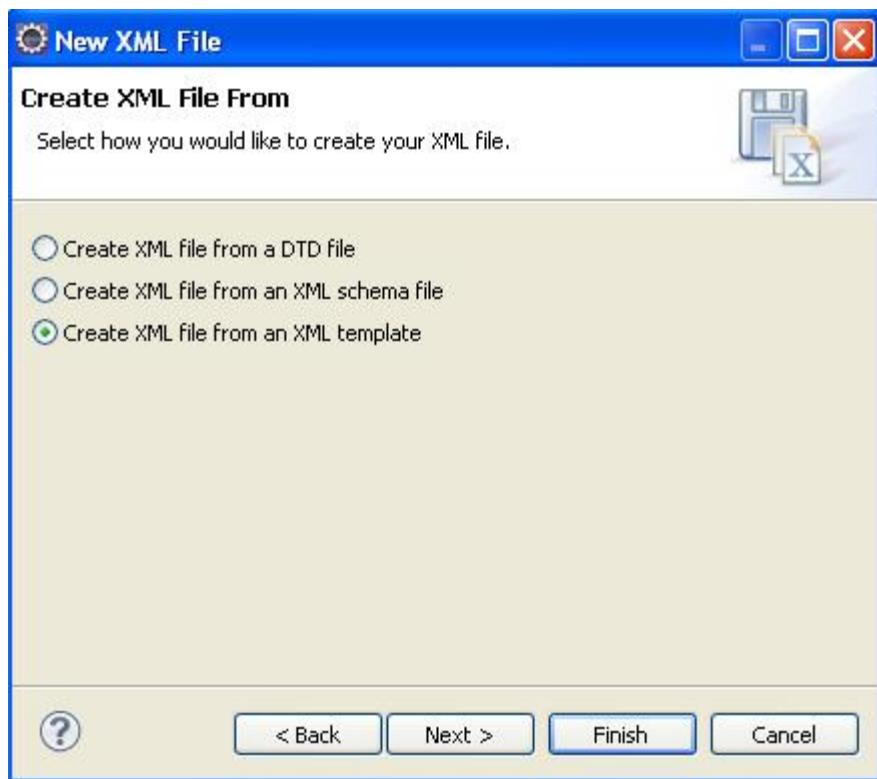
**Figure 141 New XML File Dialog Box**



The New XML File: Create XML dialog box appears.

4. Select **Create XML file from an XML template** and click **Finish**

**Figure 142 New XML File: Create XML Dialog Box**



The Employee.hbm.xml file is created in the EmpInfo/src directory.

5. Modify the Employee.hbm.xml file to add the mapping details. The Employee.hbm.xml file must appear as:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping
package="com.hp.empinfo.domain">
<class name="Employee">
<id name="empid">
<column name="emp_id"/>
</id>
<property name="firstname">
<column name="first_name"/>
</property>
<property name="lastname">
<column name="last_name"/>
</property>
<property name="age">
<column name="age"/>
</property>
<property name="email">
<column name="email"/>
</property>
</class>
</hibernate-mapping>
```

### Removing the EmployeeRowMapper.java File

The EmplInfo application uses the EmployeeRowMapper.java class for mapping various database table entities. Because Hibernate maps database table entities using the Hibernate mapping file

(.hbm file), the EmplInfo application will no longer use the EmployeeRowMapper.java file for mapping purposes. Therefore, you must delete the EmployeeRowMapper.java file from the com.hp.empinfo.domain package.

## Adding Dependency JAR Files

To incorporate the features of Hibernate in the EmplInfo application, the following JAR files, other than the ones added while developing EmplInfo application, must be added in the EmplInfo project library:

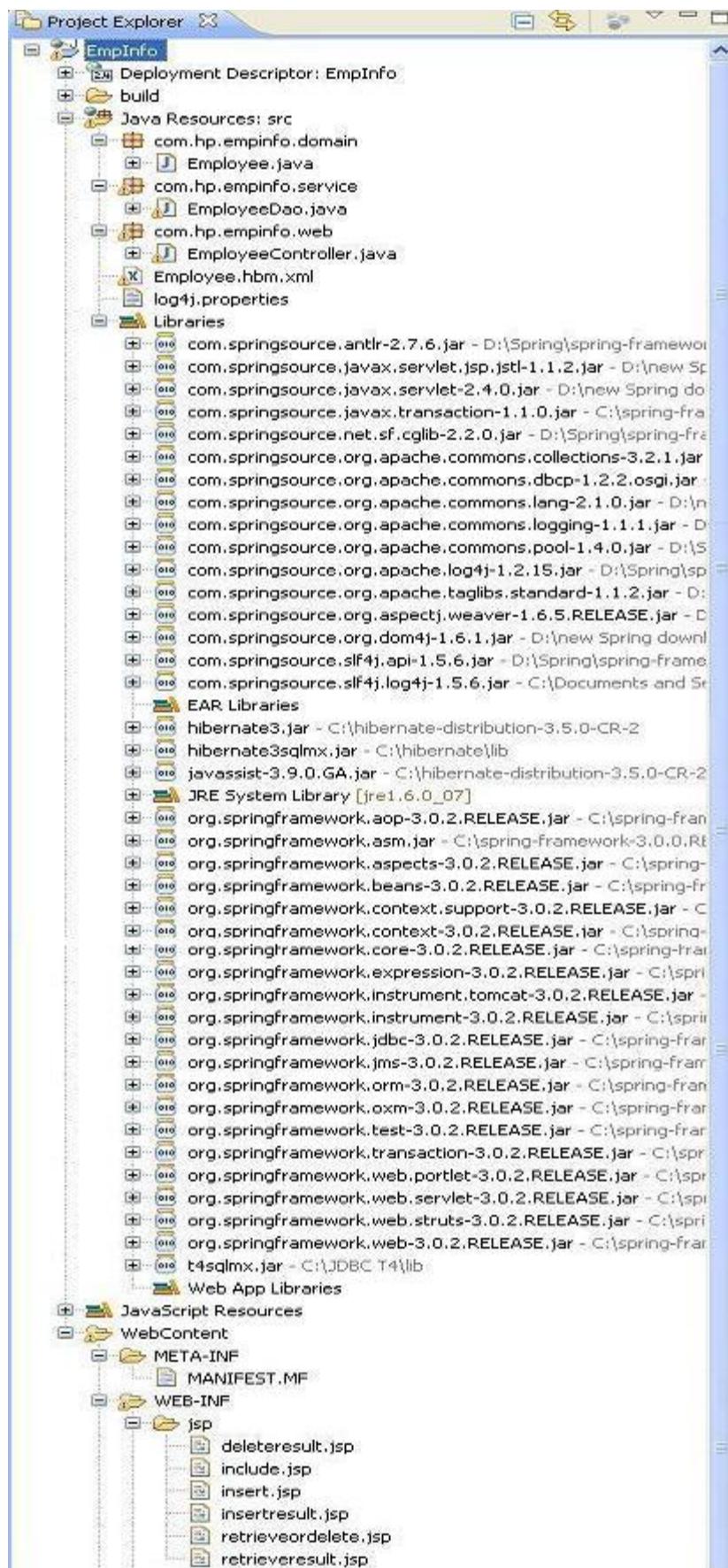
**Table 11 Dependency JAR Files**

Dependency JAR Files	Source Location
com.springsource.antlr-2.7.6.jar	<Spring Dependency Home>\org.antlr\com.springsource.antlr\2.7.6
com.springsource.javax.servlet.jsp.jstl-1.1.2.jar	<Spring Dependency Home>\javax.servlet\com.springsource.javax.jsp.jstl\1.1.2
com.springsource.javax.servlet-2.5.0.jar	<Spring Dependency Home>\javax.servlet\com.springsource.javax.servlet\2.5.0
com.springsource.javax.transaction-1.1.0.jar	<Spring Dependency Home>\javax.transaction\com.springsource.javax.transaction\1.1.0
com.springsource.net.sf.cglib-2.2.0.jar	<Spring Dependency Home>\net.sourceforge.cglib\com.springsource.net.sf.cglib\2.2.0
com.springsource.org.apache.commons.collections-3.2.1.jar	<Spring Dependency Home>\org.apache.commons\com.springsource.org.apache.commons.collections\3.2.1
com.springsource.org.apache.commons.dbcp-1.2.2.osgi.jar	<Spring Dependency Home>\org.apache.commons\com.springsource.org.apache.commons.dbcp\1.2.2.osgi
com.springsource.org.apache.commons.lang-2.1.0.jar	<Spring Dependency Home>\org.apache.commons\com.springsource.org.apache.commons.lang\2.1.0
com.springsource.org.apache.commons.logging-1.1.1.jar	<Spring Dependency Home>\org.apache.commons\com.springsource.org.apache.commons.logging\1.1.1
com.springsource.org.apache.commons.pool-1.5.3.jar	<Spring Dependency Home>\org.apache.commons\com.springsource.org.apache.commons.pool\1.5.3
com.springsource.org.apache.log4j-1.2.15.jar	<Spring Dependency Home>\org.apache.log4j\com.springsource.org.apache.log4j\1.2.15
com.springsource.org.aspectj.weaver-1.6.5.RELEASE.jar	<Spring Dependency Home>\org.aspectj\com.springsource.org.aspectj.weaver\1.6.5.RELEASE
com.springsource.org.dom4j-1.6.1.jar	<Spring Dependency Home>\org.dom4j\com.springsource.org.dom4j\1.6.1
com.springsource.slf4j.api-1.5.6.jar	<Spring Dependency Home>\org.slf4j\com.springsource.slf4j.api\1.5.6
hibernate3.jar	<Hibernate Home>\
hibernate3sqlmx.jar	<Hibernate Home>\lib
javassist-3.9.0.GA.jar	<Hibernate Home>\lib\required
com.springsource.slf4j.log4j-1.5.6.jar	<Hibernate Repository Home>\org\slf4j\com.springsource.slf4j.log4j\1.5.6

To add the dependency JAR files in the project library path and to resolve the J2EE module dependency on these JAR files, follow the instructions in the [Adding Dependency JAR Files in the Project Library Path](#) section of the “[Getting Started with Spring](#)” (page 75) chapter.

The EmplInfo application is now modified to integrate Hibernate into a Spring application (EmplInfo) to handle database operations.

**Figure 143 EmplInfo: Project Explorer View**



## Deploying EmplInfo on NonStop

This section describes the following tasks:

1. “Creating the EmplInfo WAR File on Windows” (page 379)
2. “Deploying the EmplInfo WAR File in NSJSP on NonStop” (page 380)

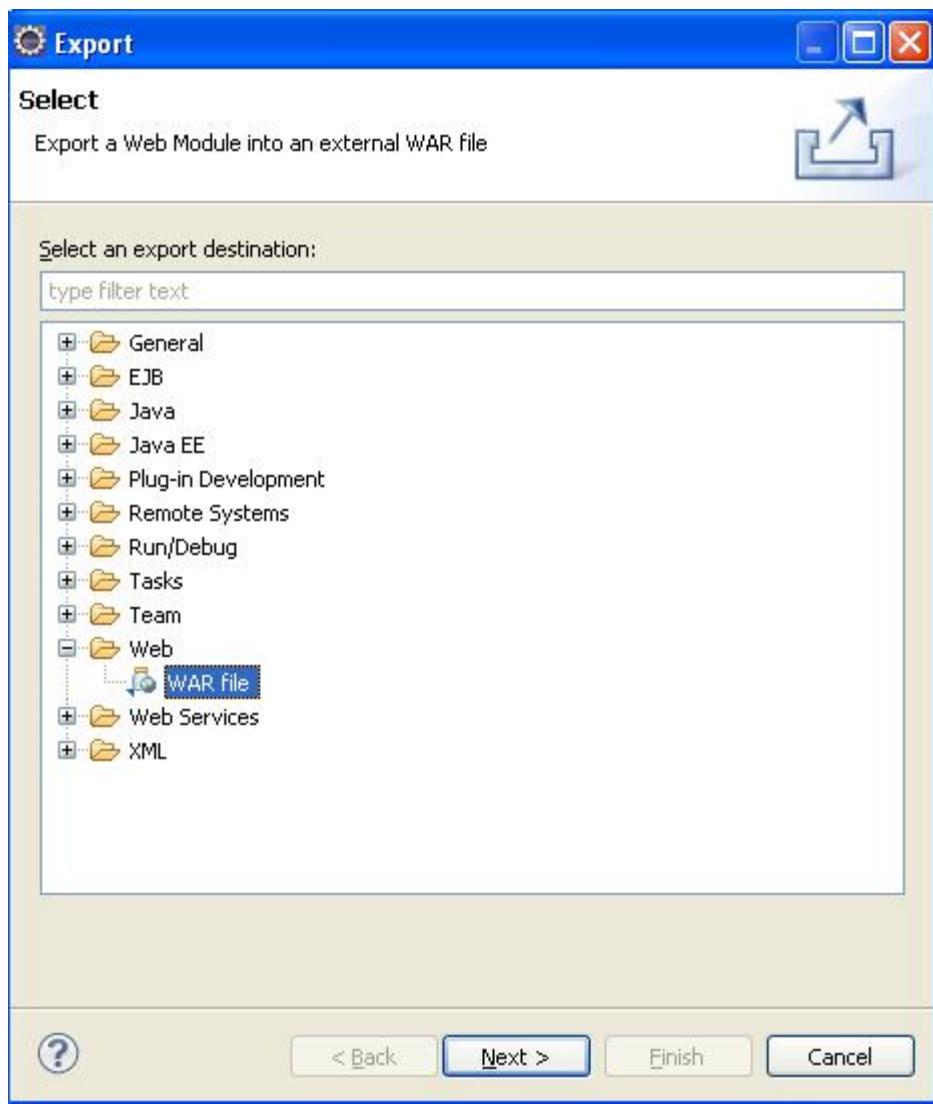
### Creating the EmplInfo WAR File on Windows

A WAR file is essential to deploy the web application. It contains property and configuration files, Java class file, and JSPs of the web application.

To create the application WAR file, complete the following steps:

1. On the Project Explorer frame, right-click **EmplInfo** and select **Export > Export**.  
The Export dialog box appears.
2. From the list of folders, select **Web > WAR file** and click **Next**.

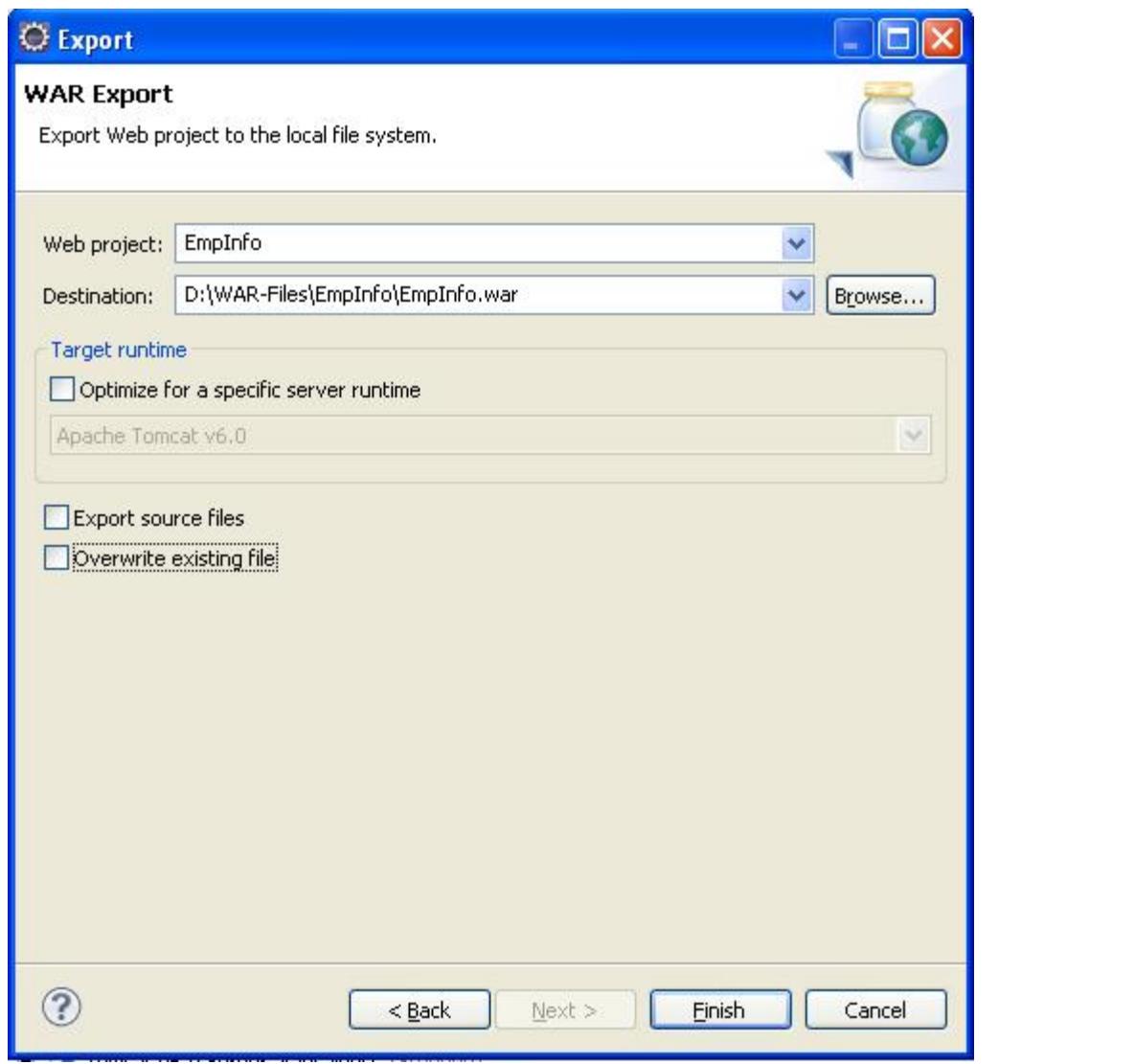
**Figure 144 Export Dialog Box**



The WAR Export dialog box appears.

3. In the **Web project** field, type **EmplInfo** and browse to the destination where you want to save the WAR file.

**Figure 145 The WAR Export dialog box**



4. Click **Finish**. The WAR file is created.

**NOTE:** If you have specified an existing name for the WAR file, the **Finish** button is disabled. In this case, change the name of the WAR file. If you want use the existing name, select the **Overwrite existing file** check box.

## Deploying the EmplInfo WAR File in NSJSP on NonStop

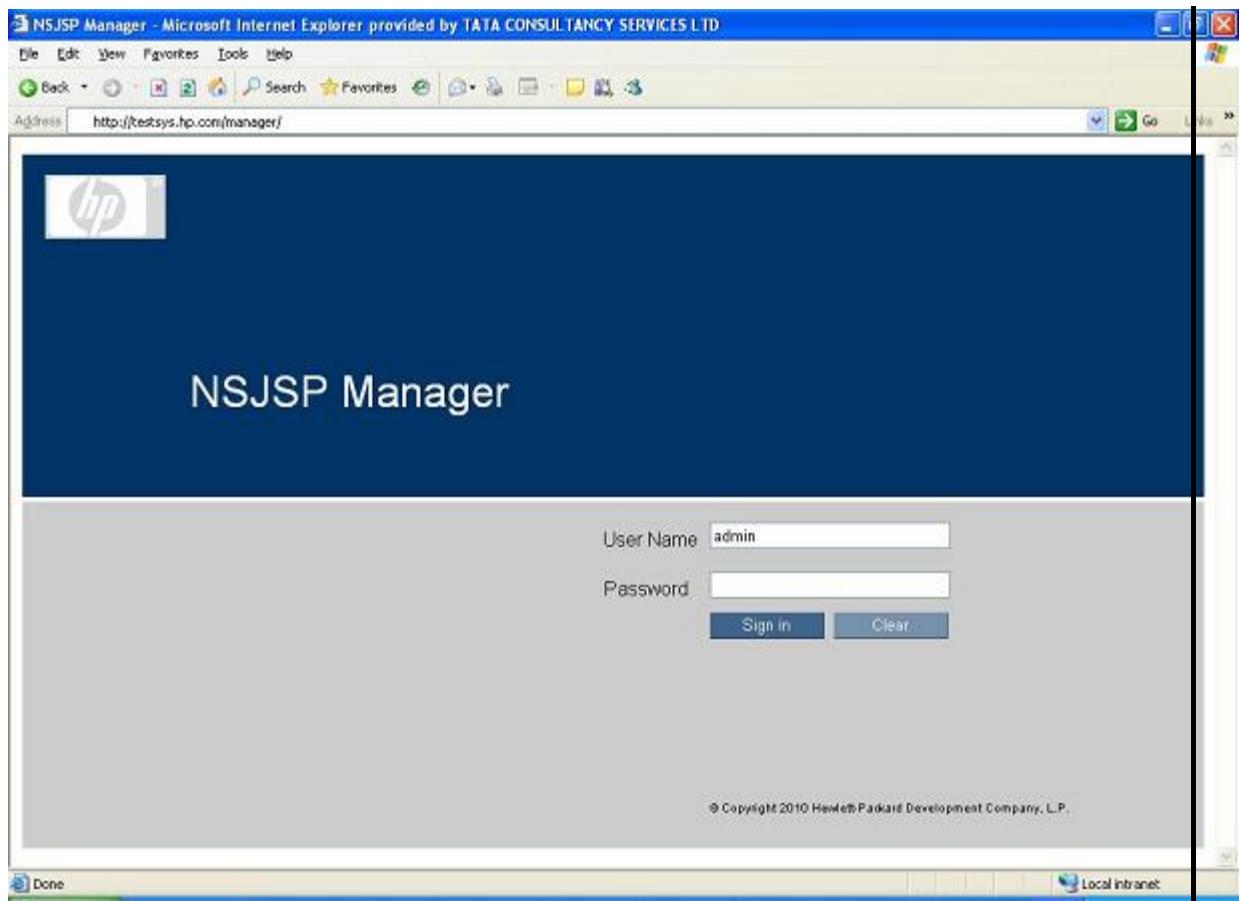
To deploy the EmplInfo WAR file on the NonStop system, complete the following steps:

1. Go to [http://<IP Address of the iTP WebServer>:<port#>/<manager\\_directory>](http://<IP Address of the iTP WebServer>:<port#>/<manager_directory>).

The **NSJSP Manager Login** screen appears.

Figure 146 shows the **NSJSP Manager Login** screen.

**Figure 146 NSJSP Manager Login Screen**

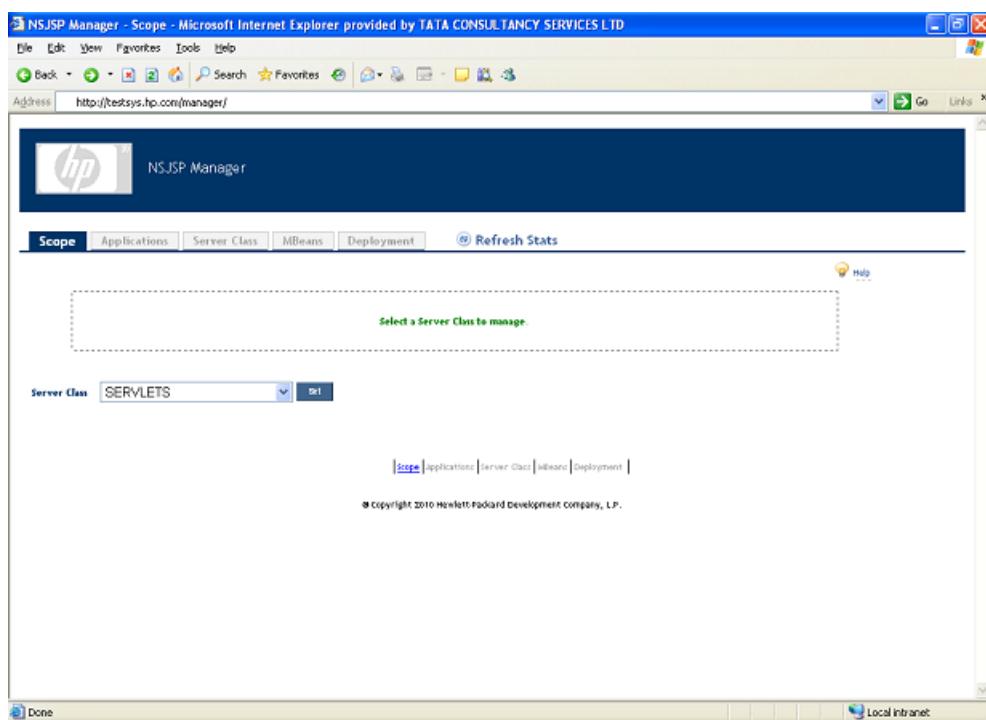


2. Enter the **User name:** and **Password:** and click **OK**.

The **NSJSP Manager** screen appears.

Figure 147 shows the **NSJSP Manager** screen.

**Figure 147 NSJSP Manager Screen**

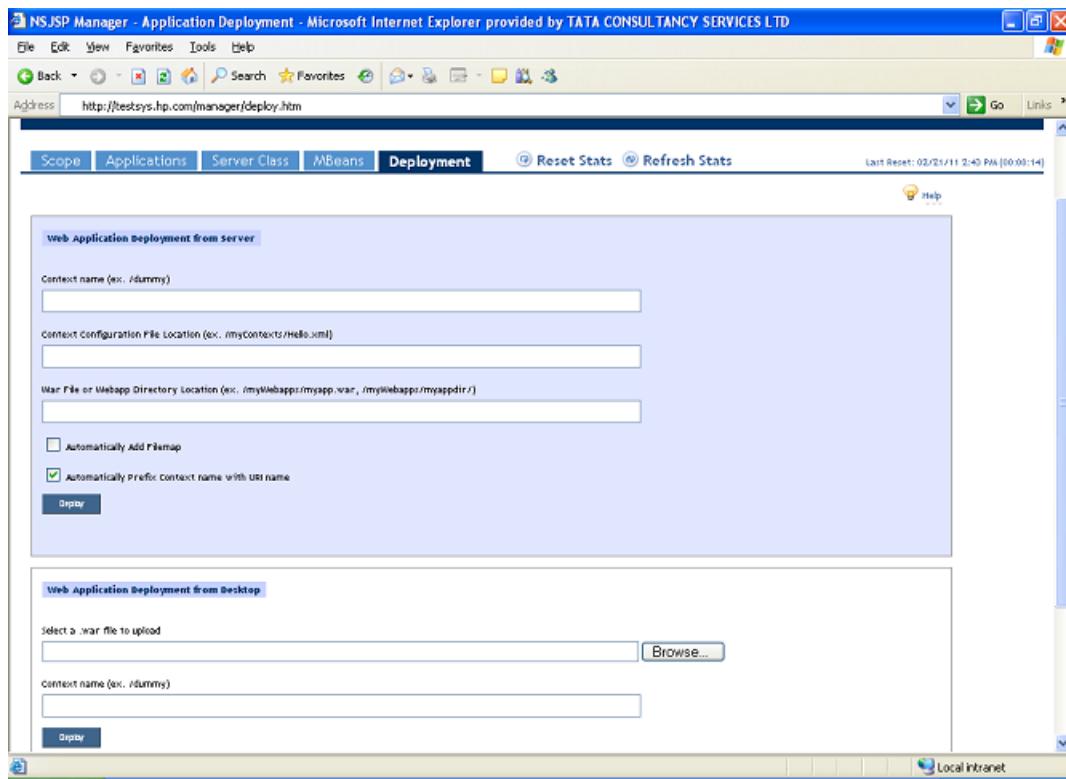


3. Select **SERVLETS** for the **Server Class** and click **Set**.

The **Host (in server.xml)** tab is enabled and populated with **localhost**. Click **Set**, which enables the **Deployment** tab on the **NSJSP Manager** screen.

Figure 148 shows the **Deployment** tab of **NSJSP Manager**.

**Figure 148 NSJSP Manager Screen - Deployment tab**



4. In the **Deployment** tab, complete the following steps in the **Web Application Deployment from Desktop** section:

1. In the **Select a .war file to upload** field, click **Browse...** and locate the `EmplInfo.war` file on the Windows system.
2. **(Optional)** In the **Context name** field, enter a name for the application context.
3. Click **Deploy**.

`EmplInfo` is deployed on NSJSP and is listed under **Applications** tab of the **NSJSP Manager** screen.

**NOTE:** HP recommends that you use the NSJSP manager for deployment. Deployment using the FTP or any other mechanism is not recommended. For more information on using the NSJSP manager, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

## Running EmplInfo on NonStop

To run `EmplInfo` on the NonStop system, click `/<servlet directory>/EmplInfo` path under **Applications** in the **NSJSP Manager** screen.

# 20 Integrating JPA with Hibernate into Spring

This chapter describes how a Spring application can use the Java Persistence API (JPA) with Hibernate for its database operation. This is demonstrated by modifying the EmplInfo application (developed in “[Getting Started with Spring](#)” (page 75) chapter) using the Eclipse Galileo IDE.

The following topics are discussed in this section:

- “[Why Integrate JPA with Hibernate in Spring](#)” (page 383)
- “[Example of Integrating JPA with Hibernate into Spring](#)” (page 383)

## Why Integrate JPA with Hibernate in Spring

JPA combines the best features from each of the persistence mechanisms, such as serialization, JDBC, Java Data Objects (JDO), proprietary object-relational mapping tools, object databases, and EJB 2.x entity beans.

JPA enables you to:

- store and retrieve persistent data
- use advanced object-oriented concepts such as inheritance
- avoid vendor lock-in
- focus on relational databases

Spring provides the complete EJB container contract for JPA, allowing JPA to be used under a Spring managed services layer (with all the AOP and DI richness of Spring) in any environment. Because Hibernate is compliant with JPA, its features can also be leveraged within the Spring JPA implementation.

## Example of Integrating JPA with Hibernate into Spring

The EmplInfo application (developed in the “[Integrating Hibernate into Spring](#)” (page 367) chapter) is a Spring application that uses Hibernate for handling database operations.

This section describes how to use JPA with Hibernate for handling the database operations of the EmplInfo application.

### Prerequisites

The prerequisites of integrating JPA with Hibernate in Spring are:

- Basic knowledge of the Spring and Hibernate frameworks.
- EmplInfo application (explained in the “[Integrating Hibernate into Spring](#)” (page 367) chapter) developed on the Windows system.
- Following software installed on the NonStop system:
  - NonStop iTP WebServer version T8996H02 or later
  - NSJSP version T1222H60 or later
  - JDBC Type 2 driver version T1275H50 or later, or JDBC Type 4 driver version T1249V11 or later

- NonStop SQL/MX version T1050H23 or later
- NSJ version T2766H60 or later
- Following software installed on the Windows system:
  - JDK version 1.5 or later
  - JDBC Type 4 driver version T1249V11 or later
  - Eclipse Galileo IDE version 3.3.1.1 or later

## Activities involved

Integrating the EmplInfo Spring application with JPA and Hibernate, involves the following tasks:

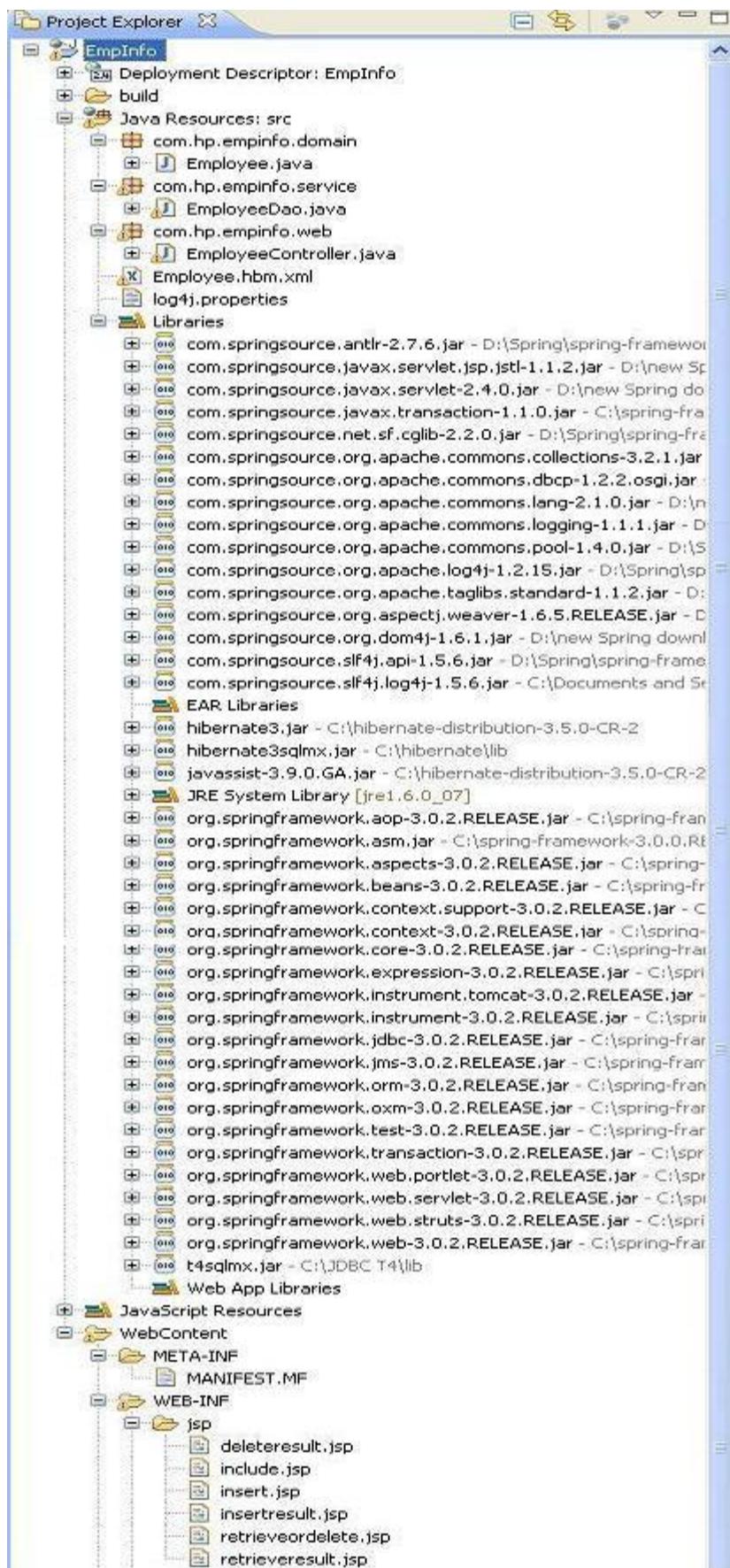
- “[Modifying EmplInfo on Windows using Eclipse Galileo IDE](#)” (page 384)
- “[Deploying EmplInfo on NonStop](#)” (page 394)
- “[Running EmplInfo on NonStop](#)” (page 397)

## Modifying EmplInfo on Windows using Eclipse Galileo IDE

The EmplInfo application developed in the “[Integrating Hibernate into Spring](#)” (page 367) chapter is the starting point to demonstrate the integration of JPA with Hibernate in Spring.

Following is the screen shot of the final structure of the EmplInfo application that was developed in the “[Integrating Hibernate into Spring](#)” (page 367) chapter.

**Figure 149 Project Explorer View**



Modifying the EmpInfo application involves the following activities:

1. "Creating the persistence.xml File" (page 386)
2. "Creating the EntityEmployee.java File" (page 386)
3. "Modifying the applicationContext.xml File" (page 388)
4. "Modifying the EmployeeDao.java File" (page 389)
5. "Removing the Employee.hbm.xml File" (page 391)
6. "Adding Dependency JAR Files" (page 391)

## Creating the persistence.xml File

Create the persistence.xml file to define a persistence unit called Empinfo to define a set of classes and their mapping characteristics.

To create persistence.xml, complete the following steps:

1. Create a folder named **META-INF** in the EmpInfo/src directory, as explained in the "Configuring the JSP Standard Tag Library" (page 95) section in the "Getting Started with Spring" (page 75) chapter.
2. Create a new XML file persistence.xml in the EmpInfo/src/META-INF directory, as explained in the "Creating the EmpInfo-servlet.xml File" (page 86) section in the "Getting Started with Spring" (page 75) chapter.
3. Modify the persistence.xml file to provide the persistence unit named Empinfo and Hibernate-related configuration.

After modification, the persistence.xml file appears as:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
  version="1.0">

  <persistence-unit name="Empinfo">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>com.hp.empinfo.domain.EntityEmployee</class>
    <properties>
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.Oracle10gDialect" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true" />
      <property name="hibernate.jdbc.batch_size" value="100" />
    </properties>
  </persistence-unit>
</persistence>
```

## Creating the EntityEmployee.java File

EntityEmployee.java is a JPA Persistence class. This class is used to map the database entities using JPA annotations such as @Entity, @Id, and @Basic.

To create the EntityEmployee.java class under the com.hp.empinfo.domain package, complete the following steps:

1. Create the Java class EntityEmployee.java, as explained in the "Creating the Controller for EmpInfo" (page 91) section in the "Getting Started with Spring" (page 75) chapter.
2. Modify the EntityEmployee.java class to add the application properties and their getter and setter methods to map with the Employee database table.

After modification, the EntityEmployee.java appears as:

```
package com.hp.empinfo.domain;

import javax.persistence.Basic;
import javax.persistence.Entity;
import javax.persistence.Id;
```

```

import javax.persistence.Table;

@Entity
@Table(name = "Employee")
public class EntityEmployee {
    @Id
    private int emp_id;
    @Basic
    private String first_name;
    @Basic
    private String last_name;
    @Basic
    private int age;
    @Basic
    private String email;

    public EntityEmployee() {
        super();
    }
    public EntityEmployee(int emp_id, String first_name, String last_name,
        int age, String email) {
        this.emp_id = emp_id;
        this.first_name = first_name;
        this.last_name = last_name;
        this.age = age;
        this.email = email;
    }
    public int getEmp_id() {
        return emp_id;
    }

    public void setEmp_id(int emp_id) {
        this.emp_id = emp_id;
    }

    public String getFirst_name() {
        return first_name;
    }

    public void setFirst_name(String first_name) {
        this.first_name = first_name;
    }

    public String getLast_name() {
        return last_name;
    }

    public void setLast_name(String last_name) {
        this.last_name = last_name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}

```

```
    }  
}
```

## Modifying the applicationContext.xml File

Modify the EmpInfo/WebContent/WEB-INF/applicationContext.xml file, so that it uses JPA EntityManagerFactory to connect to the employee SQL/MX database table, in place of Hibernate SessionFactory.

To modify the applicationContext.xml file, complete the following steps:

1. Specify the EntityManagerFactory in the applicationContext.xml file as shown:

```
<bean id="entityManagerFactory"  
      class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">  
  <property name="dataSource" ref="dataSource" />  
  <property name="persistenceUnitName" value="Empinfo" />  
</bean>
```

2. Remove the `<bean>` tag with the sessionFactory ID from the applicationContext.xml file.

```
<bean id="sessionFactory"  
      class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">  
  <property name="mappingResources">  
    <list>  
      <value>Employee.hbm.xml</value>  
    </list>  
  </property>  
  <property name="hibernateProperties">  
    <props>  
      <prop key="hibernate.dialect">org.hibernate.dialect.SqlmxDialect</prop>  
      <prop key="hibernate.show_sql">true</prop>  
    </props>  
  </property>  
  <property name="dataSource">  
    <ref bean="dataSource"/>  
  </property>  
</bean>
```

3. Replace the reference of Hibernate sessionFactory with the JPA entityManagerFactory in the bean instance of the EmployeeDao class, as shown:

```
<bean id="empdao" class="com.hp.springapp.service.EmployeeDao">  
  <property name="entityManagerFactory">  
    <ref bean="entityManagerFactory" />  
  </property>  
</bean>
```

After modification, the applicationContext.xml file should appear as:

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans  
  xmlns="http://www.springframework.org/schema/beans"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation=  
  "http://www.springframework.org/schema/beans  
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">  
  
  <bean id="dataSource"  
        class="org.springframework.jdbc.datasource.DriverManagerDataSource"  
        >  
    <property name="driverClassName">  
      <value>${jdbc.driver}</value>  
    </property>  
    <property name="url">  
      <value>${jdbc.url}</value>  
    </property>
```

```

<property name="username">
    <value>${jdbc.user}</value>
</property>
<property name="password">
    <value>${jdbc.password}</value>
</property>
    <property name="connectionProperties">
        <props>
            <prop key="catalog">
                ${jdbc.catalog}
            </prop>
            <prop key="schema">
                ${jdbc.schema}
            </prop>
        </props>
    </property>
</bean>

<bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="persistenceUnitName" value="Empinfo" />
</bean>

<bean id="empdao" class="com.hp.empinfo.service.EmployeeDao">
    <property name="entityManagerFactory">
        <ref bean="entityManagerFactory" />
    </property>
</bean>

<bean id="propertyConfigurer"
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations" value="/WEB-INF/jdbc.properties"/>
</bean>
</beans>

```

## Modifying the EmployeeDao.java File

Modify the EmployeeDao.java file, to use JPA in place of Hibernate for the insert, delete, and retrieve operations.

To modify the EmployeeDao.java file, complete the following steps:

1. Instantiate the JPA as shown:

```

private EntityManagerFactory entityManagerFactory;

public void setEntityManagerFactory(
    EntityManagerFactory entityManagerFactory) {
    this.entityManagerFactory = entityManagerFactory;
}

```

2. Add the following import statements to enable the JPA EntityManagerFactory and to use the EntityEmployee.java class file.

```

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import com.hp.empinfo.domain.EntityEmployee;

```

3. Remove the following import statements that were used by Hibernate:

```

import org.hibernate.SessionFactory;
import org.hibernate.classic.Session;

```

4. Modify the following methods to use JPA EntityManagerFactory to perform database operations:

- a. public Employee getDetail(int empid) throws SQLException
- b. public void insertDetail(int empid, String firstname, String lastname, int age, String email) throws SQLException
- c. public String deleteEmployee(int empid)

After modification, the EmployeeDao.java file should appear as:

```
package com.hp.empinfo.service;

import java.sql.SQLException;
import java.util.Iterator;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import com.hp.empinfo.domain.EntityEmployee;
import com.hp.empinfo.domain.Employee;

public class EmployeeDao {

    private EntityManagerFactory entityManagerFactory;

    public void setEntityManagerFactory(
        EntityManagerFactory entityManagerFactory) {
        this.entityManagerFactory = entityManagerFactory;
    }

    public Employee getDetail(int empid) throws SQLException {
        EntityManager em = entityManagerFactory.createEntityManager();
        EntityTransaction t = em.getTransaction();
        t.begin();
        List<EntityEmployee> l = em.createQuery(
            "from EntityEmployee where emp_id=" + empid).getResultList();
        Iterator<EntityEmployee> i = l.iterator();
        EntityEmployee entityEmployee = i.next();

        Employee employee = new Employee();
        employee.setEmpid(entityEmployee.getEmp_id());
        employee.setFirstname(entityEmployee.getFirst_name());
        employee.setLastname(entityEmployee.getLast_name());
        employee.setAge(entityEmployee.getAge());
        employee.setEmail(entityEmployee.getEmail());
        em.flush();
        t.commit();
        return employee;
    }

    public void insertDetail(int empid, String firstname, String lastname,
                           int age, String email) throws SQLException {

        EntityManager em = entityManagerFactory.createEntityManager();
        EntityTransaction t = em.getTransaction();
        t.begin();
        EntityEmployee entityEmployee = new EntityEmployee(empid, firstname,
            lastname, age, email);
        em.persist(entityEmployee);
        em.flush();
        t.commit();
    }

    public String deleteEmployee(int empid) {

        EntityManager em = entityManagerFactory.createEntityManager();
        EntityTransaction t = em.getTransaction();
        t.begin();
```

```

        EntityEmployee entityEmployee = em.find(EntityEmployee.class,
            new Integer(empid));
        em.remove(entityEmployee);
        em.flush();
        t.commit();
        return "Employee deleted";
    }
}

```

## Removing the Employee.hbm.xml File

JPA annotations (`EntityEmployee.java` class) have been used to map the database entities. Hence, the Hibernate Mapping file (`Employee.hbm.xml`) is no longer needed and is removed.

## Adding Dependency JAR Files

Adding the dependency JAR files involves the following tasks:

1. “Downloading Hibernate Entity Manager” (page 391)
2. “Adding Dependency JAR Files” (page 391)

### Downloading Hibernate Entity Manager

To use JPA with Hibernate, the `javassist.jar` and `ejb3-persistence.jar` dependency JAR files present in the Hibernate Entity Manager package must be downloaded.

To download the Hibernate Entity Manager, complete the following steps:

1. Go to [http://sourceforge.net/project/showfiles.php?group\\_id=40712&package\\_id=156160](http://sourceforge.net/project/showfiles.php?group_id=40712&package_id=156160)  
A page displaying a list of Hibernate Entity Manager Releases appears.
2. Click **3.2.0.GA** Release.
3. Click `hibernate-entitymanager-3.2.0.GA.zip` and download it to a location on the Windows system.
4. After the download is complete, unzip the `hibernate-entitymanager-3.2.0.GA.zip` file to a location on the Windows system.

For example, this location could be `<Hibernate Entity Manager Home>`.

## Adding Dependency JAR Files

To incorporate the features of JPA in the EmplInfo application, the following JAR files, other than the ones added while developing EmplInfo application, must be added in the EmplInfo project library.

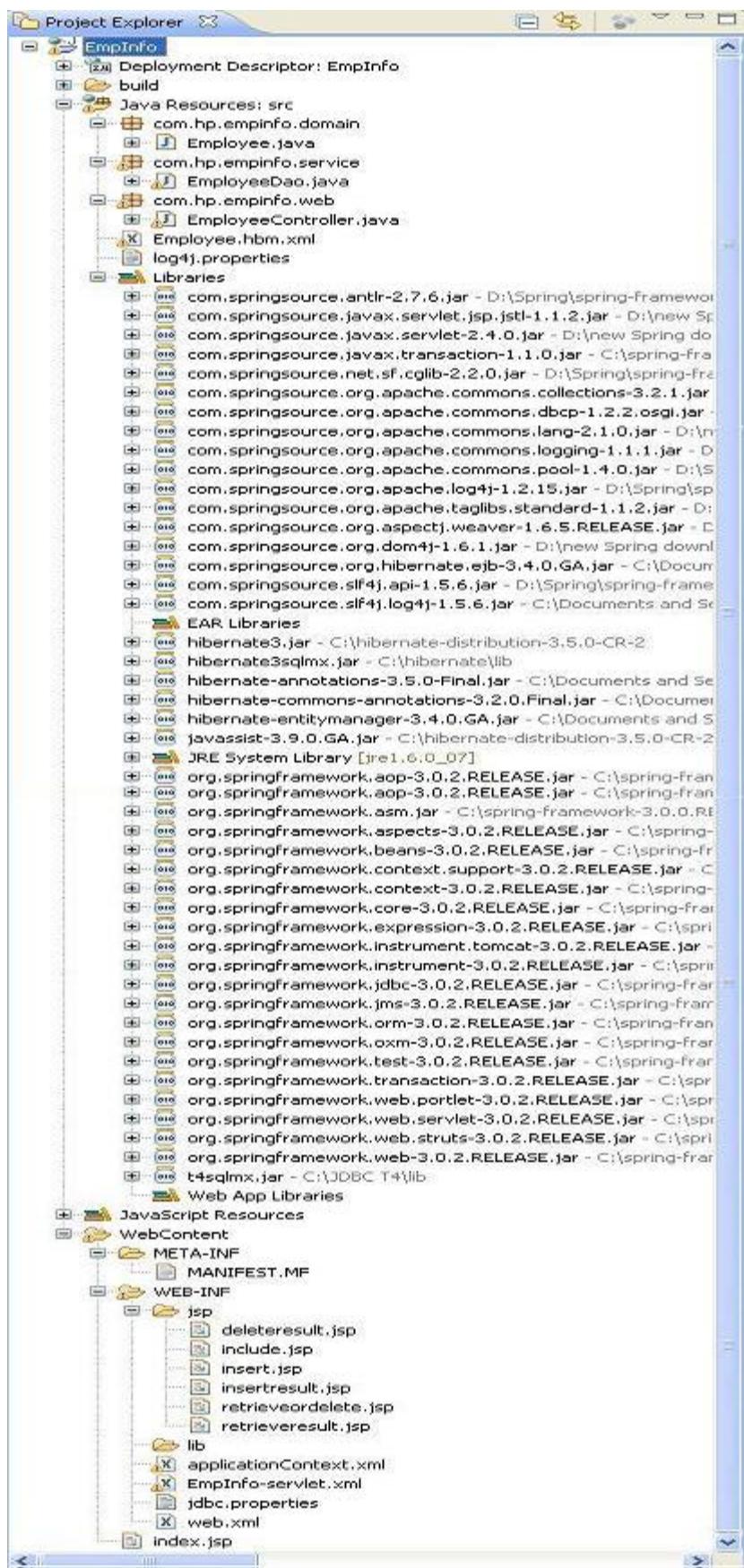
**Table 12 Dependency JAR Files**

Dependency JAR Files	Source Location
<code>com.springsource.org.hibernate.ejb-3.4.0.GA.jar</code>	<code>&lt;Hibernate Repository Home&gt;\org\hibernate\com.springsource.org.hibernate.ejb\3.4.0.GA</code>
<code>hibernate-jpa-2.0-api-1.0.0.Final.jar</code>	<code>&lt;Hibernate Repository Home&gt;\javax\persistence\hibernate-jpa-2.0-api\1.0.0.Final</code>
<code>com.springsource javax.persistence-1.0.0.jar</code>	<code>&lt;Hibernate Repository Home&gt;\javax\persistence\com.springsource.javax.persistence\1.0.0</code>
<code>hibernate-entitymanager-3.4.0.GA.jar</code>	<code>&lt;Hibernate Repository Home&gt;\org\hibernate\hibernate-entitymanager\3.4.0.GA</code>

To add the dependency JAR files in the project library path and to resolve the J2EE module dependency on these JAR files, follow the instructions in the “[Adding Dependency JAR Files in the Project Library Path](#)” (page 89) section in the “[Getting Started with Spring](#)” (page 75) chapter.

The EmplInfo application is now modified to integrate JPA with Hibernate into a Spring application.

**Figure 150 Project Explorer View**



## Deploying EmplInfo on NonStop

This section describes the following tasks:

1. “Creating the EmplInfo WAR File on Windows” (page 394)
2. “Deploying the EmplInfo WAR File in NSJSP on NonStop” (page 395)

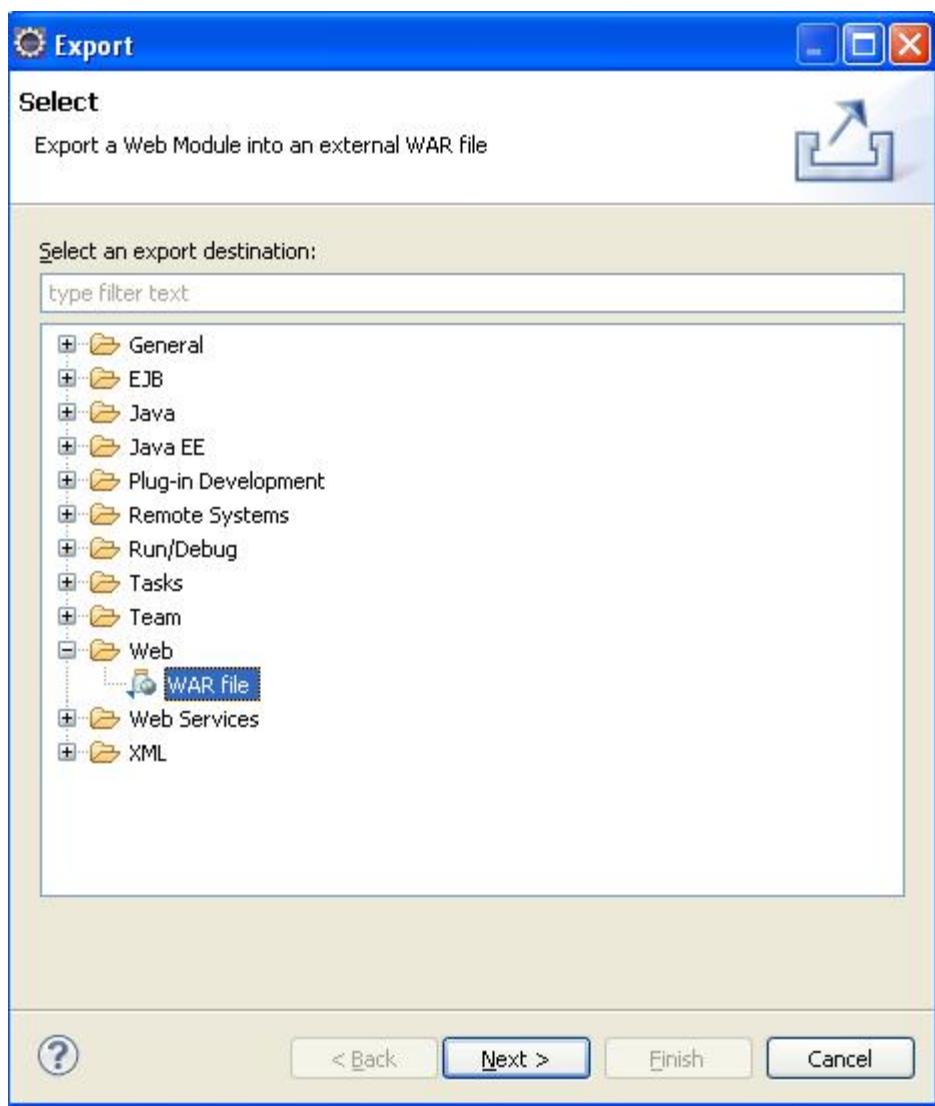
### Creating the EmplInfo WAR File on Windows

A WAR file is essential to deploy the web application. It contains property and configuration files, Java class file, and JSPs of the web application.

To create the application WAR file, complete the following steps:

1. On the Project Explorer frame, right-click **EmplInfo** and select **Export > Export**.  
The Export dialog box appears.
2. From the list of folders, select **Web > WAR file** and click **Next**.

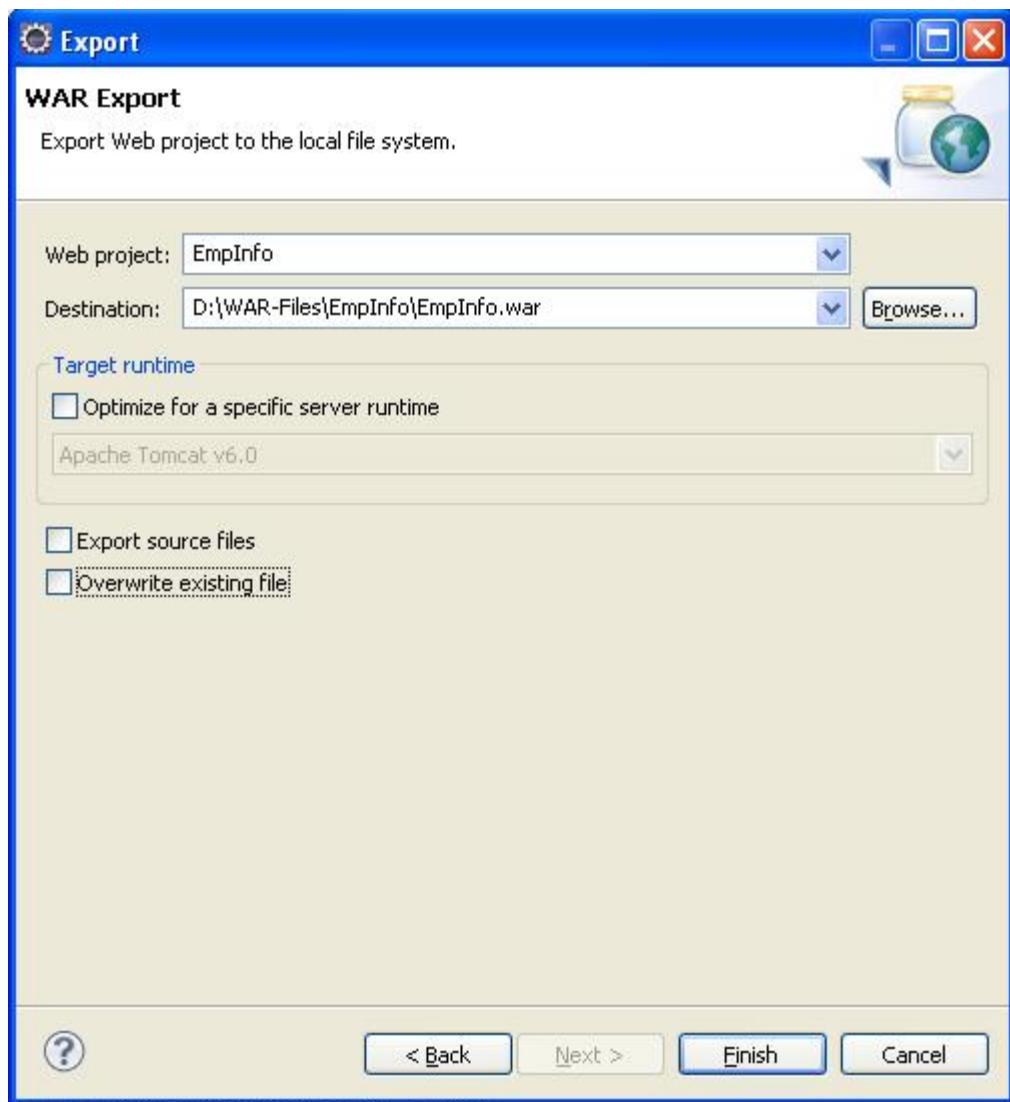
**Figure 151 Export Dialog Box**



The WAR Export dialog box appears.

3. In the **Web project** field, type **EmplInfo** and browse to the destination where you want to save the WAR file.

**Figure 152 The WAR Export dialog box**



4. Click **Finish**. The WAR file is created.

**NOTE:** If you have specified an existing name for the WAR file, the **Finish** button is disabled. In this case, change the name of the WAR file. If you want use the existing name, select the **Overwrite existing file** check box.

## Deploying the EmpInfo WAR File in NSJSP on NonStop

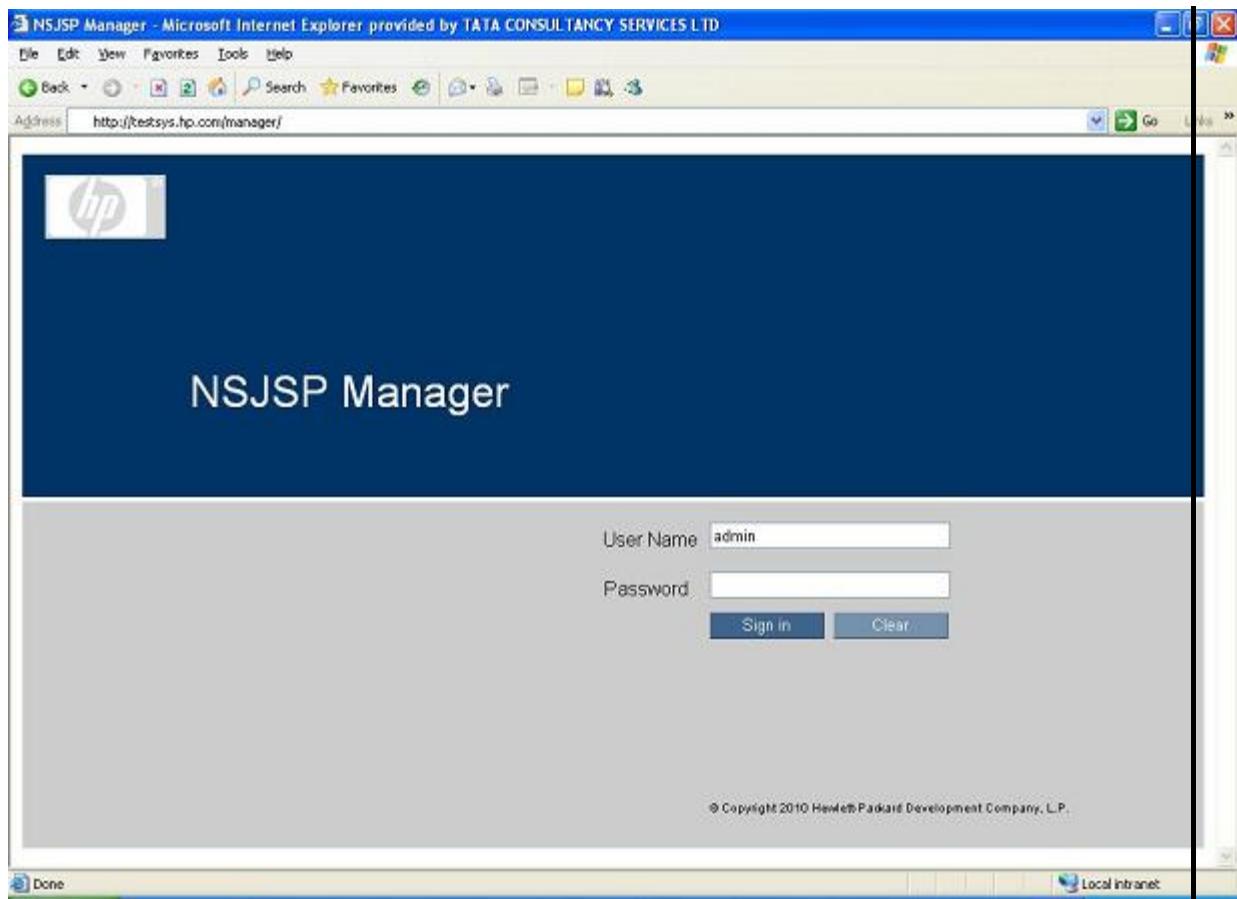
To deploy the EmpInfo WAR file on the NonStop system, complete the following steps:

1. Go to [http://<IP Address of the iTP WebServer>:<port#>/<manager\\_directory>](http://<IP Address of the iTP WebServer>:<port#>/<manager_directory>).

The **NSJSP Manager Login** screen appears.

Figure 153 shows the **NSJSP Manager Login** screen.

**Figure 153 NSJSP Manager Login Screen**

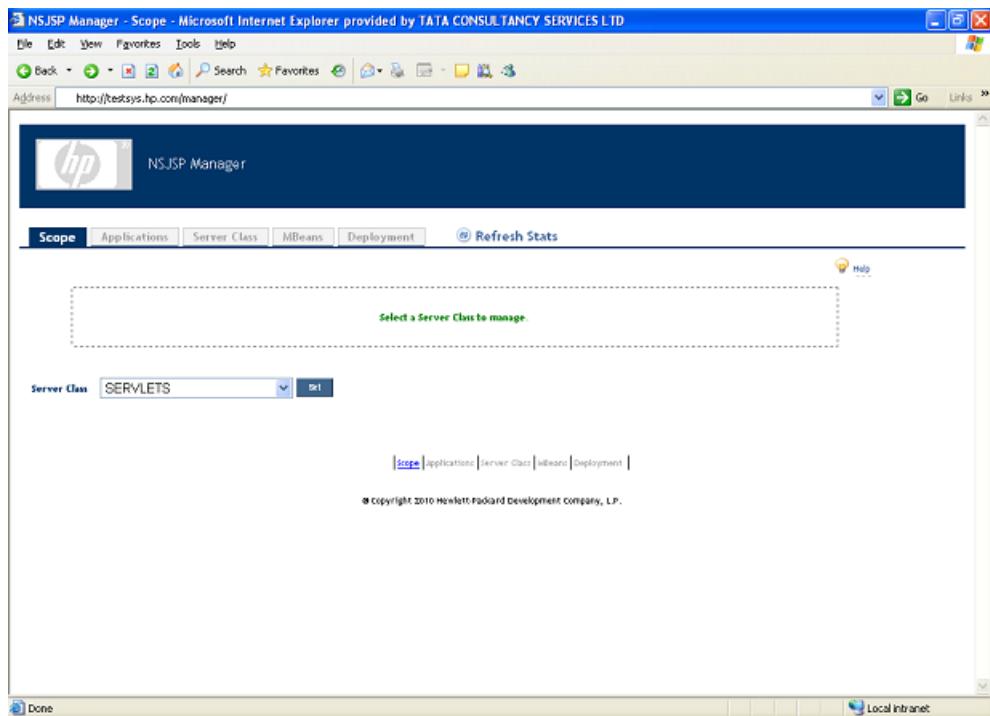


2. Enter the **User name:** and **Password:** and click **OK**.

The **NSJSP Manager** screen appears.

Figure 154 shows the **NSJSP Manager** screen.

**Figure 154 NSJSP Manager Screen**

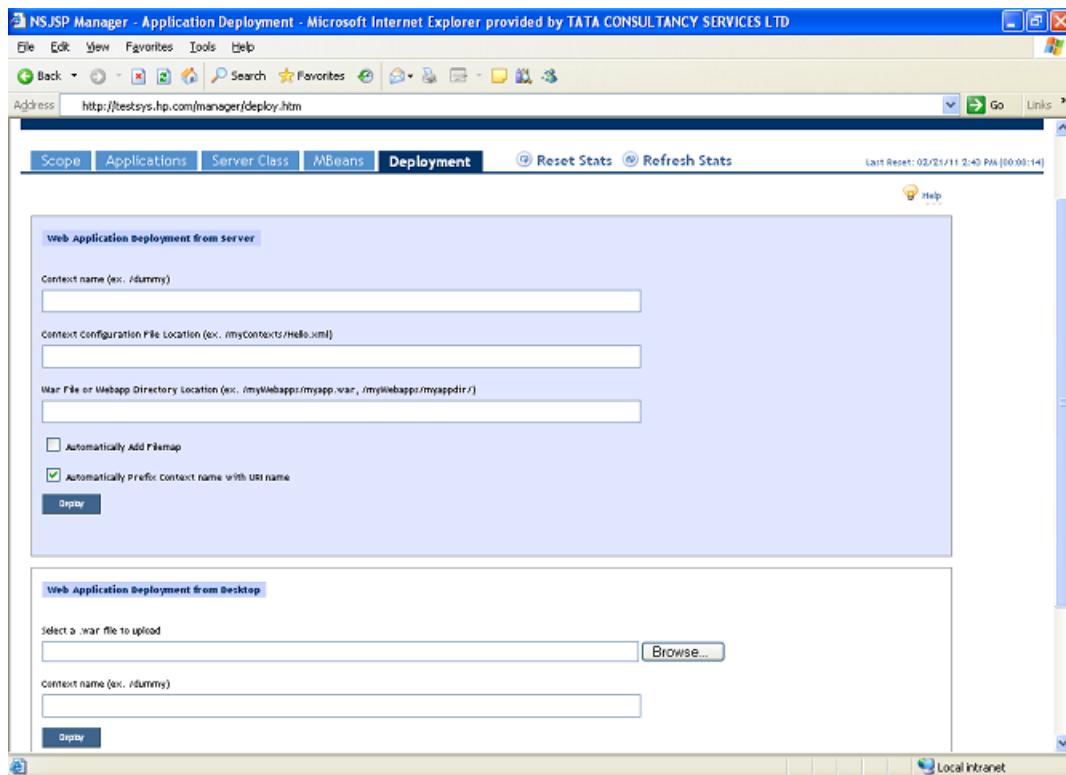


- Select **SERVLETS** for the **Server Class** and click **Set**.

The **Host (in server.xml)** tab is enabled and populated with **localhost**. Click **Set**, which enables the **Deployment** tab on the **NSJSP Manager** screen.

Figure 155 shows the **Deployment** tab of **NSJSP Manager**.

**Figure 155 NSJSP Manager Screen - Deployment tab**



- In the **Deployment** tab, complete the following steps in the **Web Application Deployment from Desktop** section:

- In the **Select a .war file to upload** field, click **Browse...** and locate the `EmplInfo.war` file on the Windows system.
- (Optional) In the **Context name** field, enter a name for the application context.
- Click **Deploy**.

`EmplInfo` is deployed on NSJSP and is listed under **Applications** tab of the **NSJSP Manager** screen.

**NOTE:** HP recommends that you use the NSJSP manager for deployment. Deployment using the FTP or any other mechanism is not recommended. For more information on using the NSJSP manager, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

## Running EmplInfo on NonStop

To run `EmplInfo` on the NonStop system, click `/<servlet directory>/EmplInfo` path under **Applications** in the **NSJSP Manager** screen.

You can now add, search, and delete employee details as explained in the “[Getting Started with Spring](#)” (page 75) chapter.

# 21 Integrating Axis2/Java into Spring

This section describes how a Spring application can use Axis2/Java for exposing itself as a web service. This is demonstrated by modifying the EmplInfo application (developed in the [Getting Started with Spring](#) chapter) using the Eclipse Galileo IDE.

The following topics are discussed in this section:

- [Why Integrate Axis2/Java into Spring](#)
- [Example of Integrating Axis2/Java with Spring](#)

## Why Integrate Axis2/Java into Spring

Using a Simple Object Access Protocol (SOAP) protocol is the best way to exchange information within the applications on the decentralized and distributed environments.

Spring is a lightweight container and flexible MVC Web application framework, and is created to address the complexity of enterprise application development. It is easy to use Axis2/Java to generate the SOAP web service with classes available for configuration and initialization using the Spring framework.

This section describes the development of a Java class to implement various business operations (such as insert, delete, and retrieve of employee details) using the Spring framework. The Axis2/Java framework is used to expose it as a web service, which generates the WSDL files for the implementation.

## Example of Integrating Axis2/Java with Spring

The EmplInfo application (described in the [Getting Started with Spring](#) chapter) is a web application that is deployed directly in NSJSP.

This section describes how to expose the EmplInfo application as a web service running under Axis2/Java.

### Prerequisites

The prerequisites for integrating the EmplInfo application with Axis2/Java are:

- Basic knowledge of the Spring and Axis2/Java frameworks.
- EmplInfo application (explained in the [Getting Started with Spring](#) chapter) developed on your Windows system.
- Following software installed on your NonStop system:
  - NonStop iTP WebServer
  - NonStop Servlets for Java Server Pages (NSJSP)
  - JDBC Type 2 driver for NonStop SQL/MX
  - NonStop SQL/MX
  - NonStop Server for Java (NSJ)
- Following software installed on your Windows system:
  - Java Development Kit (JDK)
  - JDBC Type 4 driver for NonStop SQL/MX
  - Eclipse Galileo IDE

## Activities involved

Integrating the EmplInfo application with Axis2/Java, involves the following activities:

- Modifying EmplInfo on Windows using Eclipse Galileo IDE
- Deploying EmplInfo on NonStop
- Running EmplInfo Web Service on NonStop
- Developing EmplInfoClient on Windows
- Running EmployeeInfo on NonStop

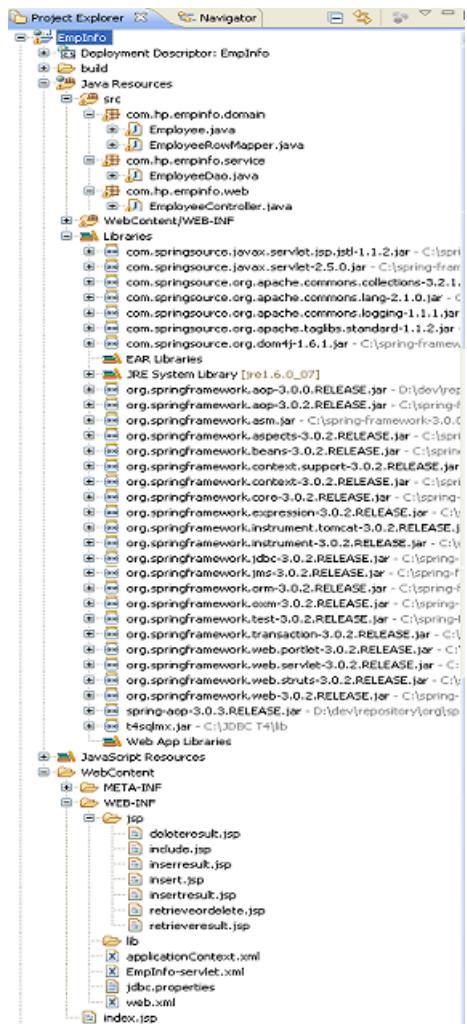
## Modifying EmplInfo on Windows using Eclipse Galileo IDE

The EmplInfo application, developed in the [Getting Started with Spring](#) chapter, is the starting point to demonstrate the integration of Axis2/Java with Spring applications.

Following is the screen shot of the final structure of the EmplInfo application that was developed in the [Getting Started with Spring](#) chapter.

Figure 156 shows the Project Explorer View.

**Figure 156 Project Explorer View**



Modifying the EmplInfo application involves the following activities:

1. Creating the `SpringInit.java` File
2. Creating the `services.xml` File
3. Modifying the `applicationContext.xml` File

- [4. Creating the build.xml File](#)
- [5. Adding Dependency JAR Files](#)

## Creating the SpringInit.java File

Create a Java class (`SpringInit.java`) to initialize the `EmplInfo` application context when the application is exposed under Axis2/Java.

**NOTE:** The `SpringInit.java` is a `ServiceLifeCycle` class and its `startUp` method gets called when it is being loaded. This is required to expose `EmplInfo` as a webservice under Axis2/Java.

To create `SpringInit.java`, complete the following steps:

- [1. Create `SpringInit.java` in the `com.hp.empinfo.service` package, as explained in \[Creating the Controller for EmplInfo\]\(#\) section in \[Getting Started with Spring\]\(#\) chapter.](#)
- [2. Modify the `SpringInit.java` file to include the `startUp` method.](#)

After modification, the `SpringInit.java` file must appear as:

```
package com.hp.empinfo.service;

import org.apache.axiom.om.OMElement;
import org.apache.axis2.engine.ServiceLifeCycle;
import org.apache.axis2.context.ConfigurationContext;
import org.apache.axis2.context.OperationContext;
import org.apache.axis2.context.ServiceContext;
import org.apache.axis2.description.AxisService;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class SpringInit implements ServiceLifeCycle {

    private static Log logger = LogFactory
        .getLog(SpringInit .class);

    // The web service
    public OMElement springInit(OMElement ignore) {
        return null;
    }

    public void init(ServiceContext serviceContext) {
    }

    public void setOperationContext(OperationContext arg0) {
    }

    public void destroy(ServiceContext arg0) {
    }

    /**
     * this will be called during the deployment time of the service. irrespective
     * of the service scope this method will be called
     */
    public void startUp(ConfigurationContext ignore, AxisService service) {
        ClassLoader classLoader = service.getClassLoader();
        ClassPathXmlApplicationContext appCtx = new
            ClassPathXmlApplicationContext(new String[] {"applicationContext.xml"}, false);
        appCtx.setClassLoader(classLoader);
        appCtx.refresh();
        if (logger.isDebugEnabled()) {
            logger.debug("\n\nstartUp() set spring classloader via axisService.getClassLoader() ... ");
        }
    }
    /**
     * this will be called during the deployment time of the service. irrespective
     * of the service scope this method will be called
     */
    public void shutDown(ConfigurationContext ignore, AxisService service) {
    }
}
```

## Creating the services.xml File

Create the Axis2/Java Deployment Descriptor file (services.xml) to instantiate the SpringInit and EmpInfo services.

To create the services.xml file, complete the following steps:

1. Create the services.xml file in the EmpInfo/WebContent/META-INF directory, as explained in the [Creating the EmpInfo-servlet.xml File](#) section in the [Getting Started with Spring](#) chapter.
2. Modify the services.xml file to include the instances SpringInit and EmpInfo services.

After modification, the services.xml file must appear as:

```
<serviceGroup>
    <service name="SpringInit" class="com.hp.empinfo.service.SpringInit">
        <description>
            This web service initializes Spring.
        </description>
        <parameter name="ServiceClass" >com.hp.empinfo.service.SpringInit</parameter>
        <parameter name="ServiceTCCL" >composite</parameter>
        <parameter name="load-on-startup" >true</parameter>
        <operation name="springInit">
            <messageReceiver class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
        </operation>
    </service>
    <service name="EmpInfo">
        <description>
            Spring EmpInfo Axis2 AAR deployment
        </description>
        <parameter name="ServiceClass" >com.hp.empinfo.service.EmployeeDao</parameter>
        <parameter name="ServiceObjectSupplier"
>org.apache.axis2.extensions.spring.receivers.SpringAppContextAwareObjectSupplier</parameter>
        <parameter name="SpringBeanName" >empdao</parameter>
        <operation name="deleteEmployee">
            <messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
        </operation>
        <operation name="getDetail">
            <messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
        </operation>
        <operation name="insertDetail">
            <messageReceiver class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
        </operation>
    </service>
</serviceGroup>
```

## Modifying the applicationContext.xml File

Modify the applicationContext.xml file to:

- Add a bean definition for integrating Axis2/Java in the EmpInfo application
- Edit the JDBC resource location

To modify the applicationContext.xml file, complete the following steps:

1. Specify the applicationContext bean as shown:

```
<bean id="applicationContext"
    class="org.apache.axis2.extensions.spring.receivers.ApplicationContextHolder" />
```

2. Edit the bean instance of the PropertyPlaceholderConfigurer class to change the location of the jdbc.properties file, as shown:

```
<bean id="propertyConfigurer"
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations" value="jdbc.properties"/>
</bean>
```

After modifications, the applicationContext.xml file should appear as:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="applicationContext"
        class="org.apache.axis2.extensions.spring.receivers.ApplicationContextHolder" />
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName">
            <value>${jdbc.driver}</value>
        </property>
```

```

<property name="url">
    <value>${jdbc.url}</value>
</property>
<property name="username">
    <value>${jdbc.username}</value>
</property>
<property name="password">
    <value>${jdbc.password}</value>
</property>
<property name="connectionProperties">
    <props>
        <prop key="catalog">
            ${jdbc.catalog}
        </prop>
        <prop key="schema">
            ${jdbc.schema}
        </prop>
    </props>
</property>
</bean>
<bean id="empdao" class="com.hp.empinfo.service.EmployeeDao">
    <property name="dataSource">
        <ref bean="dataSource" />
    </property>
</bean>
<bean id="propertyConfigurer"
      class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations" value="jdbc.properties"/>
</bean>
</beans>

```

## Creating the build.xml File

Create the build.xml file to build Axis2/Java archive file of EmpInfo application.

To create the build.xml file, complete the following steps:

1. Create the build.xml file in the EmpInfo directory, as explained in the [Creating the EmpInfo-servlet.xml File](#) section in [Getting Started with Spring](#) chapter.
2. Modify the build.xml file to add the mapping details.

After modification, the build.xml file should appear as:

```

<project name="EmpInfo" basedir=". " default="generate.service">

<property environment="env"/>
<property name="service-name" value="EmpInfoService" />
<property name="dest.dir" value="target" />

<property name="axis2.home" value="" />
<property name="spring.home" value="" />

<property name="dest.dir.classes" value="${dest.dir}/classes" />

<path id="build.class.path">
    <fileset dir="${axis2.home}/lib">
        <include name="*.jar" />
    </fileset>
    <fileset dir="${dest.dir.classes}/lib">
        <include name="*.jar" />
    </fileset>
    <fileset dir="${spring.home}/dist">
        <include name="*.jar" />
    </fileset>
</path>

<path id="master-classpath">
    <fileset dir="${dest.dir.classes}/lib">
        <include name="*.jar" />
    </fileset>
</path>

<target name="clean">
    <delete dir="${dest.dir}" />
</target>

<target name="prepare" depends="clean">

    <mkdir dir="${dest.dir}" />
    <mkdir dir="${dest.dir.classes}" />
    <mkdir dir="${dest.dir.classes}/META-INF" />
    <mkdir dir="${dest.dir.classes}/lib"/>

    <antcall target="copy.jars"/>
</target>
```

```

<target name="generate.service" depends="prepare">
    <copy file="WebContent/META-INF/services.xml" tofile="${dest.dir.classes}/META-INF/services.xml"
        overwrite="true" />
    <copy file="WebContent/WEB-INF/applicationContext.xml" tofile="${dest.dir.classes}/applicationContext.xml"
        overwrite="true" />
    <copy file="WebContent/WEB-INF/jdbc.properties" tofile="${dest.dir.classes}/jdbc.properties"
        overwrite="true" />

    <javac debug="on" srcdir="src" destdir="${dest.dir.classes}"
        includes="com/hp/empinfo/domain/**,com/hp/empinfo/service/**">
        <classpath refid="build.class.path" />
    </javac>

    <jar basedir="${dest.dir.classes}" destfile="${dest.dir}/${service-name}.aar" />
</target>

<target name="copy.jars">
    <copy todir="${dest.dir.classes}/lib">
        <fileset dir="${spring.home}/dist/modules">
            <include name="*.jar"/>
        </fileset>
    </copy>
</target>
</project>

```

- 3.** Edit the build.xml file to specify the values of <Axis2 Home> and <Spring Home> as shown:

```

<property name="axis2.home" value="" />
<property name="spring.home" value="" />

```

For example, if <Axis2 Home> is C:\axis2-1.5.2 and <Spring Home> is C:\spring-framework-3.0.0.RELEASE, edit the above lines as shown:

```

<property name="axis2.home" value=" C:\axis2-1.5.2" />
<property name="spring.home" value=" C:\spring-framework-3.0.0.RELEASE"/>

```

## Adding Dependency JAR Files

To expose EmplInfo as an Axis2/Java web service, the following JAR files, other than the ones added while developing EmplInfo application, must be added in the EmplInfo project library:

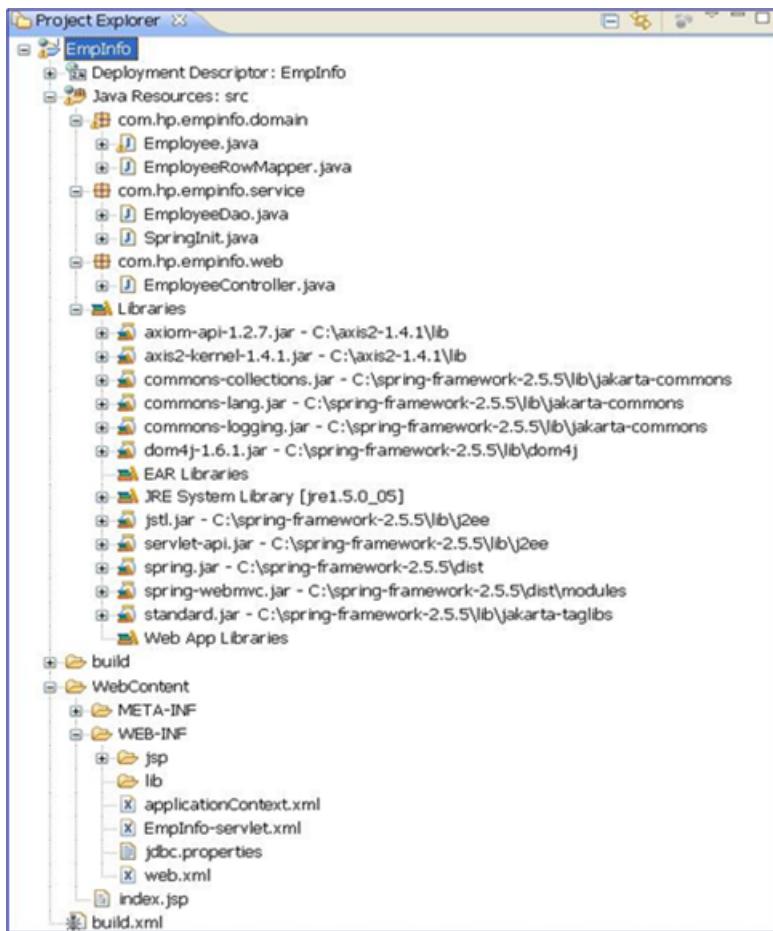
**Table 13 Dependency JAR Files**

Dependency JAR Files	Source Location
axiom-api-1.2.9.jar	<Axis2 Home>\lib
axis2-kernel-1.5.2.jar	<Axis2 Home>\lib

The EmplInfo application is now modified to integrate Axis2/Java.

Figure 157 shows the EmplInfo: Project Explorer View.

**Figure 157 Project Explorer View**



## Deploying EmplInfo on NonStop

This section describes the following tasks:

1. Creating the EmplInfo AAR File on Windows
2. Deploying the EmplInfo AAR File in Axis2/Java on NonStop

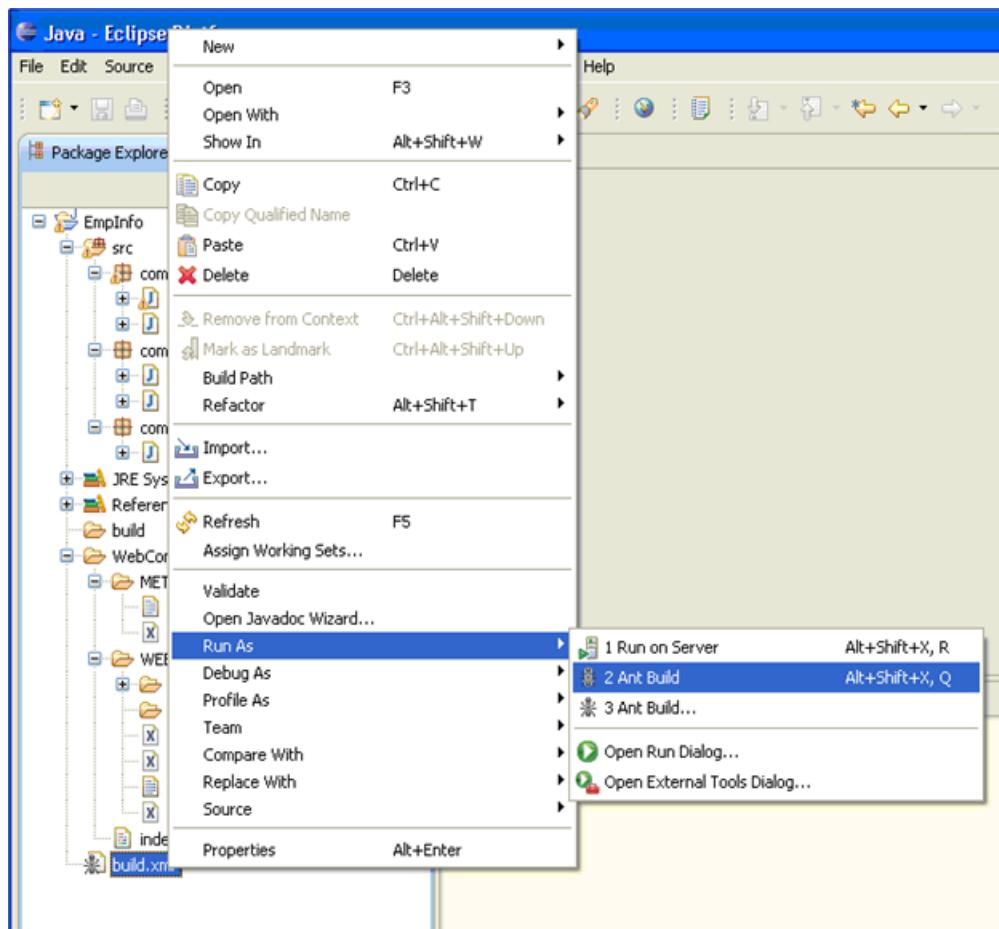
### Creating the EmplInfo AAR File on Windows

To create the EmplInfo AAR (EmplInfoService.aar) file, complete the following steps:

1. On the Project Explorer frame, right-click the EmpInfo/build.xml file and select **Run As > Ant Build**.

Figure 158 shows the Project Explorer Frame.

**Figure 158 Project Explorer Frame**



On selecting **Ant Build**, the following output appears on the console and target directory under EmpInfo is created.

---

**NOTE:** The `EmpInfoService.aar` file is created inside the target directory.

Figure 159 shows the Console dialog box.

**Figure 159 Console Dialog Box**

```
<terminated> EmpInfo build.xml [Ant Build] C:\Program Files\Java\jre1.5.0_05\bin\javaw.exe (Apr 15, 2009 4:41:03 PM)
Buildfile: D:\sash_usr\spring\EmpInfo\build.xml
clean:
[delete] Deleting directory D:\sash_usr\spring\EmpInfo\target
prepare:
[mkdir] Created dir: D:\sash_usr\spring\EmpInfo\target
[mkdir] Created dir: D:\sash_usr\spring\EmpInfo\target\classes
[mkdir] Created dir: D:\sash_usr\spring\EmpInfo\target\classes\META-INF
[mkdir] Created dir: D:\sash_usr\spring\EmpInfo\target\classes\lib
copy_jars:
[copy] Copying 3 files to D:\sash_usr\spring\EmpInfo\target\classes\lib
generate.service:
[copy] Copying 1 file to D:\sash_usr\spring\EmpInfo\target\classes\META-INF
[copy] Copying 1 file to D:\sash_usr\spring\EmpInfo\target\classes
[copy] Copying 1 file to D:\sash_usr\spring\EmpInfo\target\classes
[javac] Compiling 4 source files to D:\sash_usr\spring\EmpInfo\target\classes
[jar] Building jar: D:\sash_usr\spring\EmpInfo\target\EmpInfoService.aar
BUILD SUCCESSFUL
Total time: 1 second
```

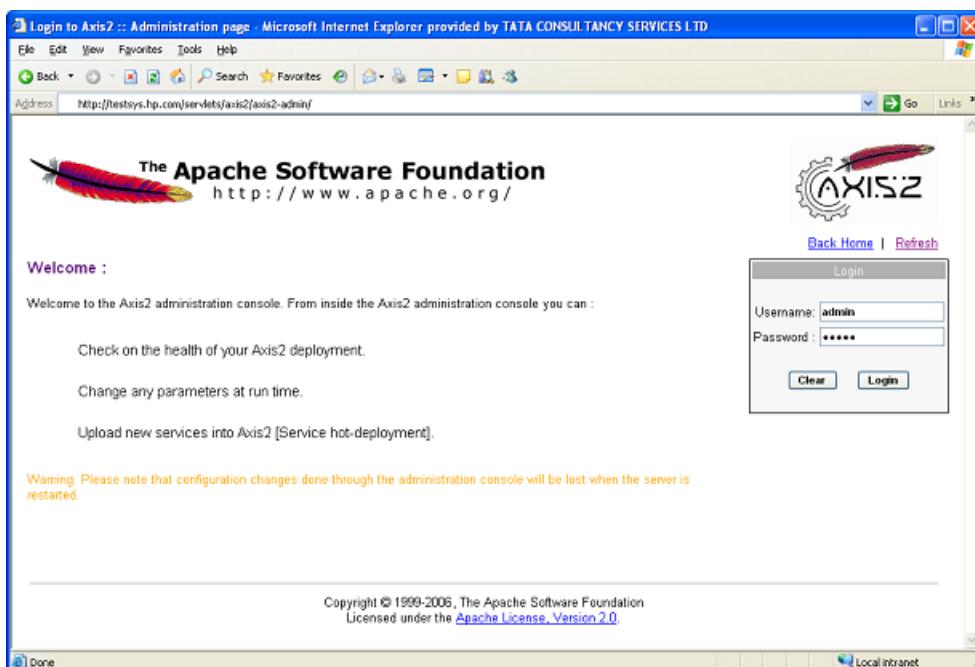
2. Refresh the EmpInfo Project and locate the EmpInfoService.aar under EmpInfo/target directory.

## Deploying the EmpInfo AAR File in Axis2/Java on NonStop

To deploy the EmpInfo AAR (EmpInfoService.aar) file on your NonStop system, complete the following steps:

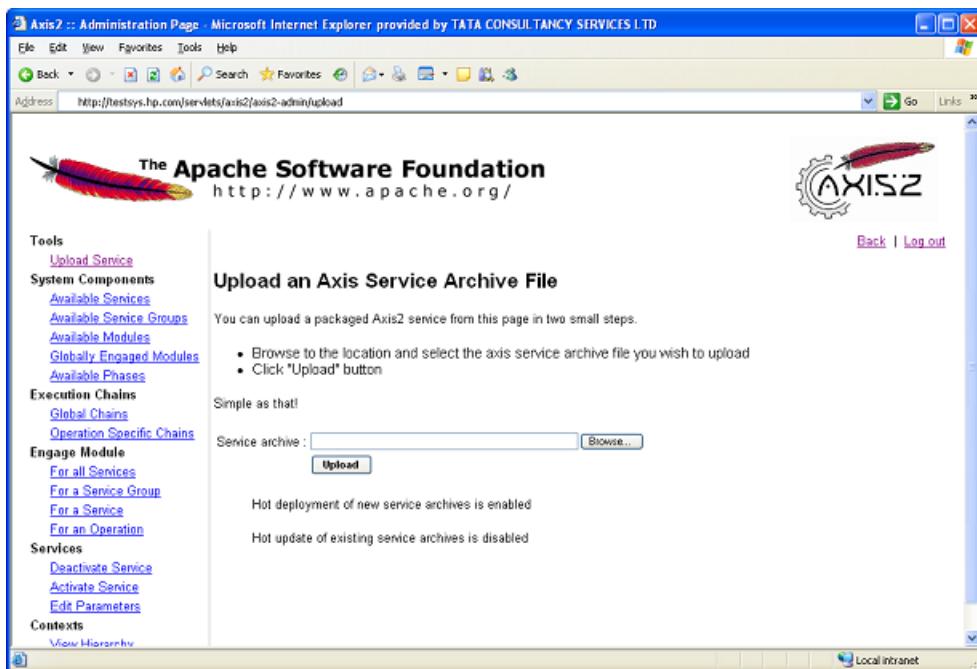
1. Go to the Axis2/Java admin login page using the following URL: [http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/axis2-admin/](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/axis2-admin/).

**Figure 160 Axis2/Java Administration Login Page**



2. Enter the username and password for the admin login to Axis2/Java. The default password for username admin is axis2.
3. In the **Tools** section, click **Upload Service**. In the **Service Archive** field, enter the complete address for the EmpInfoService.aar file or click **Browse...** to locate and select the file.

**Figure 161 Axis2/Java Administration Upload Page**



4. Click **Upload** to upload the service in Axis2/Java.

This completes deploying the EmplInfo and it is listed in [http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/services/listServices](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/services/listServices).

## Running EmplInfo Web Service on NonStop

To run the EmplInfo web service on your NonStop system, complete the following steps:

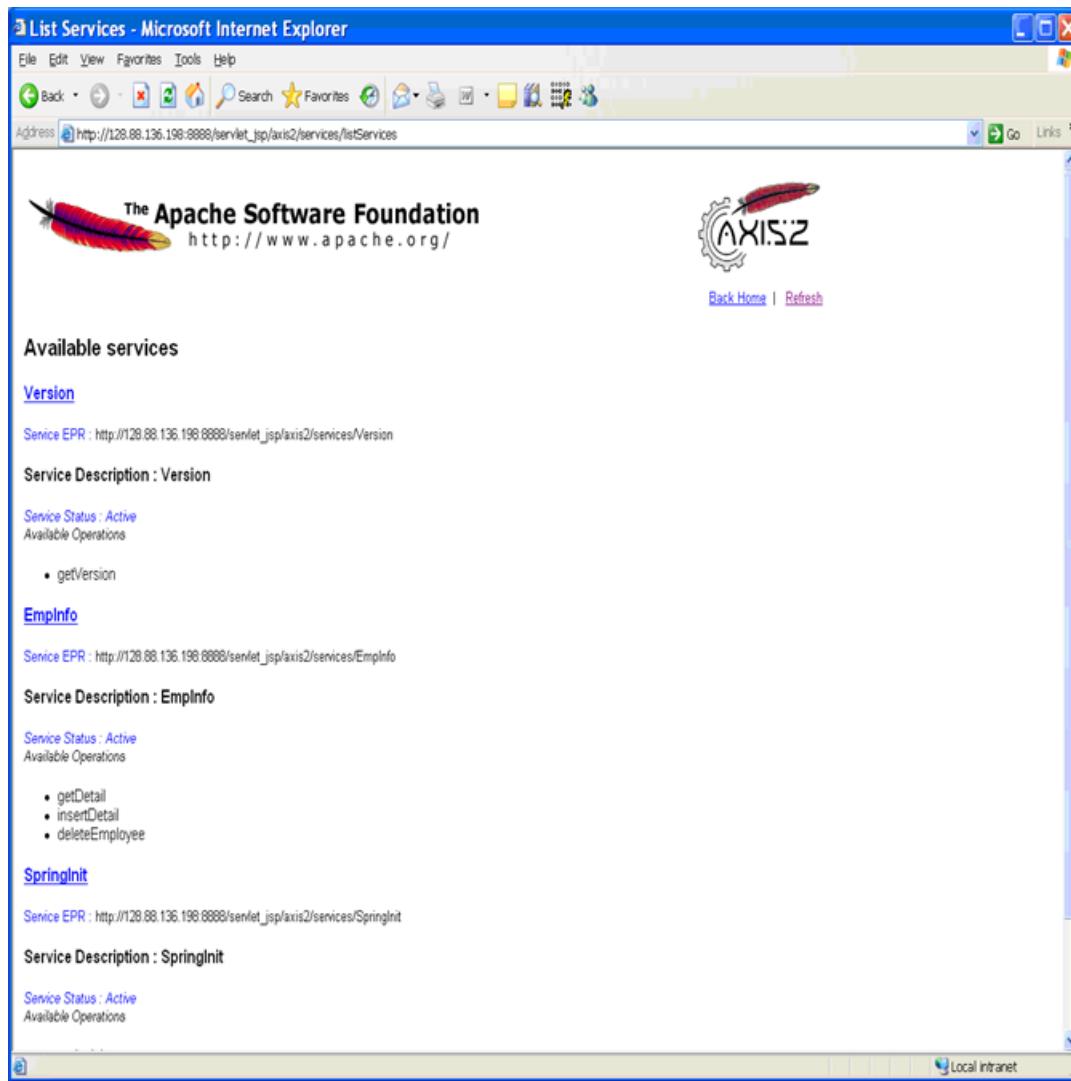
1. Access the EmplInfo web service using the following URL:

[http://<IP Address of the iTP WebServer>:<port#>/<servlet\\_directory>/axis2/services/listServices](http://<IP Address of the iTP WebServer>:<port#>/<servlet_directory>/axis2/services/listServices).

Details such as EPR, Service Description, Service Status, and Available Operations are also shown for the EmplInfo web service. The SpringInit service initializes the EmplInfo application context on startup.

Figure 162 shows the List Services dialog box.

**Figure 162 List Services Dialog Box**



## Developing EmplInfoClient on Windows

To invoke the EmplInfo web service on your NonStop system, you need to create Axis2/Java client on your Windows system. The APIs of the client are used to invoke the service.

To create an Axis2/Java client for EmplInfo web service, we will be using Eclipse Galileo IDE. Developing the Axis2/Java client (EmplInfoClient) for EmplInfo web service involves the following steps:

1. Creating the EmplInfoClient Project in Eclipse
2. Creating the Package for EmplInfoClient

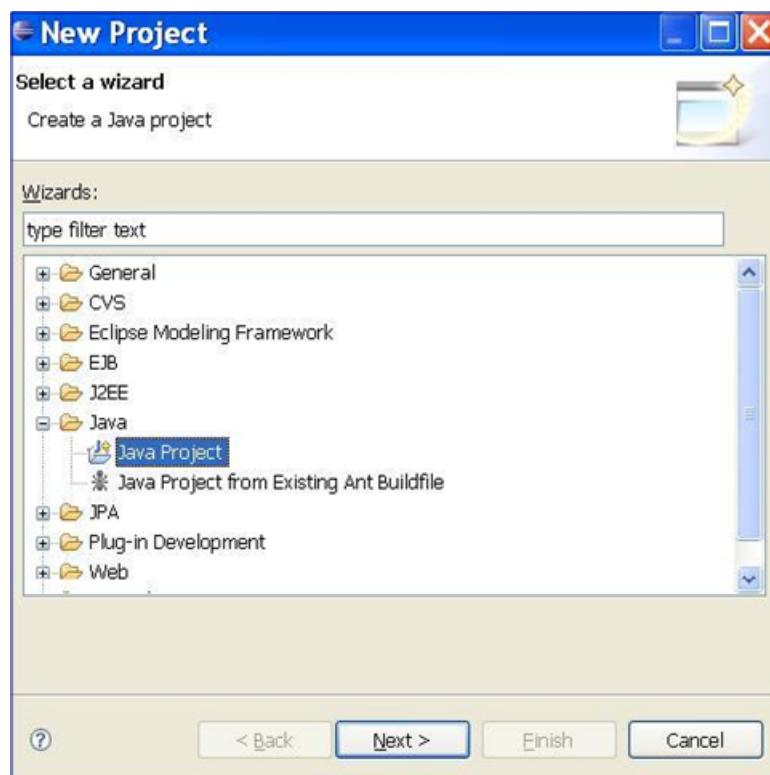
- 3. Creating the `EmpInfoClient.java` File**
- 4. Modifying the `EmpInfoClient.java` File**
- 5. Adding Dependency JAR Files**

## Creating the `EmplInfoClient` Project in Eclipse

To create the client project in Eclipse, complete the following steps:

- 1. Click **File > New > Java Project**.**  
The New Project dialog box appears.
- 2. From the list of folders, select **Java > Java Project** and click **Next**.**  
Figure 163 shows the New Project dialog box.

**Figure 163 New Project Dialog Box**

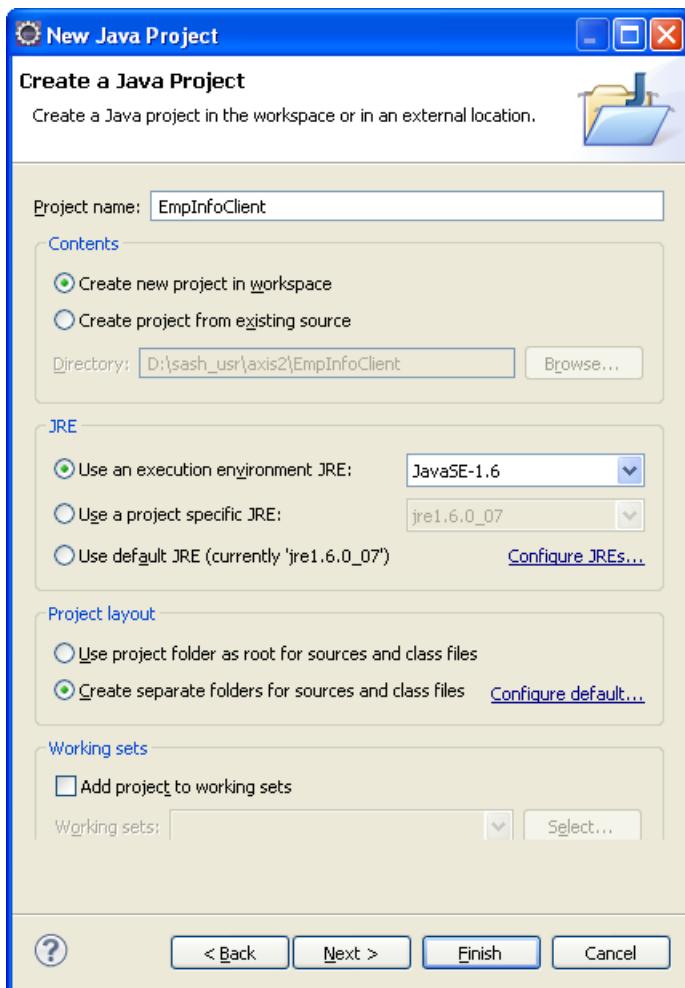


The New Java Project dialog box appears.

- 3. In the Project name field, enter `EmpInfoClient` (or specify the name of the project that you wish to create) and ensure that the directory for this project is created inside the workspace that you have selected or created.**

Figure 164 shows the New Java Project dialog box.

**Figure 164 New Java Project Dialog Box**

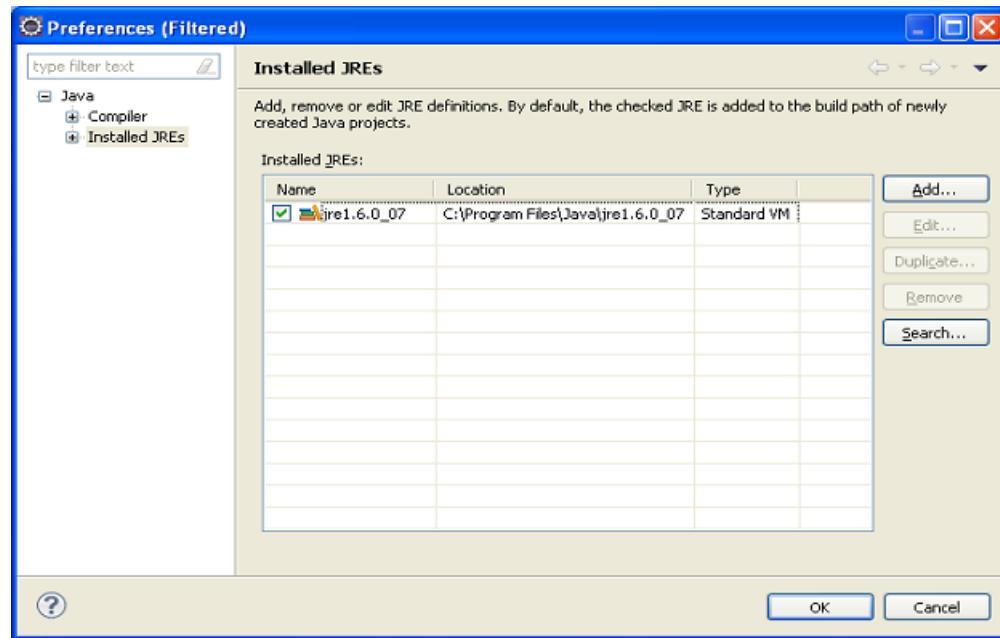


4. Confirm that the JRE System Library is set to JRE version 1.5 or later. If JRE version is not 1.5 or later, complete the following steps to select JRE 1.5 or later:
  - a. Click **Windows > Preferences**.
  - b. From the type filter text, select **Java > Installed JREs**.  
The JRE Options: Preferences dialog box appears.
  - c. Select **JRE 1.5** or later and click **OK**.

**NOTE:** In this section, JRE 1.6 version is used to create the client project.

Figure 165 shows the JRE Options: Preferences dialog box.

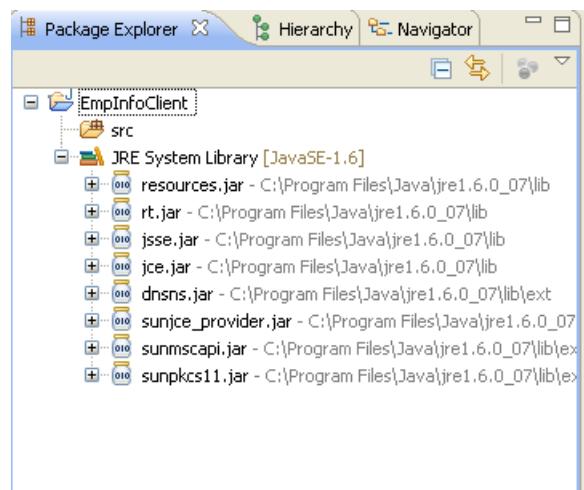
**Figure 165 JRE Options: Preferences Dialog Box**



- d. Click **Finish** in Figure 164 to create the Java project.

Figure 166 shows the Java: Project Explorer View.

**Figure 166 Java: Project Explorer View**



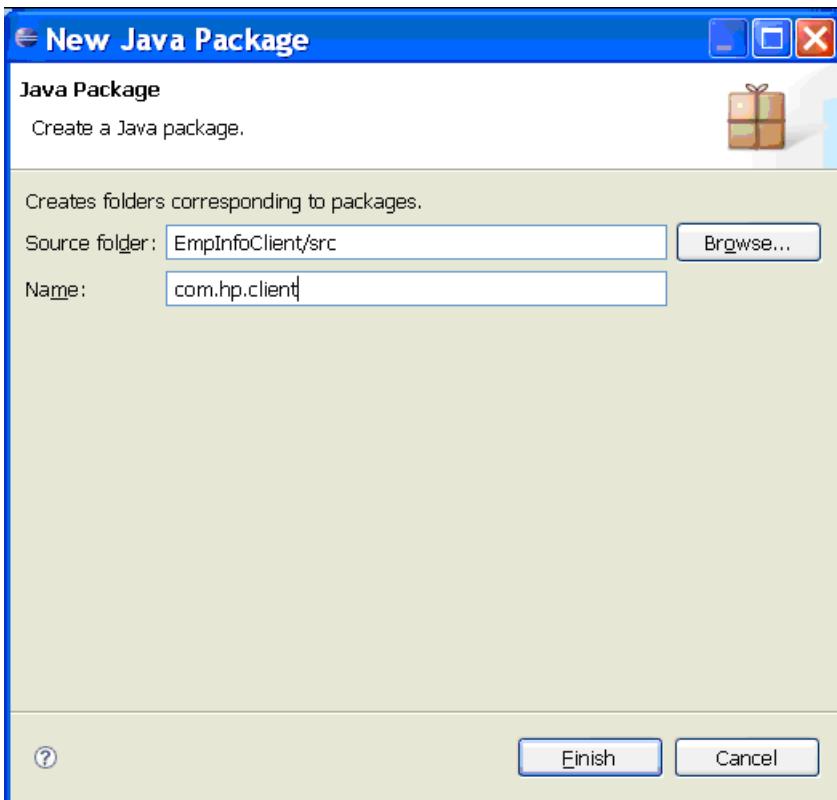
### Creating the Package for EmplInfoClient

To create the package for client project, complete the following steps:

1. On the Project Explorer frame, right-click the **EmplInfoClient** and select **New > Package**.  
The New Java Package dialog box appears.
2. In the Name field, enter **com.hp.client** and ensure that the Source folder is set to **EmplInfoClient/src** and click **Finish**.

Figure 167 shows the New Java Package dialog box

**Figure 167 New Java Package Dialog Box**



The package for client project is created.

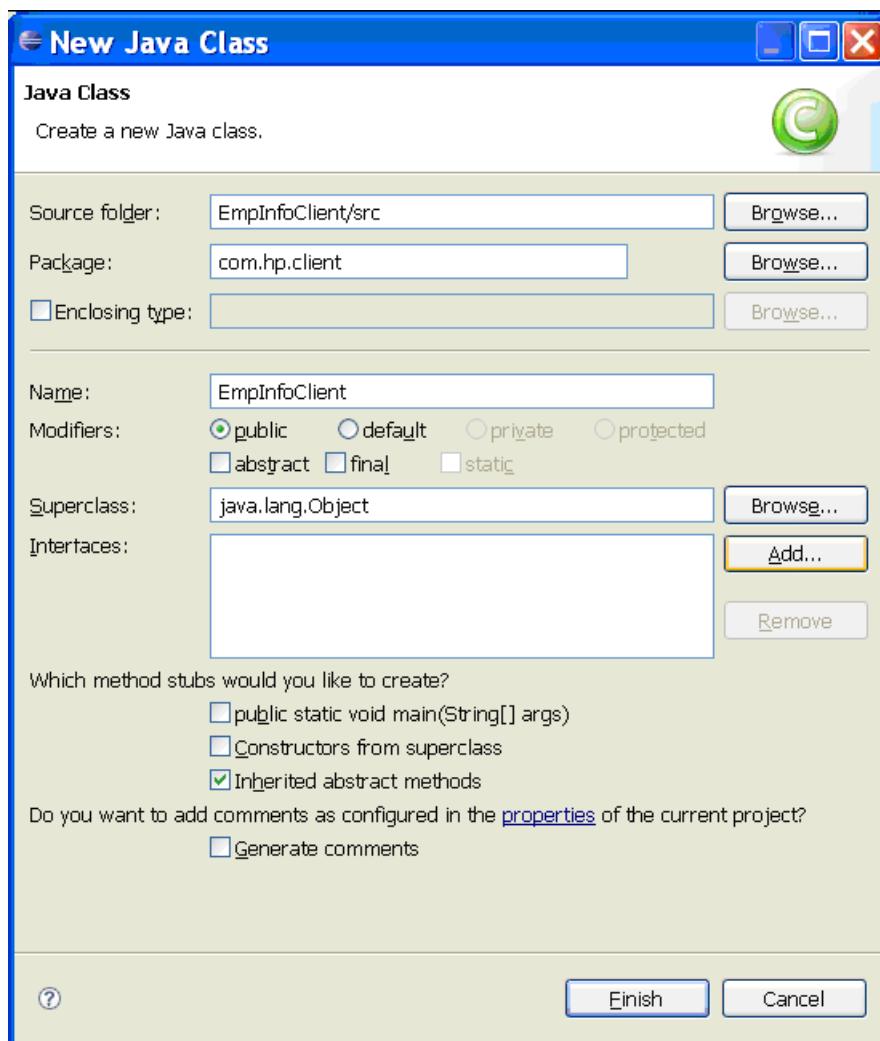
### Creating the `EmpInfoClient.java` File

To create the `EmpInfoClient.java` class file under the `EmplInfoClient` package, complete the following steps:

1. On the Project Explorer frame, right-click the **com.hp.client package** and select **New > Class**.  
The New Java Class dialog box appears.
2. In the Name field, enter `EmpInfoClient` and click **Finish**.

Figure 168 shows the New Java Class dialog box.

**Figure 168 New Java Class Dialog Box**



The EmpInfoClient.java class file under the EmpInfoClient package is created.

### Modifying the EmpInfoClient.java File

Modify the newly created class file EmpInfoClient.java file so that it appears as:

```
package com.hp.client;

import java.util.Scanner;

import org.apache.axiom.om.OMAbstractFactory;
import org.apache.axiom.om.OMELEMENT;
import org.apache.axiom.om.OMFactory;
import org.apache.axiom.om.OMNamespace;
import org.apache.axis2.AxisFault;
import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;

public class EmpInfoClient {
    public static void main(String [] args) throws AxisFault {

        String a;
        ServiceClient client = new ServiceClient();
        // create option object
        Options opts = new Options();
        // setting target EPR
        opts.setTo(new EndpointReference("http://<IP address of iTP WebServer>:<Port#>/<servlet
directory>/axis2/services/EmpInfo"));
        // Setting action ,and which can be found from the wsdl of the service
        Scanner scan=new Scanner(System.in);
        do
        {
            System.out.println("To Insert Employee press 1 \n"+"For Employee Details press 2 \n"+
"\"To Delete Employee press 3\"");
        }
    }
}
```

```

int choice=scan.nextInt();
if (choice==1 )
{
    opts.setAction("urn:insertDetail");
    client.setOptions(opts);
    client.sendRobust(createPayLoad1());
    System.out.println("Request Sent Successfully");
}
else if (choice==2 )
{ opts.setAction("urn:getDetail");
    client.setOptions(opts);
    OMElement res2 = client.sendReceive(createPayLoad2());
    System.out.println(res2);
}
else if (choice==3)
{
    opts.setAction("urn:deleteEmployee");
    client.setOptions(opts);
    OMElement res3 = client.sendReceive(createPayLoad3());
    System.out.println(res3);
}
System.out.println("Do You Wish To Continue: Press Y for Yes ");
a = scan.next();
}
while(a.equalsIgnoreCase("Y"));
}

public static OMElement createPayLoad1() {
OMFactory fac = OMAbstractFactory.getOMFactory();
OMNamespace omNs = fac.createOMNamespace(
    "http://service.empinfo.hp.com", "com");
    OMElement method = fac.createOMEElement("insertDetail", omNs);
    OMElement element1 = fac.createOMEElement("empid", omNs);
    Scanner scan=new Scanner(System.in);
    System.out.println("Enter Employee Id :");
    element1.setText(""+scan.nextInt());
    System.out.println("Enter First Name :");
    OMElement element2 = fac.createOMEElement("firstname", omNs);
    element2.setText(scan.next());
    System.out.println("Enter Last Name :");
    OMEElement element3 = fac.createOMEElement("lastname", omNs);
    element3.setText(scan.next());
    System.out.println("Enter Age :");
    OMEElement element4 = fac.createOMEElement("age", omNs);
    element4.setText(""+scan.nextInt());
    System.out.println("Enter Email :");
    OMEElement element5 = fac.createOMEElement("email", omNs);
    element5.setText(scan.next());
    method.addChild(element1);
    method.addChild(element2);
    method.addChild(element3);
    method.addChild(element4);
    method.addChild(element5);
    return method;
}

public static OMElement createPayLoad2() {
OMFactory fac = OMAbstractFactory.getOMFactory();
OMNamespace omNs = fac.createOMNamespace(
    "http://service.empinfo.hp.com", "com");

    OMElement method = fac.createOMEElement("getDetail", omNs);
    Scanner scan=new Scanner(System.in);
    System.out.println("Enter Employee Id :");
    OMElement element = fac.createOMEElement("empid", omNs);
    element.setText(""+scan.nextInt());
    method.addChild(element);

    return method;
}

public static OMElement createPayLoad3() {
OMFactory fac = OMAbstractFactory.getOMFactory();
OMNamespace omNs = fac.createOMNamespace(
    "http://service.empinfo.hp.com", "com");

    OMElement method = fac.createOMEElement("deleteEmployee", omNs);
    Scanner scan=new Scanner(System.in);
    System.out.println("Enter Employee Id :");
    OMElement element = fac.createOMEElement("empid", omNs);
    element.setText(""+scan.nextInt());
    method.addChild(element);

    return method;
}
}

```

---

**NOTE:** Remember to replace <IP address of iTP WebServer> with the IP address and <Port#> of the iTP WebServer with the port number of the iTP WebServer under which your EmplInfo webserver is running.

---

## Adding Dependency JAR Files

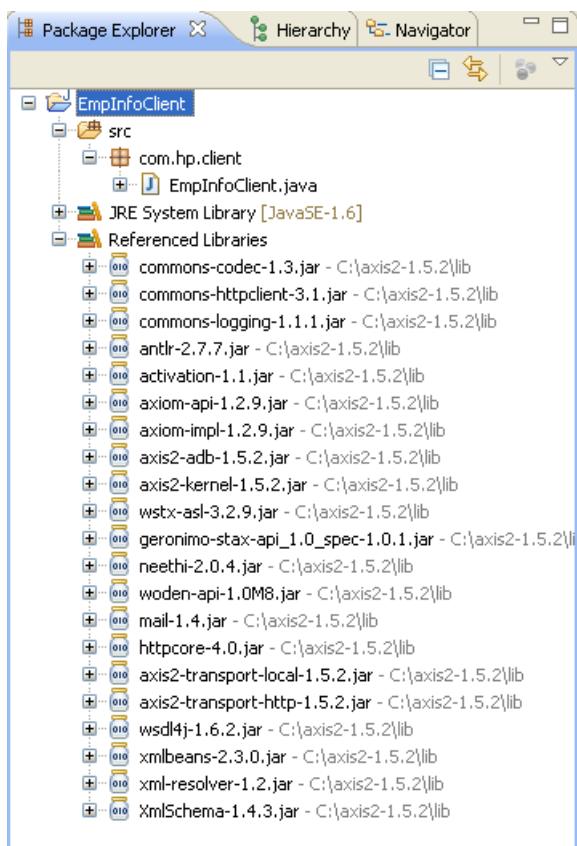
Add the following Axis2/Java dependency JAR files to the location <Axis2/Java Home>/lib in the Java Build Path of the EmplInfoClient project to compile and link the project:

- commons-codec-1.3.jar
- commons-httpclient-3.1.jar
- commons-logging-1.1.1.jar
- antlr-2.7.7.jar
- activation-1.1.jar
- axiom-api-1.2.9.jar
- axiom-impl-1.2.9.jar
- axis2-adb-1.5.2.jar
- axis2-kernel-1.5.2.jar
- wstx-asl-3.2.9.jar
- geronimo-stax-api\_1.0\_spec-1.0.1.jar
- neethi-2.0.4.jar
- woden-api-1.0M8.jar
- mail-1.4.jar
- httpcore-4.0.jar
- axis2-transport-local-1.5.2.jar
- axis2-transport-http-1.5.2.jar
- wsdl4j-1.6.2.jar
- xmlbeans-2.3.0.jar
- xml-resolver-1.2.jar
- XmlSchema-1.4.3.jar

To add the dependency JAR files in the project library path and to resolve the J2EE module dependency on these JAR files, follow the instructions in [Adding Dependency JAR Files in the Project Library Path](#) section of the [Getting Started with Spring](#) chapter.

Figure 169 shows the Project Explorer View.

**Figure 169 Project Explorer View**



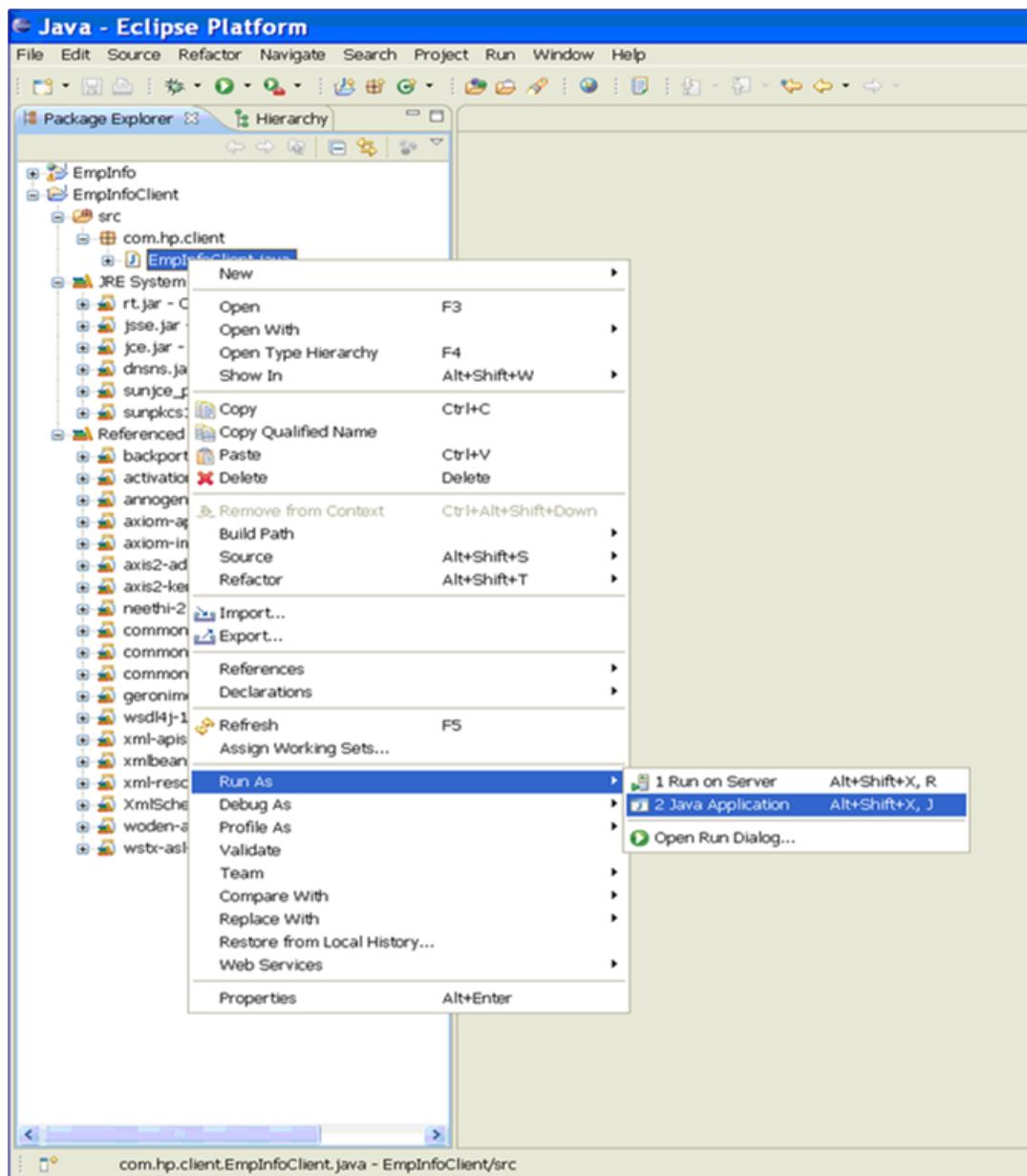
## Running EmpInfoClient on Windows

To run the EmpInfoClient on windows, complete the following steps:

1. On the Project Explorer frame, right-click the EmpInfoClient.java file and select **Run As > Java Application**.

Figure 170 shows the Java Eclipse Platform.

**Figure 170 Java Eclipse Platform**

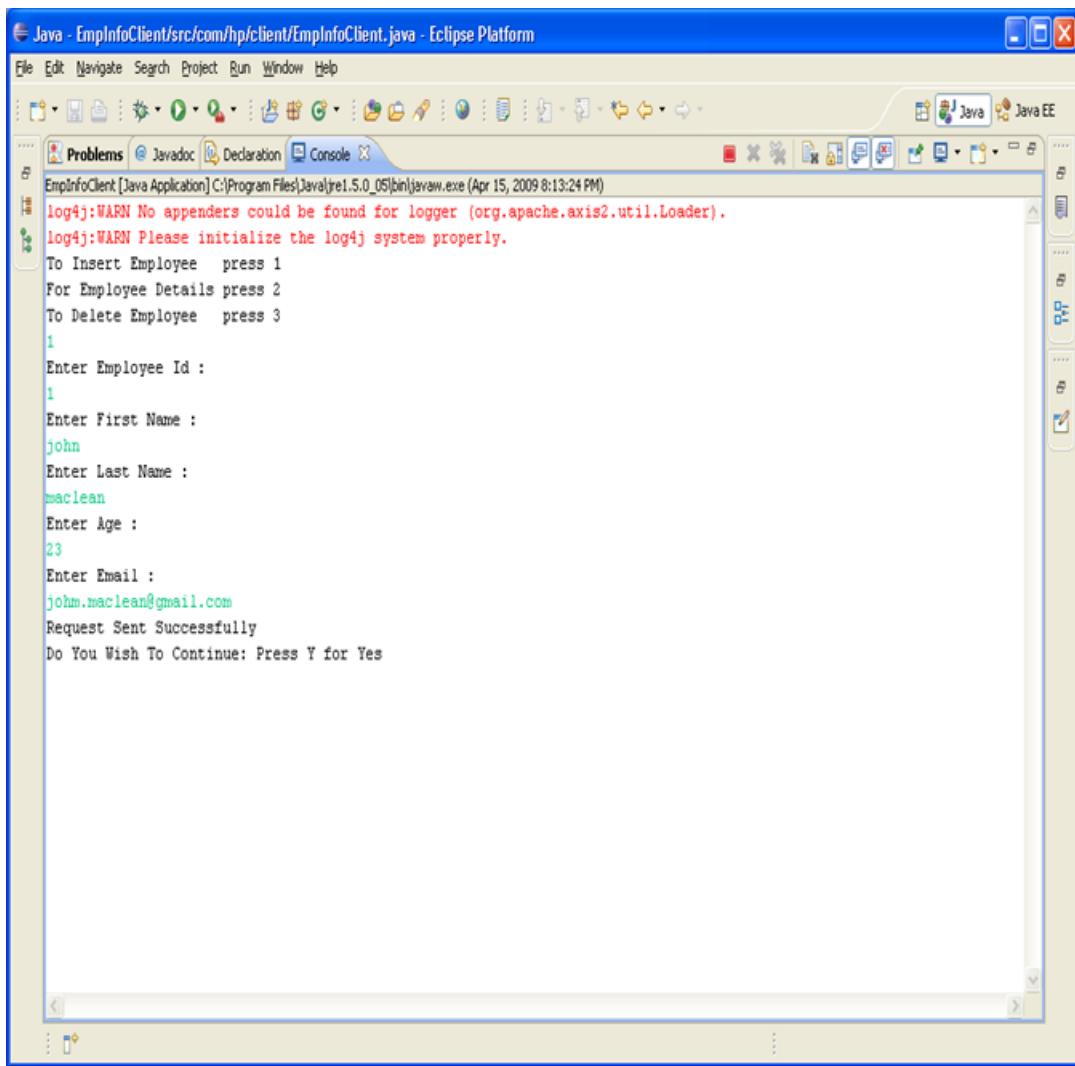


The Console is displayed.

2. Press 1 to insert Employee. Specify the employee details as prompted.

Figure 171 shows the Console: Insert Employee.

**Figure 171 Console: Insert Employee**



The screenshot shows the Eclipse IDE interface with the title bar "Java - EmplInfoClient/src/com/hp/client/EmplInfoClient.java - Eclipse Platform". The menu bar includes File, Edit, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, and Print. The left sidebar has tabs for Problems, Javadoc, Declaration, and Console. The main area is the Console tab, which displays the following text:

```
EmplInfoClient [Java Application] C:\Program Files\Java\jre1.5.0_05\bin\javaw.exe (Apr 15, 2009 8:13:24 PM)
log4j:WARN No appenders could be found for logger (org.apache.axis2.util.Loader).
log4j:WARN Please initialize the log4j system properly.
To Insert Employee press 1
For Employee Details press 2
To Delete Employee press 3
1
Enter Employee Id :
1
Enter First Name :
john
Enter Last Name :
maclean
Enter Age :
23
Enter Email :
john.maclean@gmail.com
Request Sent Successfully
Do You Wish To Continue: Press Y for Yes
```

The message Request Sent Successfully is displayed.

3. Press Y to continue.
4. Press 2 to retrieve Employee details. Specify the Employee ID of the employee whose details you wish to retrieve.

Figure 172 shows the Console: Employee Details.

**Figure 172 Console: Employee Details**

The screenshot shows the Eclipse IDE interface with the title bar "Java - EmplInfoClient/src/com/hp/client/EmplInfoClient.java - Eclipse Platform". The menu bar includes File, Edit, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, Find, and Run. The left sidebar shows the Problems, Javadoc, Declaration, and Console tabs, with the Console tab currently active. The main workspace shows Java code for an "EmplInfoClient" class. The console output window displays the following text:

```
EmpInfoClient [Java Application] C:\Program Files\Java\jre1.5.0_05\bin\javaw.exe (Apr 15, 2009 8:13:24 PM)
log4j:WARN No appenders could be found for logger (org.apache.axis2.util.Loader).
log4j:WARN Please initialize the log4j system properly.

To Insert Employee press 1
For Employee Details press 2
To Delete Employee press 3

1
Enter Employee Id :
1
Enter First Name :
john
Enter Last Name :
maclean
Enter Age :
23
Enter Email :
john.maclean@gmail.com
Request Sent Successfully
Do You Wish To Continue: Press Y for Yes
y
To Insert Employee press 1
For Employee Details press 2
To Delete Employee press 3

2
Enter Employee Id :
1
<ns:getDetailResponse xmlns:ns="http://service.empinfo.hp.com"><ns:return xmlns:ax24="http://domain.empinfo.hp.com/x:24">
Do You Wish To Continue: Press Y for Yes
```

5. Press **Y** to continue.
6. Press **3** to delete Employee. Specify the Employee Id of the employee you wish to delete.

Figure 173 shows the Console: Delete Employee.

Figure 173 Console: Delete Employee

The screenshot shows the Eclipse IDE interface with the title bar "Java - EmpInfoClient/src/com/hp/client/EmpInfoClient.java - Eclipse Platform". The menu bar includes File, Edit, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, and Find. The left sidebar shows the Problems, Javadoc, Declaration, and Console tabs, with the Console tab selected. The main workspace contains code for an Axis2 client. The console output is as follows:

```
<terminated> EmpInfoClient [Java Application] C:\Program Files\Java\jre1.5.0_05\bin\javaw.exe (Apr 15, 2009 8:13:24 PM)
Enter First Name :
john
Enter Last Name :
maclean
Enter Age :
23
Enter Email :
johm.maclean@gmail.com
Request Sent Successfully
Do You Wish To Continue: Press Y for Yes
y
To Insert Employee press 1
For Employee Details press 2
To Delete Employee press 3
2
Enter Employee Id :
1
<ns:getDetailResponse xmlns:ns="http://service.empinfo.hp.com"><ns:return xmlns:ns="http://service.empinfo.hp.com">
Do You Wish To Continue: Press Y for Yes
y
To Insert Employee press 1
For Employee Details press 2
To Delete Employee press 3
3
Enter Employee Id :
1
<ns:deleteEmployeeResponse xmlns:ns="http://service.empinfo.hp.com"><ns:return xmlns:ns="http://service.empinfo.hp.com">
Do You Wish To Continue: Press Y for Yes
n
```

# 22 Integrating MyFaces into Spring

This section describes how a Spring application can use MyFaces to create Views. This is demonstrated by modifying the EmplInfo application (developed in [Getting Started with Spring](#) chapter) using the Eclipse Galileo IDE.

The following topics are discussed in this section:

- [Why Integrate MyFaces into Spring](#)
- [Example of Integrating Axis2/Java with Spring](#)

## Why Integrate MyFaces into Spring

MyFaces and Spring have their individual usefulness and capabilities. For example, MyFaces when used for implementing Presentation Layer and Spring when used for implementing Business Layer, makes a perfect framework for the development of web application.

This is because MyFaces enables you to create a rich graphical user interface for web application and Spring uses Plain Old Java Object (POJO) and servlet classes, and other configuration files to process the business logic in the Business Layer. With features such as IOC, DI, and AOP, the Spring framework provides an ideal solution for implementing the Business Layer.

## Example of Integrating MyFaces into Spring

The EmplInfo application (described in the [Getting Started with Spring](#) chapter) is a Spring application that uses Spring managed views.

This section describes how to use MyFaces for managing views of the EmplInfo application.

### Prerequisites

The prerequisites for integrating the EmplInfo application with MyFaces are:

- Basic knowledge of the Spring and MyFaces frameworks.
- EmplInfo application (explained in the [Getting Started with Spring](#) chapter) developed on your Windows system.
- Following software installed on your NonStop system:
  - NonStop iTP WebServer
  - NonStop Servlets for Java Server Pages (NSJSP)
  - JDBC Type 2 driver for NonStop SQL/MX
  - NonStop SQL/MX
  - NonStop Server for Java (NSJ)
- Following software installed on your Windows system:
  - Java Development Kit (JDK)
  - JDBC Type 4 driver for NonStop SQL/MX
  - Eclipse Galileo IDE

### Activities involved

Integrating the EmplInfo application with MyFaces, involves the following activities:

- [Modifying EmplInfo on Windows using Eclipse Galileo IDE](#)
- [Deploying EmplInfo on NonStop](#)
- [Running EmplInfo on NonStop](#)

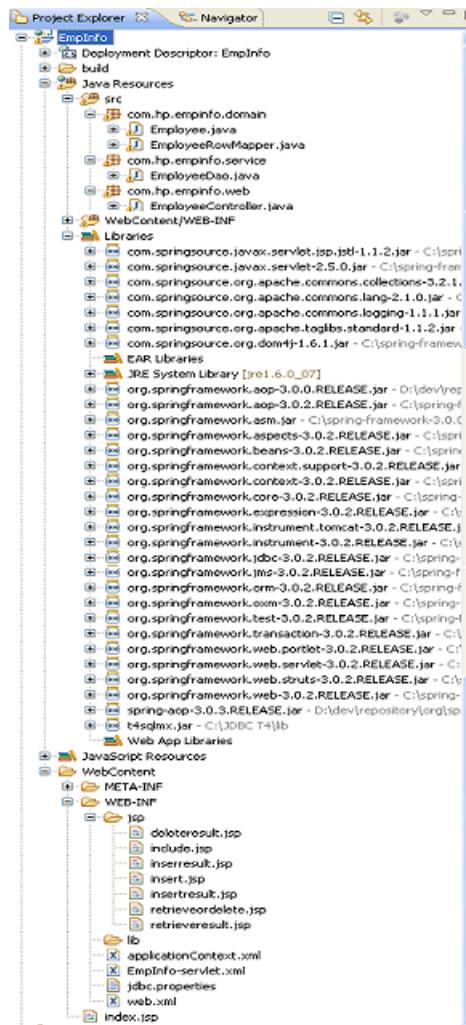
## Modifying EmplInfo on Windows using Eclipse Galileo IDE

The EmplInfo application developed in the [Getting Started with Spring](#) chapter is the starting point to demonstrate the integration of MyFaces with Spring applications.

Following is the screen shot of the final structure of the EmplInfo application that was developed in the [Getting Started with Spring](#) chapter.

Figure 174 shows the Project Explorer View.

**Figure 174 Project Explorer View**



Modifying the EmplInfo application involves the following tasks:

1. Removing the Folders and Files from the EmplInfo Application
2. Modifying the web.xml File
3. Modifying the web.xml File
4. Modifying the EmployeeDao.java File
5. Creating the EmpBean.java File
6. Creating the MessageFactory.java File
7. Creating the message.properties File
8. Creating views using MyFaces tag libraries
9. Modifying the index.jsp File
10. Creating the faces-config.xml File
11. Adding Dependency JAR Files

## Removing the Folders and Files from the EmplInfo Application

Remove the following folders and files from the EmplInfo application:

- jsp folder in EmpInfo/WebContent/WEB-INF.
- EmpInfo-servlet.xml file in EmpInfo/WebContent/WEB-INF.
- EmployeeController.java in EmpInfo/src/com/hp/empinfo/web.

## Modifying the web.xml File

Modify the web.xml file to set the Faces servlet and its mapping.

To modify the web.xml file, complete the following steps:

1. Configure WebApplicationContext to use the Spring ContextLoaderServlet. To do so, add the following code in the web.xml file:

```
<servlet>
    <servlet-name>context</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

2. Set the class for the Faces Controller servlet of the EmplInfo application by adding the following code in the web.xml file.

```
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

This ensures that the web.xml file has a Faces servlet entry

(javax.faces.webapp.FacesServlet), which serves as a Faces Controller servlet.

---

**NOTE:** The Faces Controller servlet intercepts all Faces requests.

3. Specify the URL pattern as \*.jsf in the <servlet-mapping/> tag as shown below:

```
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```

---

**NOTE:** This servlet mapping definition is for mapping the URL patterns. It ensures that any URL with the .jsf extension is routed to the Faces Servlet (the Faces Controller servlet).

4. Set the extension filter by adding the following code:

```
<filter>
    <filter-name>extensionsFilter</filter-name>
    <filter-class>
        org.apache.myfaces.component.html.util.ExtensionsFilter
    </filter-class>
    <init-param>
        <param-name>uploadMaxFileSize</param-name>
        <param-value>100m</param-value>
    </init-param>
    <init-param>
        <param-name>uploadThresholdSize</param-name>
        <param-value>100k</param-value>
    </init-param>
```

```
    </init-param>
</filter>
```

5. Set the filter mapping as shown below:

```
<filter-mapping>
<filter-name>extensionsFilter</filter-name>
<url-pattern>*.jsf</url-pattern>
</filter-mapping>
<filter-mapping>
<filter-name>extensionsFilter</filter-name>
<url-pattern>/faces/*</url-pattern>
</filter-mapping>
```

After modification, the web.xml file appears as:

```
<?xml version="1.0"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
version="2.4">

<servlet>
<servlet-name>context</servlet-name>
<servlet-class>
org.springframework.web.servlet.DispatcherServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>

<!-- End Spring configuration -->

<context-param>
<param-name>javax.faces.CONFIG_FILES</param-name>
<param-value>/WEB-INF/faces-config.xml</param-value>
</context-param>

<context-param>
<param-name>javax.faces.STATE_SAVING_METHOD</param-name>
<param-value>client</param-value>
</context-param>

<!-- Extensions Filter -->
<filter>
<filter-name>extensionsFilter</filter-name>
<filter-class>
org.apache.myfaces.component.html.util.ExtensionsFilter
</filter-class>
<init-param>
<param-name>uploadMaxFileSize</param-name>
<param-value>100m</param-value>
</init-param>
<init-param>
<param-name>uploadThresholdSize</param-name>
<param-value>100k</param-value>
</init-param>
</init-param>
</filter>

<filter-mapping>
<filter-name>extensionsFilter</filter-name>
<url-pattern>*.jsf</url-pattern>
</filter-mapping>
<filter-mapping>
<filter-name>extensionsFilter</filter-name>
<url-pattern>/faces/*</url-pattern>
</filter-mapping>

<servlet>
<servlet-name>Faces Servlet</servlet-name>
<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
<servlet-name>Faces Servlet</servlet-name>
<url-pattern>*.jsf</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
```

```
</welcome-file-list>
</web-app>
```

## Creating the IEmployeeDao.java File

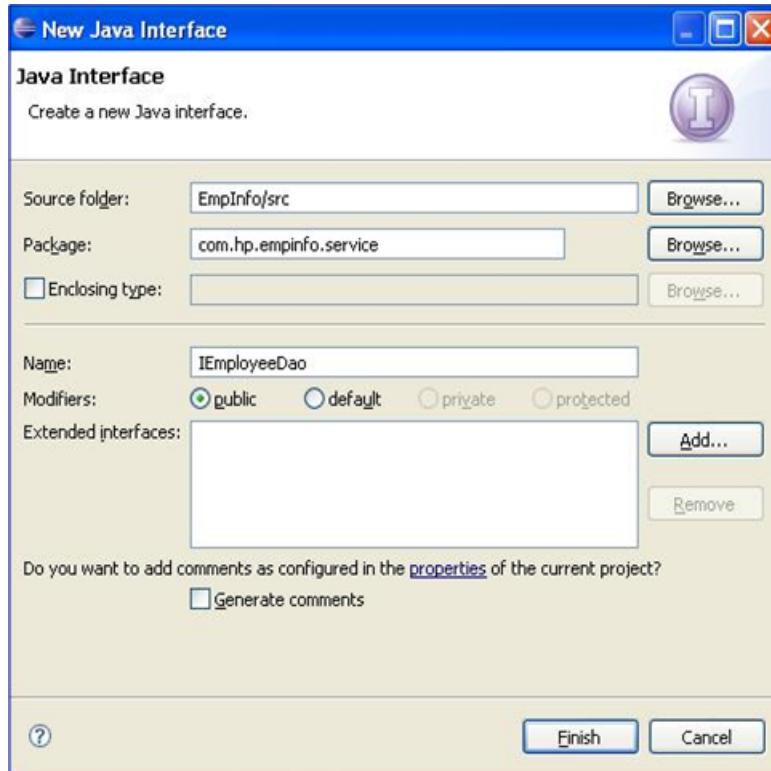
Create a Java interface named `IEmployeeDao.java` to implement the `EmployeeDao.java` class. The `IEmployeeDao.java` is placed with its implementation class in the `com.hp.empinfo.service` package.

To create the `IEmployeeDao.java` file, complete the following steps:

1. On the Project Explorer frame, right-click **EmpInfo** and select **New > Interface**.  
The New Java Interface dialog box appears.
2. In the Source folder field, enter `EmpInfo/src` and in the Package field, enter `com.hp.empinfo.service`.
3. In the Name field, enter `IEmployeeDao` and click **Finish**.

Figure 175 shows the New Java Interface dialog box.

**Figure 175 New Java Interface Dialog Box**



The `IEmployeeDao.java` file is created.

4. Modify the `IEmployeeDao.java` file to declare the methods implemented in the `EmployeeDao.java` file.

After modifications, the `IEmployeeDao.java` file must appear as shown:

```
package com.hp.empinfo.service;
import java.sql.SQLException;

import com.hp.empinfo.domain.Employee;

public interface IEmployeeDao {

    /**Retrieve employee Details corresponding to provided employeeId*/
    public Employee getDetail(int empid) throws SQLException ;

    /**Inserting employee Details*/
    public void insertDetail(int empid, String firstname, String lastname,
```

```

        int age, String email) throws SQLException ;
    /**Deleting Employee corresponding to given employeeId**/
    public String deleteEmployee(int empid) throws SQLException;
    /**Checking whether the employee corresponding to provided employeeId already exists in the database**/
    public Employee checkEmp(int empid) throws SQLException;
}

```

## Modifying the EmployeeDao.java File

Modify the EmployeeDao.java file in com.hp.empinfo.service package to add a new method checkEmp(int empid) for checking whether the employee with the given empid already exists.

After modification, the EmployeeDao.java file must appear as:

```

package com.hp.empinfo.service;

import java.sql.SQLException;

import org.springframework.dao.EmptyResultDataAccessException;
import org.springframework.jdbc.core.support.JdbcDaoSupport;

import com.hp.empinfo.domain.Employee;
import com.hp.empinfo.domain.EmployeeRowMapper;

public class EmployeeDao extends JdbcDaoSupport implements IEmployeeDao {

    public Employee getDetail(int empid) throws SQLException {
        Employee employee;
        employee = (Employee) getJdbcTemplate().queryForObject(
            "select * from employee where empid =?",
            new Object[] { empid }, new EmployeeRowMapper());
        return employee;
    }

    public void insertDetail(int empid, String firstname, String lastname,
        int age, String email) throws SQLException {
        getJdbcTemplate().update("insert into employee values(?, ?, ?, ?, ?)",
            new Object[] { empid, firstname, lastname, age, email });
    }

    public String deleteEmployee(int empid) {
        getJdbcTemplate().update("delete from employee where empid= ?",
            new Object[] { empid });
        return "Employee deleted";
    }

    public Employee checkEmp(int empid) throws SQLException {
        try{
            Employee employee ;

            employee = (Employee) getJdbcTemplate().queryForObject(
                "select * from employee where empid =?",
                new Object[] { empid }, new EmployeeRowMapper());
            return employee;
        }
        catch (final EmptyResultDataAccessException e) {
            return null;
        }
    }
}

```

```
}
```

## Creating the ServiceFinder.java File

The ServiceFinder.java file is a Business Services class that contains the code to interact with the data tier. The ServiceFinder service is used to get the Spring managed beans from WebApplicationContext.

To create the ServiceFinder.java file in the com.hp.empinfo.web package, complete the following steps:

1. Create the ServiceFinder.java file under the com.hp.empinfo.web package, as explained in the [Creating the Controller for EmplInfo](#) section in the [Getting Started with Spring](#) chapter.
2. Modify the ServiceFinder.java file to integrate the MyFaces views with the Spring business logic tier.

After modification, the ServiceFinder.java file must appear as shown:

```
package com.hp.empinfo.web;

import javax.faces.context.FacesContext;
import org.springframework.context.ApplicationContext;
import org.springframework.web.jsf.FacesContextUtils;

public class ServiceFinder {

    public static Object findBean(String beanName) {
        FacesContext context = FacesContext.getCurrentInstance();

        ApplicationContext appContext = FacesContextUtils
            .getWebApplicationContext(context);

        Object o = appContext.getBean(beanName);

        return o;
    }
}
```

## Creating the EmpBean.java File

The EmpBean.java file is used as a bean class for the view pages. This bean contains all the properties related to the fields in the view pages and their setter and getter method.

To create the EmpBean.java file in the com.hp.empinfo.web package, complete the following steps:

1. Create the EmpBean.java file under the com.hp.empinfo.web package, as explained in the [Creating the Controller for EmplInfo](#) section in [Getting Started with Spring](#) chapter.
2. Modify the EmpBean.java file to contain the properties for the fields in the view.

After modification, the EmpBean.java file must appear as shown:

```
package com.hp.empinfo.web;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;

import com.hp.empinfo.service.IEmployeeDao;
```

```

import com.hp.empinfo.web.ServiceFinder;

public class EmpBean {
    private int empid;
    private String firstname;
    private String lastname;
    private int age;
    private String email;
    private String rord;
    boolean exist = false;
    boolean state = false;
    boolean state1 = false;
    boolean ntexist = false;

    public boolean isExist() {
        return exist;
    }

    public void setExist(boolean exist) {
        this.exist = exist;
    }

    public int getEmpid() {
        return empid;
    }

    public void setEmpid(int empid) {
        this.empid = empid;
    }

    public String getFirstname() {
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getLastname() {
        return lastname;
    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getRord() {
        return rord;
    }

    public void setRord(String rord) {
        this.rord = rord;
    }

    public String register() throws Exception {
        String status = "failure";
        exist = false;

        if (validateData()) {
            IEmployeeDao dao = (IEmployeeDao) ServiceFinder.findBean("empdao");

            if (dao.checkEmp(getEmpid()) != null) {
                exist = true;
                status = "failure";
            } else {

```

```

com.hp.empinfo.domain.Employee employee = new com.hp.empinfo.domain.Employee();

// Set employee id
employee.setEmpid(getEmpid());

// Set employee Firstname
employee.setFirstname(getFirstname());

// Set employee Lastname
employee.setLastname getLastname());

// Set employee age
employee.setAge(getAge());

// Set employee email
employee.setEmail(getEmail());
/** Inserting Employee Details */
dao.insertDetail(employee.getEmpid(), employee.getFirstname(),
    employee.getLastname(), employee.getAge(), employee
        .getEmail());
status = "success";
resetInsert();
}
}

return status;
}

private boolean validateData() {
boolean status = true;
MessageFactory mf = new MessageFactory();
FacesContext ctx = FacesContext.getCurrentInstance();

// Checking Employee FirstName
if ((firstname.length() < 2) {
    ctx.addMessage("InsertForm:firstname", new FacesMessage(
        FacesMessage.SEVERITY_ERROR, mf
            .getMessage("errorFirstName"), null));
    status = false;
}

// Checking Employee LastName
if ((lastname.length() < 2) {
    ctx.addMessage("InsertForm:lastname", new FacesMessage(
        FacesMessage.SEVERITY_ERROR,
        mf.getMessage("errorLastName"), null));
    status = false;
}
// Checking Employee Age
if (age > 100) {
    ctx.addMessage("InsertForm:age", new FacesMessage(
        FacesMessage.SEVERITY_ERROR, mf.getMessage("errorAge"),
        null));
    status = false;
}

// Checking Email address
Pattern p = Pattern.compile(".+@.+\\.[a-z]+");
Matcher m = p.matcher(email);
boolean matchFound = m.matches();

if (!matchFound) {
    ctx.addMessage("InsertForm:email", new FacesMessage(
        FacesMessage.SEVERITY_ERROR, mf.getMessage("errorEmail"),
        null));
    status = false;
}
// End of checking Email address

return status;
}

public String update() throws Exception {
String s1 = null;
state1 = false;
state = false;
ntexist = false;
IEmployeeDao dao = (IEmployeeDao) ServiceFinder.findBean("empdao");
if (dao.checkEmp(getEmpid()) != null) {

```

```

exist = true;
s1 = getRord();
if (s1.equals("Delete")) {
    /** Deleting employee Details* */
    dao.deleteEmployee(getEmpid());
    state = true;
} else if (s1.equals("Retrieve")) {
    /** Retrieving employee Details* */
    com.hp.empinfo.domain.Employee employee = dao
        .getDetail(getEmpid());
        setEmpid(employee.getEmpid());
        setAge(employee.getAge());
        setEmail(employee.getEmail());
        setFirstname(employee.getFirstname());
        setLastname(employee.getLastname());
        resetUpdate();
} else {
    s1 = "Incorrect";
    state1 = true;
}

} else {
    ntexist = true;
}

return s1;
}

public boolean isState() {
    return state;
}

public void setState(boolean state) {
    this.state = state;
}

public boolean isState1() {
    return state1;
}

public void setState1(boolean state1) {
    this.state1 = state1;
}

public boolean isNtexist() {
    return ntexist;
}

public void setNtexist(boolean ntexist) {
    this.ntexist = ntexist;
}

public String resetInsert() {
    setAge(0);
    setEmail(null);
    setEmpid(0);
    setFirstname(null);
    setLastname(null);
    setRord(null);
    exist = false;
    return "insert";
}

public String resetUpdate() {
    setEmpid(0);
    setRord(null);
    state = false;
    state1 = false;
    return "update";
}
}

```

## Creating the MessageFactory.java File

The MessageFactory.java class loads the message bundle of the specific locale and is used in the Bean class to set the appropriate message for different fields when required.

To create the MessageFactory.java file in the com.hp.empinfo.web package, complete the following steps:

1. Create the MessageFactory.java file under the com.hp.empinfo.web package, as explained in the [Creating the Controller for EmplInfo](#) section in the [Getting Started with Spring](#) chapter.
2. Modify the MessageFactory.java file to include the method for receiving messages from the message bundle.

After modification, the MessageFactory.java file must appear as shown:

```
package com.hp.empinfo.web;

import java.util.*;

import javax.faces.context.FacesContext;

public class MessageFactory {
    ResourceBundle bundle;
    Locale locale;

    public MessageFactory() {
        locale = FacesContext.getCurrentInstance().getViewRoot().getLocale();
        bundle = ResourceBundle.getBundle("com.hp.empinfo.web.message",
            locale);
    }

    public String getMessage(String key) {
        return bundle.getString(key);
    }
}
```

## Creating the message.properties File

The message.properties file contains the message strings that are displayed in the views depending on the logic implemented in the Bean class ([EmpBean.java](#)).

To create the message.properties file in the com.hp.empinfo.web package, complete the following steps:

1. Create the message.properties file in the com.hp.empinfo.web package as explained in the [Creating the jdbc.properties File](#) section in the [Getting Started with Spring](#) chapter.
2. Modify the message.properties file to include the required message strings.

After modification, the message.properties file must appear as shown:

```
# Registration Page

errorFirstName=Employee First Name can not be less than 2 characters.
errorLastName=Employee Last Name can not be less than 2 characters.
errorEmail=Invalid Email Address.
errorAge=Age cannot exceed 100.
already_registered_msg=Employee already exists.
doesnot_exists_msg=Employee doesnot exists.
deleted_successfully_msg=Employee deleted
incorrect_field_value2_msg=Please give proper field value Retrieve/Delete.

# Message.properties file of JSF
javax.faces.component.UIInput.REQUIRED=Cannot be blank
```

## Creating views using MyFaces tag libraries

To create views using the MyFaces tag libraries involves the following tasks:

1. [Creating Pages folder](#)
2. [Creating JSPs](#)

- [3. Modifying JSPs](#)
- [4. Creating the mycss.css File](#)

### Creating Pages folder

Create a new folder **pages** under EmpInfo/WebContent as explained in the [Creating a folder to store JSPs](#) section in the [Getting Started with Spring](#) chapter.

### Creating JSPs

Create the following JSP files under EmpInfo/WebContent/pages as explained in the [Creating the index.jsp File](#) section in the [Getting Started with Spring](#) chapter.

- welcome.jsp
- insert.jsp
- empUpdate.jsp
- empDetail.jsp
- success.jsp

### Modifying JSPs

To modify the above-created JSPs, complete the following steps:

#### welcome.jsp

This page contains the links to insert page and empUpdate page.

Replace the contents of the existing welcome.jsp file with the code given below:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>

<f:view>

<html>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

<head>
<title>Record Insertion</title>
<link href="mycss.css" rel="stylesheet" type="text/css" />
</head>
<body>
<center><h:form id="WelcomeForm">

<h:panelGrid width="100%" columns="1" border="0">
<h:dataTable id="dt1" border="0" cellpadding="0" cellspacing="0" var="ab">
<h:column>
<f:facet name="header">
<h:outputText value="SpringFaces" styleClass="style4" />
</f:facet>
</h:column>
</h:dataTable>
</h:panelGrid>

<h:panelGrid width="175px" columns="3" border="0" cellspacing="0" cellpadding="0">
<t:htmlTag value="div">
<h:panelGroup>
<br>
</br>

<h:dataTable id="dt2" border="0" cellpadding="0" cellspacing="0" width="250" var="gh">
<h:column>
```

```

<f:facet name="header">
    <h:outputText value="Welcome To Employee Management Service "
        styleClass="style1" />
</f:facet>
</h:column>
</h:dataTable>

<br>
<br>
<t:commandLink id="Link1" action="#{EmpBean.resetInsert}"
    immediate="true">
    <h:outputText value="Insert New Employee details  >>" />
</t:commandLink>
<br>
<br>

<t:commandLink id="Link2" action="#{EmpBean.resetUpdate}" immediate="true">
    <h:outputText value="Update the existing employee details  >>" />
</t:commandLink>

</h:panelGroup>
</t:htmlTag>

</h:panelGrid>

</h:form></center>
</body>
</html>
</f:view>

```

### [insert.jsp](#)

This page enables you to insert a new employee detail in the database. It contains the EmployeeID, firstname, lastname, age, and email fields.

You should provide the required values in each of the fields. If you enter incorrect values in any of the fields, the respective error message appears on the page.

Replace the contents of the existing `insert.jsp` file with the code given below:

```

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<f:loadBundle basename="com.hp.empinfo.web.message" var="message" />
<f:view>

<html>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

<head>
<title>Record Insertion</title>
<link href="mycss.css" rel="stylesheet" type="text/css" />
</head>
<body>
<center><h:form id="InsertForm">

<h:panelGrid width="100%" columns="1" border="0"
    style="padding-left:10px; padding-top:10px; " styleClass="top_bg">
    <h:dataTable id="dt1" border="0" cellpadding="0" cellspacing="0"
        var="ab">
        <h:column>
            <f:facet name="header">
                <h:outputText value="SpringFaces" styleClass="style4" />
            </f:facet>
        </h:column>
    </h:dataTable>
</h:panelGrid>

```

```

<h:panelGrid width="175px" columns="3" border="0" cellspacing="0"
cellpadding="0">

<h:panelGroup>
<h:dataTable id="dt2" border="0" cellpadding="0" cellspacing="0"
width="250" var="gh">
<h:column>
<f:facet name="header">
<h:outputText value="Insert Employee Details "
styleClass="style1" />
</f:facet>
</h:column>
</h:dataTable>

<h:dataTable id="dt3" border="0" cellpadding="0" cellspacing="0"
width="250" var="gh">
<h:column>
<f:facet name="header">
<h:outputText value="#{message.already_registered_msg}"
style="color:red; font-weight: bold;" 
rendered="#{EmpBean.exist}" />
</f:facet>
</h:column>
</h:dataTable>

<h:panelGrid width="100px" columns="2" border="0" cellspacing="2"
cellpadding="0">

<h:outputText value="Employee Id" styleClass="style2" />
<h:panelGroup>
<h:inputText id="Empid" value="#{EmpBean.empid}" required="true"
size="27" />
<f:verbatim>
<br />
</f:verbatim>
<h:message for="Empid" styleClass="errors" />
</h:panelGroup>

<h:outputText value="firstname" styleClass="style2" />
<h:panelGroup>
<h:inputText id="firstname" value="#{EmpBean.firstname}" size="27"
required="true" />
<f:verbatim>
<br />
</f:verbatim>
<h:message for="firstname" styleClass="errors" />
</h:panelGroup>

<h:outputText value="lastname" styleClass="style2" />
<h:panelGroup>

<h:inputText id="lastname" value="#{EmpBean.lastname}" size="27"
required="true" />
<f:verbatim>
<br />
</f:verbatim>
<h:message for="lastname" styleClass="errors" />
</h:panelGroup>

<h:outputText value="age" styleClass="style2" />

```

```

<h:panelGroup>
    <h:inputText id="age" value="#{EmpBean.age}" required="true"
        size="27" />
    <f:verbatim>
        <br />
    </f:verbatim>
    <h:message for="age" styleClass="errors" />
</h:panelGroup>

<h:outputText value="email" styleClass="style2" />
<h:panelGroup>
    <h:inputText id="email" value="#{EmpBean.email}" size="27"
        required="true" />
    <f:verbatim>
        <br />
    </f:verbatim>
    <h:message for="email" styleClass="errors" />
</h:panelGroup>

<h:outputText value=" " />
<h:commandButton value="submit" action="#{EmpBean.register}" />

</h:panelGrid>

<br>
</br>
<t:commandLink id="Link1" action="home" immediate="true">
    <h:outputText value="Link to Home page >>" />
</t:commandLink>

</h:panelGroup>
</h:panelGrid>

</h:form></center>
</body>
</html>
</f:view>

```

### [empUpdate.jsp](#)

This page enables you to retrieve or delete the existing employee details from the database. It contains the EmployeeID and Action (Retrieve/Delete) fields.

You should provide the required values in each of the fields. If you enter incorrect values in any of the fields, the respective error message appears on the page.

Replace the contents of the existing `empUpdate.jsp` file with the code given below:

```

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<f:loadBundle basename="com.hp.empinfo.web.message" var="message"/>
<f:view>

<html>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

<head>
<title>Record Updation</title>
<link href="mycss.css" rel="stylesheet" type="text/css"/>
</head>
<body ><center>
<h:form id="UpdateForm">

<h:panelGrid width="100%" columns="1" border="0">
    <h:dataTable id="dt1" border="0" cellpadding="0" cellspacing="0" var="ab">
        <h:column>

```

```

<f:facet name="header">
<h:outputText value="SpringFaces" styleClass="style4"/>
</f:facet>
</h:column>
</h:dataTable>
</h:panelGrid>

<h:panelGrid width="175px" columns="3" border="0" cellspacing="0" cellpadding="0">
<h:panelGroup>

    <h: dataTable id="dt2" border="0" cellpadding="0" cellspacing="0" width="250" var="gh">
        <h:column>
            <f:facet name="header">
                <h:outputText value="Record Update " styleClass="style1"/>
            </f:facet>
        </h:column>
    </h: dataTable>

    <h: dataTable id="dt3" border="0" cellpadding="0" cellspacing="0" width="250" var="gh">
        <h:column>
            <f:facet name="header">
                <h:outputText value="#{message.doesnot_exists_msg}" style="color:red; font-weight: bold;" rendered="#{EmpBean.ntexist}"/>
            </f:facet>
        </h:column>
    </h: dataTable>

<h:panelGrid width="100px" columns="2" border="0" cellspacing="2" cellpadding="0">
    <h:outputText value="Employee Id" styleClass="style2"/>
    <h:panelGroup>
        <n:inputText id="Empid" value="#{EmpBean.empid}" required="true" size="27" />
        <f:verbatim><br/></f:verbatim>
        <h:message for="Empid" styleClass="errors"/>
    </h:panelGroup>

        <h:outputText value="Action to perform (Retrieve or Delete)" styleClass="style2"/>
    <h:panelGroup>
        <h:inputText id="rord" value="#{EmpBean.rord}" required="true" size="27" />
    </h:panelGroup>

        <h:outputText value=" "/>
        <h:commandButton id="update" value="Retrieve/Delete" action="#{EmpBean.update}"/>
    </h:panelGrid>
<h: dataTable id="dt4" border="0" cellpadding="0" cellspacing="0" width="250" var="gh">
    <h:column>
        <f:facet name="header">
            <h:outputText value="#{message.deleted_successfully_msg}" style="color:red; font-weight: bold;" rendered="#{EmpBean.state}"/>
        </f:facet>
    </h:column>
</h: dataTable>
<h: dataTable id="dt5" border="0" cellpadding="0" cellspacing="0" width="250" var="gh">
    <h:column>
        <f:facet name="header">
            <h:outputText value="#{message.incorrect_field_value2_msg}" style="color:red; font-weight: bold;" rendered="#{EmpBean.state1}"/>
        </f:facet>
    </h:column>
</h: dataTable>
<br>
<br>
<t:commandLink id="Link1" action="home" immediate="true">
    <h:outputText value="Link to Home page >>" />
</t:commandLink>

    </h:panelGroup>
</h:panelGrid>

</h:form>
</center>
</body>
</html>
</f:view>

```

### [empDetail.jsp](#)

This page displays the details of an employee whose Employee ID (with retrieve option selected) is provided in the empUpdate.jsp page.

Replace the contents of the existing empDetail.jsp file with the code given below:

```

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>

<f:view>
<html>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

<head>
<title>Detail</title>
<link href="mycss.css" rel="stylesheet" type="text/css" />
</head>
<body>
<center><h:form id="EmpdetailForm">

<h:panelGrid width="100%" columns="1" border="0">
  <h:dataTable id="dt1" border="0" cellpadding="0" cellspacing="0"
    var="ab">
    <h:column>
      <f:facet name="header">
        <h:outputText value="SpringFaces" styleClass="style4" />
      </f:facet>
    </h:column>
  </h:dataTable>
</h:panelGrid>

<h:panelGrid width="175px" columns="3" border="0" cellspacing="0"
  cellpadding="0">

  <h:panelGroup>
    <h:dataTable id="dt2" border="0" cellpadding="0" cellspacing="0"
      width="250" var="gh">
      <h:column>
        <f:facet name="header">
          <h:outputText value="Displaying Employee Details "
            styleClass="style1" />
        </f:facet>
      </h:column>
    </h:dataTable>

    <t: dataTable id="dt3" value="" var="item" bgcolor="#F1F1F1"
      border="10" cellpadding="5" cellspacing="3" first="0" rows="4"
      width="50%" dir="LTR" frame="border" rules="all"
      summary="This is a JSF code to create dataTable."
      rowClasses="TableRow1,TableRow2" columnClasses="TableColumn"
      styleClass="TableClass" headerClass="TableHeader"
      footerClass="TableFooter">
      <f:facet name="header">
        <h:outputText value="Employee Information" />
      </f:facet>
      <t:column style="color:green; font-weight:bold"
        headerstyle="background-color:#99CCFF;">
        <f:facet name="header">
          <t:outputText value="Employee Id" />
        </f:facet>
        <t:outputText value="#{EmpBean.empid}"></t:outputText>
      </t:column>
      <t:column headerstyle="background-color:#99CCFF;">
        <f:facet name="header">
          <t:outputText value="Firstname" />
        </f:facet>
        <t:outputText value="#{EmpBean.firstname}"></t:outputText>
      </t:column>
      <t:column headerstyle="background-color:#99CCFF;">
        <f:facet name="header">
          <t:outputText value="Lastname" />
        </f:facet>
        <t:outputText value="#{EmpBean.lastname}"></t:outputText>
      </t:column>
      <t:column headerstyle="background-color:#99CCFF;">
        <f:facet name="header">

```

```

        <t:outputText value="Age" />
    </f:facet>
    <t:outputText value="#{EmpBean.age}"></t:outputText>
</t:column>
<t:column headerstyle="background-color:#99CCFF;">
    <f:facet name="header">
        <t:outputText value="Email" />
    </f:facet>
    <t:outputText value="#{EmpBean.email}"></t:outputText>
</t:column>
<f:facet name="footer">
    <h:outputText value="The End" />
</f:facet>
</t:dataTable>
<br></br>

<t:commandLink id="Link1" action="#{EmpBean.resetUpdate}" immediate="true">
    <h:outputText value=" Back >>" />
</t:commandLink>
<br>
</br>
<t:commandLink id="Link2" action="home" immediate="true">
    <h:outputText value="Link to Home page >>" />
</t:commandLink>

</h:panelGroup>
</h:panelGrid>
</h:form></center>
</body>
</html>
</f:view>

```

### success.jsp

This page appears when the insert operation is successfully completed.

Replace the contents of the existing success.jsp file with the code given below:

```

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<f:view>
    <html>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

        <head>
            <title>Welcome</title>
            <link href="mycss.css" rel="stylesheet" type="text/css" />
        </head>
        <body>
            <center><h:form>
                <h:panelGrid width="100%" columns="1" border="0">
                    <h: dataTable id="dt1" border="0" cellpadding="0" cellspacing="0"
                        var="ab">
                        <h:column>
                            <f:facet name="header">
                                <h:outputText value="SpringFacse" styleClass="style4" />
                            </f:facet>
                        </h:column>
                    </h: dataTable>

                </h:panelGrid>
                <h:panelGrid width="100%" columns="1" border="0">
                    <f:verbatim>&nbsp;</f:verbatim>
                    <h:outputText value=" " />
                    <h:outputText value=" " />
                </h:panelGrid>
            </h:form></center>
        </body>
    </f:view>

```

```

<h:outputText value=" " />
</h:panelGrid>

<h:outputText value=" DataBase is successfully Updated . . . . "
    style="color:green; font-weight:bold" />
<h:panelGroup>
    <br>
    </br>
    <t:commandLink id="Link1" action="back" immediate="true">
        <h:outputText value=" Back >>" />
    </t:commandLink>
    <br>
    </br>

    <t:commandLink id="Link2" action="home" immediate="true">
        <h:outputText value="Link to Home page >>" />
    </t:commandLink>

</h:panelGroup>

</h:form></center>
</body>
</html>
</f:view>

```

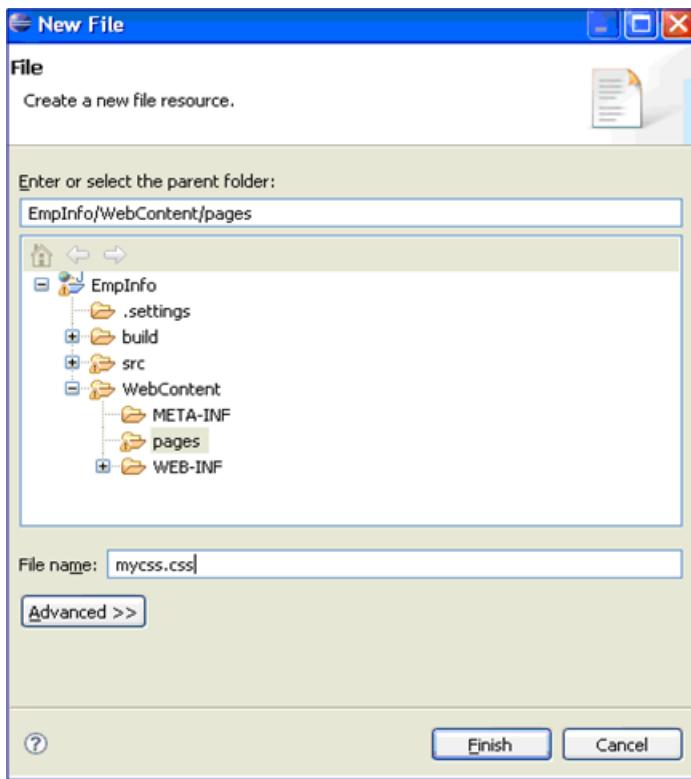
### [Creating the mycss.css File](#)

This file is the cascading style sheet which defines the style of the JSP pages.

To create the mycss.css file, complete the following steps:

1. On the Project Explorer frame, right-click **EmplInfo** and select **New > File**.  
The New File dialog box appears.
2. In the File name, enter **mycss.css** and in the parent folder, select **EmplInfo/WebContent/pages**.  
[Figure 176](#) shows the New File dialog box.

**Figure 176 New File Dialog Box**



**3. Click **Finish**.**

The mycss.css file is created.

**4. Add the following code to the mycss.css file:**

```
body{  
background-color:#fff2f2;  
margin-left:0;  
margin-right:0;  
margin-top:0;  
margin-bottom:0;  
}  
.TableRow1 {  
background-color: #D0E6E0;  
}  
.TableRow2 {  
background-color: #E8F7F1;  
}  
.TableColumn {  
text-align: center  
}  
.TableClass {  
font-family : verdana, Geneva, Arial, Helvetica, sans-serif;  
font-size: 13px;  
color: #000000;  
padding: 2;  
border-style: solid;  
border-width: 2px;  
}  
.TableHeader {  
color: #000000;  
background-color: #F1F1F1;  
padding: 3;  
text-align: center;  
border: none;  
}  
.TableFooter {  
background-color: #F1F1F1;  
}
```

```

.style1 {
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-weight: bold;
    font-size: 16px;
    color: green;
    font-weight:bold;
}
.style2 {
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-size: 10px;
    font-weight: bold;
}
.style3 {font-size: 13px; font-family: Verdana, Arial, Helvetica, sans-serif;}
.style4 {
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-weight:bold;
    color: #FF0000;
}
.errors {
    font-style: italic;
    color: green;
}

```

## Modifying the index.jsp File

Modify EmpInfo/index.jsp by removing the existing code and adding the redirect tag to the faces-enabled page welcome.jsp, as shown below:

```
<% response.sendRedirect("pages/welcome.jsf"); %>
```

## Creating the faces-config.xml File

This file contains configuration details for the MyFaces implementation.

To create the faces-config.xml file, complete the following steps

1. Create the faces-config.xml file in the EmpInfo/WebContent/WEB-INF directory as explained in the [Creating the EmpInfo-servlet.xml File](#) section in the [Getting Started with Spring](#) chapter.
2. Modify the faces-config.xml file to include information about message bundle, backing bean, and navigation rules.

After modification, the faces-config.xml file must appear as shown:

```

<?xml version="1.0"?>

<!DOCTYPE faces-config PUBLIC
    "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
    "http://java.sun.com/dtd/web-facesconfig_1_0.dtd" >

<faces-config>

    <application>
        <locale-config>
            <default-locale>en</default-locale>
        </locale-config>
        <message-bundle>com.hp.empinfo.web.message</message-bundle>
    </application>
    <managed-bean>
        <managed-bean-name>EmpBean</managed-bean-name>
        <managed-bean-class>
            com.hp.empinfo.web.EmpBean
        </managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>

```

```

<navigation-rule>
  <from-view-id>/pages/insert.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/pages/success.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>failure</from-outcome>
    <to-view-id>/pages/insert.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>home</from-outcome>
    <to-view-id>/pages/welcome.jsp</to-view-id>
  </navigation-case>

</navigation-rule>

<navigation-rule>
  <from-view-id>/pages/welcome.jsp</from-view-id>
  <navigation-case>
    <from-outcome>insert</from-outcome>
    <to-view-id>/pages/insert.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>update</from-outcome>
    <to-view-id>/pages/empUpdate.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/pages/empUpdate.jsp</from-view-id>
  <navigation-case>
    <from-outcome>Retrieve</from-outcome>
    <to-view-id>/pages/empDetail.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>Delete</from-outcome>
    <to-view-id>/pages/empUpdate.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>Incorrect</from-outcome>
    <to-view-id>/pages/empUpdate.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>home</from-outcome>
    <to-view-id>/pages/welcome.jsp</to-view-id>
  </navigation-case>

</navigation-rule>
<navigation-rule>
  <from-view-id>/pages/empDetail.jsp</from-view-id>
  <navigation-case>
    <from-outcome>home</from-outcome>
    <to-view-id>/pages/welcome.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>update</from-outcome>
    <to-view-id>/pages/empUpdate.jsp</to-view-id>
  </navigation-case>

</navigation-rule>

<navigation-rule>
  <from-view-id>/pages/success.jsp</from-view-id>
  <navigation-case>

```

```

<from-outcome>home</from-outcome>
<to-view-id>/pages/welcome.jsp</to-view-id>
</navigation-case>
<navigation-case>
<from-outcome>back</from-outcome>
<to-view-id>/pages/insert.jsp</to-view-id>
</navigation-case>

</navigation-rule>

</faces-config>

```

## Adding Dependency JAR Files

Download the following JAR files from <http://archive.apache.org/dist/commons/>:

- commons-el.jar
- commons-fileupload-1.0.jar
- commons-validator-1.3.1.jar
- tomahawk-1.1.8.jar

**NOTE:** If you have already downloaded and installed these JAR files, you need not download them again.

Download and extract the zip files of the above-mentioned JAR files to a location on your Windows system. Assume this location to be <MyFaces Dependencies>.

To incorporate the features of MyFaces in the EmplInfo application, the following JAR files, other than the ones added while developing EmplInfo application, must be added in the EmplInfo project library path:

**Table 14 Dependency JAR Files**

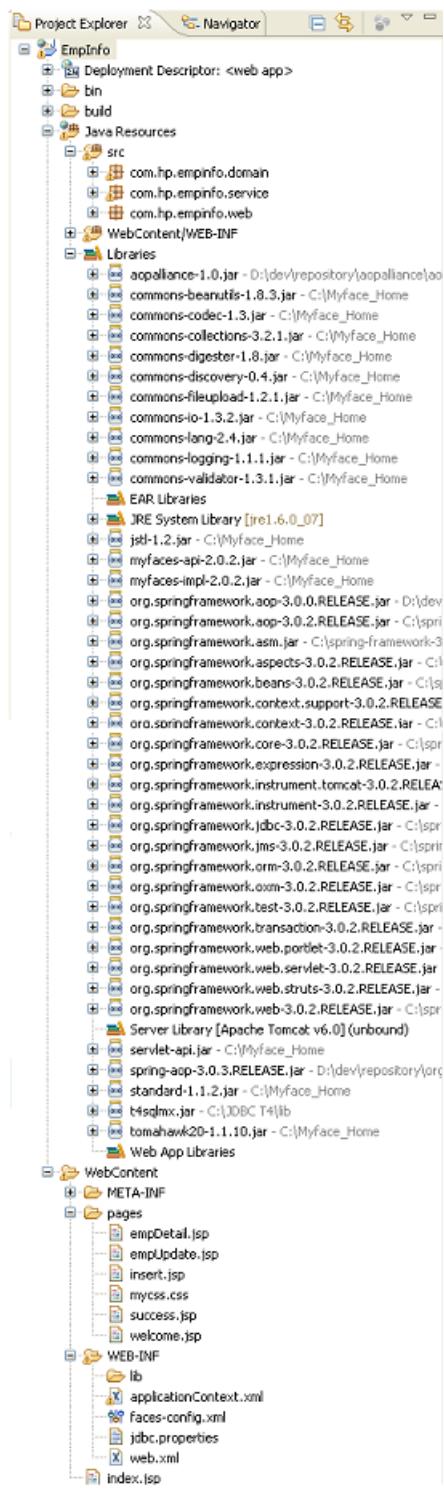
Dependency JAR Files	Source Location
commons-beanutils-1.7.0.jar	<MyFaces Home>\lib
commons-codec-1.3.jar	<MyFaces Home>\lib
commons-digester-1.8.jar	<MyFaces Home>\lib
commons-discovery-0.4.jar	<MyFaces Home>\lib
commons-el.jar	<MyFaces Dependencies>\commons-el-1.0
commons-fileupload-1.0.jar	<MyFaces Dependencies>\commons-fileupload-1.0
commons-validator-1.3.1.jar	<MyFaces Dependencies>\commons-validator-1.3.1
log4j-1.2.15.jar	<Spring Home>\lib\log4j
myfaces-api-1.2.5.jar	<MyFaces Home>\lib
myfaces-impl-1.2.5.jar	<MyFaces Home>\lib
tomahawk-1.1.8.jar	<MyFaces Dependencies>\tomahawk-1.1.8\lib

To add these dependency JAR files in the project library path and to resolve the J2EE module dependency on these JAR files, follow the instructions as explained in the [Adding Dependency JAR Files in the Project Library Path](#) section in the [Getting Started with Spring](#) chapter.

The EmplInfo application is now modified to integrate MyFaces into a Spring application.

Figure 177 shows the Project Explorer View.

## Figure 177 Project Explorer View



## Deploying EmplInfo on NonStop

This section describes the following tasks:

1. Creating the EmplInfo WAR File on Windows
2. Deploying the EmplInfo WAR File in NSJSP on NonStop

## Creating the EmplInfo WAR File on Windows

A WAR file is essential to deploy the web application. It contains property and configuration files, Java class file, and JSPs of the web application.

To create the application WAR file, complete the following steps:

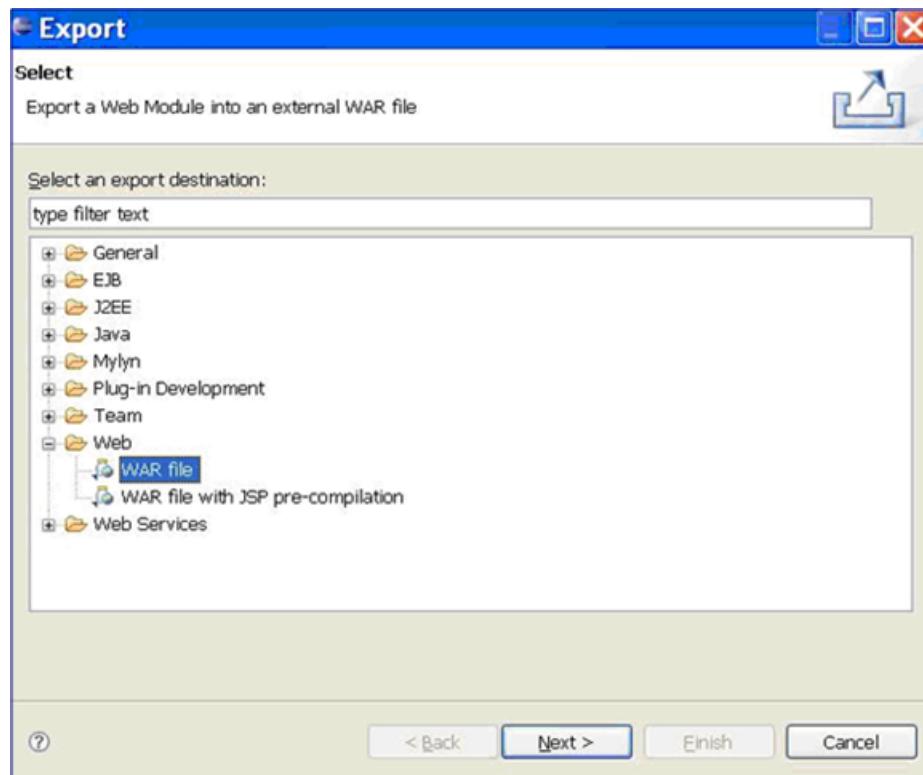
1. On the Project Explorer frame, right-click **EmplInfo** and select **Export > Export**.

The Export dialog box appears.

2. From the list of folders, select **Web > WAR file** and click **Next**.

Figure 178 shows the Export dialog box.

**Figure 178 Export Dialog Box**

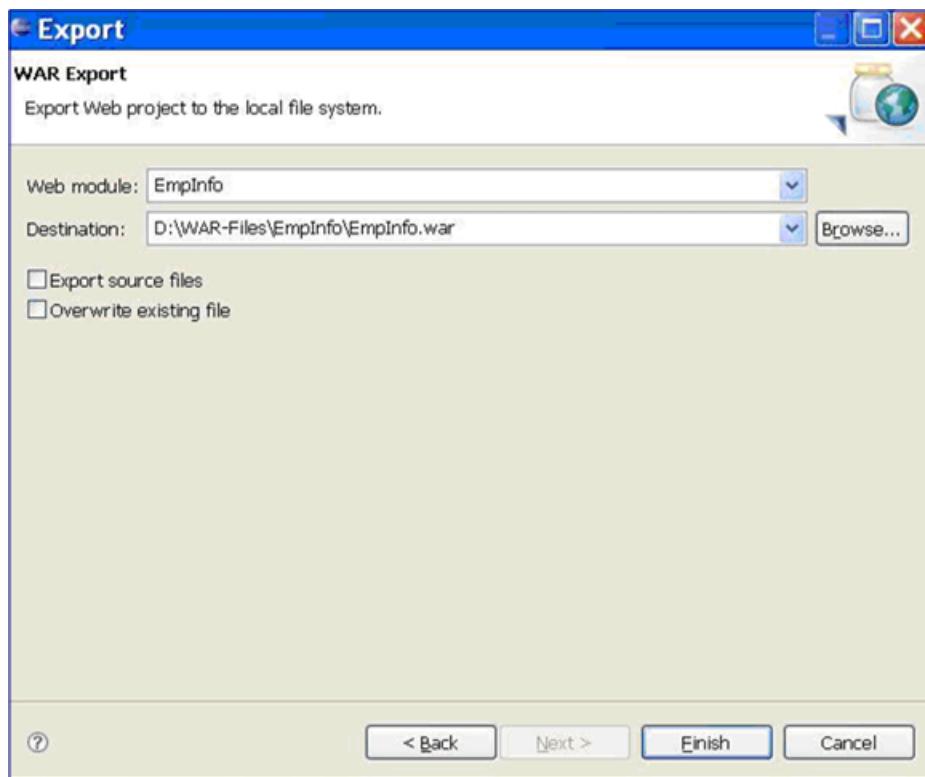


The WAR Export dialog box appears.

3. In the Web module field, enter **EmplInfo** and browse to the destination where you want to save the WAR file.

Figure 179 shows the WAR Export dialog box.

**Figure 179 The WAR Export dialog box**



4. Click **Finish**. The WAR file is created.

**NOTE:** If you have specified an existing name for the WAR file, the **Finish** button is disabled. In this case, change the name of the WAR file. If you want use the existing name, select the **Overwrite existing file** check box.

## Deploying the EmplInfo WAR File in NSJSP on NonStop

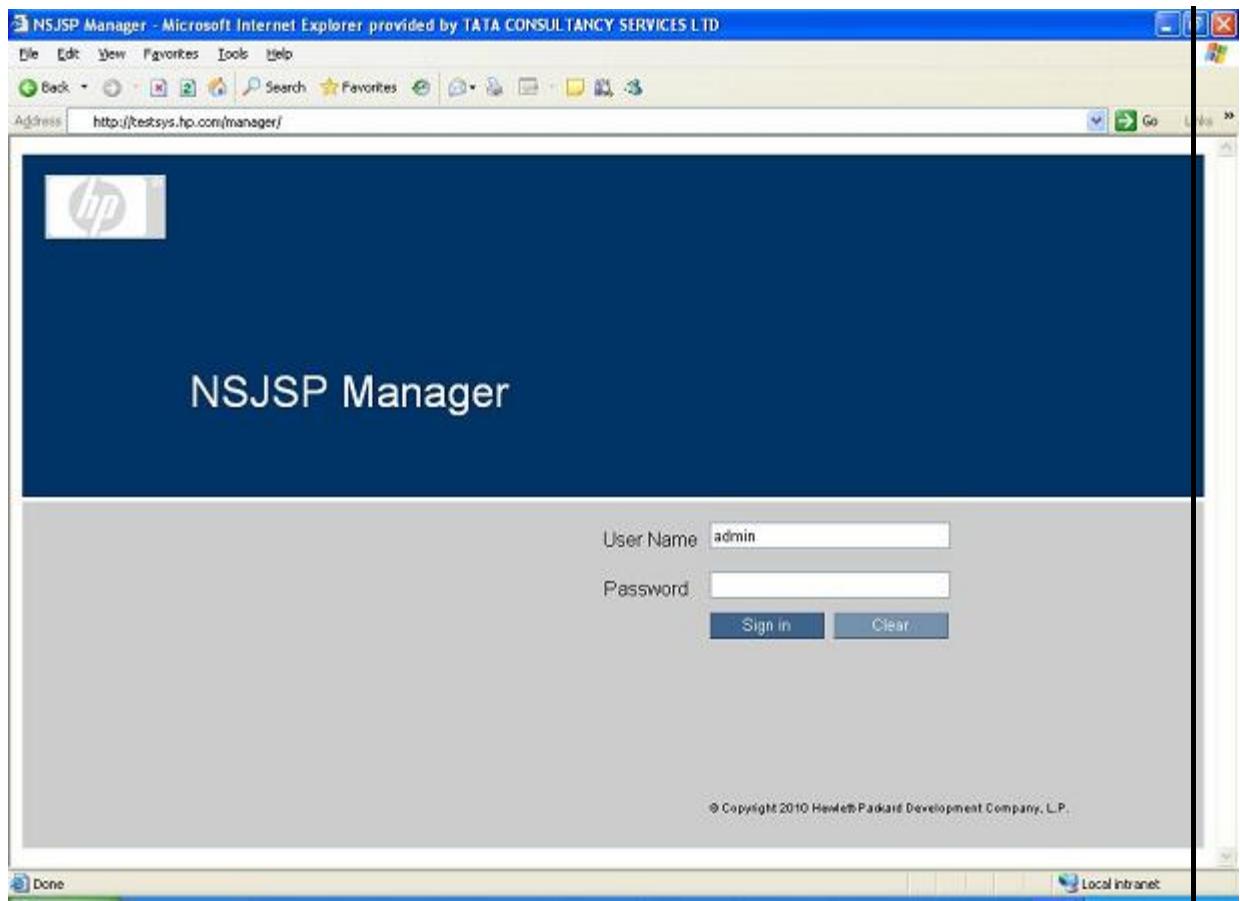
To deploy the EmplInfo WAR file on your NonStop system, complete the following steps:

1. Go to [http://<IP Address of the iTP WebServer>:<port#>/<manager\\_directory>](http://<IP Address of the iTP WebServer>:<port#>/<manager_directory>).

The **NSJSP Manager Login** screen appears.

Figure 180 shows the **NSJSP Manager Login** screen.

**Figure 180 NSJSP Manager Login Screen**

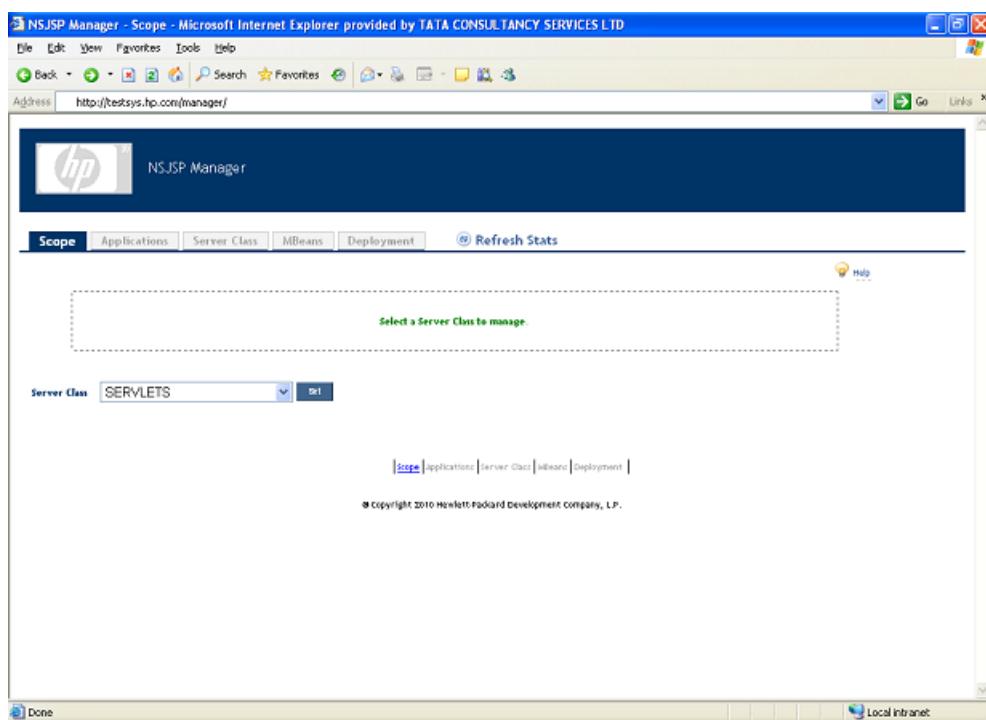


2. Enter the **User name:** and **Password:** and click **OK**.

The **NSJSP Manager** screen appears.

Figure 181 shows the **NSJSP Manager** screen.

**Figure 181 NSJSP Manager Screen**

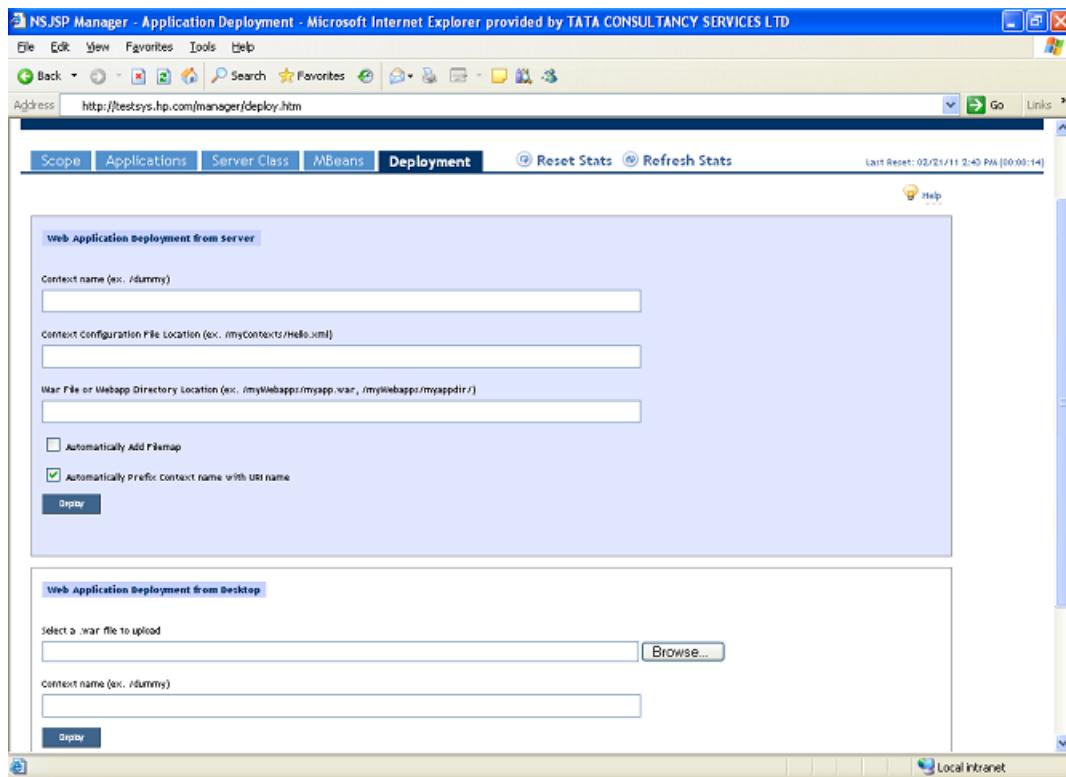


- Select **SERVLETS** for the **Server Class** and click **Set**.

The **Host (in server.xml)** tab is enabled and populated with **localhost**. Click **Set**, which enables the **Deployment** tab on the **NSJSP Manager** screen.

Figure 182 shows the **Deployment** tab of **NSJSP Manager**.

**Figure 182 NSJSP Manager Screen - Deployment tab**



- In the **Deployment** tab, complete the following steps in the **Web Application Deployment from Desktop** section:

- In the **Select a .war file to upload** field, click **Browse...** and locate the **EmplInfo.war** file on the Windows system.
- (Optional) In the **Context name** field, enter a name for the application context.
- Click **Deploy**.

EmplInfo is deployed on NSJSP and is listed under **Applications** tab of the **NSJSP Manager** screen.

**NOTE:** HP recommends that you use the NSJSP manager for deployment. Deployment using the FTP or any other mechanism is not recommended. For more information on using the NSJSP manager, see the *NonStop Servlets for JavaServer Pages (NSJSP) 6.1 System Administrator's Guide*.

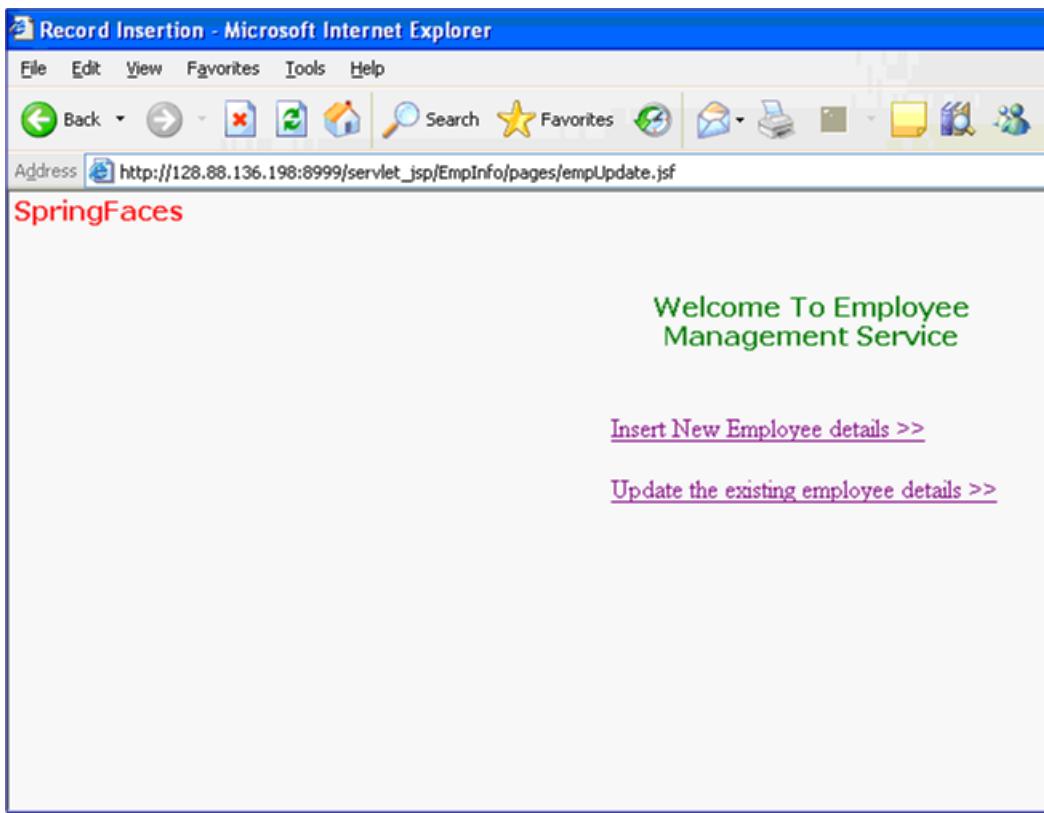
## Running EmplInfo on NonStop

To run EmplInfo on your NonStop system, click the **/< servlet directory>/EmplInfo** path under **Applications** in the **NSJSP Manager** screen.

When you access the above-mentioned URL, the Welcome to Employee Management Service screen appears.

Figure 183 shows the Welcome to Employee Management Service screen.

**Figure 183 Welcome to Employee Management Service Screen**



To insert new employee details, complete the following steps:

1. On the Welcome to Employee Management Service screen, click **Insert New Employee Details**. The **Insert Employee Details** screen appears.
- Figure 184 shows the Insert Employee Details screen.

**Figure 184 Insert Employee Details Screen**

Record Insertion - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites Home Print Mail

Address http://128.88.136.198:8999/servlet\_jsp/EmpInfo/pages/welcome.jsf

SpringFaces

Insert Employee Details

Employee Id: 0

firstname:

lastname:

age: 0

email:

submit

[Link to Home page >>](#)

2. Enter the required Employee details and click **Submit**.

Figure 185 shows an example of valid employee details.

**Figure 185 Insert Employee Details Screen**

Record Insertion - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites Home Print Mail

Address http://128.88.136.198:8999/servlet\_jsp/EmpInfo/pages/welcome.jsf

SpringFaces

Insert Employee Details

Employee Id: 1

firstname: test

lastname: user

age: 23

email: test@user.com

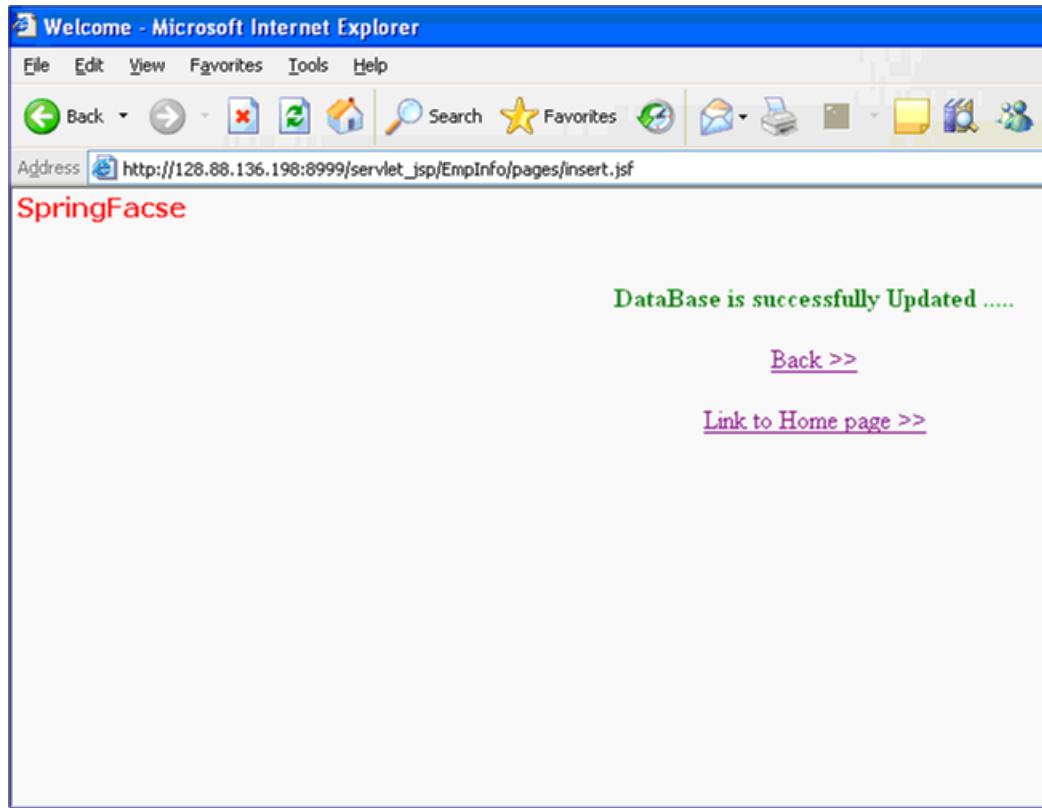
submit

[Link to Home page >>](#)

If the values in each field are valid, Database is successfully updated message appears.

Figure 186 shows the Success screen.

**Figure 186 Success Screen**



If you enter an existing Employee ID, the **Insert Employee Details** screen shows Employee already exists message.

Figure 187 shows the message in the Insert screen.

**Figure 187 Message Screen**

The screenshot shows a Microsoft Internet Explorer window titled "Record Insertion - Microsoft Internet Explorer". The address bar displays the URL [http://128.88.136.198:8999/servlet\\_jsp/EmpInfo/pages/insert.jsf](http://128.88.136.198:8999/servlet_jsp/EmpInfo/pages/insert.jsf). The page content is titled "SpringFaces" and "Insert Employee Details". A red error message "Employee already exists." is displayed above the form fields. The form fields include "Employee Id" (value: 1), "firstname" (value: user), "lastname" (value: test), "age" (value: 23), and "email" (value: test@user.com). A "submit" button is at the bottom right. Below the form is a link "Link to Home page >>".

If you enter invalid field values, the corresponding error messages appear on the **Insert Employee Details** screen.

Figure 188 shows the Error Message screen.

**Figure 188 Error Message Screen**

The screenshot shows a Microsoft Internet Explorer window titled "Record Insertion - Microsoft Internet Explorer". The address bar displays the URL [http://128.88.136.198:8999/servlet\\_jsp/EmpInfo/pages/insert.jsf](http://128.88.136.198:8999/servlet_jsp/EmpInfo/pages/insert.jsf). The page title is "SpringFaces". The main content is titled "Insert Employee Details". It contains fields for "Employee Id" (value: 2), "firstname" (value: a, with an error message: "Employee First Name can not be less than 2 characters."), "lastname" (value: b, with an error message: "Employee Last Name can not be less than 2 characters."), "age" (value: 102, with an error message: "Age cannot exceed 100."), and "email" (value: test, with an error message: "Invalid Email Address."). A "submit" button is present at the bottom. At the bottom of the page, there is a link "Link to Home page >>".

To update the existing employee details, complete the following steps:

1. Click **Update the existing employee details** on the Welcome screen. The Record Update screen appears.

Figure 189 shows the Record Update screen appears.

**Figure 189 Record Update Screen**

The screenshot shows a Microsoft Internet Explorer window titled "Record Updation - Microsoft Internet Explorer". The address bar contains the URL [http://128.88.136.198:8999/servlet\\_jsp/EmpInfo/pages/welcome.jsf](http://128.88.136.198:8999/servlet_jsp/EmpInfo/pages/welcome.jsf). The page itself has a header "SpringFaces" and a main section titled "Record Update". This section contains two input fields: "Employee Id" with value "0" and "Action to perform (Retrieve or Delete)" with an empty text area. Below these is a button labeled "Retrieve/Delete". At the bottom of the page is a link "Link to Home page >>".

2. Enter the required Employee ID and Action to be performed (Retrieve/Delete) and click **Retrieve/Delete**.

If you enter **Retrieve**, the Employee Information screen appears.

Figure 190 shows the Employee Information screen.

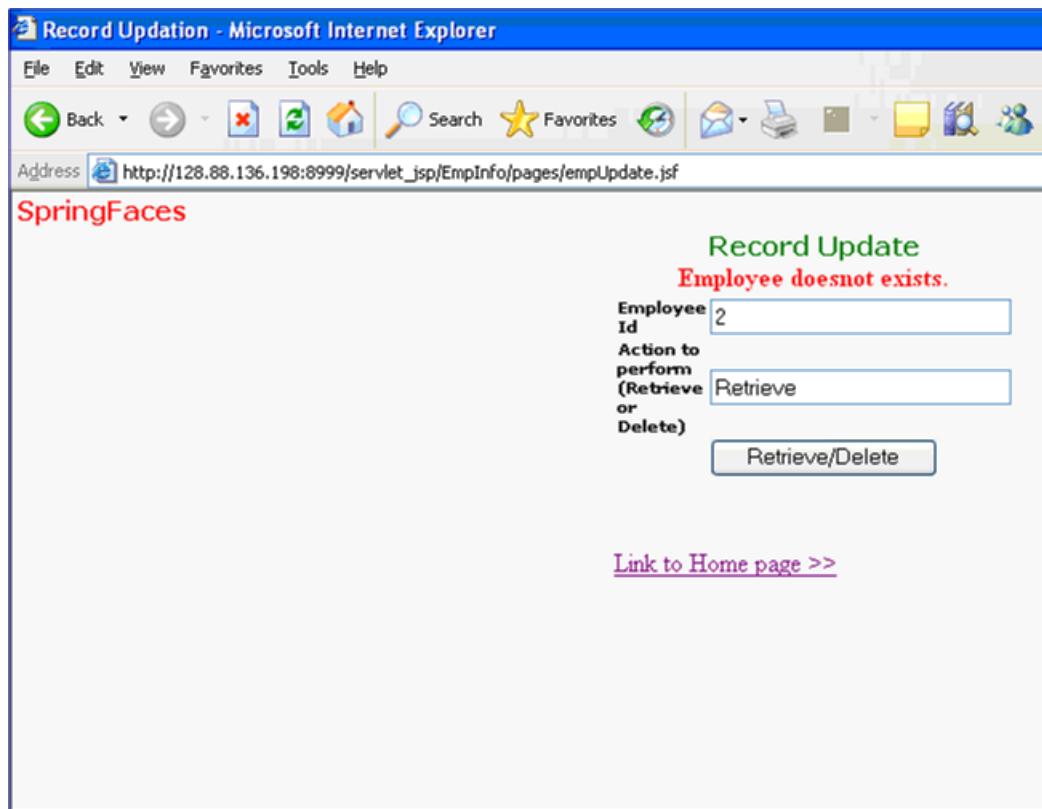
**Figure 190 Employee Information Screen**

The screenshot shows a Microsoft Internet Explorer window titled "Detail - Microsoft Internet Explorer". The address bar contains the URL [http://128.88.136.198:8999/servlet\\_jsp/EmpInfo/pages/empUpdate.jsf](http://128.88.136.198:8999/servlet_jsp/EmpInfo/pages/empUpdate.jsf). The page has a header "SpringFaces" and a section titled "Displaying Employee Details". Below this is a table titled "Employee Information" with columns: Employee Id, Fristname, Lastname, Age, and Email. A single row is shown with values: 1, test, user, 23, and test@user.com. At the bottom of the table is the message "The End". Below the table are links "Back >>" and "Link to Home page >>".

If you enter a non-existing Employee ID, the Record Update screen appears with the message Employee does not exist.

Figure 191 shows the Record Update: Message Screen.

**Figure 191 Record Update: Message Screen**



If you enter invalid field values, the corresponding error messages appears on the Record Update screen.

Figure 192 shows the Record Update: Error Message screen.

**Figure 192 Record Update: Error Message Screen**

The screenshot shows a Microsoft Internet Explorer window titled "Record Updation - Microsoft Internet Explorer". The address bar displays the URL [http://128.88.136.198:8999/servlet\\_jsp/EmpInfo/pages/empUpdate.jsf](http://128.88.136.198:8999/servlet_jsp/EmpInfo/pages/empUpdate.jsf). The page header "SpringFaces" is displayed in red. The main content area has a title "Record Update". It contains two input fields: "Employee Id" with value "1" and "Action to perform (Retrieve or Delete)" with value "delet". Below these fields is a button labeled "Retrieve/Delete". A red error message "Please give proper field value" and "Retrieve/Delete." is centered on the page. At the bottom, there is a link "[Link to Home page >>](#)".

3. If you wish to delete an Employee ID, complete the following details:

- In the Employee ID field, enter the employee ID you want to delete and in the Action to perform field, enter **Delete**. Click **Retrieve/Delete**.

Figure 193 shows the Record Update: Delete Employee screen.

**Figure 193 Record Update: Delete Employee Screen**

The screenshot shows a Microsoft Internet Explorer window titled "Record Updation - Microsoft Internet Explorer". The address bar displays the URL [http://128.88.136.198:8999/servlet\\_jsp/EmpInfo/pages/empUpdate.jsf](http://128.88.136.198:8999/servlet_jsp/EmpInfo/pages/empUpdate.jsf). The page content is titled "SpringFaces" and contains a "Record Update" form. The form has two input fields: "Employee Id" with value "1" and "Action to perform (Retrieve or Delete)" with value "Delete". Below the form is a button labeled "Retrieve/Delete". A message "Employee deleted" is displayed in red text. At the bottom of the page is a link "Link to Home page >>".

The Employee ID is deleted.

## 23 Frequently Asked Questions

- 23.23.1.1 Why do I get "blobTableName property is not set or set to null value or set to invalid value OR clobTableName property is not set or set to null value or set to invalid value error(s)" while running a JDBC application on NonStop system?  
BLOB and CLOB are not native data types in SQL/MX database. But, database administrators can create SQL/MX tables that have BLOB or CLOB columns by using the JDBC drivers for SQL/MX database or special SQL syntax in MXCI.  
Before running the JDBC application that uses BLOB or CLOB data through the JDBC API, the database administrator must create a LOB table. A LOB table actually contains the BLOB or CLOB data in chunks. The JDBC application specifies a LOB table name either through a system parameter or through a Java Property object by using one of the following properties, depending on the LOB column type:
- |      |               |
|------|---------------|
| BLOB | blobTableName |
| CLOB | clobTableName |
- 23.23.1.2 Why do I get the error "The statement was not prepared " while creating table containing a LOB column in the SQL/MX database on a NonStop system?  
Because BLOB and CLOB are not native data types in the SQL/MX database, you need to execute the following SQL Query to enable the creation of tables that have column containing LOB data type:  
`CONTROL QUERY DEFAULT JDBC_PROCESS 'TRUE'`
- 23.23.1.3 My application uses the auto-increment feature provided by Hibernate. However, the application fails to perform any insert operation. What can be the reason for this?  
NonStop SQL/MX does not support the auto-increment feature at database level. Therefore, if you mention "identity" or "sequence" as the generator class in the Hibernate application, it will not work.  
You can either disable the auto-incrementing feature in Hibernate and manually enter the values or use generator class other than "identity" or "sequence" in the Hibernate application such as "native" or "increment".
- 23.23.1.4 My application is using Hibernate for database operations on NonStop system. However, the operations performed by the application are not being reflected in the SQL/MX database. Why?  
The following reasons can be attributed to the replication of the problem:
- You are using a transaction API, but you have not committed the transaction. Without committing the transactions, the database will not be updated.
  - If you are not using transaction APIs, you have either not flushed or closed the session using which the database operations are performed.
- Ensure that you commit a transaction while using any transaction API and either flush the session or close the session.
- 23.23.1.5 What type of connection pooling should I use while using Hibernate as the ORM tool on a NonStop system?  
Hibernate offers built-in connection pooling, but it is non-scalable and non-fault-tolerant. Thus, it is not recommended that you do not use Hibernate's build-in connection pooling. Instead, use an external connection pool defined behind a DataSource such as Commons DBCP or C3P0.
- 23.23.1.6 Why do I get the error "The statement was not prepared" while a JDBC application execute an SQL statement on SQL/MX database on a NonStop system?

One of the most common reasons for this error is the use of "Reserved Words". There are a set of words reserved to be used by NonStop SQL/MX. To prevent syntax errors, avoid using these words as identifiers in NonStop SQL/MX SQL statements. In case you intend to use a reserved word as a column or table name, you must enclose the reserved word in double quotes ("") to access that column or object.

- 23.23.1.7 I see "org.apache.axis2.AxisFault: The endpoint reference (EPR) for the Operation not found" error while running a web service on Axis2/Java. What can be the reason?

This exception means that Axis2/Java could not find the service and the operation that the request is headed to. The endpoint reference mentioned in the request message is either invalid or empty. This led to a failure in selecting the intending operation to be invoked.

Make sure that the request contains the information that Axis2/Java could use to dispatch the request to the correct service and operation. Make sure that the configurations in services.xml file are as per the message body.

- 23.23.1.8 I have successfully deployed Axis2/Java WAR (axis2.war) under NSJSP. Why am I experiencing the error message "There was a problem in Axis2 version service, may be the service not available or some thing has gone wrong. But this does not mean system is not working! Try to upload some other service and check to see whether it is working " when I click on **Validate** link in the Axis2/Java Home Page? Make sure that the **Version** service exists on the system. On the home page, click the **Version** link under the **Services** section and verify that the WSDL for the Version service exists.

In case the version service is also not up, there were problems during the deployment. Rebuild the Axis2/Java WAR file on the Windows system and redeploy under NSJSP.

- 23.23.1.9 Why do I get a "ClassNotFoundException exception : org.springframework.instrument.classloading.tomcat.TomcatInstrumentableClassLoader" error when I run my Spring web application on a NonStop system?
- The Spring framework uses the concept of Load Time Weaving (LTW). It is the process of modification of byte code the moment a class is loaded from disk and placed into memory by a class loader. Thereafter, the byte code is evaluated and, if necessary, modified before it is made available to the application. The application context file contains the reference of class loader to be used.
- If the application context contains the reference for the class loader (TomcatInstrumentableClassLoader), make sure that the spring-tomcat-weaving.jar file is present in the <NSJSP Installation Directory>/lib directory.

- 23.23.1.10 I am using MyFaces for the application UI. Although I have used proper tags in the JSP file, the view of the page is not as expected. What can be the reason for this? Every JSF tag corresponds to a reference library. While using various JSF tags for UI components of the applications, you must ensure that all the required JSF tag libraries have been imported in the JSP file. For example: If you are using a core JSF UI component, you must have the following lines in the beginning of the JSP page to import the tag library of JSF core:

```
<%@ taglib uri="http://java.sun.com/jsf/core"
```

- 23.23.2.1 Why do I get a "ClassNotFoundException exception :<class file reference>" error when I run my SASH applications on a NonStop system?
- The application uses classes which are defined in one or more JAR files. This error appears when any of these dependency JAR files is not added into the CLASSPATH.

If the application is a web application developed on Windows using Eclipse Galileo IDE, then you should add all the dependency JAR files in the project library path and resolve the J2EE module dependency on these JAR files using the steps described in the “[Adding Dependency JAR Files in the Project Library Path](#)” (page 89) of the “[Getting Started with Spring](#)” (page 75) chapter. However, if the application is not a web application, you need to add the dependency JAR files into the CLASSPATH.

- 23.23.2.2** Why do I get the Resource not found error while accessing the home page of any web application deployed under NSJSP 6.0?

This error can be attributed to various reasons, such as absence of files in the specified location, insufficient permissions on the file, improper deployment and so on. However, you can perform the following checks:

1. Make sure that you have followed all the deployment steps correctly.
2. Verify that the application is deployed successfully. You can ensure this by checking that the application folder is successfully created under the *<NSJSP Installation Directory>/webapps* directory. However, if the application folder is not created, check whether the values of *unpackWARs* and *autoDeploy* are set to true in the *server.xml* file under *<NSJSP Installation Directory>/conf* directory.
3. If the application WAR file is properly deployed, then check the configurations in the *web.xml* file of the application.
4. If the above steps do not yield any results, then restart the iTP WebServer.

- 23.23.2.3** I have deployed my web application archive (WAR) file in NSJSP; however, I am not able to access the application using the web browser. How can I verify if the environment for my application is properly running or not?

Following is the list of the actions to confirm that the environment for the application is proper:

1. Verify that the application was successfully deployed in NSJSP. To verify, confirm the creation of application folder in *<NSJSP Installation Directory>/webapps* directory.
2. Check that the iTP WebServer is started. You can execute the  
`ps -A | grep httpd`  
command on OSS to check that the instance of iTP WebServer used by you is running.
3. Check that the servlet instance and httpd processes are running under the PATHMON, without any error. You can use `status server *` inside the iTP WebServer PATHMON.
4. Check that the rout processes are running on the system. You can use `status *,prog *.*.rout` on the guardian environment.

- 23.23.2.4** Why do I get the following error while running Hibernate sample applications?

```
INFO SettingsFactory:340 - Check Nullability in Core
(should be disabled when Bean Validation is on): enabled
INFO SessionFactoryImpl:199
- building session factory Exception in thread "main"
org.hibernate.HibernateException: Unable to instantiate default tuplizer
[org.hibernate.tuple.entity.PojoEntityTuplizer] at
org.hibernate.tuple.entity.EntityTuplizerFactory.constructTuplizer(EntityTuplizerFactory.java:110)

at
org.hibernate.tuple.entity.EntityTuplizerFactory.constructDefaultTuplizer(EntityTuplizerFactory.java:135)

at org.hibernate.tuple.entity.EntityEntityModeToTuplizerMapping.<init>
(EntityEntityModeToTuplizerMapping.java:80) at org.hibernate.tuple.entity.EntityMetamodel.<init>
(EntityMetamodel.java:323)
```

This error message is displayed if `javassist-3.9.0.GA.jar` is not present in the CLASSPATH. Copy the `javassist-3.9.0.GA.jar` file to the CLASSPATH and run the sample application again.

# Glossary

## A

<b>Arglist</b>	Specifies the argument list passed to an object at the start time or at the runtime.
<b>AXIOM</b>	AXIOM (AXIls2 Object Model) is the base of Axis2, where any incoming SOAP message is represented as an object model called AXIOM inside Axis2. The advantage of AXIOM over other XML inforset representations is that it is based on PULL parser technique, while most others are based on PUSH parser technique. The main difference of PULL over PUSH is that in PULL technique invoker has the full control on the parser and can ask for the next event, where as in the case of PUSH, when the parser is asked to proceed it will fire the events until it reach the end of the document.
<b>Axis2/Java</b>	A core engine for Web services. It is a complete re-design and re-write of the widely used Apache Axis. Provides the capabilities to add web service interfaces to web applications and simultaneously, it can act as standalone server application.

## B

<b>BeanFactory</b>	The BeanFactory provides an advanced configuration mechanism capable of managing beans (objects) of any nature, using potentially any kind of storage facility. The ApplicationContext builds on top of the BeanFactory (a subclass) and adds other functionality, such as easier integration with the Spring AOP features, message resource handling (for use in internationalization), event propagation, declarative mechanisms to create the ApplicationContext and optional parent contexts, and application-layer specific contexts such as the WebApplicationContext, among other enhancements.
--------------------	--

## C

<b>Connector Element</b>	Represents a connector component that supports HTTP/1.1 protocol.
<b>Connector Threads</b>	Specifies the connector instances of NSJSP
<b>Cookies</b>	Small string of text stored on a computer by a web browser. Web servers send it to the web clients as an HTTP header for the purpose of authentication and session tracking.

## E

<b>End Point Reference</b>	An endpoint reference (EPR) is a combination of Web services (WS) elements that define the address for a resource in a Simple Object Access Protocol (SOAP) header. In this context, an endpoint is any user device connected to a network. Endpoints can include personal computers (PCs), personal digital assistants (PDAs) and specialized equipment such as inventory scanners and point-of-sale terminals.  An EPR consists of a Uniform Resource Identifier (URI), message reference parameters and data concerning the interface to be used. The EPR may also contain a set of policies relevant to the endpoint such as its intended behavior and capabilities.
<b>Executor Element</b>	Represents a thread pool that can be shared between components in NSJSP.

## H

<b>Hibernate</b>	Is an object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database
<b>HTTP Clients</b>	Is a software application for retrieving, presenting, and traversing information resources on the World Wide Web using HTTP protocol. Commonly called as web browsers

## I

<b>Inversion of Control</b>	Inversion of control, or IoC, is an abstract principle describing an aspect of some software architecture designs in which the flow of control of a system is inverted in comparison to procedural programming.
-----------------------------	---

In traditional programming the flow is controlled by a central piece of code. Using Inversion of Control this central control as a design principle is left behind. Although the caller will eventually get its answer, how and when is out of control of the caller. It is the callee who decides to answer how and when.

## J

**J2EE Environments** J2EE Environment consist of J2EE platform that represents a single standard for implementing and deploying multi-tier enterprise applications. Typically, such applications are configured as a client tier to provide the user interface, one or more middle tiers that provide client services and business logic for an application, and a data tier consisting of backend enterprise information systems (EIS) that provide data management.

**Java Runtime Arguments** A Java application can accept any number of arguments from the command line. This allows the user to specify configuration information when the application is launched. The user enters command-line arguments when invoking the application and specifies them after the name of the class to be run. For example, suppose a Java application called Sort sorts lines in a file. To sort the data in a file named `friends.txt`, a user would enter: `java Sort friends.txt`.

**JavaBean** JavaBeans technology is the component architecture for the Java 2 Platform, Standard Edition (J2SE). Components (JavaBeans) are reusable software programs that you can develop and assemble easily to create sophisticated applications. JavaBeans technology is based on the JavaBeans specification.

## M

### Model-View-Controller

Model–View–Controller (MVC) is an architectural pattern used in software engineering. Successful use of the pattern isolates business logic from user interface considerations, resulting in an application where it is easier to modify either the visual appearance of the application or the underlying business rules without affecting the other. In MVC, the model represents the information (the data) of the application; the view corresponds to elements of the user interface such as text, checkbox items, and so forth; and the controller manages the communication of data and the business rules used to manipulate the data to and from the model.

**Module File Caching** The Module File Caching (MFC) feature shares the prepared statement plans among the NonStop SQL/MX database connections. It helps in reducing the SQL compilation time during the steady state of the JDBC or ODBC application, thereby reducing resource consumption.

**MyFaces** An Apache Software Foundation project that creates and maintains an open-source JavaServer Faces implementation.

## N

**NonStop** The HP operating system environment, which consists of core and system services. The operating system does not include any application program interfaces (APIs).

**NSJSP** NonStop Servlet for Java Server Pages 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.

## P

**PATHMON** The central controlling process for a NonStop TS/MP application.

**Persistent Manager** Represents the session manager that creates and maintains HTTP sessions as requested by the associated web application.

## S

**Scout for NonStop Servers** Scout is a vital tool for anyone involved in NonStop server system management. Scout for NonStop Servers enables you to view, research and download Software Product Revisions (SPRs) as well as to request Site Update Tapes (SUTs) and Independent Product (IP) CDs for the systems. In

addition, you can use it to view Hotstuff messages, Support Notes, SPRs for special consideration, contents of available RVUs, and to review what software products are licensed for the systems.

### **SessionBasedLoadBalancing**

A feature of NSJSP by virtue of which NSJSP Container maintains sessions for load balancing purposes.

### **SOAP**

SOAP, Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on Extensible Markup Language (XML) as its message format, and usually relies on other Application Layer protocols (most notably Remote Procedure Call (RPC) and HTTP) for message negotiation and transmission. SOAP can form the foundation layer of a web services protocol stack, providing a basic messaging framework upon which web services can be built.

### **Spring**

An open source application framework for the Java platform. Apart from the core features for application development, it provides extensions for building web applications on top of the Java Enterprise platform.

### **Struts**

Apache Struts is an open-source web application framework for developing Java EE web applications. It uses and extends the Java Servlet API to encourage developers to adopt a model-view-controller (MVC) architecture. The goal of Struts is to cleanly separate the model (application logic that interacts with a database) from the view (HTML pages presented to the client) and the controller (instance that passes information between view and model). Struts provides the controller (a servlet known as ActionServlet) and facilitates the writing of templates for the view or presentation layer (typically in JSP, but XML/XSLT and Velocity are also supported). The web application programmer is responsible for writing the model code, and for creating a central configuration file struts-config.xml which binds together model, view and controller.

## T

### **TS/MP Specific Parameters**

The parameters which are offered by the TS/MP for tuning the applications on NonStop systems.

## W

### **WebWork**

WebWork is a Java web-application development framework. It is built specifically with developer productivity and code simplicity in mind, providing robust support for building reusable UI templates, such as form controls, UI themes, internationalization, dynamic form parameter mapping to JavaBeans, robust client and server side validation, and much more.

# Index

## Symbols

\$datavol, 42  
\$RECEIVE, 53, 171, 241, 308  
-DSaveSessionOnCreation, 57, 175, 245, 312  
-DSessionBasedLoadBalancing, 58, 176, 246, 313  
-Xmx, 62, 180, 250, 317  
-Xmx maximum-stack-size, 62, 180, 250, 317  
-Xnklassgc, 62, 180, 250, 317  
-Xss, 62, 180, 250, 317  
/ <servlet directory>/EmplInfo, 382  
/dbconfig, 223  
/etc, 135, 223  
/insert.htm, 88  
/mxci, 135  
16-processor, 55, 173, 243, 310  
<\$datavol>, 47, 161, 166  
<aop:config>, 355  
<Axis2 Home>, 292, 403  
<body>, 97  
<c:out>, 97  
<Eclipse IDE Installation directory>, 253  
<h:commandButton>, 277  
<h:selectBooleanCheckbox>, 269  
<h:selectOneMenu value="#{resumeBean.bgColor}">, 269  
<h:selectOneMenu>, 269  
<Hibernate Entity Manager Home>, 391  
<Hibernate Home>, 154  
<id>, 212  
<mapping resource>, 182  
<My SASH Home>, 37, 154, 233, 292  
<MyFaces Dependencies>, 263, 443  
<MyFaces Home>, 233  
<navigation-rule>, 277  
<node.\$vol>, 48, 161, 166  
<NonStop Axis2 Home>, 292  
<NonStop Hibernate Home>, 154  
<NonStop SASH Home>, 42  
<NonStop Spring Home>, 37  
<NSJSP Deployment Directory>, 37  
<NSJSP Deployment Directory>/conf/server.xml, 53, 171, 241, 308  
<param-value>, 136  
<Rampart Home>, 292  
<Sandesha2 Home>, 292  
<servlet-mapping/>, 262  
<servlet-mapping>, 85  
<servlet-name/>, 86  
<session-factory>, 181, 213  
<soap12>, 305  
<soap>, 305  
<Spring Dependency Home>, 37  
<Spring Home>, 37, 403  
<Spring Home>/lib/c3p0, 72  
<subvol reference>, 48, 161, 166  
<tx:advice>, 355

<user>, 47, 161, 166  
<webapps>, 262  
<welcome-file-list>, 262  
<welcome-file>, 85, 262  
@Basic, 386  
@Entity, 386  
@Id, 386  
\annotations, 155  
\bin, 293  
\conf, 293  
\db, 46  
\dbconfig, 159, 164  
\dist, 37  
\docs, 37  
\documentation, 155  
\entitymanager, 155  
\etc, 40, 164  
\lib, 155, 234, 293  
\project, 155  
\projects, 37  
\repository\modules, 293  
\repository\services, 293  
\samples, 293  
\setup, 159  
\src, 38, 40, 46, 159, 164, 235  
\webapp, 293  
\WebContent, 235

## A

Adding Dependency JAR Files, 262  
Ant, 24  
AOP Configurations, 354  
| AOP Proxy Factory Bean, 354  
Apache C3P0, 190  
Apache C3P0 0.9.1.2, 190  
Apache JMeter, 53, 171, 241, 308  
API package, 189  
applicationContext-dataSource.xml, 137  
applicationContext-hibernate.xml  
Changes to applicationContext-dataSource.xml, 137  
applicationContext.xml, 62, 106, 351, 370, 401  
archive file (.aar), 318  
Arglist, 57, 175, 245, 312  
auctioncat, 160  
auctionsch, 160  
Average Load, 52, 170, 240, 307  
Average Response Time, 52, 170, 240, 307  
axis2.war, 293  
axis2.xml, 317  
Axis2/Java, 292, 319  
Configuring, 307  
Configuring iTP WebServer for Axis2/Java, 308  
Configuring NSJSP for Axis2/Java, 312  
Determining Application Parameters, 307  
Determining Maximum Capacity of NSJSP Instance, 307

Installing Axis2/Java, 292  
     Building, 293  
     Deploying, 294  
     Downloading, 292  
     Running, 296  
     TemperatureConverter, 319  
**AxisClient**, 324, 342  
**AxisClient/src**, 328

**B**

**BankService**, 301  
**BankService.aar**, 301  
**Bean Instance**, 355  
**Bid.hbm.xml**, 184  
**build.xml**, 402  
**Business Layer**, 421  
**Business Logic**, 102

**C**

**c2fConversion**, 320  
**c3p0-0.9.1.2.jar**, 72  
**c3p0-0.9.1.jar**, 190  
**Caching**, 190  
**Caveat Emptor**, 159  
     Building, 159  
     Customizing, 221  
         Added Files, 221  
         Modified File, 223  
     Deploying, 163  
     Running, 163  
     Setting, 161  
**Caveatemptor.jar**, 160  
**Caveatemptor\_script.sql**, 222  
**checkEmp**, 426  
**CLASSPATH**, 72, 190  
**CLASSPATHs**, 162  
**com.hp.empinfo.domain**, 376  
**com.hp.empinfo.service**, 351, 425  
**com.hp.empinfo.web**, 92  
**com.hp.imp**, 206  
**com.mchange.v2.c3p0.ComboPooledDataSource**, 64, 66  
**com.tandem.jta**, 189  
**com.tandem.sqlmx.SQLMXDriver**, 64, 181, 183  
**commons-el.jar**, 263  
**commons-lang-2.4.jar**, 263  
**Configuration**, 275  
     faces-config.xml, 275  
**Connection Pooling**, 72, 189  
**Connection URL**, 63, 183  
**Connector**, 53, 171, 241, 308  
**Connector Element**, 59, 177, 247, 314  
**Connector Threads**, 58, 176, 246, 313  
**ContextLoaderListener**, 111  
**ContextLoaderServlet**, 423  
**Controller**, 91  
**createPayLoad1()**, 330  
**createPayLoad2()**, 330  
**css**, 265  
**customize.jsp**

**Creating customize.jsp**, 268  
**Modifying customize.jsp**, 268

**D**

**Data Access Object**, 103  
**Data Access Object (DAO)**, 367  
**dataAccessContext-local.xml**, 140  
     Changes to dataAccessContext-local.xml, 140  
**DataList**, 234  
**deleteEmployee**, 354  
**deleteresult.jsp**, 116, 118  
**Dependency JAR Files**, 359, 376, 391, 403, 415, 443  
**DispatcherServlet**, 85  
**Documents Distribution**, 293  
**dynamic capacity**, 52, 170, 240, 307

**E**

**Eclipse Galileo IDE**, 29  
**eclipse-SDK-3.3.1-win32.zip**, 29  
**eclipse.exe**, 76, 197, 253  
**EJB 2.x entity beans**, 383  
**ejb3-persistence.jar**, 391  
**EmpBean.java**, 427  
**empDetail.jsp**, 436  
**EmplInfo**, 75, 367, 383, 398, 421  
     Deploying, 129  
     Deploying EmplInfo, 362, 379, 394, 404, 444  
     Developing, 76  
     Modifying EmplInfo, 349, 368, 384, 399, 422  
     Running, 133  
     Running EmplInfo, 366, 382, 397, 408, 448  
**EmplInfo-servlet.xml**, 86, 100  
     Creating EmplInfo-servlet.xml, 86  
**EmplInfo/src**, 372  
**EmplInfo/WebContent**, 81  
**empinfocat**, 109  
**EmplInfoClient**, 409  
**EmplInfoClient.java**, 412  
**empinfosch**, 109  
**EmplInfoService.aar**, 404, 406  
**Employee.hbm.xml**, 182, 209, 372, 391  
**Employee.java**, 102, 206  
**EmployeeController**, 92  
**EmployeeController.java**, 93, 351, 353  
**EmployeeDao**, 370  
**EmployeeDao class**, 106  
**EmployeeDao.java**, 119, 371  
**EmployeeInfo**  
     Developing, 197  
         Adding Dependency JAR Files, 201  
         Creating Class File, 204  
         Creating Main Class, 206  
         Creating New Java Project, 198  
         Creating Package, 204  
     Running, 220  
     Setting Up, 218  
**EmployeeInfo**,  
     Developing  
         Creating the Eclipse Workspace, 76, 197

employeeinfocat, 215  
employeeinfosch, 215  
EmployeeRowMapper, 118, 375  
empUpdate.jsp, 435  
EntityEmployee.java, 386  
Environment.HBM2DDL\_AUTO, 223  
error-maximum-connection, 57, 175, 245, 312  
eventcat, 165  
EventManager  
    Customizing, 223  
        Added Directories, 223  
        Added Files, 224  
        Modified File, 225  
eventmanager\_script.sql, 224  
Eventmgr, 163  
eventsch, 165  
Executor, 53, 171, 241, 308  
Executor Element, 59, 177, 247, 314

**F**

f2cConversion, 320  
faces-config.xml, 441  
FaultHandling, 301  
    Building, 301  
    Deploying, 301  
    Running, 303

**G**

getEmployeeDetail, 354  
Global Configuration, 317

**H**

handleRequest, 110  
HandlerMapping, 88  
Hibernate  
    Configuring  
        Determining Maximum Capacity of NSJSP Instance, 170  
        EmployeeInfo, 196  
    Hibernate, 153, 196  
    Configuring, 52, 170  
        Configuring iTP WebServer for Hibernate, 171  
        Configuring NSJSP for Hibernate, 175  
        Determining Application Parameters, 170  
    Installing Hibernate, 153  
        Copying, 157  
        Downloading Hibernate Dependency JARs, 155  
        Downloading Hibernate Distribution, 154  
        Including Hibernate Dialect, 157  
    Hibernate Dialect, 182  
    Hibernate Entity Manager, 391  
    Hibernate Query Language (HQL), 207  
    Hibernate SessionFactory, 370, 388  
    hibernate-configuration-3.0.dtd, 183  
    hibernate-entitymanager-3.2.0.zip, 391  
    Hibernate.cfg.xml, 225  
    hibernate.cfg.xml, 180, 212  
    hibernate.connection.catalog, 160, 165  
    hibernate.connection.password, 160, 164

hibernate.connection.schema, 160, 165  
hibernate.connection.username, 160, 164  
hibernate.properties, 160, 164, 180, 213, 221, 224  
hibernatehome, 162  
HP LoadRunner, 53, 171, 241, 308  
HTTP clients, 53, 171, 241, 308  
HTTP cookies, 53, 171, 241, 308  
httpd, 240  
httpd.config, 54, 172, 242, 309

**I**

IEmployeeDao.java, 425  
include.jsp, 96  
index.jsp, 441  
    Creating index.jsp, 80, 258  
    Modifying index.jsp, 83  
insert.jsp, 94, 433  
insertEmployee, 354  
insertresult.jsp, 103  
int empid, 426  
Integrating Axis2/Java, 398  
Integrating Hibernate, 367  
Integrating JPA with Hibernate, 383  
Integrating MyFaces, 421  
Inversion of Control (IOC), 367  
Item.hbm.xml, 184  
Item.xml, 139  
    Changes to Item.xml, 140  
IVY, 27

**J**

Java Data Objects (JDO), 367, 383  
Java Development Kit (JDK), 23  
Java Persistence API (JPA), 383  
Java Runtime Arguments, 61, 179, 249, 316  
JavaServer Faces (JSF), 231  
javassist.jar, 391  
javax.faces.webapp.FacesServlet, 423  
javax.transaction, 189  
JDBC Configuration, 144  
JDBC Type 2 Driver, 181  
JDBC Type 2 Driver Class, 63  
JDBC Type 2 Driver for NonStop SQL/MX, 22  
JDBC Type 4 Driver, 65, 185  
JDBC Type 4 Driver for NonStop SQL/MX, 26  
jdbc.properties, 41, 62, 138, 140, 401  
    Changes to jdbc.properties, 138, 140  
jdbc: protocol, 63, 182  
jdbc:sqlmx://, 63, 182  
JdbcDaoSupport, 107  
JPA EntityManagerFactory, 388  
JPetStore, 46  
    Building, 46  
    Customizing, 139  
        Added File, 139  
        Modified Files, 139  
    Deploying, 48, 167  
    Running, 51  
    Setting, 47

jpetstore.war, 47  
jpetstore\_dataload\_script.sql, 48  
jpetstore\_tables\_script.sql, 47  
jpetstorecat, 48  
jpetstoresch, 48  
JSESSIONID, 57, 175, 245, 312  
JSP pages, 95  
JSP Standard Tag Library (JSTL), 95  
jstl.jar, 263  
JTAFactory, 189

## K

kilobytes, 62, 180, 250, 317

## M

main(), 330  
Main.java, 223  
Managed Beans  
    Creating Managed Beans, 271  
managed beans, 427  
Maven, 25  
maxIdleTime, 53, 171, 241, 308  
Maximum Load, 52, 170, 240, 307  
maximum-heap-size, 62, 180, 250, 317  
maximum-stack-size, 62, 180, 317  
Maxlinks, 53, 171, 240, 308  
Maxservers, 52, 54, 170, 172, 240, 242, 307, 309  
maxThreads, 53, 171, 241, 308  
megabytes, 62, 180, 250, 317  
message.properties, 431  
MessageFactory.java, 430  
MessageReceivers, 321  
META-INF, 318, 321  
minSpareThreads, 53, 171, 241, 308  
ModelAndView, 91  
Module Configuration, 318  
Module File Caching, 73, 190  
    Configuring NonStop SQL/MX DataSource, 73, 190  
    Modifying the Hibernate Application, 191  
    Modifying the Spring Application, 74  
module.xml, 318  
MTOM, 303  
    Building, 303  
    Configuring, 305  
    Deploying, 304  
    Running, 305  
MTOMSample, 303  
MTOMSample.wsdl, 305  
MVC Web application, 398  
mycss.css, 439  
MyFaces, 233  
    Configuring, 240  
        Configuring iTP WebServer for MyFaces, 241  
        Configuring NSJSP for MyFaces, 245  
        Determining Application Parameters, 240  
        Determining Maximum Capacity of NSJSP Instance, 240  
    Installing MyFaces, 233  
    Installing MyFaces Tomahawk

    Downloading, 286  
    Installing MyFaces Trinidad  
        Copying, 285, 286  
        Downloading, 285  
    MyFaces distribution, 234  
    MyFaces Getting Started, 252  
    myfaces-components, 234  
        Building, 235  
        Deploying, 235  
        Running, 237  
    myfaces-components.war, 235  
    myfaces-impl-2.0.2.jar, 234  
    myfaces-impl1-2.0.2.jar, 234

## N

Navigation model, 252  
New Java Class, 272  
    ResumeBean  
        Creating, 272  
        Modifying Java Class, 273  
New Package, 271  
    coreservlets, 271  
node.\$vol, 42  
NonStop iTP WebServer, 19  
NonStop Platform, 52, 170, 240, 307  
NonStop Server for Java (NSJ), 22  
NonStop Servlets for Java Server Pages (NSJSP), 20  
NonStop SQL/MX, 21  
NonStop system, 18, 36, 75, 153, 233, 252, 292, 319  
Numstatic, 52, 54, 170, 172, 240, 242, 307, 309

## O

OMELEMENT, 330  
onSubmit, 110  
Order.xml, 141  
    Changes to Order.xml, 141  
org.apache.catalina.core.StandardThreadExecutor, 59, 177, 247, 314  
org.apache.catalina.Executor, 59, 177, 247, 314  
org.apache.commons.dbcp.BasicDataSource, 64, 66  
org.hibernate.auction.Main, 163  
org.hibernate.dialect.SqlmxDialect, 182  
org.springframework.jdbc.datasource.DriverManagerDataSource, 64, 66  
org.springframework.transaction.PlatformTransactionManager, 348  
org.springframework.web.servlet.mvc, 92  
Overview, 196, 319

## P

Package, 91  
PATHMON, 54, 172, 242, 309  
persistence.xml, 386  
Persistent Manager, 58, 176, 246, 313  
PetClinic, 40  
    Building, 40  
    Customizing, 135  
        Added Directory, 135  
        Added File, 135

Modified Files, 136  
Deploying, 43  
Running, 45  
Setting, 42  
`petclinic.hbm.xml`, 137  
Changes to `petclinic.hbm.xml`, 137  
`petclinic.war`, 41  
`petclinic_dataload_script.sql`, 43  
`petclinic_tables_script.sql`, 43  
placeholder, 64  
Plain Old Java Object (POJO), 421  
platform-independent, 367  
PlatformTransactionManager, 356  
PlatformTransactionManagerImplementation, 348  
`Pom.xml`, 141  
Changes to `Pom.xml`, 142  
Prerequisites, 18, 36, 75, 153, 196, 233, 252, 292, 319, 349, 367, 383, 398, 421  
Presentation Layer, 421  
Processors, 55, 173, 243, 310  
PropertyPlaceholderConfigurer, 64, 66, 401

**R**

Radview Webload, 53, 171, 241, 308  
RawXML MessageReceivers, 321  
Receive Depth, 60, 178  
results, 265  
ResultSet, 118  
resumeBean, 269  
`ResumeBean.java`, 273  
`retrieveordelete.jsp`, 116  
`retrieveresult.jsp`, 116  
RowMapper, 118  
RPC MessageReceivers, 321

**S**

same-color.jsp  
Creating `same-color.jsp`, 269  
Modifying `same-color.jsp`, 269  
`sample-mtom.aar`, 304  
`SAMPLES.zip`, 77, 254  
schema location, 355  
`server.xml`, 59, 177, 247, 314  
Service Configuration, 318  
`ServiceFinder.java`, 427  
ServiceLifeCycle, 400  
`services.xml`, 318, 320, 401  
servlet, 421  
`servlet-api.jar`, 263  
`servlet.ssc`, 57, 175, 245, 312  
Session, 367  
session-ID, 57, 175, 245, 312  
SessionBasedLoadBalancing, 57, 312  
SessionFactory, 182, 367  
`SetDatabase.java`, 135, 139  
setenv, 221  
`show-preview.jsp`  
Creating `show-preview.jsp`, 269  
Modifying `show-preview.jsp`, 269

Simple Object Access Protocol (SOAP), 398  
Simple POJO, 102  
SimpleFormController class, 92  
SkinSelector, 252  
Deploying, 278  
Developing, 252  
Running, 282  
Source Distribution, 293  
`sourceforge.net`, 154  
Spring, 36  
Configuring  
Configuring iTP WebServer for Spring, 53  
Configuring NSJSP for Spring, 57  
Determining Application Parameters, 52  
Determining Maximum Capacity of NSJSP Instance, 52  
Installing Spring, 36  
Copying, 39  
Downloading, 38  
Downloading Spring, 37  
Spring Beans, 367  
Spring Getting Started, 75  
Spring JDBC API, 75  
Spring Transaction Management, 348  
Declarative Transaction Management, 348  
Programmatic Transaction Management, 348  
Spring Transaction Manager, 348  
Spring-MVC framework, 75  
`SpringInit.java`, 400  
SQL/MX Database, 181  
`sscaux`, 249  
SsionBasedLoadBalancing, 175, 245  
**Standard Binary Distribution**, 293  
`standard.jar`, 263  
startUp, 400  
static capacity, 52, 170, 240, 307  
`style.css`  
Creating `style.css`, 266  
Modifying `style.css`, 267  
subvol reference, 42  
`success.jsp`, 438  
Sun Microsystems JTA, 189  
SVN, 27

**T**

`t2jdbc`, 220  
`t4sqlmx.jar`, 220  
`t: dataList`, 238  
`t:inputDate`, 238  
`t:inputSecret`, 238  
`t:inputText`, 238  
`t:newspaperTable`, 238  
`t:panelTab`, 238  
`t:panelTabbedPane`, 238  
`t:popup`, 238  
`t:validateCreditCard`, 238  
`t:validateEmail`, 238  
`t:validateEqual`, 238  
`TabbedPane`, 234

taglibs, 97  
TANDEM\_RECEIVE\_DEPTH, 53, 54, 171, 172, 241, 242, 308, 309  
target, 161  
TemperatureConverter, 320  
TemperatureConverter Client  
    Developing, 324, 342  
        Adding Dependency JAR Files, 332, 343  
        Creating Class Files, 329  
        Creating Client Project, 324, 342  
        Creating Java Package, 328  
        Creating the Client stub, 343  
        Modifying Class File, 330  
    Running, 335  
TemperatureConverter Web Service  
    Deploying, 321, 339  
    Developing, 320  
        Creating Axis2/Java AAR File, 321  
        Creating Deployment Descriptor, 320  
        Creating Web Service Implementation Class, 320  
    Running, 323, 340  
TemperatureConverter.aar, 321, 339  
TemperatureConverter.java, 320  
TemperatureConverterClient.java, 330  
Test.java, 206  
TestManager.java, 207  
thread pool, 53, 171, 241, 308  
timestamp, 141  
Tomahawk, 234  
Transaction Advice, 354  
Transaction Management, 188, 190  
    Declarative, 66  
    Programmatic, 66  
Transaction Manager, 354  
TransactionDefinition, 348  
TransactionStatus, 348  
TransactionTemplate, 348  
TS/MP, 52, 170, 240

**U**  
URL-rewriting, 57, 175, 245, 312  
user, 42  
User Interface (UI), 252

**V**  
ViewResolver, 100  
Views  
    Creating Views, 265

**W**  
Web Archive (WAR) Distribution, 293  
Web Interface, 102  
Web Service Description Language (WSDL), 319  
WEB-INF, 97  
web.xml, 136, 423  
    Changes to web.xml, 136  
    Modifying web.xml, 84, 261  
WebApplicationContext, 423, 427  
welcome.jsp, 432, 441

Windows system, 23, 36, 75, 153, 196, 233, 252, 292, 319  
WSDL, 301, 398

**X**

XML namespace, 355

